UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MÁRCIA GONÇALVES DE OLIVEIRA

NÚCLEOS DE AVALIAÇÕES DIAGNÓSTICA E FORMATIVA PARA REGULAÇÃO DA APRENDIZAGEM DE PROGRAMAÇÃO

VITÓRIA
2013

MÁRCIA GONÇALVES DE OLIVEIRA

NÚCLEOS DE AVALIAÇÕES DIAGNÓSTICA E FORMATIVA PARA REGULAÇÃO DA APRENDIZAGEM DE PROGRAMAÇÃO

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Doutor em Engenharia Elétrica.

Orientador: Prof. Dr. Elias de Oliveira.

VITÓRIA

2013

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Oliveira, Márcia Gonçalves de, 1976 -

O48a NÚCLEOS DE AVALIAÇÕES DIAGNÓSTICA E FORMATIVA PARA REGULAÇÃO DA APRENDIZAGEM DE PROGRAMAÇÃO/ Márcia Gonçalves de Oliveira. – 2012.

130 f. : il.

Orientador: Elias Silva de Oliveira.

Tese de Doutorado – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Avaliação semi-automática. 2. Avaliação Diagnóstica. 3. Avaliação Formativa. 4. Programação. 5. Aprendizagem. I. Oliveira, Elias Silva de. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

MÁRCIA GONÇALVES DE OLIVEIRA

NÚCLEOS DE AVALIAÇÕES DIAGNÓSTICA E FORMATIVA PARA REGULAÇÃO DA APRENDIZAGEM DE PROGRAMAÇÃO

Tese de Doutorado submetida ao programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção do Grau de Doutor em Engenharia Elétrica.

Aprovada em _____ de _____ de 2013.

COMISSÃO EXAMINADORA

_____

Prof. Dr. Elias de Oliveira (Orientador)

Universidade Federal do Espírito Santo (UFES)

_____

Prof. Dr. Marcelo Eduardo Vieira Segatto (Co-rientador)

Universidade Federal do Espírito Santo (UFES)

_____

Prof[a]. Dra. Lucia Maria Martins Giraffa

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

_____

Prof[a]. Dra. Eliana Zandonade

Universidade Federal do Espírito Santo (UFES)

_____

Prof[a]. Dra. Renata Guizzardi

Universidade Federal do Espírito Santo (UFES)

_____

Prof. Dr. Evandro Ottoni Teatini Salles

Universidade Federal do Espírito Santo (UFES)

_____

Prof. Dr. Orivaldo de Lira Tavares

Universidade Federal do Espírito Santo (UFES)

”O deserto e o lugar solitário se alegrarão disto;

e o ermo exultará e florescerá como a rosa. Abundantemente florescerá...”(Isaías 35:1-2)

Agradecimentos

Agradeço a Deus por todas as dificuldades passadas e pelas vitórias alcançadas. A Ele ofereço esta tese como gesto de sincera adoração.

Agradeço aos meus pais por todos os esforços que realizaram para a minha formação e por sempre me apoiarem. Eu tenho pais excepcionais que são sempre presentes em minha vida e me orientam para o bem-viver.

Agradeço ao Professor Elias por todas as orientações que me foram dadas que muito contribuíram não só para a vida acadêmica, mas principalmente para a vida.

Agradeço à Fundação de Apoio à Pesquisa no Espírito Santo (FAPES) pelo financiamento das pesquisas desta tese.

Agradeço à professora Dra. Cristina Rangel, do Departamento de Informática da Universidade Federal do Espírito Santo, por sua relevante contribuição na realização dos experimentos desta tese.

Agradeço ao meu colega Dr. Patrick Marques Ciarelli pela parceria na realização de alguns experimentos desta tese.

Agradeço a todos os alunos, coordenadores e professores dos cursos de Ciência da Computação, Engenharia da Computação, Engenharia Elétrica e Estatística da Universidade Federal do Espírito Santo que participaram dos experimentos desta tese.

Que Deus recompense com sabedoria e felicidade a todos esses que me ajudaram a concluir essa importante etapa da minha vida.

Declarações

Este trabalho resulta das pesquisas das seguintes publicações:

- De Oliveira, Márcia Gonçalves; Marques Ciarelli, Patrick; Oliveira, Elias. Recommendation of programming activities by multi-label classification for a formative assessment of students. Expert Systems with Applications, 2013.
- Elias Oliveira, Oliveira, M.G., Ciarelli, P.M. Recommending the Right Activities based on the Needs of Each Student. In:5th. International Conference on Knowledge Discovery and Information Retrieval, 2013, Vilamoura, Algarve-Portugal.
- Oliveira, M.; Fraga, N.E.B; BARBOSA, L.; Elias Oliveira. O Sistema NADID para Avaliação de Dificuldades de Aprendizagem no Processo de Indexação de Documentos. In: IV Workshop sobre Avaliação e Acompanhamento da Aprendizagem em Ambientes Virtuais (WAVALIA), 2011, Aracaju. Anais do XXII SBIE - XVII WIE, 2011.
- BARBOSA, L. ; Oliveira, M. Metodologia ANEA para Avaliação *Online* de Lógica de Programação. In: SBIE-WIE 2011, Aracaju. Anais do XXII SBIE - XVII WIE, 2011.
- Elias Oliveira; Fraga, N.E.B; Oliveira, M.; Marchesi, R. Z. . Uma Tecnologia de Agrupamento de Respostas para Redução de Esforço de Correção de Atividades em Sistema *Online* de Apoio à Avaliação Formativa em Indexação. In: Encontro Nacional de Pesquisa em Ciência da Informação (ENANCIB), 2010, Rio de Janeiro. Anais do XI ENANCIB, 2010.

- Oliveira, M.; Elias Oliveira. Avaliações Metacognitivas *Online* para Nivelamento de Alunos. In: 15º Congresso Internacional ABED de Educação a Distância (CIAED), 2009, Fortaleza. Anais do 15º CIAED, 2009.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

API *Application Programming Interface* -Interface de Programação de Aplicativos

ASCII *American Standard Code for Information Interchange* - Código Padrão Americano para Intercâmbio de Informações

AST *Abstract Syntactic Tree* - Árvore Sintática Abstrata

AS Avaliador SQL

EPR Erro Padrão Residual

GAME *Generic Automated Marking Environment* - Ambiente Genérico de Avaliação Automática

ISIM Índice de Similaridade Interna

LMS *Learning Management Systems* - Sistemas de Gestão da Aprendizagem

MMQ Método dos Mínimos Quadrados

MRLM Modelo de Regressão Linear Múltiplo

MRLS Modelo de Regressão Linear Simples

NAD Núcleo de Avaliação Diagnóstica

NAF Núcleo de Avaliação Formativa

RNA Redes Neurais Artificiais

SBC Sociedade Brasileira de Computação

SOAP Sistema *Online* de Atividades de Programação

SQL *Structured Query Language* - Linguagem de Consulta Estruturada

STI Sistema Tutor Inteligente

TEL *Technology Enhanced Learning* - Tecnologia de Aprendizagem Melhorada -

TGS Teoria Geral dos Sistemas

TRI Teoria de Resposta ao Item

Resumo

A programação de computadores é um conhecimento considerado complexo porque para ser aprendido requer a combinação de várias habilidades cognitivas e extensa prática. Uma vez que muitos estudantes chegam às universidades mal formados nas habilidades essenciais para a resolução de problemas, é previsível que enfrentem dificuldades em desenvolver programas de computador. Dessa forma, as disciplinas de programação são as que mais respondem pelos altos índices de reprovação e até de evasão em cursos superiores de Informática. Como as turmas de programação ficam mais numerosas a cada ano pelo aumento do número de vagas e pelo adicional de alunos reprovados, demanda-se muito esforço manual e cognitivo do professor para corrigir grandes quantidades de exercícios. Além disso, em turmas com grande número de alunos, é praticamente inviável para um professor realizar um acompanhamento individual da aprendizagem de seus alunos. Com os objetivos de reduzir esforços do professor na correção de exercícios e de oferecer melhores condições de aprendizagem para alunos de cursos de programação, propomos neste trabalho um sistema de monitoramento e regulação da aprendizagem de programação. Esse sistema é formado por dois núcleos: o Núcleo de Avaliação Diagnóstica (NAD) e o Núcleo de Avaliação Formativa (NAF), implementados ambos através de técnicas de reconhecimento de padrões como o *Clustering*, a Regressão Linear e o Algoritmo de Classificação Multilabel ML-kNN. O NAD realiza as funções de correção semi-automática de exercícios e de mapeamento de perfis. O NAF, por sua vez, possui as funções de controle de estabilidade de desempenhos e de recomendação de atividades para alunos cujo perfis revelem dificuldades de aprendizagem. Neste trabalho os núcleos de avaliação foram aplicados no contexto de aprendizagem da Linguagem C. Os resultados de aplicação dos núcleos de avaliação em turmas reais de programação demonstram que é possível automaticamente reduzir o esforço de correção de exercícios em até 70% e imitar professores nas recomendações de atividades em cerca de 90% das vezes. Em resumo, a contribuição deste trabalho para o domínio da programação de computadores é oferecer um mecanismo de diagnóstico e regulação das variáveis que caracterizam a aprendizagem de programação possibilitando ao professor realizar melhor gestão da aprendizagem de seus alunos.

Palavras-chave: Avaliação Diagnóstica, Avaliação Formativa, Avaliação Semi-automática de Exercícios, Recomendação de Atividades, Aprendizagem de Programação.

Abstract

The computers programming knowledge is considered complex because it requires a combination of several cognitive skills and extensive practice to be learned. Once many students come to universities malformed in essential skills to solve problems, it is expected that they face difficulties to develop computer programs. Thus, programming disciplines are those which respond more for the high failure rates and even evasion of computers science courses. As the programming classrooms are more numerous every year by increasing the number of vacancies and the additional students who failed, manual and cognitive effort is demanded from the teacher to correct a lot of exercises. Furthermore, in classrooms with a lot of students, it is almost impossible for a teacher to make an individual learning monitoring of their students. In order to reduce teacher's efforts to correct exercises and to provide better learning conditions for students of programming courses, we propose in this thesis a system for monitoring and regulation of learning programming. This system consists of two cores: Diagnostic Assessment Core (DAC) and Formative Assessment Core (FAC), both implemented by pattern recognition technologies such as Clustering, Linear Regression and *ML-kNN* Multilabel Classification Algorithm. The DAC performs the functions of semi-automatic correction of exercises and mapping of students'profile. The FAC, in turn, has the functions of stability control of student's performances and of activities recommendation for students whose profiles indicate learning difficulties. In this thesis the assessment cores were applied in the learning context of C Language. The results of applying assesment cores in actual programming classrooms demonstrate that it is possible automatically to reduce the correction effort of exercises up to 70% and to mimic the recommendations of teachers around 90% of the times. In summary, the contribution of this work to the computer programming field is to provide a mechanism for diagnosis and adjustment of variables that characterize the learning programming allowing the teacher to make better learning management of their students.

Keywords: Diagnostic Assessment, Formative Assessment, Semiautomatic Assessment of Exercises, Recommendation of Activities, Programming Learning.

Capítulo 1

Introdução

A programação de computadores é o processo de escrever em linguagem formal instruções sequenciadas logicamente com o propósito de resolver um problema através de uma solução automatizada. Esse processo de programar envolve uma série de atividades como, por exemplo, criar uma estratégia, representar formalmente uma solução e revisar o processo de resolução de um problema. Mas, para realizar tais atividades, demanda-se daqueles que praticam a programação uma série de habilidades cognitivas (PEA; KURLAND, 1984), a começar pela complexa habilidade de compreender o problema que se pretende resolver. Por isso, a programação de computadores é considerada um conhecimento de difícil aprendizagem.

Para se ter uma ideia da complexidade de programar, para desenvolver um programa de computador, são necessárias as seguintes atividades cognitivas (PEA; KURLAND, 1984):

1. Compreender o problema
2. Planejar ou projetar uma solução
3. Escrever código do plano de solução
4. Compreender a escrita do programa
5. Depurar o programa

A atividade de compreender o problema consiste em interpretar a especificação do domínio ou o enunciado de um problema que se pretende resolver. Planejar ou projetar uma solução é esboçar uma sequência lógica de passos para resolver o problema. Escrever o código do plano de solução é uma atividade de abstração, uma vez que o plano de solução será descrito formalmente para ser processado por máquina. Compreender a escrita do programa é ser capaz de escrever o plano de solução formalmente em uma linguagem de programação, o que exige do programador um conhecimento bem formado da linguagem utilizada. A ação de depurar um programa, por sua vez, é uma habilidade de alto nível e construtiva (PEA; KURLAND, 1984) porque combina outras habilidades como a capacidade de analisar, de sintetizar informações, de reorganizar instruções, de operar logicamente e de observar em detalhes as construções de códigos para identificação e correção de erros.

Uma vez que o conhecimento de programação é dependente da combinação de tantas habilidades, os seus processos de ensino e de aprendizagem deverá contemplar o desenvolvimento dessas habilidades, se verdadeiramente objetivar formar programadores competentes.

No entanto, ao iniciar um curso de programação, professores assumem que os alunos já chegam pelo menos bem desenvolvidos nas habilidades de compreensão textual e de raciocinar logicamente. No decorrer do curso, porém, as dificuldades de aprendizagem evidenciam as deficiências nessas habilidades. Demonstra-se, inclusive, que as dificuldades de aprendizagem de programação têm uma forte relação entre as habilidades de compreensão das construções de uma linguagem de programação e o desenvolvimento de uma seqüência lógica para resolver um problema (EBRAHIMI, 1994).

Mas, sendo os conteúdos de disciplinas de programação muito extensos, os professores priorizam concluir o conteúdo programático do curso em vez de treinar as habilidades fundamentais para a aprendizagem bem sucedida da programação.

De acordo com Antunes (2001), o desenvolvimento das habilidades de um domínio do conhecimento pode ser realizado a partir dos próprios conteúdos ministrados em um curso. No caso da programação de computadores, entendemos que isso pode ser feito através da aplicação de tarefas, uma vez que a programação se aprende com a prática, e a prática

melhora a taxa de processamento de informações de um aprendiz (ANDERSON, 2000). Dessa forma, a ideia é que tarefas sejam planejadas com atividades de programação que reforcem o domínio de conteúdos e ao mesmo tempo favoreçam o desenvolvimento das habilidades fundamentais no processo de programar. Nesse caso, os desempenhos dos alunos nas tarefas podem ser mapeados em variáveis ou componentes de habilidades que possam ser monitoradas e controladas de forma que se alcancem êxitos na aprendizagem de programação.

Para Anderson (2000), monitorando cuidadosamente as componentes de uma habilidade, é possível levar estudantes rapidamente ao domínio de habilidades mais complexas.

Esse controle das componentes das habilidades envolvidas no processo de programar pode ser realizado através da avaliação, que é um mecanismo de retorno para sintonizar aprendizagens (NEVADO; CARVALHO; MENEZES, 2007). Esse mecanismo de retorno pode ser representado pelos modelos de avaliação diagnóstica e formativa (PERRENOUD, 1999). A avaliação diagnóstica deve ter o papel de identificar habilidades e dificuldades de aprendizagem além de reconhecer perfis de alunos. A avaliação formativa, por sua vez, deve consistir de *feedbacks* e ajustes nos processos de ensino e de aprendizagem para alcançar objetivos traçados (PERRENOUD, 1999; BALLESTER, 2003).

Para avaliar, entretanto, é necessário obter as medidas das variáveis de avaliação que podem ser, por exemplo, as notas atribuídas às tarefas por professores.

Uma vez que as turmas de programação são geralmente compostas por uma grande quantidade de alunos, é inviável para um professor atribuir notas a uma grande quantidade de exercícios, trabalhos e provas de programação, pois isso demanda-lhe grande esforço físico e cognitivo. Por não terem condições de corrigir tantos exercícios, os professores acabam reduzindo a quantidade de programas a serem entregues pelos alunos (PIMENTEL; FRANCA; OMAR, 2003). Em outros casos, os professores aplicam e corrigem trabalhos e provas, mas as listas de exercícios, embora sejam aplicadas, em geral, não são corrigidas.

A deficiência nas habilidades envolvidas no processo de programar, a pouca prática de exercícios e a carência de *feedbacnks* imediatos nas atividades comprometem diretamente a aprendizagem de programação. Uma possível evidência disso é que as disciplinas de

programação são as que mais respondem pelos altos índices de reprovação e até de evasão em cursos superiores de informática.

Considerando o prejuízo que as reprovações e desistências ocasionam em muitas instituições e a complexidade dos processos de ensino e de aprendizagem de programação, vários estudos chamam a atenção para esses problemas e propõem tecnologias educacionais como apoio aos processos de ensino e de aprendizagem de programação (PILLAY, 2003; PIMENTEL; FRANCA; OMAR, 2003; POWERS et al., 2006; PIMENTEL et al., 2007; BARBOSA; OLIVEIRA, 2011).

As avaliações diagnóstica e formativa podem ser boas estratégias contra o fracasso na aprendizagem de programação, mas são difíceis de serem realizadas manualmente em turmas com um grande número de alunos (BALLESTER, 2003). No entanto, várias tecnologias inteligentes, principalmente para *web*, têm sido desenvolvidas como o apoio a esses modelos de avaliações e têm promovido êxitos de aprendizagem (ANDERSON, 2000; PACHECO, 2005; MAZZA; DIMITROVA, 2007; CASTELLANO et al., 2007; OLIVEIRA; OLIVEIRA, 2008a). Entre essas tecnologias estão os sistemas de reconhecimento de padrões, que têm sido amplamente aplicados como suporte eficaz às avaliações diagnóstica e formativa (CHEN; CHEN, 2005; CASTELLANO et al., 2007; PIMENTEL et al., 2007; MARINAGI; KABURLASOS, 2006; OLIVEIRA; OLIVEIRA, 2008a; OLIVEIRA; CIARELLI; OLIVEIRA, 2013).

Sabendo que a avaliação diagnóstica e formativa são estratégias que melhoram a aprendizagem (PERRENOUD, 1999), desenvolvemos em tecnologias computacionais da *web* e de reconhecimento de padrões, uma metodologia para monitorar e regular as componentes de aprendizagem de programação.

Para realizar a avaliação diagnóstica, desenvolvemos o Núcleo de Avaliação Diagnóstica (NAD) que implementa as funcionalidades de avaliação semi-automática de exercícios de programação e de mapeamento de perfis de aprendizagem por componentes de habilidades. No NAD, utilizamos algoritmos de *clustering* (JAIN; MURTY; FLYNN, 1999) para identificação de perfis e a técnica de regressão linear, para a predição de notas de exercícios (HAIR; TATHAM; BLACK, 1998).

Para realizar a avaliação formativa, desenvolvemos o Núcleo de Avaliação Formativa (NAF), que implementa as funcionalidades de controle de estabilidade por realimentação dos estados de aprendizagem e a recomendação de atividades de programação conforme os perfis de aprendizagem. No NAF, a tarefa de recomendação de atividades é reformulada em uma tarefa de classificação *multilabel* (ou multirrotulada) em que, a cada perfil de aluno, são associadas uma ou mais classes de atividades de programação a serem recomendadas.

Em resumo, a proposta metodológica deste trabalho para monitoramento e controle da aprendizagem de programação consiste em representar perfis de alunos por componentes de habilidades, predizer notas de exercícios conforme esses perfis e recomendar-lhes as atividades de programação mais adequadas.

Os objetivos dessa proposta são reduzir esforços de professores na correção de atividades de programação e oferecer mecanismos de diagnóstico e regulação da aprendizagem que auxiliem professores na gestão das aprendizagens de seus alunos.

Este capítulo está organizado conforme a ordem a seguir. Na Seção 1.1, descrevemos alguns problemas dos processos de ensino e de aprendizagem de programação que pretendemos contemplar com a nossa proposta metodológica. Na Seção 1.2, listamos os objetivos deste trabalho. Na Seção 1.3, apresentamos nossa metodologia de gestão dos processos de ensino e de aprendizagem de programação. Na Seção 1.4, concluimos apresentando as contribuições deste trabalho que o consolidam como uma tese de doutorado. Na Seção 1.5, explicamos como este trabalho é organizado como um todo.

1.1 O problema

A programação de computadores é considerada um conhecimento de difícil aprendizagem porque requer a combinação de uma série de habilidades como a compreensão textual, o raciocínio lógico, a observação de detalhes e a abstração de informações. Como o desenvolvimento de programas de computador é um complexo processo de resolver problemas (MENEZES et al., 2008) e nem sempre os estudantes são bem desenvolvidos

nessas habilidades, a aprendizagem de programação requer extensa prática de exercícios e que essa prática favoreça o desenvolvimento das habilidades de programar.

A prática da programação, para ser bem sucedida, no entanto, precisa ser assistida pelo professor para que ele possa intervir nos pontos de dificuldades demonstrados pelos seus alunos. Mas avaliar, diagnosticar os problemas de aprendizagem e dar *feedbacks* são tarefas que demandam grande esforço manual e cognitivo do professor, a começar pelo trabalho de corrigir muitas atividades em turmas com grande quantidade de alunos. Dessa forma, não conseguindo atender à demanda de fornecer *feedbacks* imediatos para os exercícios, a tendência é que professores reduzam cada vez mais o número de exercícios aplicados em suas turmas, o que afeta diretamente a aprendizagem dos alunos (IHANTOLA et al., 2010).

Atentando para a complexidade da aprendizagem de programação, nos últimos anos, muitas tecnologias e metodologias têm sido desenvolvidas e propostas para apoiar os processos de ensino e de aprendizagem de programação. Hoje temos sofisticadas tecnologias para apresentar conteúdos de programação (POWERS et al., 2006; ANDERSON; MCLOUGHLIN, 2007), para instruir (PILLAY, 2003), para predizer comportamentos (MAVRIKIS, 2010), para dar instruções e exercícios personalizados de acordo com os perfis de alunos (CASTELLANO et al., 2007; PIMENTEL et al., 2007) e até para avaliação automática de exercícios de programação (NAUDE; GREYLING; VOGTS, 2010; SOUZA; MALDONADO; BARBOSA, 2011; WANG et al., 2011; ROMLI; SULAIMAN; ZAMLI, 2010). Mas carecemos de tecnologias que de fato ofereçam uma prática assistida, isto é, planejada, monitorada, controlada e com *feedback* imediato no domínio de conhecimento da programação de computadores.

É preciso, desse modo, criar tecnologias para regular o processo de aprendizagem dos alunos para que êxitos de aprendizagem sejam de fato alcançados. Isso pode ser feito através da avaliação, que é um método de adquirir e processar evidências necessárias para melhorar o ensino (BLOOM; HASTINGS; MADAUS, 1975) e um mecanismo de retorno para ajustar aprendizagens (NEVADO; CARVALHO; MENEZES, 2007).

A grande quantidade de alunos com diversificados perfis de aprendizagem em turmas de programação, as deficiências de muitos alunos nas habilidades envolvidas na prática de

programar e as dificuldades de professores fornecerem *feedbacks* imediatos e acompanharem individualmente a aprendizagem de seus alunos são problemas que devem ser contemplados pelas novas tecnologias. Isso porque esses problemas respondem pelos altos índices de reprovação e desistências em cursos de programação, o que implica em má formação de profissionais da informática e em prejuízos para instituições que oferecem esses cursos.

1.2 Objetivos

As pesquisas apresentadas neste trabalho têm como objetivos gerais reduzir esforços do professor na correção de exercícios e oferecer melhores condições de aprendizagem aos alunos de cursos de programação.

Para o professor, oferecemos um sistema de avaliação semi-automática que seleciona um conjunto mínimo de exercícios para serem pontuados pelo professor e, a partir desses exercícios, prediz as notas para outros exercícios semelhantes.

Para o aluno, oferecemos um modelo de avaliações diagnóstica e formativa apoiado por tecnologias de reconhecimento de padrões que identificam perfis de alunos e recomendam-lhes atividades de acordo com as suas dificuldades de aprendizagem.

Sabendo que as avaliações diagnóstica e formativa são estratégias que melhoram a aprendizagem (PERRENOUD, 1999; ANDERSON, 2000; OLIVEIRA; OLIVEIRA, 2008a), desenvolvemos uma metodologia de avaliação para o domínio de conhecimento da programação de computadores que, apoiada por tecnologias computacionais, reúne os seguintes objetivos específicos:

1. Representar componentes de habilidades por desempenhos em atividades
2. Corrigir de forma semi-automática exercícios de programação
3. Mapear alunos em perfis de aprendizagem
4. Descobrir estados de aprendizagem dos alunos

5. Realimentar estados de aprendizagem dos alunos

As Funções 1-3 são ações de avaliação diagnóstica e as Funções 4 e 5 são ações de avaliação formativa. A Função 1 significa transformar conteúdos em instrumentos do aprimoramento das habilidades de programação (ANTUNES, 2001). A ideia é cada atividades de programação seja contemplada como um conjunto de componentes de habilidades, isto é, variáveis, que informem êxitos ou dificuldades de aprendizagem em diferentes conteúdos de programação.

A Função 2 consiste em analisar programas de computador desenvolvidos por alunos e, tendo como referências algumas notas atribuídas por professor a outros exemplos de programas, predizer notas para esses exercícios.

A Função 3 consiste em classificar alunos em perfis de aprendizagem conforme os desempenhos obtidos em cada componente de habilidade.

A Função 4, a partir das medidas das componentes de habilidades obtidas por um aluno em um exercício, define um estado de aprendizagem que informa se, para cada componente de habilidade, foram alcançados êxitos ou não de aprendizagem.

A Função 5, de acordo com o estado de aprendizagem informado pela Função 4, deve recomendar atividades de programação continuamente até que o conjunto de componentes de habilidades, isto é, os estados de aprendizagem, alcancem um patamar de desempenhos considerados satisfatórios para a aprendizagem.

1.3 Metodologia

Para alcançar os objetivos apresentados na seção anterior, desenvolvemos uma metodologia de avaliações diagnóstica e formativa para monitorar e realimentar o processo de aprendizagem de programação.

A avaliação diagnóstica, segundo Haydt (2002), refere-se à identificação do nível inicial de conhecimento dos alunos em uma área do conhecimento e à verificação das características

individuais e grupais desses alunos no processo de aprendizagem. Uma vez identificadas as características individuais, a avaliação diagnóstica deve reconhecer habilidades e classes de dificuldades de aprendizagem bem como identificar classes de perfis de alunos.

A avaliação formativa consiste de *feedbacks* e ajustes nos processos de ensino e de aprendizagem realizados para alcançar objetivos traçados (PERRENOUD, 1999; BALLESTER, 2003). A ideia de avaliação formativa leva o professor a observar melhor seus alunos, a compreender melhor seu funcionamento, de modo a ajustar e individualizar suas intervenções (PERRENOUD, 1999). A avaliação formativa, estando ligada à gestão e à otimização das aprendizagens, deve perseguir os seguintes objetivos: a regulação da aprendizagem, a gestão dos erros e a consolidação dos êxitos (PERRENOUD, 1999).

Para implementar essas funcionalidades de avaliações diagnóstica e formativa, desenvolvemos um sistema de avaliação composto por dois núcleos: o Núcleo de Avaliação Diagnóstica (NAD) e o Núcleo de Avaliação Formativa (NAF), ambos implementados em tecnologias de reconhecimento de padrões como o *clustering*, a regressão linear e algoritmos de classificação (DUDA; HART; STORK, 2001; ZHANG; ZHOU, 2007; MANNING; RAGHAVAN; SCHUTZE, 2008; BAEZA-YATES; RIBEIRO-NETO, 1999; JAIN; MURTY; FLYNN, 1999; HAIR; TATHAM; BLACK, 1998).

Os nossos padrões de avaliação são as atividades de programação em Linguagem de programação C desenvolvidas por alunos, que são representados por seus desempenhos em atividades.

Escolhemos a Linguagem C para aplicação desta metodologia porque é uma linguagem de programação adotada em muitos cursos de programação introdutória. No entanto, a metodologia desenvolvida neste trabalho pode ser estendida para outras linguagens de programação, especificamente de paradigma de programação estruturado, como a Linguagem C (SCHILDT; MAYER, 2006).

O NAD realiza as funções de correção semi-automática de exercícios de programação e de mapeamento de alunos em perfis de aprendizagem de acordo com suas componentes de habilidades. A correção semi-automática é realizada pela técnicas de *clustering* e por modelos de regressão linear. O *clustering* reúne em grupos os exercícios de programação

com soluções semelhantes e indica que características explicam as semelhanças entre esses padrões. O modelo de regressão linear prediz as notas dos padrões em cada agrupamento com base nessas características e em exemplos de exercícios pontuados por um professor. Já o mapeamento dos alunos com base em seus desempenhos é também realizado por *clustering*, que identifica perfis de alunos e classes de dificuldades de aprendizagem.

O NAF, por sua vez, possui as funções de controle de estabilidade (ou nivelamento) de aprendizagens e de recomendação de atividades para alunos que apresentam dificuldades de aprendizagem, conforme os estados de aprendizagem apontados pelas componentes de habilidades.

O NAD e o NAF comunicam-se com o mundo externo através do Sistema *Online* de Atividades de Programação (SOAP), que é um sistema *web* que possibilita ao professor disponibilizar tarefas para suas turmas e, ao aluno, realizar submissões de exercícios. Os exercícios submetidos, que são programas de computador em Linguagem C, são compilados e executados em uma infraestrutura de máquinas virtuais para execução paralela e segura desses programas. Para acompanhamento da aprendizagem dos alunos, o SOAP emite relatórios de diagnósticos do processo de aprendizagem e da execução dos programas submetidos.

A Figura 1.1 representa a nossa proposta metodológica. Essa arquitetura mostra como o NAD e o NAF interagem com o SOAP para monitoramento e controle da aprendizagem de programação.

De acordo com a Figura 1.1, as atividades dos alunos, reunidas em tarefas, são recebidas e processadas pelos núcleos NAD e NAF. O NAD processa essas atividades gerando planilhas e mapas de perfis de aprendizagem que são entregues ao NAF na forma de componentes de habilidades C_i e, ao SOAP, na forma de relatórios. Esses relatórios são visualizados por professores para que tenham um acompanhamento e melhor gestão do processo de aprendizagem de seus alunos.

O NAF, com base no diagnóstico dos alunos fornecido pelo NAD, sugere atividades de recuperação para que os alunos com dificuldades possam melhorar seus desempenhos. Essas atividades sugeridas são apresentadas ao aluno através do sistema SOAP.

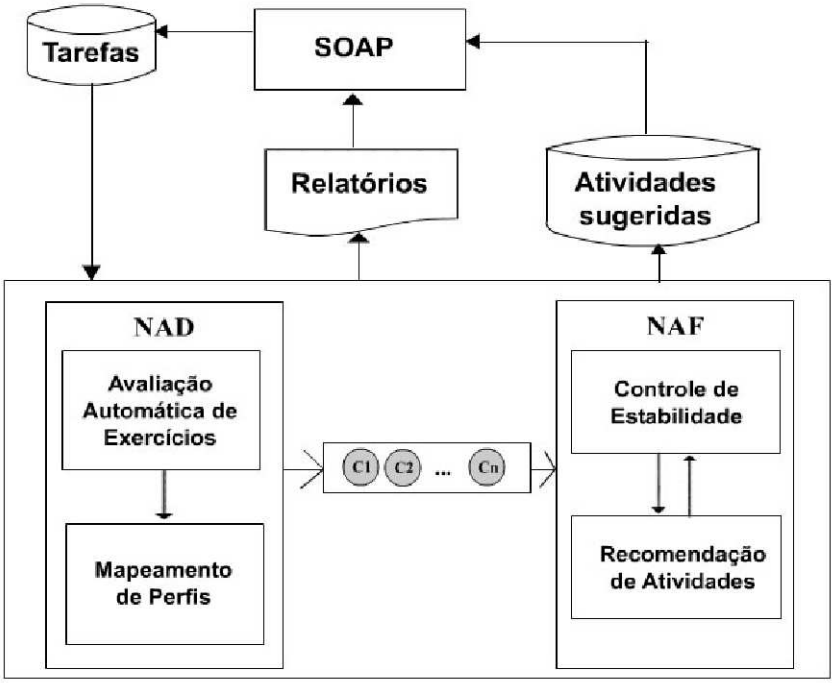

Figura 1.1: Arquitetura de avaliação semi-automática da aprendizagem de programação

Em síntese, a nossa metodologia combina as técnicas de reconhecimento de padrões de aprendizagem supervisionada e não-supervisionada para implementar uma estratégia de gestão da aprendizagem de programação por avaliações diagnóstica e formativa. Essa estratégia possibilita predizer notas, classificar alunos e recomendar-lhes atividades conforme os seus perfis. Os detalhes de implementação dos núcleos de avaliação são apresentados no Capítulo 6 e do sistema SOAP, no Capítulo 7.

1.4 Contribuições

As principais contribuições deste trabalho para os domínios de conhecimento de reconhecimento de padrões e da aprendizagem de programação são as seguintes:

1. O nosso modelo de avaliação aponta para uma avaliação fina, do tipo clínica e dinâmica para captar a multidimensionalidade do objeto aprendido para se coletar uma pluraridade de informações (RAPHAEL; CARRARA, 2002).

2. Desenvolvemos uma estratégia inovadora de avaliação semi-automática de exercícios para o domínio de conhecimento da programação de computadores (NAUDE; GREYLING; VOGTS, 2010).
3. Desenvolvemos uma estratégia semi-automática de recomendação de atividades que transforma a tarefa de recomendação em uma tarefa de classificação *multilabel* tendo como referência recomendações de um especialista humano (OLIVEIRA; CIARELLI; OLIVEIRA, 2013).
4. Oferecemos uma estratégia de avaliação semi-automática que possibilita a prática assistida da programação com *feedbacks* mais rápidos em turmas com grande quantidade de alunos e onde se aplicam grandes quantidades de exercícios.
5. Combinamos técnicas de reconhecimento de padrões de abordagens não-supervisionada e supervisionada para predição de notas e recomendação de atividades de programação em conformidade com padrões de professores.
6. Formamos uma ampla base de exercícios de programação resolvidos por alunos e avaliados por professores para realização de experimentos de avaliação e de recomendação semi-automática de atividades.

1.5 Organização do trabalho

Este trabalho está organizado conforme a ordem a seguir. No Capítulo 2, explicamos a fundamentação teórica deste trabalho baseada na Teoria Geral dos Sistemas (TGS) e nas teorias educacionais de avaliações diagnóstica e formativa. No Capítulo 3, relatamos o estado da arte em avaliação automática de programação. No Capítulo 4, apresentamos uma revisão de literatura dos sistemas de recomendação, como são aplicados em contextos educacionais e os trabalhos relacionados. No Capítulo 5, destacamos as técnicas de Reconhecimento de Padrões aplicadas neste trabalho. No Capítulo 6, descrevemos os núcleos de avaliações diagnóstica e formativa NAD e NAF, respectivamente. No Capítulo 7, apresentamos o sistema SOAP. No Capítulo 8, detalhamos os experimentos e os resultados do NAD. No Capítulo 9, relatamos os experimentos e resultados do NAF. No Capítulo 10, concluímos com as considerações finais e propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

A avaliação é um método de adquirir e processar evidências necessárias para melhorar o ensino e a aprendizagem (BLOOM; HASTINGS; MADAUS, 1975; BALLESTER, 2003) e um mecanismo de retorno para sintonizar aprendizagens (NEVADO; CARVALHO; MENEZES, 2007) constituído de três fases: coleta de informações, análise e tomada de decisão (BALLESTER, 2003).

A fase de coleta de informações consiste em reunir informações do processo de aprendizagem de alunos. Essas informações podem ser obtidas através de atividades, interações e comportamentos de alunos em seu processo de aprendizagem. A análise dessas informações envolve relacionar, quantificar, classificar, predizer ou inferir informações para diagnosticar um processo de aprendizagem. Já a tomada de decisão, baseada na análise de informações, consiste de ações de reorientação do ensino e de remediação do processo de aprendizagem com o objetivo de guiar alunos para um estado estável de êxito.

Mas, para que a avaliação de fato agregue qualidade ao processo de ensino e de aprendizagem, ela deve ter algumas das características próprias de toda avaliação autêntica (TARDIF, 1996; BALLESTER, 2003):

- A avaliação não inclui senão tarefas contextualizadas
- A avaliação aborda problemas complexos.
- A avaliação deve contribuir para que os estudantes desenvolvam mais competências

- A tarefa e suas exigências são conhecidas antes da situação de avaliação
- As informações extraídas da avaliação devem considerar as aptidões dos estudantes, seus conhecimentos anteriores e seu grau atual de domínio das competências visadas.
- Os mesmos procedimentos de avaliação são exigidos a todos os alunos e o apoio necessário deve estar disponível para aqueles que têm mais dificuldades.

A avaliação, como instrumento adequado para regular e adaptar o ensino às necessidades e dificuldades de estudantes, deve cumprir três funções didático-pedagógicas: diagnóstica, formativa e somativa (BALLESTER, 2003).

A avaliação diagnóstica deve ter o papel de identificar, através de um conjunto de variáveis representantes de perfis de alunos, habilidades e dificuldades de aprendizagem bem como classificar perfis conforme as medidas das variáveis de avaliação. A avaliação formativa, por sua vez, deve consistir de *feedbacks*, intervenções e ajustes no processo de ensino-aprendizagem para alcançar objetivos traçados (PERRENOUD, 1999; BALLESTER, 2003). Já a avaliação somativa tem as funções de classificação e orientação de alunos, de reforçar êxitos e verificar resultados de um processo aprendizagem (BALLESTER, 2003; OLIVEIRA; OLIVEIRA, 2008a).

Para o cognitivismo, a aprendizagem é contemplada não a partir de comportamentos observáveis, mas sim, do planejamento da ação e do tratamento das informações coletadas (PERRAUDEAU-DELBRIEL, 2009). O processo de aprendizagem pode, sob esse ponto de vista, ser tratado como um sistema que recebe informações como entradas através do ensino, gera saídas e é realimentado pelo planejamento da ação em função das informações coletadas nas saídas geradas.

A abordagem sistêmica, que é uma aplicação da Teoria Geral dos Sistemas (TGS) (L.BERTALANFY; VON, 1973), trata uma organização como um sistema composto de partes que interagem de forma cooperativa e organizada para alcançar um objetivo e que se realimenta a partir das informações que produz. Assim, um sistema, do ponto de vista organísmico aplicável em qualquer domínio do conhecimento, tem como componentes genéricos as entradas, os mecanismos de processamento, as saídas, a realimentação, o

ambiente e o limite entre um sistema e o seu ambiente (L.BERTALANFY; VON, 1973; REYNOLDS; STAIR, 2010).

Uma vez que a aprendizagem está no centro de uma rede de variáveis cujas interdependências agem sobre os efeitos (PERRAUDEAU-DELBRIEL, 2009), de acordo com a abordagem sistêmica, a avaliação pode ser considerada um mecanismo realimentador dessa rede de variáveis. Esse mecanismo reuniria, desse modo, um conjunto de ações de avaliação diagnóstica e formativa realizadas por um professor para ajustar o processo de aprendizagem até que este alcance um estado de êxito. São exemplos de ações de avaliação de um professor para realimentação de um processo de aprendizagem (BALLESTER, 2003):

- Realizar um diagnóstico inicial
- Readaptar o ensino em função dos resultados de diagnóstico da aprendizagem
- Garantir uma sequência adequada do processo de aprendizagem
- Detectar erros e dificuldades de aprendizagem
- Criar mecanismos de regulação
- Reforçar êxitos;
- Verificar resultados

Os estudos de Perrenoud (2001) sustentam essa abordagem sistêmica da avaliação para promover êxitos de aprendizagem, reforçando que a maioria das pessoas é capaz de aprender coisas complexas, desde que ajustem-se constantemente às exigências, aos objetivos próximos, às situações didáticas e aos seus recursos disponíveis. Entendemos, dessa forma, que a avaliação não deve ser um fim no processo de ensino e de aprendizagem, mas um meio de conduzi-lo em favor da aprendizagem.

Com essa visão sistêmica, defendemos que monitorando cuidadosamente os componentes individuais de uma habilidade e fornecendo *feedbacks*, é possível conduzir os estudantes ao rápido domínio de habilidades complexas (ANDERSON, 2000).

Mas, embora a utilização da abordagem sistêmica na prática avaliativa garanta êxitos de aprendizagem (PERRENOUD, 1999; ANDERSON, 2000; OLIVEIRA, 2009), na realidade,

é praticamente inviável (BALLESTER, 2003). Isso porque é muito difícil diagnosticar as causas das dificuldades de aprendizagem, enfrentar uma grande quantidades de tarefas demandadas pela avaliação formativa para turmas com grande quantidade de alunos e acompanhar a aprendizagem individual quando há ampla diversidade de perfis de alunos em uma turma (BALLESTER, 2003; OLIVEIRA; ZANDONADE; OLIVEIRA, 2008).

Dessa forma, o acompanhamento individualizado e contínuo dos aprendizes por avaliações diagnóstica e formativa, principalmente em turmas com um número elevado de alunos, só será possível se for auxiliado por computadores através de sistemas inteligentes (PIMENTEL; FRANCA; OMAR, 2003).

Nos últimos anos, várias tecnologias computacionais inteligentes têm sido desenvolvidas como apoio aos modelos de avaliações diagnóstica e formativa e têm promovido sucessos de aprendizagem (ANDERSON, 2000; XU; CHEE, 2003; PACHECO, 2005; MAZZA; DIMITROVA, 2007; HUNTER et al., 2013). As técnicas de reconhecimento de padrões também têm sido reconhecidas e aplicadas como suporte eficaz às avaliações diagnóstica e formativa (CHEN; CHEN, 2005; CASTELLANO et al., 2007; PIMENTEL et al., 2007; MARINAGI; KABURLASOS, 2006; OLIVEIRA; OLIVEIRA, 2008a; GUO et al., 2013).

Este trabalho apresenta uma tecnologia inteligente de apoio às avaliações diagnóstica e formativa para o domínio de aprendizagem de programação de computadores. Através da avaliação diagnóstica foram coletadas informações de atividades de programação desenvolvidas por alunos para realizar mapeamento de perfis de aprendizagem. Analisando as informações dos perfis formados e recomendando atividades de acordo com esses perfis, foi realizada a avaliação formativa.

Para descrever a fundamentação teórica deste trabalho, este capítulo está organizado conforme a ordem a seguir. Na Seção 2.1 descrevemos a abordagem sistêmica da avaliação fundamentada na Teoria Geral dos Sistemas. Na Seção 2.2, apresentamos os fundamentos teóricos da avaliação diagnóstica. Na Seção 2.3, apresentamos os fundamentos teóricos da avaliação formativa. Na Seção 2.4, fazemos um relato das tecnologias que implementam as ideias de avaliações dignóstica e formativa. Na Seção 2.5, concluimos com as considerações finais da fundamentação teórica deste trabalho.

2.1 A abordagem sistêmica da avaliação

A TGS foi elaborada por Ludwig Bertalanffy em 1936 e apresentada em 1937 na Universidade de Chicago. A ideia da TGS é integrar a ciência por um modelo genérico de sistema aplicável em diferentes áreas do conhecimento (L.BERTALANFY; VON, 1973). Ao contrário da visão mecanicista que trata um sistema como partes e processos, a TGS busca o ponto de vista organísmico, isto é, que um sistema é um conjunto de partes integradas e organizadas que, interagindo entre si e o ambiente, trabalham em função de um objetivo comum (L.BERTALANFY; VON, 1973).

De acordo com Reynolds e Stair (2010), a função básica de um sistema é converter seus insumos (materiais, energia, trabalho, informações) retirados de seu ambiente em produtos (bens, serviços, informações). Para realizar essa função, um sistema têm como componentes genéricos as entradas, o processamento, as saídas, a realimentação, o ambiente e o limite entre o sistema e o ambiente (L.BERTALANFY; VON, 1973; REYNOLDS; STAIR, 2010). Uma ilustração desses componentes é mostrada na Figura 2.1.

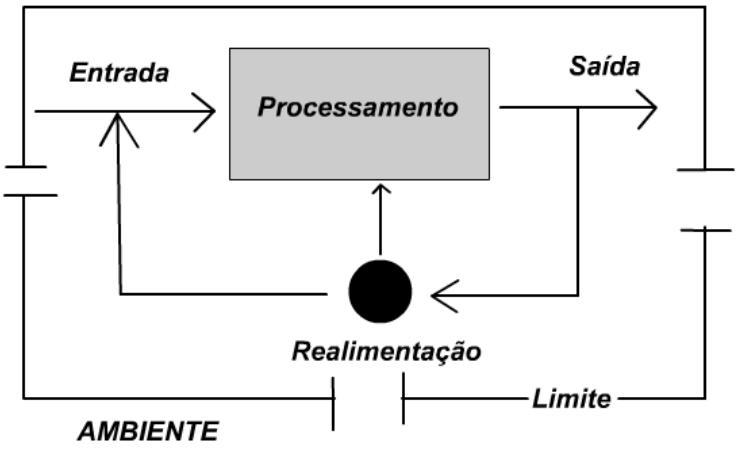

Figura 2.1: Os componentes genéricos de um sistema

Na Figura 2.1 a *entrada* do sistema, de acordo com Reynolds e Stair (2010), é uma atividade de captar e reunir dados brutos. O *processamento* é o ato de converter ou processar dados brutos em informações úteis através de cálculos, operações de comparação, de execução de ações e armazenamento de dados. A *saída* envolve a produção de informações úteis, normalmente na forma de documentos e relatórios, mas a saída de um sistema pode tornar-se entrada de outro. A *realimentação*, por sua vez, é a informação gerada pelo sistema

utilizada para fazer mudanças na entrada ou nas atividades de processamento (REYNOLDS; STAIR, 2010).

Esses componentes do sistema, segundo Reynolds e Stair (2010), são definidos como ações e não como objetos de um sistema. Preferimos essa abordagem porque entendemos que o processo de aprendizagem, o ensino e a avaliação, vistos como partes de um sistema, representam de fato ações.

Dessa forma, a avaliação, segundo a abordagem sistêmica e as definições de Reynolds e Stair (2010), representaria a realimentação de um processo de aprendizagem, cuja entrada é o ensino e as saídas são os desempenhos, que podem representar a própria aprendizagem como resultado de um processo (PERRAUDEAU-DELBRIEL, 2009). Dessa forma, os processos que se pressupõem de uma aprendizagem são observáveis especialmente pela modificação do desempenho (PERRAUDEAU-DELBRIEL, 2009). A Figura 2.2 ilustra o ensino, o processo de aprendizagem e a avaliação como partes de um sistema.

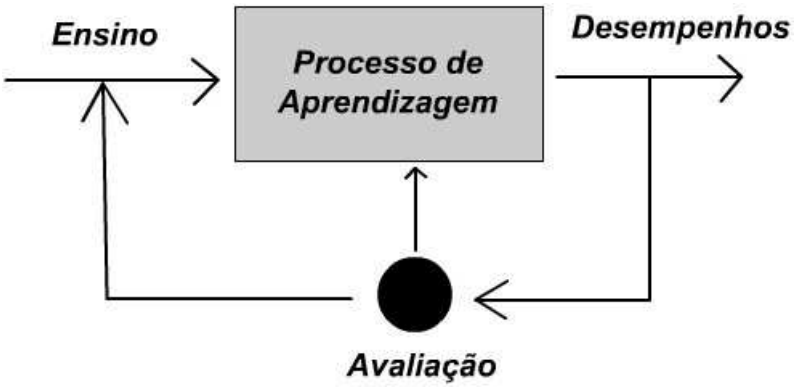

Figura 2.2: Avaliação sistêmica

De acordo com a Figura 2.2, é através da avaliação que as informações de saída, que são os desempenhos, são analisadas e o ensino e as atividades de aprendizagem são reorientados. Nesse caso, a avaliação diagnóstica seria responsável pela coleta e análise de informações de um processo de aprendizagem representado por uma rede de variáveis de avaliação (PERRAUDEAU-DELBRIEL, 2009). A avaliação formativa, por sua vez, realizaria os ajustes nessas variáveis através do ensino e de intervenções na prática de exercícios no processo de aprendizagem.

Em um processo de aprendizagem, segundo Anderson (2000), as diferenças individuais

entre estudantes estão relacionadas à quantidade de informações que eles absorveram e não necessariamente às suas capacidades inatas. Além disso, a taxa de processamento de informações pode ser melhorada com a prática de exercícios. Por isso, segundo a abordagem sistêmica, o processo de aprendizagem pode ser ajustado continuamente por mecanismos de avaliações diagnóstica e formativa através da aplicação de atividades planejadas que ofereçam informações sobre a aprendizagem e dêem condições para que esta seja melhorada (PERRENOUD, 1999; ANDERSON, 2000; BALLESTER, 2003).

Na Figura 2.3 apresentamos o modelo de avaliação (PERRENOUD, 1999; BALLESTER, 2003; OLIVEIRA; ZANDONADE; OLIVEIRA, 2008) em que se baseia a nossa proposta metodológica a ser aplicada no processo de aprendizagem de programação de computadores.

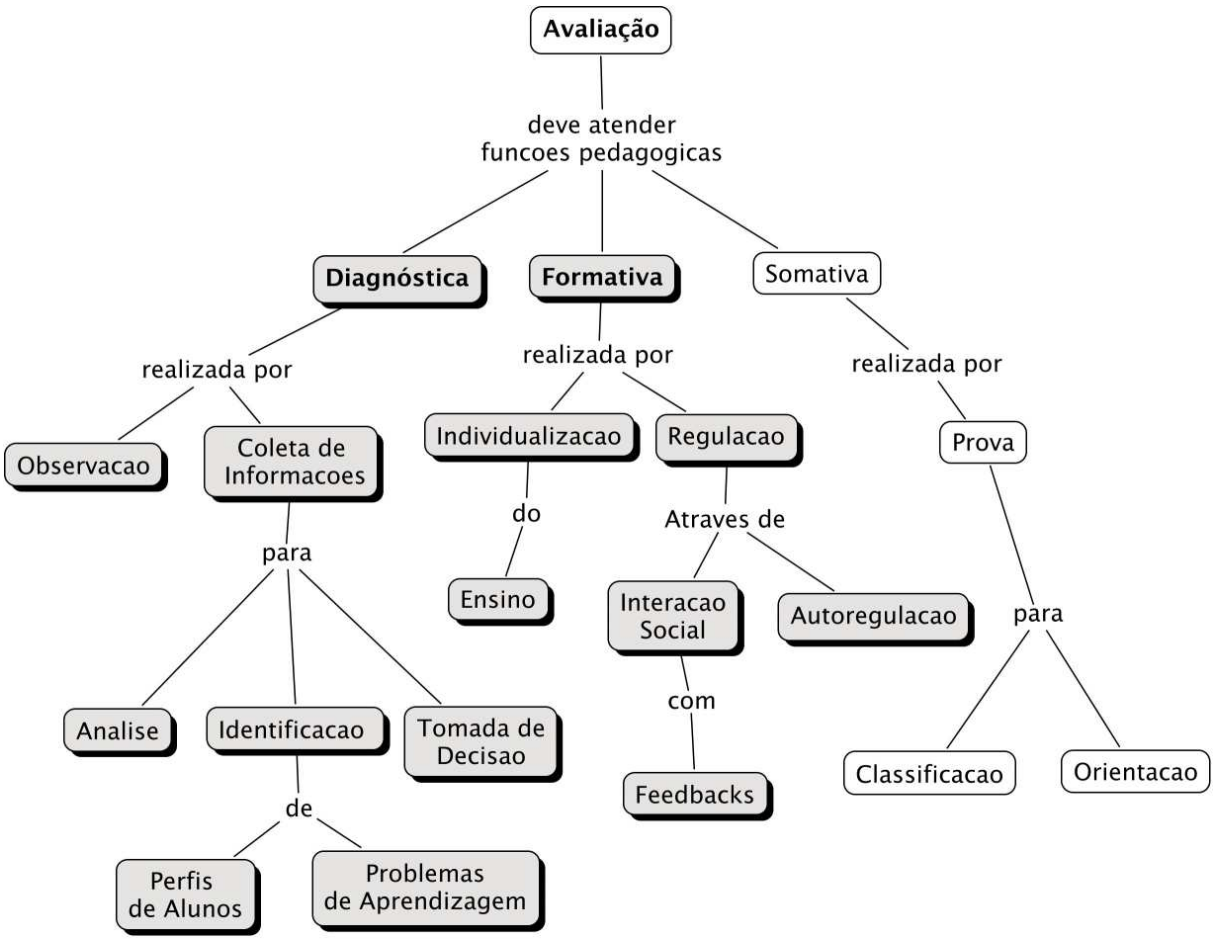

Figura 2.3: Modelo de Avaliação (OLIVEIRA; ZANDONADE; OLIVEIRA, 2008)

No Modelo de Avaliação em mapa conceitual da Figura 2.3, são apresentados, no

primeiro nível, as funções pedagógicas da avaliação que são as avaliações diagnóstica, formativa e somativa. No nível seguinte, apontamos os conceitos que explicam como são realizadas essas três avaliações. Nos níveis seguintes, apresentamos as finalidades das avaliações diagnóstica, formativa e somativa em um processo de ensino e de aprendizagem.

Neste trabalho, a nossa proposta visa alcançar as áreas escurecidas do modelo de avaliação da Figura 2.3 correspondentes às avaliações diagnóstica e formativa, áreas essas em geral negligenciadas em vários processos avaliativos da aprendizagem. As avaliações diagnóstica e formativa são apresentadas nas seções a seguir.

2.2 A avaliação diagnóstica

A função diagnóstica da avaliação, segundo Haydt (2002), refere-se à identificação do nível inicial de conhecimento dos alunos em uma área do conhecimento e à verificação das características individuais e grupais desses alunos em um processo educacional. Para Haydt (2002), a avaliação diagnóstica deve ser realizada no início de um curso a fim de verificar conhecimentos prévios, habilidades e dificuldades de aprendizagem (ANDERSON, 2000).

No entanto, como a avaliação diagnóstica tem a finalidade de comparar os comportamentos finais dos alunos com um comportamento desejado (MIZUKAMI, 1986), entendemos que a avaliação diagnóstica deve ser realizada continuamente em um processo de aprendizagem. Isso para que o ensino seja melhorado ao longo de um curso pela análise do que são os elementos individuais levantados por uma avaliação diagnóstica para tomar decisões e para remediar problemas de aprendizagem (ANDERSON, 2000).

Para que a avaliação diagnóstica identifique os conhecimentos prévios, componentes de habilidades e dificuldades de aprendizagem torna-se necessária uma avaliação fina, do tipo clínica e dinâmica para captar a multidimensionalidade do objeto aprendido para se coletar uma pluraridade de informações (RAPHAEL; CARRARA, 2002).

Com base nessa idéia, a nossa proposta metodológica visa sintetizar um processo de aprendizagem em uma matriz cognitiva (MAZZA; DIMITROVA, 2007) que mapeie

as aprendizagens dos alunos de uma turma em variáveis de avaliação e classes de perfis (OLIVEIRA, 2009). Essa matriz deve informar grupos de alunos com perfis de aprendizagens similares e auxiliar professores na instrução adaptativa (PIMENTEL; FRANCA; OMAR, 2003).

De acordo com Anderson (2000), as análises de processamento de informações procuram examinar os passos pelos quais o estudante chega à decisão de uma resposta para uma pergunta e o tempo necessário para essa pessoa dar cada passo. As informações obtidas a partir da avaliação diagnóstica devem, portanto, permitir a exploração do conhecimento e sintetizá-lo em um perfil ou modelo do aluno que informe (BALLESTER, 2003):

- O grau de aquisição dos requisitos de aprendizagem
- As ideias e modelos de raciocínio e estratégias de atuação
- As atitudes e hábitos adquiridos com relação à aprendizagem
- As representações das tarefas propostas

A avaliação diagnóstica, como instrumento de coleta e análise de informações de um processo de aprendizagem, deve ser utilizada para informar, através de variáveis de avaliação, o estado de aprendizagem dos alunos. Mas o diagnóstico será inútil se não der lugar a uma ação apropriada (PERRENOUD, 1999). Por isso, a avaliação diagnóstica deve ser realizada para auxiliar a avaliação formativa contribuindo, assim, com a análise e a remediação da aprendizagem.

2.3 A avaliação formativa

O termo avaliação formativa, introduzido em 1967 por Michael Scriven, refere-se aos procedimentos realizados por professores para adaptar seu processo didático aos progressos e necessidades de aprendizagem observados em relatórios de avaliação diagnóstica (BALLESTER, 2003). A ideia de avaliação formativa leva o professor a observar melhor seus alunos, a compreender melhor seu funcionamento, de modo a ajustar e individualizar suas intervenções pedagógicas (PERRENOUD, 1999). A avaliação formativa deve, desse

modo, estar diretamente ligada à gestão e à otimização das aprendizagens dos alunos pelo professor e pelos interessados.

A avaliação formativa responde a uma concepção de ensino que considera que aprender é um longo processo por meio do qual o aluno vai reestruturando seu conhecimento a partir das atividades que executa (BALLESTER, 2003). Esse modo de avaliação intervém na aprendizagem porque em função das respostas dos alunos, é possível reajustar o ensino e, se necessário, reorientá-lo (PERRAUDEAU-DELBRIEL, 2009).

A literatura define três processos de regulação da aprendizagem: a antecipação, o controle e o ajuste (ALLAL; SAADA-ROBERT, 1992; KIRCHNER; STOLZ, 2008). A antecipação representa as orientações de ação e assegura a condução de um processo de resolução de problemas. O controle ou monitoramento implica em um processo contínuo de comparação entre um estado dado e um estado-objetivo a ser alcançado. O ajuste ou regulação, que é a conseqüência da operação de controle, introduz uma modificação nos processos de produção ao ser evidenciado uma divergência entre o estado presente e o estado-objetivo.

De acordo com Santos (2002), a regulação da aprendizagem é todo ato intencional que, agindo sobre os mecanismos de aprendizagem, contribua diretamente para a progressão e/ou redirecionamento dessa aprendizagem. Segundo Haydt (2002), a regulação está associada a um mecanismo de retro-alimentação, isto é, de *feedbacks*. Esse mecanismo permite, após identificar deficiências por avaliação diagnóstica, reformular as ações pedagógicas, visando aprimorá-las em ciclo contínuo e ascendente (HAYDT, 2002).

A finalidade principal da avaliação formativa é uma função ajustadora do processo de ensino e de aprendizagem para possibilitar que os meios de formação respondam às características dos estudantes (BALLESTER, 2003). Para ter essa função ajustadora, a avaliação formativa persegue mais três objetivos: a regulação pedagógica, a gestão dos erros e a consolidação de êxitos (PERRENOUD, 1999; BALLESTER, 2003).

O processo de regulação poderá também ser realizado pelo próprio aluno, isto é, por autorregulação. Através dela um aluno acompanha seus próprios desempenhos e, a partir dos seus erros, reorienta o seu processo de aprendizagem. Um exemplo de autorregulação

apresentado por Santos (2002) é que quando um aluno risca o que fez ou recomeça tudo de novo, ele avalia etapas intermediárias do seu trabalho.

O segundo objetivo da avaliação formativa, a gestão de erros, é realizada pelo professor e pelo aluno nos processos de regulação e autorregulação, respectivamente. A gestão de erros, ao contrário das avaliações tradicionais, deve ser orientada por uma abordagem positiva do erro. Isso porque o erro é um importante indicador para a compreensão de situações de aprendizagem (SANTOS, 2002).

A gestão de erros no processo de regulação de aprendizagens deve ser individualizada (PERRENOUD, 1999). Na regulação feita pelo professor, isso significa que os *erros* de cada aluno devem ser identificados, corrigidos e comentados (*feedbacks*). Já na autorregulação, um aluno reflete os próprios erros e, ao refazer as questões que errou, avalia e corrige as etapas do seu processo de resolução das questões.

Quanto ao terceiro objetivo da avaliação formativa, que é a consolidação de êxitos, entendemos que a regulação e a autorregulação de aprendizagens devem ser realizadas continuamente até que se alcance um estado muito bom de aprendizagem coletiva, isto é, um estado de nivelamento de aprendizagens (OLIVEIRA, 2009).

Ir em direção à avaliação formativa é, portanto, não mais fabricar tantas desigualdades, mas sim criar meios para remediar as dificuldades de aprendizagem dos alunos mais lentos e mais fracos (PERRENOUD, 1999).

Embora se reconheça a grande eficácia da avaliação formativa em promover sucessos de aprendizagem, alguns professores consideram-na impraticável, sobretudo no caso de turmas com muitos alunos ou quando um professor dá muitas aulas semanais (BALLESTER, 2003).

No entanto, o processo de avaliação formativa torna-se mais rápido e bem menos complexo se apoiado por tecnologias computacionais que automatizem algumas das ações que demandam muito esforço do professor e ofereçam recursos que o auxiliem em suas ações de avaliação formativa. Algumas tecnologias que foram utilizadas para apoiar as ações formativas de professores são apresentadas na seção a seguir.

2.4 Tecnologias de avaliação diagnóstica e formativa

Os desafios da avaliação formativa são enfrentar repetidamente a grande quantidade de tarefas, o maior número de matérias a estudar, o aumento do número de alunos por professor e a maior diversidade de estudantes (BALLESTER, 2003; OLIVEIRA; ZANDONADE; OLIVEIRA, 2008). Considerando esses desafios, questiona-se como conceber dispositivos didáticos favoráveis a uma regulação contínua da aprendizagem (PERRENOUD, 1999).

Um bom dispositivo de avaliação deve estar a serviço de uma pedagogia diferenciada e ser capaz de dar resposta aos interesses e dificuldades de cada aluno (BALLESTER, 2003).

O trabalho desenvolvido por Pimentel, Franca e Omar (2003) teve como objetivo, através de tecnologias de *clustering*, a identificação de perfis similares de alunos no ensino presencial a fim de se oferecer atendimento personalizado a grupos heterogêneos.

Da mesma forma, utilizando algoritmos de *clustering*, Oliveira, Zandonade e Oliveira (2008) desenvolveram uma metodologia de avaliações diagnóstica e formativa para monitorar e controlar a prática da classificação de documentos em cursos de Biblioteconomia analisando variáveis de perfis de alunos e classificando esses alunos segundo as semelhanças entre suas características.

O *CourseVis* é uma ferramenta desenvolvida por Mazza e Dimitrova (2007) para monitoramento de estudantes. O objetivo dessa ferramenta é auxiliar instrutores de cursos a distância no acompanhamento de seus alunos. Para isso, o *Coursevis* gera representações gráficas multidimensionais para visualização de informações sobre desempenhos, características e comportamentos dos alunos de um curso a distância.

O *Coursevis* foi desenvolvido como instrumento de avaliação diagnóstica com foco em três componentes: eficácia, eficiência e usabilidade. A eficácia significa obter uma compreensão do que acontece em turmas de cursos a distância. A eficiência é inferir rapidamente sobre as informações obtidas. Já a usabilidade indica a grande utilidade que a ferramenta pode representar para o professor obter diagnósticos e realizar intervenções em um processo de ensino e de aprendizagem.

O trabalho de Soares et al. (2008) propõe um método de realimentação, isto é, de *feedbacks*, que contemple os aspectos diagnósticos, formativos e somativos da avaliação. A realimentação consiste em comparar progressos entre as avaliações diagnósticas por análises feitas antes e depois de uma avaliação formativa.

O método proposto por Soares et al. (2008) foi desenvolvido para a aprendizagem de matemática. Para a aplicação desse método de realimentação, foi utilizado o programa *WIMS*. Esse programa possibilita especificar atividades através de folhas de exercícios, *feedbacks* de desempenhos, de registros como o tempo da questão, o número de tentativas e a alternância entre erros e acertos. Além disso, segundo Soares et al. (2008), o *WINS* possibilita melhor reflexão sobre resultados das avaliações. Isso porque as análises a *priori* e a *posteriori* dos resultados contribuem para orientar as ações formativas de um professor.

Uma metodologia de avaliação formativa para cursos de engenharia é apresentada por Pacheco (2005) através da formação de mapas cognitivos difusos que combinam os mapas cognitivos de Axelrod com a lógica difusa. Essa metodologia tem como objetivo uma completa e contínua avaliação formativa do processo de aprendizagem em engenharia. Os mapas modelam matematicamente o processo educacional em engenharia e fornece uma visão ampla desse processo, permitindo diagnósticos e prognósticos e provendo dados necessários para eventuais ajustes.

Em pesquisas mais recentes, um Sistema Tutor Inteligente (STI) baseado em redes neurais tem sido aplicado para melhorar a eficácia dos procedimentos de avaliação formativa fornecendo instrução personalizada e *feedbacks* a estudantes (CASTELLANO et al., 2007). Para isso, esse sistema representa os desempenhos dos alunos em unidades de avaliação formativa, isto é, em uma matriz *MxN*. Nessa matriz, *M* representa os conceitos que devem ser aprendidos em um curso e *N* as unidades de avaliação formativa. Através dessa matriz, uma rede neural de aprendizagem por *Backpropagation* classifica situações de aprendizagem e tomadas de decisões formativas.

As tecnologias chegaram como resposta ao questionamento de Ballester (2003) sobre se é possível realizar avaliação formativa enfrentando as barreiras da grande quantidade de tarefas, do aumento do número de alunos, do aumento dos conteúdos estudados

e da diversidade entre os estudantes. Embora essas barreiras tornem impraticável a avaliação formativa por um professor, as tecnologias reduzem essas barreiras porque podem automatizar procedimentos repetitivos como a correção de exercícios e auxiliar professores no acompanhamento e na remediação individual da aprendizagem de seus alunos. Para isso, muitas tecnologias oferecem recursos de diagnósticos e de acompanhamento individual da prática de exercícios mesmo para turmas muito grandes e com a aplicação contínua de uma grande quantidade de exercícios.

Nos capítulos a seguir, são apresentadas mais técnicas e tecnologias que apóiam as ações de avaliação diagnóstica e formativa através de sistemas de avaliação semi-automática de exercícios, sistemas de recomendação e de reconhecimento de padrões.

2.5 Conclusão

Neste capítulo apresentamos a fundamentação teórica desta tese. Mostramos que o processo de aprendizagem pode ser contemplado, segundo a abordagem sistêmica, como um sistema que recebe entradas através do ensino, gera como saída um conjunto de variáveis que representam o estado de aprendizagem dos alunos e é realimentado continuamente pela avaliação.

O processo de aprendizagem pode ser compreendido como um processo de prática assistida de exercícios (ANDERSON, 2000) que se inicia após o recebimento de instruções através do ensino. Os desempenhos de atividades seriam, nesse caso, os indicadores de aprendizagem. Através de mecanismos de avaliação diagnóstica, essas informações de desempenhos poderiam ser sintetizadas em variáveis representantes de perfil ou do modelo do aluno. Essas variáveis apontariam êxitos e/ou dificuldades de aprendizagem. As variáveis que sinalizassem dificuldades de aprendizagem deveriam então ser realimentadas até que indicassem êxitos. Essa realimentação seria realizada pelas intervenções de avaliação formativa que reorientariam o ensino e as atividades aplicadas em um processo de aprendizagem.

Considerando que avaliações diagnóstica e formativa, principalmente em turmas com

um número elevado de alunos, só é possível se for auxiliado por computadores através de sistemas inteligentes (PIMENTEL; FRANCA; OMAR, 2003), desenvolvemos, segundo a abordagem sistêmica, um sistema semi-automatizado de avaliações diagnóstica e formativa para o domínio de aprendizagem da programação de computadores.

Nesse sistema, os fundamentos de avaliação diagnóstica e formativa foram implementados em tecnologias da *web* e de reconhecimento de padrões para a prática assistida da programação de computadores. O processo de aprendizagem foi sistematizado em um sistema *online* de atividades de programação, onde os alunos, após serem ensinados sobre um determinado assunto, praticariam a programação escrevendo e submetendo programas de computador como respostas a exercícios planejados por um professor.

Através de mecanismos de avaliação diagnóstica, os exercícios submetidos pelos alunos são automaticamente corrigidos pelo nosso sistema tendo como referência um conjunto de exemplos representativos pré-corrigidos por um professor. Em seguida, os resultados de desempenhos gerados são mapeados em variáveis de avaliação que reunidas formam um perfil de aluno indicando seus êxitos e dificuldades de aprendizagem. Os perfis dos alunos de uma turma são então agrupados, conforme semelhanças entre as variáveis de avaliação de cada perfil, em mapas de aprendizagem.

Através de mecanismos de avaliação formativa, os mapas de aprendizagem gerados pelos mecanismos de avaliação diagnóstica são analisados e as variáveis de perfis que indicam dificuldades de aprendizagem são realimentadas continuamente até que alcancem um estado satisfatório de aprendizagem. Essa realimentação é realizada através da recomendação das atividades mais apropriadas para alunos melhorarem seus desempenhos nos conteúdos representados por variáveis de avaliação que sinalizam dificuldades de aprendizagem.

Sabendo que a avaliação formativa é uma estratégia que melhora a aprendizagem (PERRENOUD, 1999; ANDERSON, 2000; BALLESTER, 2003), através do sistema que desenvolvemos implementando esse modelo de avaliação esperamos contribuir de forma significativa para uma melhor aprendizagem da programação de computadores.

Capítulo 3

Avaliação Automática

A avaliação automática de programação surgiu como um importante método para auxiliar professores na correção de exercícios de programação e possibilitar *feedbacks* imediatos a estudantes, principalmente em turmas numerosas (EBRAHIMI, 1994). O conceito de avaliação automática de atividades de programação foi introduzido por Hollingsworth (1960) e, desde os anos 60, soluções diversas e cada vez mais sofisticadas têm sido desenvolvidas. As soluções propostas têm avaliado itens como a execução correta de um programa (HOLLINGSWORTH, 1960; REEK, 1989; SAIKKONEN; MALMI; KORHONEN, 2001; BLUMENSTEIN et al., 2004; TANG; POON, 2009; SOUZA; MALDONADO; BARBOSA, 2011), a codificação (XU; CHEE, 2003; WU et al., 2007; WANG et al., 2011; NAUDE; GREYLING; VOGTS, 2010), o estilo de programação (REES, 1982; BERRY; MEEKINGS, 1985; JACKSON; USHER, 1997), as métricas de *software* (HUNG; KWOK; CHUNG, 1993; TRUONG; ROE; BANCROFT, 2004), o plágio (SAIKKONEN; MALMI; KORHONEN, 2001; BLUMENSTEIN et al., 2004; SANT, 2009) e até aspectos psicológicos (CURTIS et al., 1979). Mas, embora os sistemas de avaliação automática de programação tenham evoluído, conforme destacam Romli, Sulaiman e Zamli (2010), tais sistemas ainda são limitados em avaliar se de fato objetivos educacionais foram alcançados.

As principais estratégias de avaliação automática de programação são a análise dinâmica, a análise estática e a análise dinâmica e estática em conjunto. A análise dinâmica é baseada na execução de um programa e avalia a corretude através da testagem funcional e estrutural

de programas. A avaliação de análise estática é baseada na escrita do código-fonte e avalia itens como erros de programação de ordem sintática, semântica e estrutural, estilo de programação, métricas de código-fonte e complexidade de código. Já a análise dinâmica e estática combina as estratégias da análise dinâmica e da análise estática para avaliar programas de computador. Cada uma dessas abordagens e suas aplicações são explicadas mais detalhadamente na Seção 3.1.

A Tabela 3.1, baseada na revisão de trabalhos de avaliação automática de programação realizada por Romli, Sulaiman e Zamli (2010), ilustra como as propostas de avaliação automática de programação evoluíram desde a década de 60 até o presente trabalho. Na Tabela 3.1, são apresentados os trabalhos mais representativos de cada década, o método ou estratégia de avaliação, o fator de qualidade que explica e diferencia cada trabalho, a versatilidade das características utilizadas para avaliação de programas e como acontece a geração de dados de teste em cada proposta.

Embora várias soluções de sistemas de avaliação de exercícios de programação, tanto de análise dinâmica quanto estática, tenham sido desenvolvidas nas últimas décadas (MALMI; KORHONEN; SAIKKONEN, 2002; DOUCE; LIVINGSTONE; ORWELL, 2005; ALA-MUTKA, 2005; ROMLI; SULAIMAN; ZAMLI, 2010; IHANTOLA et al., 2010), são apontadas ainda as seguintes demandas no domínio de avaliação automática de programação:

1. Gerar representações alternativas de programas de computador além das árvores sintáticas abstratas (XU; CHEE, 2003; WU et al., 2007) e dos grafos (NAUDE; GREYLING; VOGTS, 2010).
2. Reduzir a dependência de modelos de soluções (os oráculos) fornecidos por especialistas, isto é, por professores.
3. Tratar a variabilidade de soluções dos exercícios de programação (NAUDE; GREYLING; VOGTS, 2010).
4. Estabelecer medições que garantam a confiabilidade dos sistemas de avaliação automática de forma que motivem professores a utilizá-los.

Avaliação Automática de Programação

Trabalhos	**Método**	**Categoria de teste**	**Fator de qualidade**	**Versatilidade de características**	**Geração de dados de teste**
Hollingsworth (1960)	Dinâmico	Caixa-preta	Corretude		
Rees (1982)	Estático		Estilo de programação		
Hung, Kwok e Chung (1993)	Estático		Métricas	X	
Jackson e Usher (1997)	Dinâmico/ Estático	Caixa preta/ Caixa Branca	Análise estrutural, corretude, eficiência, estilo, complexidade, adequação a dados de testes	X	Manual
Saikkonen, Malmi e Korhonen (2001)	Dinâmico	Caixa preta/ Caixa Branca	Corretude, Estilo de programação, tempo de execução, plágio	X	Semi-automática
Truong, Roe e Bancroft (2004)	Dinâmico/ Estático	Caixa preta/ Caixa Branca	Métricas da Engenharia de Software, similaridade estrutural, corretude, estados das variáveis	X	Manual
Rahman et al. (2008)	Estático		Similaridade não-estrutural, análise de pseudo-código	X	
Wu et al. (2007)	Estático		Análise semântica e estrutural	X	
Gerdes, Jeuring e Heeren (2010)	Dinâmico	Caixa-preta	Corretude, estratégias de programação, normalização de código		
Tang e Poon (2009)	Dinâmico	Caixa-preta	Corretude (Por padrões de *tokens*)	X	Manual
Sant (2009)	Dinâmico/ Estático	Caixa-preta	Estilo de programação, complexidade, funcionalidade, plágio	X	Manual
Naude, Greyling e Vogts (2010)	Estático		Similaridade estrutural (Grafo)		
Este Trabalho	Dinâmico/ Estático	Caixa-preta	Similaridade Vetorial	X	Manual

Tabela 3.1: Trabalhos de avaliação automática de programação

5. Orientar a avaliação automática por uma perspectiva pedagógica (IHANTOLA et al., 2010), isto é, segundo aquilo que seria interessante para o professor avaliar.
6. Desenvolver estratégias de prevenção contra códigos maliciosos submetidos por alunos em sistemas de avaliação automática.
7. Controlar com maior rigor os plágios de exercícios
8. Integrar os sistemas de avaliação automática a sistemas *Learning Management Systems* - Sistemas de Gestão da Aprendizagem (LMS) (SULEMAN, 2008). Uma justificativa para essa integração é a prevenção contra a submissão de códigos maliciosos (IHANTOLA et al., 2010).

Nas seções a seguir, apresentamos o estado da arte da avaliação automática no Brasil e no mundo e as propostas desenvolvidas desde a década de 60 para atender a essas demandas. Para isso, este capítulo está organizado conforme a ordem a seguir. Na Seção 3.1, destacamos as estratégias de avaliação automática de programação bem como as vantagens, desvantagens e aplicações de cada uma delas. Na Seção 3.2, apresentamos a avaliação automática de programação no Brasil. Na Seção 3.3, destacamos os trabalhos relacionados e concluimos com as considerações finais.

3.1 Estratégias de avaliação automática

A avaliação de programas de computador deve contemplar a corretude e a construção do código-fonte do programa (GERDES; JEURING; HEEREN, 2010). A corretude indica o funcionamento correto do programa. Avaliar a corretude é testar se um programa atende os requisitos para os quais foi criado. Já a avaliação de código-fonte analisa as boas práticas de programação (GERDES; JEURING; HEEREN, 2010). f A avaliação automática deve justificar-se pedagogicamente (IHANTOLA et al., 2010). Para isso, um bom sistema de avaliação automática de programas de alunos iniciantes em programação deve atender os seguintes requisitos (ROMLI; SULAIMAN; ZAMLI, 2010):

- Realizar testagens suficientes

- Checar se programas estão de acordo com a especificação
- Reconhecer erros sintáticos e semânticos
- Fornecer *feedback* imediato

As estratégias de avaliação automática propostas que visam atender a esses requisitos são as estratégias de análise dinâmica, análise estática e análise dinâmica e estática. Nas subseções a seguir, são apresentadas as descrições de cada uma dessas abordagens bem como suas aplicações, vantagens e desvantagens.

3.1.1 Análise dinâmica

A análise dinâmica é a estratégia de avaliação automática de atividades de programação que envolve executar programas por um conjunto de dados de testes, tendo como principal objetivo descobrir erros de execução em um programa (ZIN; FOXLEY, 1994). A análise dinâmica tem como vantagem informar se um programa funciona corretamente e se atende os propósitos para os quais foi desenvolvido através dos testes da caixa-preta e da caixa-branca. O teste da caixa-preta avalia o comportamento externo do programa, isto é, se a saída está correta de acordo com as entradas fornecidas. Já o teste da caixa-branca avalia o comportamento interno do componente de software a partir do código, isto é, avaliando se os componentes de um programa funcionam.

A testagem da caixa-preta, também chamada de teste funcional, é orientada a dados ou à entrada e saída. Esse teste, no entanto, não leva em consideração o comportamento interno de um programa de computador, mas apenas o seu comportamento externo. Dessa forma, na testagem dinâmica, os dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado contra um modelo de saída previamente definido (ROMLI; SULAIMAN; ZAMLI, 2010).

A testagem da caixa-branca, também chamada de teste estrutural ou orientado à lógica, avalia o comportamento interno do componente de software. Essa técnica trabalha diretamente sobre o código-fonte do componente de software para avaliar aspectos tais como:

teste de condição, boas práticas de programação, teste de fluxo de dados, teste de ciclos, teste de caminhos lógicos e códigos nunca executados (ROMLI; SULAIMAN; ZAMLI, 2010).

Embora a análise dinâmica ofereça a grande vantagem de avaliar se um programa funciona corretamente, é um modelo que oferece muitas desvantagens. A primeira delas é oferecer *feedback* limitado (NAUDE; GREYLING; VOGTS, 2010) quando, mais do que avaliar a corretude de um programa, objetiva-se avaliar a capacidade de programar de um aluno. Nesse caso, como a avaliação dinâmica não contempla os processos realizados na construção de um programa, não se pode afirmar, com base apenas no resultado correto, se um aluno é bom programador ou não. Além disso, como a análise dinâmica pode ser sensível a pequenos erros (NAUDE; GREYLING; VOGTS, 2010), ainda que um aluno tenha realizado corretamente todos os passos para resolver um problema, um simples erro pode provocar falhas em todos os testes dinâmicos e prejudicar a avaliação do aluno. Nesse caso, a maior parte do processo de programação desenvolvido corretamente pelo aluno é anulada injustamente na avaliação dinâmica.

Uma outra desvantagem da análise dinâmica é que a testagem pode ser duvidosa porque os alunos podem imitar a saída padrão e porque alguns programas podem compilar com sucesso, mas gerar uma saída errada (WU et al., 2007).

Para Zin e Foxley (1994), a análise dinâmica é também deficiente porque há muitos tipos de problemas onde não existe um oráculo, isto é, o mecanismo pelo qual a corretude da saída pode ser checada. Eles assumem que o oráculo está sempre presente e que sem ele a testagem dinâmica não pode ser automatizada.

Na avaliação da saída, poderá haver ainda o problema da variabilidade de saídas geradas pelos programas submetidos por alunos como consequência da liberdade de escrita de código (NAUDE; GREYLING; VOGTS, 2010). Em outros, essa variabilidade de saídas pode ser espontânea, como as saídas geradas por Algoritmos Genéticos (FERNEDA; SMIT, 2012).

Um requisito essencial apontado por Ala-Mutka (2005) para os sistemas de análise dinâmica é garantir ambientes seguros, isto é, *sandboxes ou caixas de areia*, para a execução sem riscos dos programas submetidos por alunos. Um exemplo de execução de risco em sistema de avaliação automática é a submissão de programas em *loop* que gravam dados em

arquivos. O alto consumo de tempo de processamento e de memória podem causar tão-logo consideráveis danos ao sistema de avaliação e até a paralisação da sua infra-estrutura. Estratégias de prevenção e de ação devem, portanto, ser criteriosamente desenvolvidas contra códigos de execução perigosa submetidos a sistemas de avaliação automática que utilizam a abordagem de análise dinâmica.

Destacamos entre as diversas abordagens de avaliação dinâmica o *Scheme-robo* (SAIKKONEN; MALMI; KORHONEN, 2001), o *Generic Automated Marking Environment* - Ambiente Genérico de Avaliação Automática (GAME) (BLUMENSTEIN et al., 2004), a abordagem de testagem automática de Tang e Poon (2009) e o *Progtest* (SOUZA; MALDONADO; BARBOSA, 2011).

O *Scheme-robo* de Saikkonen, Malmi e Korhonen (2001) é um sistema de avaliação automática de exercícios de programação funcional que utiliza as testagens da caixa-preta e da caixa-branca para avaliar a corretude, o estilo de programação, o tempo de execução e possíveis plágios. A corretude é avaliada por pesquisa de palavras-chave, onde especificam-se palavras-chave que devem ser rejeitadas. O estilo de programação é avaliado pela análise da estrutura do programa cujo código é reduzido a uma árvore sintática abstrata. Já a detecção de plágio é realizada removendo-se os nomes de variáveis, comparando-se as árvores sintáticas e, como os exercícios são pequenos, o sistema compara todas as soluções para reconhecer pares de estudantes com muitas soluções em comum.

O GAME (BLUMENSTEIN et al., 2004) oferece a vantagem de ter sido projetado para realizar avaliação de análise dinâmica de exercícios escritos em diferentes linguagens de programação, embora hoje o faça somente nas linguagens C e Java. Nesse sistema o professor deve fornecer um esquema de avaliação que inclui os critérios de avaliação e os modelos de soluções. Os itens de avaliação do GAME incluem o teste estrutural, a execução, a análise da saída por comparação das palavras-chave do arquivo de solução do professor contra as palavras-chave da saída do programa do aluno e detecção de plágios. Os arquivos de submissão dos alunos incluem arquivos de entrada, de saída, da saída correta e um arquivo de roteiro de execução. Os resultados de avaliação automática do GAME apresentados por Blumenstein et al. (2004) mostram que grande parte das notas atribuídas pelo sistema em duas bases de exercícios (uma em Linguagem C e outra em Linguagem Java) coincidem

com as notas atribuídas por professores, segundo critérios de execução correta.

A abordagem de testagem automática de Tang e Poon (2009) propõe avaliar a corretude do programa dos alunos com base em padrões de *tokens* nas saídas uma vez que é um processo difícil comparar as saídas geradas de programas de alunos contra os modelos de saída dos gabaritos fornecidos por professores. Quando um programa do estudante é testado, sua produção real de cada caso de teste é também convertida em uma seqüência de *tokens* a partir da qual os relevantes são extraídos. Esses *tokens* relevantes são comparados com o padrão de *tokens* derivado da produção esperada. A saída real é considerada correta se e somente se a comparação retorna um sucesso.

O *ProgTest* é um sistema mais recente de avaliação de análise dinâmica (SOUZA; MALDONADO; BARBOSA, 2011). A ideia do *ProgTest* é fornecer apoio automatizado para avaliar os programas e casos de testes submetidos por alunos. O processo do *ProgTest* consiste de compilar programas do professor e do aluno, executar critério funcional de testes e executar o critério estrutural de testes. Os parâmetros de avaliação, segundo Souza, Maldonado e Barbosa (2011), são a passagem pelos casos de testes do aluno (funcional), a passagem pelos casos de teste do professor e pelos casos de testes do aluno (estrutural-teste de fluxo de controle e de dados). O *ProgTest*, dessa forma, além de avaliar a corretude, integra técnicas de testagem com conceitos de programação introdutória. No entanto, no *ProgTest* a carga de trabalho do professor aumenta porque, além dos modelos de soluções, o professor deve fornecer também os casos de testes. Para programas mais complexos e com alta variabilidade de soluções, a especificação de atividades através do *ProgTest* fica, dessa forma, praticamente inviável.

Desde os anos 60, várias soluções de análise dinâmica foram desenvolvidas e é a abordagem de avaliação automática mais utilizada (ROMLI; SULAIMAN; ZAMLI, 2010). No entanto, embora os sistemas de avaliação de análise dinâmica tenham evoluído na sua principal vantagem, que é a avaliação de corretude, ainda preservam muitas das desvantagens e limitações do sistema original de análise dinâmica desenvolvido por Hollingworth na década de 60.

3.1.2 Análise estática

A análise estática é a estratégia de avaliação automática baseada na análise de código-fonte. Nessa abordagem de avaliação, aplicam-se várias técnicas para avaliação de estilo de programação, detecção de erros sintáticos e semânticos, análise de métricas de software, análise de similaridade estrutural, análise de similaridade não-estrutural, detector de palavras-chave, detecção de plágios e análise de diagramas (RAHMAN et al., 2008). A análise estática poderia também ser estendida para avaliar *loops* infinitos, erros lógicos e sentenças não utilizadas em programas (RAHMAN et al., 2008; BENFORD et al., 1995)

A análise estática de similaridade estrutural é realizada pela comparação entre as estruturas de programas de alunos contra as estruturas dos programas-solução fornecidos pelo professor (RAHMAN et al., 2008). O passo inicial dessa comparação é a normalização dos códigos-fonte do aluno e do professor a uma representação padrão. Essa representação pode ser realizada por árvores sintáticas e grafos (XU; CHEE, 2003; WU et al., 2007; NAUDE; GREYLING; VOGTS, 2010). Para a avaliação de corretude, os códigos normalizados do aluno são comparados estruturalmente contra os códigos do professor. Essa comparação entre as soluções do aluno e do professor é realizada por medidas de similaridade (NAUDE; GREYLING; VOGTS, 2010) ou por casamento de padrões (XU; CHEE, 2003).

A análise estática de similaridade não-estrutural, por sua vez, desconsidera a relação entre as sentenças bem como a ordem delas para comparar as soluções desenvolvidas por alunos contra as soluções-modelos do professor. Nessa abordagem, códigos-fontes do professor e aluno podem ser reduzidos a pseudo-códigos ou a outras representações que não levem em conta as relações entre as sentenças e, sem a parte de declaração de variáveis, obtêm-se os índices de similaridades entre a solução do aluno e as soluções do professor (RAHMAN et al., 2008). A nota da solução do aluno estará associada ao maior índice percentual de similaridade obtido na comparação com as soluções-modelos do professor.

A atribuição de notas em muitos sistemas de análise estática apoia-se em métricas da Engenharia de Software (HUNG; KWOK; CHUNG, 1993; BENFORD et al., 1995; CURTIS et al., 1979; MOREIRA; FAVERO, 2009), em índices de similaridades com as

soluções-modelo (RAHMAN et al., 2008) e em modelos de predição de notas como a regressão linear (MOREIRA; FAVERO, 2009; NAUDE; GREYLING; VOGTS, 2010). Mas a atribuição de notas conforme padrões de correção de professores é ainda um desafio para a abordagem de avaliação automática de análise estática de programação.

Dessa forma, raras soluções demonstram que de fato as notas atribuídas por avaliação automática atendem de fato os critérios de avaliações de professores. O método de avaliação estática proposto por Naude, Greyling e Vogts (2010), porém, dá um passo relevante para atender a essa demanda, pois os resultados do seu método mostram forte relação entre as notas preditas pelo sistema e as notas atribuídas por professores. No entanto, quando as notas são baixas, as notas preditas pelo método de Naude, Greyling e Vogts (2010) ainda divergem das notas das soluções-modelos do professor.

As principais vantagens da análise estática são o menor custo, a menor dependência do oráculo e a possibilidade de oferecer uma avaliação mais aproximada da avaliação de um professor. O custo da análise estática é menor do que o da análise dinâmica porque, uma vez que não requer compilação de código e execução, reduz a carga de trabalho de máquinas servidoras (TRUONG; ROE; BANCROFT, 2004). A dependência do oráculo, por sua vez, é reduzida quando a avaliação é realizada pelos escores levantados pelas métricas de software ou utilizam-se exemplos de programas já corrigidos por professores como base de predição de notas de outros exercícios (NAUDE; GREYLING; VOGTS, 2010).

A avaliação de análise sintática é mais justa porque contempla o processo de construção de programas. Dessa forma, um simples erro não anula todo um processo de desenvolvimento de um programa como ocorre na avaliação de análise dinâmica. No entanto, a análise estática não contempla os resultados de execução como a corretude, funcionalidade e eficiência, que são importantes itens de avaliação para muitos professores de programação (ALA-MUTKA, 2005; RAHMAN et al., 2008).

O principal desafio da avaliação de análise estática, no entanto, é tratar a variabilidade das soluções em códigos-fontes, o que amplia a carga de trabalho de professores, uma vez que estes precisam fornecer vários modelos de soluções (RAHMAN et al., 2008). De acordo com Naude, Greyling e Vogts (2010), as principais causas das variabilidades entre programas

de computador são as seguintes:

- Seleção de algoritmos, como nos casos de haver vários algoritmos para resolver um mesmo problema. Exemplo: algoritmos de ordenação.
- Escolha dos identificadores, pois é difícil mapear os nomes de variáveis e fazer a correspondência delas em diferentes programas.
- Decomposição do programa em funções, sentenças e expressões
- Ordenação de sentenças independentes, que é um problema mais para soluções de programação cuja ordem dos textos é considerada.
- Linguagens específicas e sinônimos, isto é, quando há várias formas, na mesma linguagem de programação, várias possibilidades de escrever uma mesma sentença. A Linguagem C é um exemplo de flexibilidade na construção de sentenças e expressões.
- Identidades algébricas e lógicas
- Variação de *loop*, isto é, quando há várias formas de escrever uma estrutura de controle de repetição.
- Variação de decisão, isto é, quando há várias formas de escrever estruturas de controle condicionais.

As abordagens de avaliação de análise dinâmica predominam sobre as abordagens de avaliação que aplicam apenas a análise sintática de programas (ROMLI; SULAIMAN; ZAMLI, 2010) e a maioria dos trabalhos que aplicam a análise estática utilizam-na em combinação com a análise dinâmica. Entre os trabalhos que utilizam apenas a avaliação de análise estática, destacamos os trabalhos de Hung, Kwok e Chung (1993), de Xu e Chee (2003), de Wu et al. (2007) e de Naude, Greyling e Vogts (2010).

O trabalho de Hung, Kwok e Chung (1993) utiliza quatro métricas de software para avaliar programas de computador que medem a habilidade de programar, a complexidade de código, o estilo de programação e a eficiência. Para a avaliação da habilidade de programar sugere-se como uma boa opção a métrica de número de linhas de código. Para medir a complexidade de código, aplicam-se métricas de complexidade de *McCabe* (MCCABE, 1976) e, para medir a eficiência, medições de tempos de execução. Todas essas métricas

foram implementadas em um analisador de programa testado em amostras de programas escritos em Linguagem Pascal.

O artigo de Xu e Chee (2003) descreve uma abordagem baseada em transformação para automatizar o diagnóstico dos programas dos alunos para a programação em sistemas de tutoria. Essa abordagem oferece melhorias na análise de controle de fluxo e análise de fluxo de dados. O diagnóstico automático do programa do aluno é alcançado comparando-o com um programa de amostra no nível semântico após ambos normalizados a uma forma padrão. Esse diagnóstico baseado em transformações dos programas usa, segundo Xu e Chee (2003), três formas de representação:

- Código fonte
- *Abstract Syntactic Tree* - Árvore Sintática Abstrata (AST)
- Gráfico de dependência aumentada do programa orientado a objeto (AOPDG)

A representação AOPDG tem como méritos eliminar muitas variações que existem em nível de código-fonte e em nível de AST, suportar algoritmo de casamento de padrões e ativar esse algoritmo para reconhecer mudanças comportamentais, preservando semânticas na comparação do programa do aluno com a solução-modelo (XU; CHEE, 2003).

O *AnalyseC*, proposto por Wu et al. (2007), foi desenvolvido para avaliar automaticamente exercícios de programação escritos em Linguagem C em nível estrutural e semântico. O *AnalyseC* aplica técnicas de compilação para otimização de código como a expansão *inline*, análise de fluxo de dados e análise de fluxo de controle. O *AnalyseC* faz uso de métricas de *software* e método de diagnóstico baseado em transformações estruturais AST e semânticas no código-fonte para automatizar análise e avaliação de programas. As padronizações nos códigos-fonte do aluno e do professor são feitas na declaração temporária de variáveis de uma função, nas expressões e nas estruturas de controle. Depois das transformações estruturais, os programas são representados em grafos para análise semântica. Em seguida, a solução do aluno e as soluções do professor são comparadas para avaliação.

O trabalho mais recente de análise estática é o método de avaliar programas por

similaridade de grafos proposto por Naude, Greyling e Vogts (2010). Nesse trabalho, os programas desenvolvidos por alunos são normalizados a árvores sintáticas abstratas e as variações das soluções como expressões lógicas, estruturas de controle condicional e de repetição são normalizadas. As notas são preditas com base em notas de provas de outras turmas por um modelo de regressão linear, que tem como variáveis independentes os maiores índices de similaridades entre um programa não pontuado e os programas já avaliados. Esses índices foram calculados pela medida de similaridade de grafos *AssignSim*, desenvolvida por Naude, Greyling e Vogts (2010). Os resultados dessa proposta de avaliação indicam alta correlação entre a nota do professor e a nota predita, principamente para notas mais altas. Porém, há ainda um desvio significante entre a nota do professor e a nota predita quando as notas são baixas, conforme mostram Naude, Greyling e Vogts (2010).

Assim como a análise dinâmica é dependente dos casos de testes, a análise sintática é, em geral, dependente das soluções-modelo do professor. Uma vez que é comum haver na programação diferentes soluções para um mesmo problema, é trabalhoso para o professor, em alguns casos, prever todas as possibilidades de soluções para compor os gabaritos. Além disso, nem sempre haverá uma base repetida de soluções de exercícios para servir de modelo de predição para outros exercícios.

Em grande parte dos trabalhos que aplicam a estratégia de avaliação estática, as transformações sintáticas e semânticas são realizadas para reduzir as variações dos programas dos alunos e das soluções-modelos, principalmente em linguagens flexíveis como a Linguagem C (WU et al., 2007). No entanto, as variações podem fazer parte dos critérios de avaliações de professores. Nesse caso, as transformações implicariam em perda de informações relevantes para avaliação.

As abordagens de análise sintática carecem, dessa forma, de representações que mapeiem programas em direção aos objetivos de avaliação de professores. Como esses objetivos quase sempre incluem a corretude de programas, a avaliação de análise sintática não deveria ser aplicada como uma alternativa à avaliação dinâmica, mas sim como uma estratégia complementar. O melhor, portanto, é combinar as estratégias de análise estática e dinâmica para que a avaliação automática justifique-se pedagogicamente (IHANTOLA et al., 2010).

3.1.3 Análise dinâmica e estática

A estratégia de análise dinâmica e estática é a estratégia de avaliação que combina as estratégias de avaliação estática e dinâmica para avaliar códigos-fontes e funcionamento correto de programas.

A principais vantagens da análise dinâmica e estática é reduzir as desvantagens de aplicar apenas a abordagem de análise estática ou a de análise dinâmica. A análise dinâmica e estática é, dessa forma, a estratégia mais adequada para avaliar objetivos educacionais (ROMLI; SULAIMAN; ZAMLI, 2010).

Destacamos como exemplos da estratégia de avaliação estática e dinâmica o *Ceillidth* (BENFORD et al., 1995), o *Assyst* (JACKSON; USHER, 1997), e o *Autolep* (WANG et al., 2011). O *Ceillidth* valoriza mais a análise estática e o *Assyst*, a análise dinâmica. Já o *Autolep*, equilibra as duas abordagens.

A avaliação de exercícios de programação através do sistema *Ceilidh* (BENFORD et al., 1995) inclui programas de computador em várias linguagens com métricas de avaliação estática e dinâmica, questões de múltipla-escolha, exercícios de perguntas e respostas e questões discursivas. De acordo com Benford et al. (1995), a avaliação de programas no *Ceilidh* é realizada pela análise do *layout* tipográfico, da complexidade estrutural, das fraqueza estruturais, das características do código-fonte, da corretude e da eficiência dinâmica.

Na análise do *layout* tipográfico do *Ceilidh*, são computadas estatísticas como, por exemplo, média de caracteres por linha, média de espaços por linha, número de comentários e variáveis globais. Na complexidade estrutural são avaliados itens como número de palavras reservadas, número de estruturas de controle condicional e de repetição, número de chamadas de funções e número de *gotos*. Para verificação das deficiências estruturais, utiliza-se o analisador de código C do Linux chamado *Lint* (ORCERO, 2000). As características de código-fonte são verificadas pela análise dos construtos do programa, removendo-se *strings* e comentários. A corretude dinâmica é avaliada submetendo um programa a vários casos de testes e a eficiência dinâmica, pelo número de vezes que cada linha do programa é executada.

A ferramenta de avaliação automática *Assyst*, proposta por Jackson e Usher (1997), avalia programas C por 5 parâmetros: corretude, eficiência, complexidade, estilo e adequação aos dados de teste. O professor deve especificar as saídas e os casos de testes, e o aluno deve submeter o programa e os casos de testes. No final da avaliação, um relatório é apresentado.

No *Assyst*, a avaliação de corretude é realizada verificando se um programa compila, executa e gera saída correta conforme um gabarito. A avaliação de eficiência é feita pelo tempo de execução, mas como em cursos de programação introdutória o tempo de execução é muito rápido, contam-se o número de sentenças executadas. Para a avaliação de complexidade, o *Assyst* aplica as métricas de McCabe (MCCABE, 1976). Na avaliação de estilo de programação, medem-se características de programação como tamanho do módulo, número de comentários e uso de identação. Os testes de adequação aos dados, por sua vez, são realizados por casos de testes.

O *Autolep* é um sistema de avaliação automática orientado ao treinamento de habilidades aplicado em cursos de programação de Linguagem C (WANG et al., 2011). O *Autolep* combina as estratégias de avaliação dinâmica e estática para ajudar programadores iniciantes a obterem habilidades de programação. Para a avaliação de programas no *Autolep*, o professor deve fornecer modelos de programas e casos de testes.

No *Autolep*, há de fato uma prática assistida de exercícios porque o seu ambiente permite aos estudantes escrever, digitar, corrigir e executar programas. A execução de códigos, porém, é realizada na própria máquina do aluno.

O *Autolep* pontua sobre sintaxe, semântica e funcionalidades avaliando construções de um programa, conferindo com a especificação e marcando erros lógicos e sintáticos (WANG et al., 2011). A identificação de erros sintáticos é como dos compiladores. Para verificar a similaridade semântica, o programa do estudante e os modelos de soluções são transformados em grafos.

Embora a análise estática e dinâmica seja a estratégia mais adequada para avaliar objetivos educacionais por combinar as vantagens da análise estática e da análise dinâmica, herda, na maioria das abordagens, a forte dependência dos modelos de soluções da análise estática e dos casos de testes, da análise dinâmica. Isso se torna uma grande

desvantagem para essa essa estratégia de avaliação quando os programas submetidos por alunos apresentam variabilidade de soluções, as saídas não são previsíveis e a avaliação de corretude requer muitos casos testes.

Há portanto uma forte demanda no domínio da avaliação de exercícios de programação de reduzir a dependência do oráculo, isto é, das soluções-modelos fornecidas por professores na análise dinâmica, estática e dinâmica e estática de exercícios de programação.

3.2 A avaliação automática de programação no Brasil

No Brasil, a avaliação automática de exercícios de programação é ainda inexpressiva, pois poucas soluções têm sido desenvolvidas e predominam ainda as soluções baseadas em análise dinâmica. As discussões sobre avaliação automática no Brasil chamam à atenção para duas questões (MOREIRA; FAVERO, 2009):

- Como avaliar manualmente todos os exercícios dos estudantes, para turmas grandes?
- Como fornecer um *feedback* em curto prazo, de forma que o estudante possa aperfeiçoar sua solução?

De acordo com Moreira e Favero (2009), para atender a essas duas demandas, soluções de avaliação automática até então desenvolvidas avaliam programas de computador por duas métricas: se o programa retorna o resultado esperado (NUNES; LISBOA, 2004) e a complexidade da solução (HARB et al., 2007).

A abordagem de Moreira e Favero (2009) é um avanço em relação às abordagens propostas porque tenta avaliar programas de computador por um mecanismo que interpreta códigos-fonte de programas por um conjunto de parâmetros, representados pelas métricas de complexidade da Engenharia de Software, para realizar a predição de notas (MOREIRA; FAVERO, 2009).

3.2.1 Análise dinâmica

As soluções de avaliação automática que aplicam a estratégia de análise dinâmica no Brasil predominantemente caracterizam-se como testadores automáticos de programas (NUNES; LISBOA, 2004) ou como sistemas de submissão de programas para avaliação de corretude com base em entrada e saída (CAMPOS; FERREIRA, 2004).

O testador automático proposto por Nunes e Lisboa (2004) é uma aplicação Java que testa classes também implementadas em Java e apresenta um método de atribuição às classes avaliadas. De acordo com Nunes e Lisboa (2004), a correção de uma determinada classe pode ser vista de duas maneiras: pela correção funcional e pela correção de estado. Na correção funcional, a execução é correta quando construtores e métodos de classes retornam os valores esperados e, na correção de estados, os campos privados e públicos da classe representam corretamente os estados em que se encontram. O testador automático avalia, portanto, a corretude, comparando as classes contra um conjunto-modelo de classes.

Como exemplo de sistemas de submissão de programas para avaliação de corretude com base em entrada e saída, destacamos o BOCA (CAMPOS; FERREIRA, 2004). O BOCA é um sistema web de submissão de exercícios com autenticação, controle de tempo e disponibilização de resultados em tempo real (CAMPOS; FERREIRA, 2004). O BOCA tem sido aplicado no Brasil como apoio às competições de programação promovidas pela Sociedade Brasileira de Computação (SBC). Para cada problema cadastrado no BOCA, são necessários um arquivo contendo um conjunto de entradas e outro contendo as respectivas saídas (FRANCA et al., 2011). O processo de correção automática consiste dos seguintes passos (CAMPOS; FERREIRA, 2004; FRANCA et al., 2011):

1. Receber o código-fonte do programa, que deve ser o único arquivo submetido
2. Compilar o programa-fonte
3. Gerar o programa executável, caso não haja erros
4. Executar o programa executável
5. Testar as entradas contidas no arquivo entrada
6. Comparar a saída gerada com a saída da solução-modelo

7. Envio de *feedback*, apresentando relatório de submissão e eventuais erros de compilação e comparação com a saída

Uma adaptação do sistema BOCA é o BOCA-LAB, que é um *script* que estende as funcionalidades do BOCA agregando-lhe o envio múltiplo de arquivos para compilação e o cadastro de informações referentes aos alunos (FRANCA et al., 2011).

Em relação a outros modelos de análise dinâmica (TANG; POON, 2009; SOUZA; MALDONADO; BARBOSA, 2011), as soluções propostas aqui no Brasil são limitadas, pois carecem de melhores estratégias para análise das saídas de programas executados e para avaliação de corretude por testes da caixa-preta e da caixa-branca.

3.2.2 Análise estática

As principais soluções de avaliação automática que aplicam a estratégia de análise estática no Brasil caracterizam-se pela normalização de código-fonte a uma representação padrão após análise léxica e sintática (DIAS, 2001) e pela representação de programas por métricas de complexidade da Engenharia de Software com predição de notas por regressão linear múltipla (MOREIRA; FAVERO, 2009).

O modelo de avaliação automática, o Avaliador SQL (AS), proposto por Dias (2001) decompõe o processo de avaliação em correção e classificação. No AS o processo de correção é realizado sequencialmente pelas seguintes etapas: análise léxica por representações dos termos em códigos *American Standard Code for Information Interchange* - Código Padrão Americano para Intercâmbio de Informações (ASCII), análise sintática, normalização a uma representação semântica e comparação da solução do aluno contra a solução do professor por casamento de padrões. O algoritmo que realiza essa comparação cria a seguinte lista para representar as situações de avaliação entre a solução (exercício resolvido pelo aluno) e a resposta (gabarito do professor) (DIAS, 2001):

- Existe na solução e não existe na resposta
- Existe na resposta e não existe na solução

- Existe na resposta mas não corresponde à solução

Já o processo de classificação da solução desenvolvida pelo aluno no AS, segundo Dias (2001), baseia-se no cálculo do erro relativo, com base nos pesos atribuídos aos termos de uma instrução. A fórmula de cálculo do erro considera os pesos das diferenças e dos termos corretos entre a solução do aluno e do professor.

Uma abordagem alternativa à avaliação de análise sintática baseada em transformações sintáticas e semânticas é a representação de programas por métricas de código-fonte da Engenharia de Software.

O modelo de avaliação automática proposto por Moreira e Favero (2009), por exemplo, representa programas pelas seguintes métricas de complexidade: a quantidade de uso de funções pré-definidas da linguagem, o número de palavras reservadas, o número de declarações, o número de linhas do programa, a complexidade ciclomática de McCabe (1976) e o volume de Halstead (HALSTEAD, 1977).

Considerando essas métricas como variáveis independentes de um modelo de regressão linear múltipla (HAIR; TATHAM; BLACK, 1998), o método de avaliação de Moreira e Favero (2009) estima o quanto essas variáveis influenciam na atribuição da nota, que é a variável dependente. A partir da descoberta de quanto essas variáveis influenciam por um conjunto representativo de exercícios corrigidos, as notas de outros exercícios de programação podem ser preditas. Essa predição seria realizada a partir de equação de regressão linear múltipla:

$$y = p_1.m_1 + p_2.m_2 + \ldots + p_n.m_n,$$

Onde $m_i (i = 1 \ldots n)$ são diferentes métricas e p_i são os pesos de cada métrica e y é a variável dependente representada pela nota.

O trabalho de Moreira e Favero (2009) é um passo inicial, mas relevante para a avaliação automática de exercícios de programação no Brasil. Embora os resultados apresentados indiquem fraca correlação entre as variáveis independentes e a variável dependente e o erro

estimado seja alto, o modelo pode ser melhorado. Isso poderia ser realizado adicionando e quantificando outras variáveis que podem ser consideradas importantes na atribuição de notas pelo professor.

3.3 Conclusão

Neste capítulo apresentamos o estado da arte da avaliação automática de programação no Brasil e no mundo. Os trabalhos apresentados mostraram como as abordagens de avaliação dinâmica, estática e dinâmica-estática têm sido aplicadas e como têm evoluído.

Os trabalhos apresentados mais relacionados à nossa proposta de avaliação semi-automática de exercícios de programação são os modelos de análise estática de Naude, Greyling e Vogts (2010) e de Moreira e Favero (2009) e os sistemas de avaliação de análise dinâmica e estática *Ceilidth* (BENFORD et al., 1995) e *Assyst* (JACKSON; USHER, 1997).

O nosso modelo de avaliação assemelha-se ao trabalho de Naude, Greyling e Vogts (2010) pois, como eles, desenvolvemos uma estratégia para tratar a variabilidade das soluções. Na proposta de Naude, Greyling e Vogts (2010), essa variabilidade é tratada normalizando-se estruturas e variáveis nos programas de forma que programas sejam reduzidos à sua essência estrutural e lógica.

Aplicando uma técnica de *clustering* para separar em grupos os tipos de soluções de programas e reunir em cada um desses grupos as soluções semelhantes. Entendemos que a vantagem da nossa abordagem de *cluster* em relação à proposta de Naude, Greyling e Vogts (2010) é manter a integridade das estruturas e expressões de programação construídas pelos alunos. A normalização realizada por Naude, Greyling e Vogts (2010) reduz todos os programas a uma forma padrão, descarta informações importantes do processo de construção dessas estruturas e expressões.

Uma vantagem da estratégia de Naude, Greyling e Vogts (2010) no tratamento das variabilidades é a normalização de variáveis e a identificação da correspondência destas com as variáveis de outros programas. Essa normalização é muito importante para descobrir os

programas semelhantes em relação ao uso de memórias, o que facilita a avaliação automática e considera uma importante variável de avaliação para muitos professores que é o uso de memórias.

Assim como Naude, Greyling e Vogts (2010), tivemos a preocupação de realizar experimentos para mostrar a concordância entre as notas preditas por avaliação automática e as notas atribuídas por professores. Essa experimentação promove maior aceitação por parte dos professores para aplicar modelos de avaliação automática como apoio à correção de exercícios de programação.

Em relação ao trabalho de Moreira e Favero (2009), o nosso modelo se aproxima do deles ao realizar predição de notas por regressão linear múltipla (HAIR; TATHAM; BLACK, 1998) com base em métricas de código-fonte como, por exemplo, quantidade de palavras reservadas utilizadas, esforço de codificação (HALSTEAD, 1977) e a complexidade de código (MCCABE, 1976). Mas, de forma diferente de Moreira e Favero (2009), não reduzimos as frequências das palavras reservadas a um número representando a sua quantidade total como uma única variável independente de um modelo preditor de regressão linear múltipla. Um número apenas não explica como todas as palavras reservadas influenciam na nota de um professor. Consideramos, dessa forma, que cada palavra reservada de uma linguagem de programação é uma variável independente nesse modelo preditor, assim como os símbolos utilizados nas expressões lógicas.

O *Ceilidth* (BENFORD et al., 1995) é uma referência para o nosso modelo de avaliação semi-automático de exercícios de programação por avaliar programas de computador por métricas de análise dinâmica e estática. Conforme fizeram Benford et al. (1995), realizamos contagens para explicar o processo de desenvolvimento de um programa bem como o seu funcionamento correto. No entanto, essas características que contamos em um programa, utilizamos como variáveis independentes de um modelo de regressão linear. Assim, tentamos descobrir, através de um conjunto de exemplos, quais delas influenciam nas notas atribuídas por um professor e como elas influenciam. Depois disso, o nosso modelo de predição automática tenta imitar como o professor ao predizer as notas para predição de notas de outros exemplos.

A contribuição do *Assyst* (JACKSON; USHER, 1997) para o nosso modelo de avaliação semi-automática de programas de computador está na sua avaliação de corretude. A nossa avaliação de programas é predominantemente estática, mas reconhecemos que indicadores do funcionamento correto de um programa, como a compilação e a execução, são necessários porque influenciam na forma de atribuir notas de muitos professores de programação. O *Assyst* nos dá os indicadores essenciais de análise dinâmica: se um programa compila, se executa, se é eficiente e gera a saída correta conforme um modelo de saída fornecido. Dessa forma, utilizamos essas informações de forma quantificada como características representativas de um programa de computador.

Em outras palavras, para o nosso modelo de regressão linear de predição de notas ($y = a_1.x_1 + a_2.x_2 + \ldots + a_n.x_n$), algumas das características de avaliação dinâmica do *Assyst* representam variáveis independentes $x_i (i = 0 \ldots n)$ do modelo que explicam parcialmente uma nota y atribuída por um professor.

Combinando as vantagens, reduzindo as desvantagens, melhorando as estratégias e estendendo as ideias dos trabalhos relacionados, desenvolvemos um modelo de avaliação semi-automática de programação, que é apresentado em detalhes no Capítulo 6.

Capítulo 4

Sistemas de Recomendação

Um sistema de recomendação, em diferentes contextos, pode ter várias definições (MANOUSELIS; COSTOPOULOU, 2007). Em qualquer domínio de aplicação, um sistema de recomendação tem como propósito básico selecionar, dentre um amplo conjunto de itens (produtos, serviços, ações, informações e até pessoas), aqueles que podem ser mais interessantes, úteis ou necessários para um usuário (ZAIANE, 2002).

Aplicando a ideia de compor perfis a partir de comportamentos, preferências e características de usuários, sistemas de recomendação podem ser utilizados para aprender o que interessa a usuários e fazer-lhes recomendações de acordo (MCNEE; RIEDL; KONSTAN, 2006; BELL; KOREN, 2007; PU; CHEN; HU, 2012).

Em comércio eletrônico, sistemas de recomendação podem reconhecer perfis de clientes a partir de preferências ou hábitos de cmpras para recomendar-lhes produtos e serviços de acordo com seus interesses (SCHAFER; KONSTAN; RIEDI, 1999; LINDEN; SMITH; YORK, 2003). Em sistemas médicos, perfis de pacientes podem ser associados à presença de diferentes sintomas e medicamentos podem ser recomendados conforme as combinações desses sintomas (MEISAMSHABANPOOR; MAHDAVI, 2012; CHEN et al., 2012). De forma semelhante, sistemas de *e-learning* podem mapear perfis a partir das performances obtidas por estudantes em diferentes variáveis de avaliação e oferecer a esses estudantes instrução personalizada (BAYLARI; MONTAZER, 2009).

Os principais tipos de sistemas de recomendação são os sistemas recomendadores baseados em conteúdos, baseados em filtragem colaborativa e híbridos (ADOMAVICIUS; TUZHILIN, 2005; MANOUSELIS et al., 2011). Nos sistemas recomendadores baseados em conteúdos, itens são recomendados de acordo com as similaridades com outros itens solicitados por usuários no passado (LOPS; GEMMIS; SEMERARO, 2011). Nos sistemas recomendadores que são baseados em filtragem colaborativa, os itens são recomendados a usuários como tendo sido recomendados ou disponibilizados a outros usuários com perfis similares (HERLOCKER et al., 2004; SCHAFER et al., 2007; MANOUSELIS; VUORIKARI; ASSCHE, 2010). Já os sistemas recomendadores híbridos combinam as duas abordagens (BURKE, 2002; KLASNJA-MILICEVIC et al., 2011).

Assim como os sistemas recomendadores baseados em filtragem colaborativa, os sistemas recomendadores híbridos podem ser baseados em comportamentos e preferências de usuários (PU; CHEN; HU, 2012). No entanto, uma abordagem interessante, que é uma tendência na próxima geração de sistemas de recomendação, é contemplar perfis de usuários reconhecendo o que eles precisam e não somente o que eles querem ou preferem (PARK et al., 2012). Nessa abordagem, sistemas de recomendação são mais úteis para os usuários porque eles podem ajudá-los a resolver problemas, tomar decisões, simplificar e até transformar seus perfis.

As técnicas de *Data Mining* também têm sido amplamente aplicadas em sistemas de recomendação (AMATRIAIN et al., 2011). Os modelos mais usados são os Modelos Heurísticos, o *Algoritmo* kNN, o *Clustering*, as *Regras de Associação*, a *Análise de Links*, a *Regressão*, as *Árvores de Decisão* e as *Redes Neurais* (PARK et al., 2012). O *Algoritmo* kNN é uma técnica típica de sistemas de recomendação de filtragem colaborativa porque realiza recomendações através dos seguintes passos: construir perfis de usuários a partir de suas preferências, descobrir comportamentos de usuários semelhantes a um perfil e recomendar os *top-N* itens preferidos dos vizinhos mais próximos (PARK et al., 2012).

Através dessas técnicas, os sistemas de recomendação em *e-learning* podem ajudar a melhorar a aprendizagem dos alunos mapeando seus perfis, reconhecendo suas dificuldades de aprendizagem e recomendando-lhes atividades personalizadas. Acreditamos, portanto, que esses sistemas de recomendação seriam os mais adequados para áreas em que alunos

consideram ser as mais difíceis.

Sabendo que a avaliação formativa é um modelo de avaliação que melhora a aprendizagem porque monitora e regula o processo de aprendizagem (PERRENOUD, 1998), desenvolvemos um sistema de recomendação que pode indicar classes de atividades que remediam a prática da programação. O objetivo desse sistema é recomendar as classes de atividades mais apropriadas tal que estudantes possam melhorar seus desempenhos e reduzir as suas dificuldades de aprendizagem na prática da programação.

Este capítulo está organizado conforme a ordem a seguir. Na Seção 4.1, apresentamos os sistemas de recomendação no contexto educacional. Na Seção 4.2, apresentamos os trabalhos relacionados ao nosso sistema de recomendação. Na Seção 4.3, explicamos como nossa proposta estende as propostas dos trabalhos relacionados e concluimos com as considerações finais.

4.1 Sistemas de recomendação educacionais

Há diferenças entre sistemas de recomendação com relação aos seus propósitos. Em *e-commerce*, por exemplo, a recomendação é destinada a finalizar transações comerciais, auxiliar consumidores na compra de produtos, encorajá-los a comprar outros produtos e conquistar a fidelidade dos clientes (SCHAFER; KONSTAN; RIEDI, 1999; DRACHSLER; HUMMEL; KOPER, 2009). Por outro lado, no contexto educacional, os sistemas de recomendação de *Technology Enhanced Learning* - Tecnologia de Aprendizagem Melhorada - (TEL) lidam com informações sobre aprendizes e recomenda-lhes atividades de aprendizagem com o objetivo de favorecer o desenvolvimento de habilidades (DRACHSLER; HUMMEL; KOPER, 2009).

De acordo com Manouselis et al. (2011), algumas tarefas suportadas pelos atuais sistemas recomendadores são interessantes para sistemas de recomendação TEL como, por exemplo, a recomendação de novidades, a recomendação *peer-helpers* e a recomendação de caminhos alternativos através de fontes de aprendizagem.

A tarefa de recomendação de novidades é comum em sistemas de recomendação baseados em conteúdos porque requer técnicas de recomendação que analisem similaridades entre itens para melhor recomendá-los (MANOUSELIS et al., 2011; LOPS; GEMMIS; SEMERARO, 2011). No contexto educacional, um exemplo de sistema de recomendação de novidades é aquele que sugere materiais (páginas *web*, artigos, objetos de aprendizagem, atividades, etc.) para aprendizes de acordo com as atividades que realizam em um ambiente de aprendizagem ou com os conteúdos que estão estudando (GODOY; AMANDI, 2010; GHAUTH; ABDULLAH, 2011).

A recomendação *peer-helpers* é comum em sistemas de recomendação de filtragem colaborativa porque requer explorar atributos de perfis de usuários para indicar outros usuários que compartilhem os mesmos interesses (MANOUSELIS et al., 2011; GHAUTH; ABDULLAH, 2011). Um exemplo de sistemas que realizam essa forma de recomendação são aqueles que analisam o que aprendizes estão aprendendo e indicam, para ajudá-los, outros aprendizes que estão aprendendo os mesmos conteúdos, mas que estão em níveis mais avançados de aprendizagem (BOBADILLA; SERRADILLA; HERNANDO, 2009).

A recomendação de caminhos alternativos através das fontes de aprendizagem, por sua vez, pode ser realizada por sistemas recomendadores baseados em conteúdos e de filtragem colaborativa. Um exemplo de recomendação híbrida para recomendar uma sequência ou um caminho a ser seguido através de fontes de aprendizagem selecionadas é o sistema *Protus* desenvolvido por Klasnja-Milicevic et al. (2011). O *Protus* reconhece perfis de estudantes através de hábitos e estilos de aprendizagem. A partir desses perfis o sistema forma *clusters* de estudantes, descobre padrões de comportamentos e recomenda uma sequência de atividades a partir de frequentes sequências de aprendizagem percebidas e de acordo com os perfis dos aprendizes (KLASNJA-MILICEVIC et al., 2011).

A ideia de avaliação formativa como uma estratégia de remediar uma aprendizagem deficiente através da recomendação de materiais de aprendizagem tem sido uma tendência em sistemas recomendadores educacionais (CHEN; LEE; CHEN, 2005; BAYLARI; MONTAZER, 2009). Nos sistemas que aplicam essa ideia, os perfis de aprendizes são multidimensionais e são entendidos como modelo do aluno (BRUSILOVSKY; KOBSA; NEJDL, 2007). Como resultado, os perfis de estudantes são representados por habilidades ou

dificuldades de aprendizagem, conforme são reconhecidos nos desempenhos de estudantes em atividades ou testes, e os materiais de aprendizagem mais apropriados para cada perfil de estudante são recomendados. Essa abordagem de recomendação é similar ao recomendador médico de Meisamshabanpoor e Mahdavi (2012), que reconhece doenças pelos seus sintomas e recomenda para cada paciente o seu devido tratamento.

Um exemplo de aplicação dos conceitos de avaliação formativa em sistemas recomendadores é o agente de remediação baseado em Redes Neurais Artificiais (RNA) proposto por Baylari e Montazer (2009). Esse agente, a partir das habilidades estimadas pelo método da Teoria de Resposta ao Item (TRI) aplicado em testes, reconhece níveis de habilidades em perfis de aprendizes e recomenda-lhes atividades de acordo (BAYLARI; MONTAZER, 2009).

A recomendação de caminhos alternativos através de materiais ou atividades de aprendizagem pode ser realizada por sistemas recomendadores baseados em conteúdos e em filtragem colaborativa. Um modelo de recomendação híbrida, o *Protus* proposto por Klasnja-Milicevic et al. (2011), por exemplo, recomenda uma sequência ou caminho a ser seguido selecionando conteúdos de aprendizagem para aprendizes. Para isso, o *Protus* reconhece perfis de estudantes através de hábitos e estilos de aprendizagem. A partir desses perfis, o sistema forma *clusters* de estudantes, descobre padrões e recomenda uma sequência de atividades tendo como referências as frequentes sequências de conteúdos acessadas por estudantes e os padrões de perfis (KLASNJA-MILICEVIC et al., 2011). Nessa estratégia de recomendação, a sequência de atividades sugeridas é baseada no histórico de navegação através dos conteúdos de um curso. Nesse caso, o sistema pode localizar hábitos e preferências de um aprendiz, mas não necessariamente o ajuda a resolver seus problemas de aprendizagem. Dessa forma, o *Protus* poderia também avaliar quando uma sequência de atividades que é recomendada para um aprendiz de fato representa as atividades mais necessárias que poderiam melhorar os desempenhos desse aluno e, por conseguinte, a sua aprendizagem.

Na seção a seguir, mostramos como a nossa proposta de sistema recomendador, aplicado no domínio da aprendizagem de programação de computadores estende ou inova as propostas atuais de sistemas de recomendação no contexto educacional.

Sistemas de Recomendação					
Propostas	**Recomenda**	**Representação de Perfis**	**Referência**	**Estratégia**	**Aplicação**
(KATAKIS; TSOUMAKAS; VLAHAVAS, 2008)	*Tags* para objetos de aprendizagem	—	Itens similares	**Classificação multilabel**	*Tagging* de objetos de aprendizagem
(BAYLARI; MONTAZER, 2009)	Objetos	**Desempenhos de estudantes**	**Professor**	Redes Neurais Artificiais (RNA)	Predição de *ratings* para objetos de aprendizagem
(KLASNJA-MILICEVIC et al., 2011)	Sequência de atividades	Avaliações, Padrões de navegação	Outros Estudantes	algoritmo *AprioriAll*, *Clustering*	Localizar sequência de padrões navegacionais, Predição de *ratings*
(MEISAMSHABANPOOR; MAHDAVI, 2012)	Medicamentos	Índices dos sintomas	Outros pacientes	Algoritmo kNN	Predição de índices dos sintomas
(CARRILLO; LÓPEZ; MORENO, 2013)	Livros	*Ratings*	Outros usuários	**Classificação multilabel**	Predição de *ratings*
Nossa proposta	**Classes de atividades**	**Desempenhos de estudantes**	**Professor**	**Classificação multilabel**	**Recomendação das principais atividades**

Tabela 4.1: Trabalhos relacionados

4.2 Trabalhos relacionados

O diferencial do sistema recomendador proposto neste trabalho é tratar o problema de recomendação como um problema de classificação *multilabel*. Estendemos, dessa forma, as propostas de Katakis, Tsoumakas e Vlahavas (2008), Song, Zhang e Giles (2011), López et al. (2012), Becerra, Astudillo e Mendoza (2012), que fazem recomendações através da classificação *multilabel* de acordo com a similaridade entre itens, ao recomendar itens baseando-se nas similaridades entre perfis de usuários. Da mesma forma, estendemos a proposta de Carrillo, López e Moreno (2013) porque nossas recomendações são baseadas em recomendações realizadas por um especialista humano e não apenas em *ratings* de outros usuários.

Na Tabela 4.1, são apresentados os trabalhos relacionados à nossa proposta de sistema recomendador e como combinamos e estendemos essas propostas. Os itens em negrito informam as similaridades entre a nossa proposta e os trabalhos relacionados.

Em sistemas de recomendação, a classificação *multilabel* (ZHANG; ZHOU, 2007) é, em geral, associada à tarefa de *tagging*, que consiste em associar *tags* (ou descritores) que podem ser termos ou expressões para descrever padrões (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008; SONG; ZHANG; GILES, 2011; LÓPEZ et al., 2012). Nesse caso, recomendações de *tags* são para itens e não para usuários.

O trabalho de Carrillo, López e Moreno (2013) aplica algoritmos de classificação *multilabel*, baseado em similaridades de perfis, para predizer *ratings* (ou avaliações) de usuários e recomendar livros. Nesse trabalho, o objetivo da classificação *multilabel* é associar corretamente um conjunto de rótulos ou classes, onde cada classe é um valor inteiro da avaliação do usuário entre 0 e 10 para uma nova instância, que é representada por um vetor de características (CARRILLO; LÓPEZ; MORENO, 2013).

4.3 Conclusão

Este capítulo apresentou os conceitos, ideias, abordagens e tecnologias de sistemas de recomendação bem como estes são aplicados principalmente em contextos educacionais.

Assim como qualquer outro sistema de recomendação de filtragem colaborativa, nós seguimos os típicos passos desses sistemas (PARK et al., 2012). Para isso, nós utilizamos o algoritmo ML-kNN (ZHANG; ZHOU, 2007; LÓPEZ et al., 2012), que é uma extensão do tradicional Algoritmo kNN, para recomendar classes de atividades que são as mais apropriadas para estudantes com problemas de aprendizagem.

Nossa proposta estende a proposta de Klasnja-Milicevic et al. (2011) porque tem como referência de qualidade de recomendação as métricas de avaliação de classificação *multilabel* (SCHAPIRE; SINGER, 1999; ZHANG; ZHOU, 2007). Essas métricas confirmam que o *ranking* de classes de atividades recomendadas para um estudante reúnem de fato as principais atividades que o estudante precisa fazer e também são consistentes com as recomendações realizadas por um professor.

Seguindo os princípios de avaliação formativa tal como Baylari e Montazer (2009), Meisamshabanpoor e Mahdavi (2012), nós construímos perfis multidimensionais para

reconhecer problemas de aprendizagem que são caracterizados pelos desempenhos de estudantes em diferentes variáveis de avaliação.

Finalmente, recomendamos classes de atividades que remediam o processo de aprendizagem e suportam o desenvolvimento de habilidades (DRACHSLER; HUMMEL; KOPER, 2009).

Capítulo 5

Técnicas de Reconhecimento de Padrões

Os núcleos de avaliação diagnóstica e formativa do modelo de avaliação semi-automática são formados por algoritmos de reconhecimento de padrões combinando as abordagens de aprendizagem supervisionada e não-supervisionada (DUDA; HART; STORK, 2001).

Na abordagem de aprendizagem supervisionada, um algoritmo aprende a realizar ações como a classificação e a predição através de um conjunto de exemplos chamado treino que orienta essas ações quando aplicadas a novos exemplos (DUDA; HART; STORK, 2001; HAIR; TATHAM; BLACK, 1998; MANNING; RAGHAVAN; SCHUTZE, 2008; ZHANG; ZHOU, 2007). As redes neurais e o Algoritmo kNN são exemplos de algoritmos que utilizam abordagem de aprendizagem supervisionada.

O algoritmo ML-kNN, uma extensão do kNN para classificação *multilabel*, e a Regressão Linear também são técnicas de aprendizagem supervisionada que se utilizam de exemplos para classificar padrões e para realizar previsões, respectivamente.

Na abordagem de aprendizagem não-supervisionada os padrões se auto-organizam em classes não-rotuladas, de acordo com as características semelhantes, sem precisar de exemplos para orientar a classificação desses padrões (JAIN; MURTY; FLYNN, 1999).

Os algoritmos de *clustering* são exemplos de algoritmos de aprendizagem não-supervisionada porque analisam um conjunto de características de padrões para formar agrupamentos conforme as similaridades entre as características desses padrões.

As técnicas de Reconhecimento de Padrões utilizadas neste trabalho são apresentadas conforme a ordem a seguir. Na Seção 5.1, apresentamos os algoritmos de *clustering*, os principais métodos e o software *Cluto*, que é um pacote de algoritmos de *clustering* para dados de alta dimensionalidade. Na Seção 5.2, descrevemos a técnica de Regressão Linear e como fazer a sua análise. Na Seção 5.3, apresentamos o algoritmo ML-kNN e as suas métricas de avaliação. Na Seção 5.4, concluimos explicando como essas técnicas de Reconhecimento de Padrões foram aplicadas neste trabalho.

5.1 O Clustering

O *clustering* é uma técnica que utiliza a abordagem de aprendizagem não-supervisionada para agrupamento de padrões em classes ou *clusters* considerando as características semelhantes desses padrões. A Figura 5.1 ilustra um processo de *clustering* em que os pontos dos gráficos, que representam padrões, auto-organizam-se formando agrupamentos a partir da proximidade desses pontos.

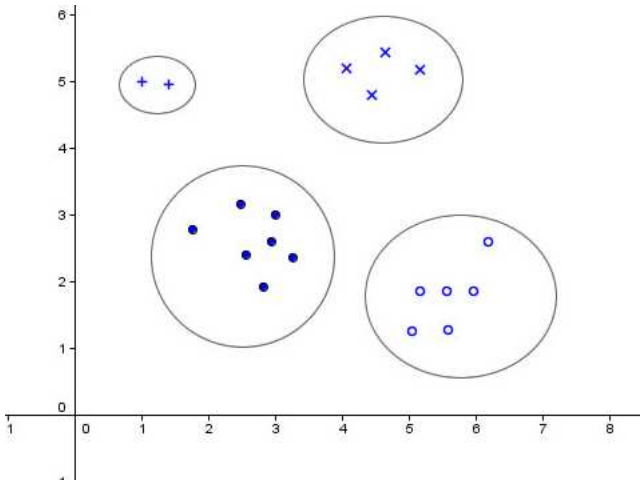

Figura 5.1: O *Clustering*

O objetivo do *clustering* é formar grupos caracterizados por alta homogeneidade entre padrões de um mesmo grupo e heterogeneidade entre padrões de grupos distintos. Isso significa que os padrões devem ser semelhantes entre si dentro do mesmo grupo e diferentes em relação aos padrões de outros grupos

Uma grande vantagem do *clustering* é facilitar a percepção e a extração de informações relevantes em grupos de padrões, principalmente em conjuntos que reúnem uma grande quantidade de dados.

A principal diferença entre o *clustering* e a classificação é que, na classificação, os padrões são reunidos por classes já conhecidas. No *clustering*, as características dos padrões são analisadas para a partir delas descobrir possíveis classes de padrões.

Uma típica atividade de *clustering* deve envolver os seguintes passos (JAIN; MURTY; FLYNN, 1999):

1. Representação de padrões
2. Definição de medidas de similaridade
3. Agrupamento
4. Abstração de dados, se necessário
5. Avaliação da saída, se necessário

A *Representação dos padrões* envolve a definição do número, tipo e modo de apresentação dos atributos que descrevem cada padrão. Em geral, um padrão é representado por um vetor cujas dimensões representam atributos ou características desse padrão.

Na representação de padrões, pode haver seleção e extração de características. A seleção de características consiste em identificar um subconjunto dos atributos de um padrão para descrevê-lo. Já a extração de características consiste em realizar transformações nos atributos de um padrão para melhor descrevê-lo.

A *Definição da medida de similaridade* é uma função de distância definida entre pares de padrões. São exemplos de medidas de similaridade as funções *cosseno*, *coeficiente de Jaccard*, *coeficiente de correlação* e *distância euclidiana* (KARYPIS, 2003).

O processo de *Agrupamento* pode ser realizado por diferentes métodos de *clustering*. As abordagens hierárquica e particional são as mais comuns.

A *Abstração de dados* é o processo de extrair uma representação compacta de um conjunto de dados direcionada para análise automática ou para o usuário. Para a

análise automática, deve ser uma representação que permita à máquina processar fácil e eficientemente. Já para o usuário, deve fornecer, preferencialmente, uma visualização gráfica que permita-lhe compreender facilmente os *clusters* formados e as suas relações. No *clustering*, uma típica abstração é uma descrição compacta de cada *cluster*, como a representação dos padrões de um *cluster* pelo seu centróide, que é o vetor-média dos padrões pertencentes a um *cluster*.

A *avaliação da saída* de um processo de *clustering* geralmente se recorre a critérios de otimização, muitas vezes definidos de forma subjetiva. O fato é que os *clusters* são produzidos a partir dos dados de entrada. Desse modo, o processo de *clustering* não pode ser validado como eficiente ou não caso os padrões de entrada em sua essência não se agrupam.

5.1.1 Os principais métodos de agrupamento k-Médias

O método de *clustering hieráquico* organiza padrões hierarquicamente obedecendo à similaridade entre esses padrões (MANNING; RAGHAVAN; SCHUTZE, 2008). Os resultados de *clustering hierárquico*, em geral, são apresentados na forma de dendrogramas, conforme a Figura 5.2. As raízes de um dendrograma representam *clusters* e as folhas, os padrões x_i desses *clusters*.

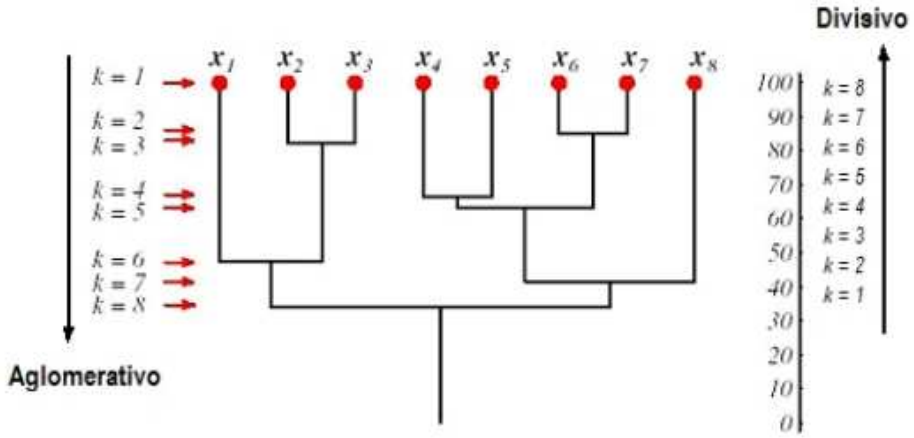

Figura 5.2: Clustering Hierárquico

Ao contrário de métodos hierárquicos, os métodos de *clustering particional* associam um conjunto de padrões a *K* grupos sem criar uma estrutura hierárquica. Desse modo, os

padrões deslocam-se entre *clusters* a cada etapa de execução de um algoritmo particional até que um critério de parada seja atingido. Esse critério pode ser a convergência dos *clusters* ou um número fixo de iterações. A convergência indica a estabilidade dos *clusters*, isto é, quando cada padrão encontra seu *cluster* mais adequado e não se desloca mais entre os demais *clusters* (MANNING; RAGHAVAN; SCHUTZE, 2008).

Entre os algoritmos de *clustering particional*, apresentamos como exemplos o tradicional *K-means* e a sua variação *Bisecting K-means* (LOOKS et al., 2007; STEINBACH; KARYPIS; KUMAR, 2000).

O *K-means* é uma técnica que utiliza o algoritmo de agrupamento de dados por K-médias, onde *K* representa o número de agrupamentos ou *clusters*. A ideia do *K-Means* é criar pontos centrais, isto é, centróides que representem cada *cluster*. O centróide de um grupo é calculado pela média ou pela mediana de todos os pontos de um agrupamento (LOOKS et al., 2007). Os centróides são recalculados a cada iteração do *K-means* até haver convergência, isto é, até que todos os pontos estejam nos grupos de seus centróides mais próximos. De acordo com Looks et al. (2007), o algoritmo *K-means* funciona da seguinte forma:

1. Selecionar K centróides aleatoriamente
2. Associar cada ponto de um conjunto de dados ao centróide mais próximo
3. Recalcular os centróides de cada *cluster* pela média ou mediana de seus pontos
4. Repetir os passos 2 e 3 até que os centróides não se alterem mais

O *K-means* tem como principais vantagens ser fácil de implementar e convergir rápido. A sua principal desvantagem é ser computacionalmente caro para grandes bases de dados (FAHIM et al., 2006).

O *Bisecting K-means* é uma variação do algoritmo *K-means*. Ele começa com um simples *cluster* e continuamente seleciona um *cluster* para dividir em dois *sub-clusters* até alcançar o número *K* de *clusters* desejados (LOOKS et al., 2007). O algoritmo *Bisecting K-means* funciona da seguinte forma (STEINBACH; KARYPIS; KUMAR, 2000):

1. Colocar um *cluster* para dividir

2. Dividir o *cluster* em dois *sub-clusters* utilizando o *K-means* tradicional.
3. Repetir o Passo 2, que é o passo de bissecção, por um número *I* de vezes e escolher a divisão que produzir *clusters* com maior similaridade.
4. Repetir os passos 1, 2 e 3 até alcançar o número de *clusters* desejado.

De acordo com Steinbach, Karypis e Kumar (2000), o *Bisecting K-means* tem apresentado melhor performance do que o *K-means* em muitos casos devido ao fato de produzir *clusters* de tamanhos mais uniformes em vez de *clusters* de tamanhos variáveis.

A técnica *K-means* e suas extensões estão entre as técnicas de *clustering* mais aplicadas em diversas áreas do conhecimento, entre elas, a recuperação de informação e a segmentação de imagens (FAHIM et al., 2006).

5.1.2 O Software Cluto

O *software Cluto 2.1.2* [1] é um pacote de *software* para *clustering* de conjuntos de dados de baixas e altas dimensões. O *Cluto* oferece três classes de algoritmos de *clustering* que operam diretamente sobre o espaço de características de objetos ou no espaço de similaridades desses objetos. Esses algoritmos são baseados nas abordagens particional, hierárquica aglomerativa e de particionamento por grafos (KARYPIS, 2003).

A característica principal do *Cluto* é que seus algoritmos tratam o problema do *clustering* como um processo de otimização que busca maximizar ou minimizar uma particular função-objetivo de *clustering*. O *Cluto* fornece sete funções-objetivos que podem ser aplicadas nos algoritmos de *clustering* hierárquico e particional (KARYPIS, 2003).

O *Cluto* também oferece ferramentas para descobrir *clusters* e compreender as relações entre os objetos associados a cada *cluster* e as relações entre diferentes *clusters*. Além disso, o *Cluto* identifica as características que melhor descrevem ou discriminam cada *cluster*.

[1] Software e documentação disponíveis em : http://glaros.dtc.umn.edu/gkhome/views/cluto

Os resultados de *clustering* podem ser compreendidos e analisados através dos recursos de visualização que o *Cluto* oferece. A Figura 5.3, por exemplo, é um gráfico de visualização de *clustering* emitido pelo *Cluto*.

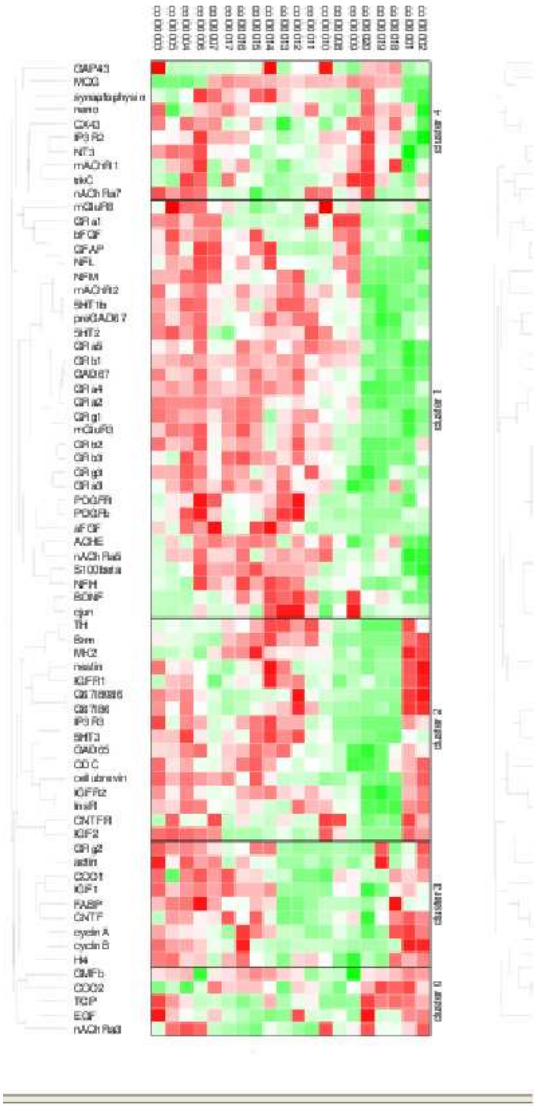

Figura 5.3: Visualização de *clusters* do *Software Cluto* (KARYPIS, 2003)

No gráfico da Figura 5.3 cada linha é uma folha ou padrão k_i de um dendrograma. Cada coluna representa cada característica desses padrões. Já os *clusters i* são apresentados como uma hierarquia desses padrões com base em medidas de similaridades. As tonalidades de cores representam quanto cada característica descreve e discrimina cada padrão de um *cluster*. Cada valor não-nulo das características dos padrões é mostrado em diferentes tonalidades de vermelho, sendo que os valores mais altos têm tonalidades mais fortes. Já para valores menores, essas tonalidades são mais fracas. A cor verde indica valores negativos

e a cor branca, por sua vez, representa valores nulos (KARYPIS, 2003).

Os pacotes de distribuição do *Cluto* consistem de dois programas, o *vcluster* e o *scluster*, que são utilizados na formação e análise de *clusters*. Esses pacotes incluem uma biblioteca que permite outras aplicações acessarem diretamente os algoritmos implementados no *Cluto*.

Os algoritmos de otimização de funções-objetivos do *Cluto* têm produzido soluções de *clustering* de alta qualidade (STEINBACH; KARYPIS; KUMAR, 2000). Com todas as vantagens oferecidas pelo *Cluto*, tanto pela eficiência quanto pela usabilidade, optamos por utilizá-lo neste trabalho para a realização dos experimentos apresentados no Capítulo 8.

5.2 A Regressão Linear

O termo *regressão* foi introduzido por Francis Galton em um famoso ensaio em que se verificou que, apesar da tendência de pais altos terem filhos altos e de pais baixos terem filhos baixos, a altura média dos filhos de pais de uma determinada altura tendia a *regredir* para a altura média de uma população como um todo (GUJARATI, 2006).

Sendo Y uma variável aleatória de interesse denominada de variável resposta e X uma variável aleatória regressora, o significado do termo *linear* de um modelo de regressão indica que a expectativa condicional de Y é uma função linear de X_i (GUJARATI, 2006).

Dessa forma, um Modelo de Regressão Linear Simples (MRLS) descreve a variável Y como uma soma de uma quantidade determinística (CHARNET et al., 1999) e uma parte aleatória. A parte determinística, que é uma reta em função de X, representa a informação de Y que é esperada a partir do conhecimento de X e, a parte aleatória, chamada *erro*, representa os inúmeros fatores que interferem no valor de Y (CHARNET et al., 1999).

A reta Y em função de X com os coeficientes β_0 e β_1, o erro ε, a variância do erro de Y, representada por σ^2, e x_i (um valor específico para X) sumarizam o MRLS a seguir (CHARNET et al., 1999).

$$Y = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Nessa equação β_0, β_1 e x_i são constantes, $E[\varepsilon_i] = 0$ e $Var[\varepsilon_i] = \sigma^2$, sabendo que a *Esperança* (E) é uma medida de tendência central e a *Variância* (*Var*), uma medida de dispersão em torno da *Esperança* (CHARNET et al., 1999).

Sob o MRLS, Y é a soma de uma constante, $\beta_0 + \beta_1 x$, com a variável aleatória ε (CHARNET et al., 1999). O modelo de regressão linear $Y = \beta_1 X + \beta_0$, com os coeficientes $\beta_0 = b$ (ou intercepto) e $\beta_1 = a$ são estimados conforme a reta da Figura 5.4.

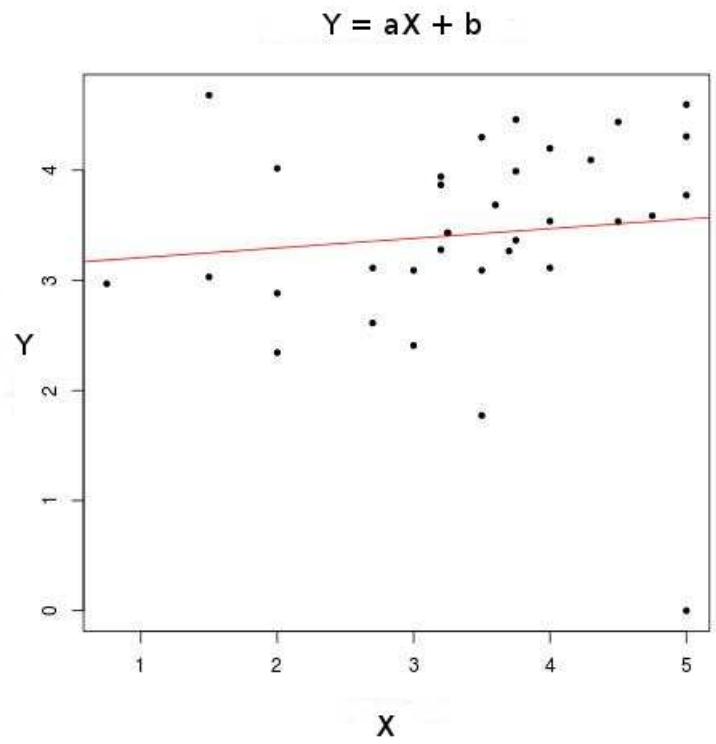

Figura 5.4: Modelo de Regressão Linear

Para escolher a reta que melhor representa um conjunto de n pontos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ da Figura 5.4, foi utilizado o Método dos Mínimos Quadrados (MMQ).

O MMQ, para um conjunto de possíveis retas, analisa as n diferenças entre cada valor y e cada valor da reta em função de x. A reta selecionada é a reta que apresenta a menor soma de quadrados de tais diferenças (CHARNET et al., 1999).

Sejam $\hat{y}_i = \hat{a} + \hat{b}x_i$, para $i = 1, \ldots, n$, os valores da reta de quadrados mínimos, ajustada ao conjunto de n pontos $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. As quantidades $e_i = y_i - \hat{y}_i$, para $i =$

$1, \ldots, n$ são denominadas *resíduos* (CHARNET et al., 1999).

Em muitas aplicações há situações em que há mais de uma variável regressora ou explicativa. Nesse caso, o modelo de regressão é chamado de Modelo de Regressão Linear Múltiplo (MRLM), definido como

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_k x_k + \varepsilon$$

Nessa equação Y é a variável dependente, $x_1, x_2, \ldots, x_k$ são as variáveis independentes, $E[Y] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_k x_k$ é a parte determinística, em que β_i determina a contribuição de cada variável independente x_i (FONSECA, 1995).

5.2.1 Análise de regressão linear

A análise de regressão é uma ferramenta analítica que ocupa-se do estudo da dependência de uma variável, a *dependente* Y, em relação a uma ou mais variáveis independentes x_i, com o objetivo de estimar e/ou prever a média (da população) ou o valor médio da variável dependente em termos dos valores conhecidos das variáveis independentes (GUJARATI, 2006; HAIR; TATHAM; BLACK, 1998).

De acordo com McCLAVE, BENSON e SINCICH (2008), os passos para análise de regressão linear são os seguintes:

1. Criar a hipótese do componente determinístico do modelo que relaciona a média $E[Y]$ às variáveis independentes $x_1, x_2, \ldots, x_k$ escolhidas para formar o modelo.
2. Estimar os parâmetros β_0, $\beta_1, \ldots, \beta_k$ a partir de dados amostrais.
3. Especificar a distribuição de probabilidade do termo erro aleatório ε e estimar o desvio padrão σ dessa distribuição.
4. Verificar se os pressupostos sobre ε estão satisfeitos e realizar modificações no modelo, se necessário.
5. Avaliar estatisticamente a utilidade do modelo.

6. Uma vez provada a utilidade do modelo, utilizá-lo para previsões e/ou estimativas.

A seleção de variáveis independentes do modelo no Passo 1 é um procedimento estatístico para escolher um conjunto de variáveis X_i que influenciam significativamente na variável dependente Y. Existem vários métodos estatísticos que auxiliam a seleção do melhor conjunto de variáveis, dentre os quais citamos: *Forward*, *Backward* e *Stepwise* (HAIR; TATHAM; BLACK, 1998; CORRAR; PAULO; FILHO, 2007).

O algoritmo *stepwise* utilizado neste trabalho para seleção de variáveis do modelo de regressão múltipla, realiza as seguintes etapas (HAIR; TATHAM; BLACK, 1998):

i Selecionar a variável independente inicial que apresenta a maior correlação com a variável dependente.

ii Se a variação explicada é estatisticamente significante, verificar se há mais variáveis disponíveis. Se não, avançar para Linha vii.

iii Se há mais variáveis disponíveis, selecionar variável independente adicional. Senão, avançar para Linha vi.

iv Se a variância explicada pelas variáveis independentes inicial e adicional não for significante, dispensar variáveis insignificantes

v Voltar a Linha iii.

vi Verificar adequação do modelo

vii Finalizar

Cada coeficiente β_i do Passo 2 representa o montante de variação na variável dependente em relação a uma unidade de variação da variável independente x_i (HAIR; TATHAM; BLACK, 1998).

No Passo 3, o desvio-padrão $\hat{\sigma}$ mede a precisão dos estimadores $\hat{\beta}_i$.

Na análise estatística do Passo 5, as inferências sobre os parâmetros β_i estimados no Passo 2 são obtidas utilizando um intervalo de confiança ou testes de hipótese (testes t). Já os coeficientes de determinação múltipla R^2 e de determinação múltipla ajustada $R_a{}^2$ indicam

quão bem a equação de previsão se ajusta aos dados amostrais (MCCLAVE; BENSON; SINCICH, 2008; GUJARATI, 2006).

O intervalo de confiança nos fornece informação sobre a precisão das estimativas, no sentido de que quanto menor a amplitude do intervalo, maior a precisão do modelo (CHARNET et al., 1999).

Para estimar o intervalo de confiança, isto é, saber quão próximo um parâmetro $\hat{\beta}_i$ estimado está de β_i, tentamos descobrir dois números, δ e α, este último entre 0 e 1, de modo que a probabilidade do intervalo aleatório $(\hat{\beta}_2 - \delta, \hat{\beta}_2 + \delta)$ conter o verdadeiro β_2 é de $1 - \alpha$ (GUJARATI, 2006). Por exemplo, se $\alpha = 0.05$, há uma probabilidade de 95% (1 - 0.05) de o intervalo incluir β_2. Simbolicamente, o estimador de intervalo pode ser descrito como (GUJARATI, 2006):

$$Pr(\hat{\beta}_2 - \delta \leq \beta_2 \leq \hat{\beta}_2 + \delta) = 1 - \alpha$$

O coeficiente de determinação R^2, é interpretado como a proporção da variabilidade dos Ys observados, explicada pelo modelo considerado. O valor de R^2 pertence ao intervalo $[0;1]$ e, quanto mais próximo de 1, melhor é o ajuste do modelo (CHARNET et al., 1999).

Quando $R^2 = 1$, todos os pontos observados se situam sobre a reta de regressão. Por outro lado, se $R^2 = 0$, as variações de Y são aleatórias e a inclusão da variável X no modelo não incorporará informação alguma sobre as variações de Y (FONSECA, 1995).

O coeficiente de determinação R^2, no MRLS, é definido pela relação expressa na equação a seguir (CHARNET et al., 1999):

$$R^2 = \frac{SQR_{eg}}{SQT}$$

Nessa equação SQR_{eg} é a *Soma de Quadrados de Regressão*, representando a variação das esperanças específicas de Y, dado x, em torno da sua média, é medida por

$$SQR_{eg} = \sum_{i=1}^{n} (\hat{y}_i - \overline{y_i})^2$$

O valor de SQT é a *Soma de Quadrados Total* (ajustada), representando a variação de Y em torno da sua média, é medido por

$$SQT = SQE + SQR_{eg} = \sum_{i=1}^{n} (y_i - \overline{y_i})^2$$

O valor de SQE é a soma de quadrados do erro representando a variação de Y em torno da reta e medido por

$$SQE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Como uma alternativa ao uso de R^2 como medida de adequação do modelo, podemos utilizar o coeficiente de determinação múltiplo ajustado representado por ${R_a}^2$. Embora a interpretação de R^2 e ${R_a}^2$ sejam similares, ${R_a}^2$ leva em consideração o tamanho da amostra n e o número de parâmetros β no modelo. Por isso, ${R_a}^2$ será menor que R^2 e, principalmente, não pode ser induzido a aproximar-se do valor 1 ao adicionar variáveis independentes ao modelo de regressão como acontece com o valor de R^2 (MCCLAVE; BENSON; SINCICH, 2008). Mas ${R_a}^2$ pode cair se as variáveis independentes acrescentadas tiverem pouco poder de explicação e/ou se os graus de liberdade se tornarem muito pequenos (HAIR; TATHAM; BLACK, 1998).

Através das estatísticas F podemos realizar um teste global do modelo envolvendo todos os parâmetros β_i (exceto β_0). Nesse teste, testamos a hipótese nula $H_0 : \beta_1 = \beta_2 = \ldots = \beta_k = 0$ e a hipótese não-nula H_a, isto é, se pelo menos um dos β_i é diferente de 0 (MCCLAVE; BENSON; SINCICH, 2008).

As quantidades necessárias para calcular o valor observado da estatística do teste são comumente dispostas em tabela denominada Tabela de Análise de Variância ou *Tabela de ANOVA*. Para o MRLS, a Tabela de ANOVA é descrita conforme a Tabela 5.1 (CHARNET et al., 1999):

ANOVA

Fonte (Fonte de variação)	GL (Graus de Liberdade)	SQ (Soma de Quadrados)	QM(Quadrado Médio)	F_0
Regressão	1	SQR_{eg}	SQR_{eg}	$\frac{SQR_{eg}}{SQE/(n-2)}$
Erro	$n-2$	SQE	$SQE/(n-2)$	
Total	$n-1$	SQT		

Tabela 5.1: Tabela ANOVA - MRLS

Na Tabela 5.1, o número $n-2$ é o número dos graus de liberdade (GL) no MRLS, que é a diferença entre o número n de amostras ou observações e o número de parâmetros βs estimados que, na Tabela 5.1 é igual a 2.

A Tabela 5.2 estende a Tabela de ANOVA do MRLS (Tabela 5.1) para o MRLM, acrescentando novas linhas, obtidas da linha do corpo da tabela, referente ao *Erro*, que é subdividida em duas novas linhas: Falta de ajuste e Erro puro (CHARNET et al., 1999).

ANOVA

Fonte (Fonte de variação)	GL (Graus de Liberdade)	SQ (Soma de Quadrados)	QM(Quadrado Médio)	F_0
Regressão	1	SQR_{eg}	SQR_{eg}	$\frac{SQR_{eg}}{SQE/(n-2)}$
Erro	$(n-2)$	SQE	$SQE/(n-2)$	
(Falta de ajuste)	$(k-2)$	SQF_a	$\frac{SQF_a}{(k-2)}$	$\frac{[SQF_a/(k-2)]}{[SQE_p/(n-k)]}$
(Erro puro)	$n-k$	SQE_p	$\frac{SQE_p}{(n-k)}$	
Total	$n-1$	SQT		

Tabela 5.2: Tabela ANOVA - MRLM

Na Tabela 5.2, o valor $(n-k)$ é o grau de liberdade do sistema para k coeficientes β ou $k-1$ variáveis x_i do MRLM. O valor SQE é a soma de quadrados dos resíduos, representando a variação em torno da reta. O valor de SQE_p é a soma dos quadrados do *erro puro*, representando a variação de Y, para X fixo, independente do modelo (CHARNET et al., 1999). O valor SQF_a, por sua vez, é a soma dos quadrados por falta de ajuste, representando a falta de ajuste do modelo (CHARNET et al., 1999).

Após ser criado, um modelo de regressão linear, conforme o Passo 6, pode ser utilizado para predizer um valor Y_0 não observado correspondente a $X = x_0$. Se aplicarmos o MRLS ajustado, dizemos que $\hat{Y_0} = \beta_0 + \hat{\beta}_1 x_0$ é uma predição de Y_0, com o erro de predição definido por (CHARNET et al., 1999; HAIR; TATHAM; BLACK, 1998):

$$Y_0 - \hat{Y_0}$$

Essa relação será a base para construção do intervalo de predição e, quando um valor de

x_0 não pertencer a esse intervalo, ocorre a extrapolação (CHARNET et al., 1999).

Para maximizar a predição a partir de um conjunto de variáveis independentes, deve-se selecionar variáveis independentes que tenham baixa multicolinearidade, isto é, baixa correlação com outras variáveis independentes do modelo, mas que apresentem alta correlação com a variável dependente.

Os fatores que afetam a amplitude do intervalo de predição são o tamanho da amostra, o desvio-padrão, o desvio de x em relação à sua média $\bar{x}$ e a variabilidade dos x_i observados. Assim, quanto maior for o tamanho da amostra n, quanto menor for a variabilidade dos x_i observados e quanto menor o desvio-padrão, menor será a amplitude do intervalo de predição. Em contraste, quanto mais x_i se afastar de $\bar{x}$, mais amplo será o intervalo de predição (FONSECA, 1995).

5.3 O Algoritmo ML-kNN

O Algoritmo ML-kNN é uma versão do tradicional Algoritmo kNN (k-Nearest Neighbour) (DUDA; HART; STORK, 2001) designado para a classificação multirrotulada de padrões (ZHANG; ZHOU, 2007).

Para entender o algoritmo, suponha que precisemos classificar um padrão X (ou d_j) em uma ou mais classes de um conjunto C de classes. Seja $TV = d_1, d_2, ..., d_{|TV|}$ um conjunto de perfis X usados para treinar e validar o Algoritmo ML-kNN.

Inicialmente, o ML-kNN identificará os k vizinhos mais próximos de X no conjunto de treino TV usando uma medida de similaridade (MANNING; RAGHAVAN; SCHUTZE, 2008). Seja $T_{i,1}(X)$ o evento em que X é rotulado na classe c_i e seja $T_{i,0}(X)$ o evento em que X não é rotulado na classe c_i. Além disso, considere que $E_{i,k_i}(X)$ ($k_i \in \{0, 1, \ldots, k\}$) denota o evento em que há exatamente k_i amostras das classes c_i entre os k vizinhos mais próximos de X. Então, baseado na regra bayesiana, a probabilidade de classificar X na classe c_i é dada por (ZHANG; ZHOU, 2007):

$$P_i(X) = \arg\max_{b=0}^{1}[P(T_{i,b})P(E_{i,k_i}(X)|T_{i,b}(X))], \tag{5.1}$$

onde $P_i(X)$ representa a probabilidade de X ser associado à classe c_i. $P(T_{i,b})$ é a probabilidade *a priori* de X ser rotulado ou não-rotulado na classe c_i e $P(E_{i,k_i}(X)|T_{i,b}(X))$ é a probabilidade *a posteriori*. Essas duas probabilidades podem ser estimadas diretamente de seu conjunto de treino TV.

Os cálculos de $P(T_{i,b})$ e $P(E_{i,k_i}(X)|T_{i,b}(X))$ são explicados a seguir. Primeiro, os valores de probabilidade $P(T_{i,b})$ são calculados na fase de treino para cada classe c_i ($i \in \{1,\ldots,|C|\}$, onde $|C|$ é o número de classes) usando a Equação 5.2:

$$P(T_{i,1}) = \frac{\delta + n_i}{2\delta + n}, \quad P(T_{i,0}) = 1 - P(T_{i,1}), \tag{5.2}$$

onde n_i é o número de amostras de treino rotuladas na classe c_i, n é o número total de amostras de treino no conjunto TV e δ é o parâmetro de *smoothing* de probabilidade. Para $\delta = 1$, é alcançado pelo *Smoothing Laplaciano*.

Em seguida, para cada amostra de treino d_j, nós computamos na fase de treino os k vizinhos mais próximos. E, para cada classe c_i e amostra d_j, nós computamos o valor de k_i. Se d_j é rotulado na classe c_i, então o valor $L_i(k_i)$ é incrementado de um; caso contrário, o valor de $\overline{L_i(k_i)}$ é incrementado de um.

$L_i(k_i)$ e $\overline{L_i(k_i)}$ contam quantas amostras de treino são associadas com a classe c_i e quantas não são, respectivamente; incluído entre os k vizinhos mais próximos são exatamente as k_i amostras que são classificadas na classe c_i.

Depois desses passos, o valor de k_i e de cada classe é computado para a amostra X, os valores de probabilidades $P(E_{i,k_i}(X)|T_{i,b}(X))$ são calculados para cada classe c_i usando as Equações 5.3 e 5.4.

$$P(E_{i,k_i}|T_{i,1}(X)) = \frac{\delta_X + L_i(k_i)}{\delta X(k+1) + \sum_{r=0}^{k} L_i(r)} \tag{5.3}$$

$$P(E_{i,k_i}|T_{i,0}(X)) = \frac{\delta_X + \overline{L_i(k_i)}}{\delta_X(k+1) + \sum_{r=0}^{k} \overline{L_i(r)}} \tag{5.4}$$

Os parâmetros que devem ser ajustados para o ML-kNN são o número de vizinhos k e o parâmetro de *smoothing* δ. Isso pode ser observado nas Equações 5.2, 5.3 and 5.4, e esse valor de δ muda levemente para as probabilidades a *priori* ($P(T_{i,b})$) e *posteriori* ($P(E_{i,k_i}(X)|T_{i,b}(X))$), respectivamente.

Em resumo, o algoritmo ML-kNN recebe como entradas o número k de vizinhos mais próximos, o parâmetro de *smoothing* δ, calcula as probabilidades a priori e posteriori de cada classe c_i, e fornece como saída a probabilidade de o padrão a ser classificado ser de cada classe c_i.

5.3.1 As métricas de avaliação

A avaliação de classificadores *single label* é geralmente conduzida pelo uso de métricas tradicionais, tais como *Precision, Recall* e *F-measure* (JONES, 1981; BAEZA-YATES; RIBEIRO-NETO, 1999). Mas a avaliação de classificadores *multilabel* é mais complexa porque considera o *ranking* de um conjunto de classes que o classificador recomenda como as mais prováveis para uma instância assim como a relevância de cada uma dessas classes para essa instância.

Sendo s o número de perfis utilizados para testes, nós apresentamos cada uma das métricas que são utilizadas para avaliar o desempenho do classificador *multilabel* (SCHAPIRE; SINGER, 1999; ZHANG; ZHOU, 2007) conforme as descrições a seguir.

hamming loss

Avalia quão frequente uma instância de teste d_j é erradamente classificada em uma classe, isto é, quando é classificada como pertinente e é não-pertinente ou quando é classificada pertinente e é, na verdade, não-pertinente. Quanto menor for o valor de *hamming loss*, mais eficaz será a recomendação de classes pelo classificador. O valor de *hamming loss* é calculado pela equação:

$$\text{hloss} = \frac{1}{s}\sum_{j=1}^{s}\frac{1}{|C|}|P_j \Delta A_j|,$$

onde $|C|$ é o número de classes, e Δ é a diferença simétrica entre o conjunto P_j de classes preditas e o conjunto A_j de classes associadas à instância de teste d_j. As classes preditas são aquelas que estão mais próximas do topo do *ranking* e que estão acima do limite (ou *threshold*) τ.

one-error

Avalia quantas vezes a classe indicada pelo classificador como a principal classe não aparece no conjunto A_j de classes pertinentes para uma instância d_j. Quanto mais próximo for o valor de *one-error* de 0, melhor será a performance do classificador. O valor de *one-error* é calculado por

$$\text{one-error} = \frac{1}{s}\sum_{j=1}^{s}\text{error}_j, \quad \text{error}_j = \begin{cases} 0, \text{ Se } [\arg\max_{c\in C} f(d_j,c)] \in A_j \\ 1, \text{ caso contrário.} \end{cases}$$

onde $[\arg\max_{c\in C} f(d_j,c)]$ retorna a classe do topo do *ranking* para a instância d_j.

coverage

Avalia quanto se deve descer no *ranking* de classes recomendadas pelo classificador para cobrir todas as classes pertinentes de uma instância. O valor de *coverage* varia de 0 a $|C|-1$. Quanto menor for o valor de *coverage*, melhor será a performance do classificador em acertar o conjunto de classes pertinentes para uma instância d_j. O valor de *coverage* é calculado pela equação

$$\text{coverage} = \frac{1}{s}\sum_{j=1}^{s}(\max_{c\in A_j} r(d_j,c) - 1),$$

onde $\max_{c\in A_j} r(d_j,c)$ retorna o *ranking* máximo para o conjunto de classes pertinentes para a instância de teste d_j.

ranking loss

Avalia a fração de pares de classes pertinentes A_j que são ordenadas em ordem inversa. Em outras palavras, *ranking loss* avalia quantas vezes, no *ranking* de classes associadas à

instância d_j, classes pertinentes estarão dispostas abaixo das classes não pertinentes. Quanto menor for o valor de *ranking loss*, melhor será a performance do classificador. O valor de *ranking loss* é calculado conforme a equação

$$\text{rloss} = \frac{1}{s}\sum_{j=1}^{s}\frac{|\{(c_k,c_l)|f(d_j,c_k) \leq f(d_j,c_l)\}|}{|A_j||\bar{A_j}|},$$

onde $(c_k,c_l) \in A_j \times \bar{A_j}$ and $\bar{A_j}$ é o conjunto complementar de A_j em C.

average precision

Avalia para cada classe $c_i \in A_j$, a fração média das classes no *ranking* de classes que são associadas a uma instância de teste d_j e que estão acima de c_i. Quanto maior for o valor de *average precision* melhor será a precisão de classificação e mais corretamente as classes recomendadas estarão próximas ao topo do *ranking*. O valor de *average precision* é calculado por

$$\text{avgprec} = \frac{1}{s}\sum_{j=1}^{s}\frac{1}{|A_j|}\sum_{k=1}^{|A_j|} precision_j(R_{jk}),$$

onde R_{jk} é o conjunto de classes ordenadas que são descidas da classe do topo do *ranking* até a posição k do *ranking*, onde está uma classe $c_i \in A_j$ para d_j, e onde $precision_j(R_{jk})$ é o número de classes pertinentes em R_{jk} dividido por $|R_{jk}|$.

Em resumo, os menores valores de *hamming loss*, *one-error*, *coverage*, e *ranking loss*, e o maior valor de *average precision* indicam a melhor performance de um classificador. A performance ideal é quando *hamming loss*, *one-error*, *coverage* e *ranking loss* são iguais a zero e quando *average precision* é igual a 1.

5.4 Conclusão

Neste capítulo apresentamos as técnicas de *clustering* e de regressão linear e o algoritmo ML-kNN. O *clustering* (JAIN; MURTY; FLYNN, 1999; STEINBACH; KARYPIS; KUMAR, 2000; MANNING; RAGHAVAN; SCHUTZE, 2008) é uma técnica

de aprendizagem não-supervisonada utilizada para agrupamento de padrões conforme uma medida de similaridade e o método de agrupamento, que pode ser hierárquico ou particional. A regressão linear é uma técnica analítica muito utilizada para explorar relações de dependência e fazer predições a partir delas (HAIR; TATHAM; BLACK, 1998; HAMALAINEN; VINNI, 2010; CHARNET et al., 1999). O algoritmo ML-kNN, que é de abordagem supervisionada, por sua vez é utilizado para, a partir de exemplos de padrões pré-classificados em uma ou mais classes, identificar possíveis classes para novos padrões conforme a similaridade com os padrões-exemplos aprendidos (ZHANG; ZHOU, 2007).

Neste trabalho o *clustering* foi utilizado para mapeamento de perfis e para auxiliar a predição de notas de atividades de alunos por um modelo de regressão linear múltipla. Para mapear o perfil de um aluno, representamos esse aluno por um vetor cujas dimensões são variáveis de avaliação da aprendizagem de programação. Como valor de cada variável foi atribuído o desempenho desse aluno em uma componente de habilidade do domínio de programação em um exercício de programação resolvido pelo aluno. O valor da variável de avaliação poderia ser também o desempenho em uma tarefa, que representa um conjunto de atividades. Para identificar os perfis semelhantes, utilizamos o método de *clustering* hierárquico (MANNING; RAGHAVAN; SCHUTZE, 2008; KARYPIS, 2003).

Os perfis que foram mapeados por desempenhos em tarefas foram utilizados como relatórios de acompanhamento da aprendizagem para identificar classes de alunos bem como suas dificuldades de aprendizagem. Os perfis mapeados por desempenhos em componentes de habilidades em um exercício de programação resolvido por um aluno, além de terem sido utilizados como relatórios de análise de atividades de programação, foram enviados ao módulo de avaliação semi-automática do NAD para serem pontuados.

Os algoritmos de *clustering*, quando recebem esses perfis, que representam soluções de alunos para um exercício, formam *clusters* a partir das similaridades entre as componentes de habilidades. Para cada *cluster* formado, foram selecionadas algumas amostras de perfis para um professor pontuar e, a partir desses exemplos pontuados, o algoritmo de predição de notas, que é baseado em um modelo de regressão linear múltipla, estimou as notas para as demais amostras do *cluster*.

Além de auxiliarem a seleção de amostras representativas como referências de predição de notas, os algoritmos de *clustering* também foram utilizados para identificar *outliers* e também para a seleção das componentes que melhor descrevessem as amostras reunidas em um *cluster*.

Os perfis baseados em componentes de habilidades também foram utilizados pelo NAF para recomendação de atividades. Para isso, o NAF utilizou um algorimo de classificação *multilabel*, o ML-kNN (ZHANG; ZHOU, 2007), para analisar os perfis e recomendar-lhes classes de atividades de programação conforme as dificuldades de aprendizagem reconhecidas nas medidas das componentes de habilidades. Dessa forma, reformulando o problema de recomendação em um problema de classificação *multilabel*, utilizamos o algoritmo ML-kNN para aprender um conjunto de perfis de alunos e as classes de atividades associadas a eles e, a partir dessa aprendizagem, associar (recomendar) classes de atividades para outros perfis de alunos.

No próximo capítulo, descreveremos em detalhes como o *clustering*, a regressão linear e o algoritmo ML-kNN foram utilizados neste trabalho para a avaliação diagnóstica e formativa de estudantes de programação.

Capítulo 6

Um Modelo de Avaliação Semi-automática de Programação

A metodologia de avaliação automática da aprendizagem de programação apresentada neste capítulo tem por objetivo oferecer um mecanismo que possibilite realizar a gestão da aprendizagem de programação por estratégias de avaliações diagnóstica e formativa (PERRENOUD, 1999). Para isso, tratamos a aprendizagem como um processo que pode ser monitorado e regulado se observado como um conjunto de variáveis ou componentes de habilidades que podem ser medidas. Segundo Anderson (2000), aplicando essa estratégia de controlar componentes de habilidades, é possível levar estudantes rapidamente ao domínio de habilidades complexas.

O Modelo de Avaliação Automática da Aprendizagem de Programação que propomos neste trabalho é representado por dois núcleos de avaliação: o Núcleo de Avaliação Diagnóstica (NAD) e o Núcleo de avaliação Formativa (NAF). O NAD é responsável pela correção semi-automática de exercícios de programação e pelo mapeamento de perfis de aprendizagem em componentes de habilidades. O NAF é responsável pela regulação dessas componentes através da recomendação contínua de atividades de programação conforme os estados de perfis dos alunos até que se alcance um nível de aprendizagem satisfatório.

Os núcleos de avaliação NAD e NAF foram implementados em algoritmos de reconhecimento de padrões escritos nas linguagens C (SCHILDT; MAYER, 2006), R

(TORGO, 2009) e *Shell Script*. Esses núcleos comunicam-se com o mundo externo através do sistema SOAP (Sistema *Online* de Atividades de Programação), que foi desenvolvido em Linguagem PHP.

A interface *web* do SOAP recebe especificações de tarefas por professores e submissões de exercícios de programação pelos alunos. Os exercícios submetidos, que são programas de computador em Linguagem C, são compilados e executados no Núcleo Executor (NE) do SOAP, que é uma estrutura de máquinas virtuais que executam os programas submetidos, isto é, realizam a avaliação dinâmica, de forma paralela e segura. Futuramente, a interface do SOAP será uma *Application Programming Interface* -Interface de Programação de Aplicativos (API), que possibilitará a integração a outros ambientes virtuais de aprendizagem como o *Moodle*, por exemplo.

A Figura 6.1 representa a nossa proposta metodológica de avaliação semi-automática da aprendizagem de programação. Essa arquitetura mostra como os núcleos NAD e NAF interagem com o SOAP para realizarem a prática assistida da programação.

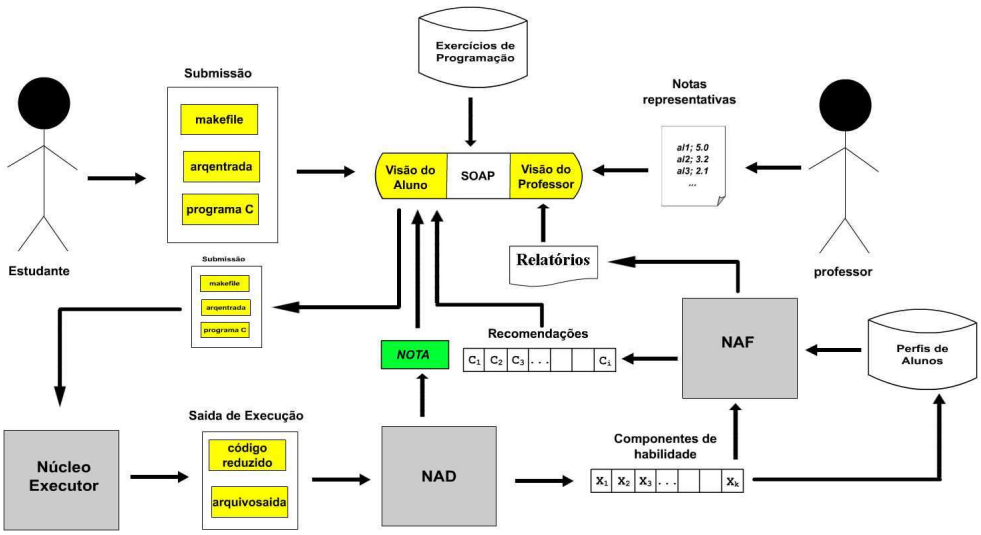

Figura 6.1: Modelo de avaliação semi-automática da aprendizagem de programação

O modelo de avaliação semi-automática da Figura 6.1 consiste de módulos funcionais que realizam as seguintes tarefas: receber programa submetido por um aluno, rodar o programa, obter um código simplificado desse programa, mapear o código reduzido e os resultados de execução do programa em componentes de habilidades, atribuir nota imitando o padrão de correção do professor, e recomendar atividades conforme as medidas das

componentes de habilidades.

Na *Visão do aluno* do SOAP, na Figura 6.1, além do programa em Linguagem C, o aluno deve submeter o arquivo *makefile* e os arquivos de entrada do programa. O *makefile* contém a sequência de instruções realizadas para rodar um programa como, por exemplo, as instruções de compilação e de execução. Já os arquivos de entrada devem conter as entradas que, em geral, os alunos fornecem por teclado ao rodar um programa.

A Figura 6.2 apresenta um exemplo de arquivo *makefile*. Nesse arquivo, a cláusula *all* chama as cláusulas *compila* e *executa* que, contêm, respectivamente, instruções para compilar e executar um programa direcionando todos os resultados para o arquivo *arquivosaida*. A cláusula *clear*, por sua vez, contém as instruções para remover os arquivos gerados a partir da execução do programa C.

```
all: compila executa

compila:
	gcc prova1.c -o executavel > compilacao.txt

executa:
	./executavel < arquivoentrada  > arquivosaida


clear:
	rm executavel arquivosaida
```

Figura 6.2: Arquivo *makefile*

Os arquivos submetidos são então enviados para o *Núcleo Executor* (Figura 6.1) que executa as instruções contidas no arquivo de *makefile*. Em seguida, o programa em Linguagem C submetido é reduzido à sua forma essencial, que consiste de palavras reservadas, símbolos da linguagem de programação e de *tokens* que informam se a compilação e a execução de um programa submetido foram realizadas com sucesso.

O código reduzido e o *arquivosaida* composto pelos resultados de execução do programa são enviados para o NAD, que os mapeia em *componentes de habilidades*. Essas componentes são variáveis $x_i (i = 1, 2, 3, ..., k)$ que representam desempenhos no uso de estruturas, palavras reservadas, símbolos e indicadores de execução, como a compilação e a execução do programa.

Conforme a Figura 6.1, a partir das *notas representativas* atribuídas por um professor a um conjunto de soluções desenvolvidas para o mesmo programa da atual *submissão*, o NAD atribui uma *nota*, que o aluno visualiza através da *Visão do aluno* do SOAP.

Na Figura 6.1, cada representação em *componentes de habilidades* é definida como um estado de perfil de aluno. Esse estado de perfil é armazenado em um repositório que contém estados de *perfis de alunos*.

As recomendações de classes de atividades, que são realizadas a partir de recomendações enviadas para outros alunos com perfis similares, são mostradas na *Visão do aluno* do SOAP quando um aluno faz uma atividade e as medidas das *componentes de habilidades* nessa atividade indicam desempenhos insuficientes. Através do NAF, essas componentes podem ser realimentadas por recomendação contínua de atividades até que os desempenhos de um aluno nessas componentes se estabilizem.

O NAF também fornece para o professor, através da *Visão do professor*, os relatórios de desempenhos dos alunos ao longo do tempo e os estados das *componentes de habilidades* em mapas de perfis de turmas que auxilia o professor a identificar classes de alunos bem como dificuldades e habilidades de aprendizagem de uma turma.

As interfaces do sistema SOAP e o seu Núcleo Executor são descritos em detalhes no Capítulo 7. Neste capítulo apresentamos os núcleos NAD e NAF. Na Seção 6.1, apresentamos a arquitetura do NAD. Na Seção 6.1.2, explicamos como os perfis de alunos são construídos para serem utilizados para processamento nos núcleos de avaliação. Na Seção 6.2, descrevemos a arquitetura do NAF. Na Seção 6.3, concluimos com as inovações e considerações finais do modelo desenvolvido neste trabalho.

6.1 Núcleo de Avaliação Diagnóstica (NAD)

O modelo de avaliação diagnóstica do NAD é formado por dois módulos: o módulo de avaliação semi-automática de exercícios de programação e o módulo de mapeamento de perfis de alunos.

O módulo de avaliação semi-automática de exercícios de programação recebe uma coleção de programas em Linguagem C reunidos em tarefas. Dessa base de exercícios resolvidos por alunos, é selecionado um conjunto de amostras representativas para o professor atribuir notas. Essa base pontuada pelo professor será a referência de predição das notas dos demais exercícios da base.

O módulo de mapeamento de perfis de alunos recebe do módulo de avaliação semi-automática uma base de exercícios de programação com suas notas correspondentes. A partir dessa base, o módulo de mapeamento de perfis gera uma matriz $P_{s\times t}$, onde s representa alunos e t é as tarefas. Cada tarefa é representada por um conjunto de atividades de programação utilizadas para avaliar habilidades de programação. Cada elemento d_{ij} da matriz P representa o desempenho do Aluno i na Tarefa j. A Tabela 6.1 representa a matriz $P_{2\times 5}$, que é um exemplo da matriz de desempenhos gerada pelo NAD.

.	t_1	t_2	t_3	t_4	t_5
s_1	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
s_2	d_{21}	d_{22}	d_{23}	d_{24}	d_{25}

Tabela 6.1: Matriz de desempenhos

Nas seções a seguir descrevemos em mais detalhes como funcionam os módulos de avaliação semi-automática de exercícios de programação e o módulo de mapeamento de perfis de alunos.

6.1.1 Avaliação semi-automática de exercícios de programação

O nosso modelo de avaliação semi-automática de exercícios de programação foi construído através do algoritmo de *clustering* chamado *Bissecting k-means* (STEINBACH; KARYPIS; KUMAR, 2000), e de um modelo de regressão linear múltipla (HAIR; TATHAM; BLACK, 1998). O algoritmo de *clustering* analisa os padrões de uma base programas em Linguagem C, separa os padrões que se diferenciam pela construção dos programas e reúne em *clusters* os padrões mais semelhantes entre si. Além disso, o algoritmo de *clustering* aponta as características que mais descrevem e mais discriminam cada *cluster*. Os valores

dessas características são utilizados para escolher os programas de um *cluster* que serão pontuados pelo professor e que servirão de referências para a predição de notas dos demais padrões desse *cluster*. Essas mesmas características são também utilizadas como variáveis independentes do nosso modelo de regressão linear múltipla para predizer as notas dos demais programas de um *cluster*. Uma vez que o modelo de aprendizagem da regressão linear é supervisionado, isto é, guiado por exemplos, ele precisa de exemplos de exercícios já pontuados por um professor. Dessa forma, o conjunto de padrões avaliados pelo professor é chamado de *treino* e o conjunto de padrões preditos a partir desse conjunto de treino é chamado de *teste*.

O Algoritmo 1 é a representação formal do processo de como o nosso modelo de avaliação semi-automática forma *clusters*, retira *outliers* (padrões atípicos), seleciona o conjunto de treino e realiza a predição de notas dos padrões reunidos em *clusters*.

A base de entrada B_e do Algoritmo 1 é uma base de atividades a_i não pontuados formada por programas em Linguagem C. A saída do Algoritmo 1 é a lista de notas L_N da base B_e.

Na Linha 2 do Algoritmo 1, a base B_e é normalizada à base B_n. A normalização da base *Be* foi realizada substituindo, em cada programa C da base B_e, os comentários e as cadeias de caracteres entre aspas por *strings-padrão*. Em seguida, normalizamos os símbolos formados por outros símbolos como, por exemplo, o símbolo $++$ (operador de incremento), que é formado pelo símbolo $+$ (operador de soma). A Tabela 6.2 apresenta exemplos de símbolos, palavras-chave e indicadores de execução normalizados a *strings-padrão*.

Com o objetivo de reduzir confusões no processamento automático, a normalização é realizada substituindo os símbolos da Tabela 6.2 por *strings-padrão*. Da mesma forma, substituimos também *tokens* que contêm outros *tokens*, como, por exemplo, o *printf* (instrução de saída da Linguagem C), que contém *int* (o tipo de dado inteiro da Linguagem C), por *strings-padrão*. Por último, em cada programa normalizado, são inseridos *strings-padrão* indicadoras de funcionamento do programa como, por exemplo, *@compila* e *@funciona*, que informam, respectivamente, se o programa compila e se funciona. A Figura 6.3 é um exemplo de programa em Linguagem C normalizado segundo os símbolos da Tabela 6.2 e os *tokens* de compilação e execução.

Algorithm 1 Algoritmo Predição de Notas

Require: $B_e = \{a_1, a_2, \ldots, a_n\}$ $\forall a_i$: Programa de computador em Linguagem C ($i = 1, \ldots n$)

Ensure: $Tamanho(L_N) = Tamanho(B_e)$ L_N : Lista de notas da Base B_e

1: **Procedure** PredizerNotas(B_e)
2: $B_n \leftarrow Normalizar(B_e)$
3: $M_I \leftarrow Indexar(B_n)$
4: $[L_c, L_f] \leftarrow Clusterizar(M_I)$
5: $M_{sim} \leftarrow MatrizSimilaridade(M_I)$
6: $i \leftarrow 0$
7: **for all** $cluster \in L_c$ **do**
8: $[clusterPuro, outliers] \leftarrow RetirarOutliers(cluster, L_f[i])$
9: $Adicionar(L_r, outliers])$
10: **if** $(Tamanho(clusterPuro) > 3)$ **then**
11: $[L_{tr}, L_{te}] \leftarrow Separar(cluster, L_f[i])$
12: $Notas_{tr} \leftarrow SolicitarNotas(L_{tr})$
13: $ModeloRL \leftarrow CriarModeloRegressao(L_{tr}, Notas_{tr})$
14: $Notas_{te} \leftarrow PredizerNotasRL(ModeloRL, L_{te}, L_f[i])$
15: **else**
16: **if** $(Tamanho(clusterPuro) >= 2)$ **then**
17: $[L_{tr}, L_{te}] \leftarrow EscolherMelhorpadrao(cluster, L_f[i])$
18: $Notas_{tr} \leftarrow SolicitarNotas(L_{tr})$
19: $Notas_{te} \leftarrow PredizerNotasMsim(L_{te}, L_{tr}, L_f[i])$
20: **else**
21: **if** $(Tamanho(clusterPuro) = 1)$ **then**
22: $Adicionar(L_r, cluster)$
23: **end if**
24: **end if**
25: **end if**
26: $i \leftarrow i + 1$
27: **end for**
28: $Notas_r \leftarrow PredizerNotasOutliers(L_r, M_{sim})$
29: $L_N \leftarrow [Notas_{tr}, Notas_{te}, Notas_r]$
30: **EndProcedure**

Símbolos normalizados		
Símbolo	Significado	*string-padrão* de normalização
==	sinal de igualdade	@igual
<=	sinal $\leq$	@menorigual
>=	sinal $\geq$	@maiorigual
!=	sinal $\neq$	@diferente
&&	operador AND	@e
||	operador OR	@ou
++	operador de incremento	@inc
--	operador de decremento	@dec
+=	operador de soma e atribuição	@somaatrib
-=	operador de subtração e atribuição	@menosatrib
*=	operador de produto e atribuição	@multatrib
/=	operador de divisão e atribuição	@divatrib
//	Comentário	@comentlinha
printf	Comando de saída	@imprimir
unsigned	Sem sinal	@unsinal
%i	símbolo para Inteiro	@ri
%d	símbolo para Inteiro	@ri
%c	símbolo para Caracter	@rc
%f	símbolo para Números reais	@rf
%s	símbolo para Strings	@rs
/*	inicio de comentário	@icoment
*/	fim de comentário	@fcoment

Tabela 6.2: Tabela de símbolos normalizados

```
@compila  @funciona

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {


        int a = 0, b = 0, cont = 1;

        @imprimir ( @textostr , &a);
        if (a > b)
        {
                b = a;
        }

        while (cont@menorigual2)
        {
                @imprimir ( @textostr , &a);

                if (a < b)
                {
                        b = a;
                }
        cont@inc;
        }

        @imprimir ( @textostr , b);



        system ( @textostr );
        return 0;
}
```

Figura 6.3: Um programa em Linguagem C normalizado

Depois da normalização da base B_e à base B_n, realizamos a indexação da base B_n gerando matriz indexada M_I (Linha 3 do Algoritmo 1). A indexação consiste em representar vetorialmente os programas da base B_n. A indexação é realizada extraindo-se de cada programa da base B_n as frequências de ocorrência das *strings-padrão* (*tokens* e símbolos normalizados), palavras reservadas e símbolos da Linguagem C (SCHILDT, 1991). Essas frequências representam as dimensões de um vetor. Esse vetor, por sua vez, representa um programa. A Figura 6.4 ilustra a representação vetorial de três programas.

Na Figura 6.4, os vetores $P1$, $P2$ e $P3$ representam três programas de computador. As dimensões $C1$, $C2$ e $C3$ são três características desses programas que podem ser uma *string-padrão*, uma palavra reservada ou um símbolo. A Matriz de Indexação M_I, onde n_l é o número de linhas, n_c é o número de colunas e c_{lc} o valor de cada característica c, pode ser representada da seguinte forma:

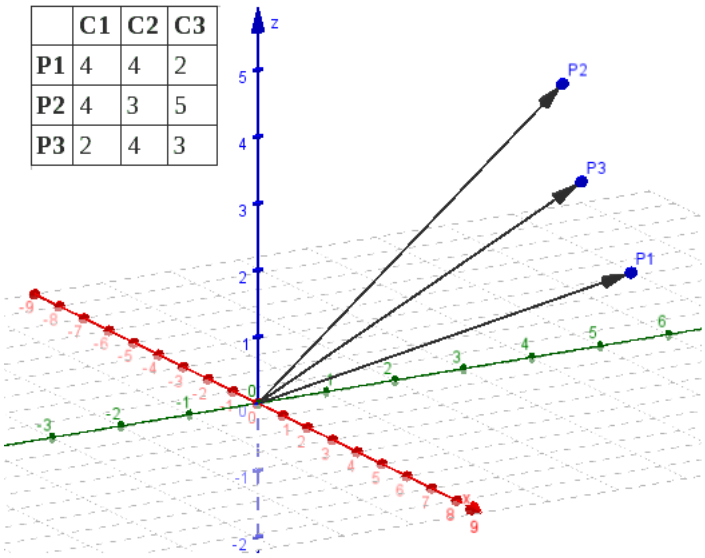

Figura 6.4: Representação vetorial de programas

$$M_I = \begin{bmatrix} c_{11} & c_{12} & c_{13} & & c_{1n_c} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2n_c} \\ c_{31} & c_{32} & c_{33} & \dots & c_{3n_c} \\ c_{41} & c_{42} & c_{43} & \dots & c_{n_l n_c} \end{bmatrix}.$$

Após a indexação, a matriz M_I é submetida ao algoritmo de *clustering* para separar seus padrões em *clusters*. De acordo com a Linha 4 do Algoritmo 1, a clusterização da matriz M_I gera a lista de *clusters* L_c e a lista L_f de características descritivas de cada *cluster*.

Para a execução do algoritmo de *clustering*, foram definidas como entradas fixas o número de *clusters* igual a onze e a medida de similaridade *cosseno*. Escolhemos onze *clusters* para ter uma possível distribuição dos padrões em dez intervalos de notas. No décimo-primeiro *cluster* seriam reunidos os padrões que o algoritmo de *clustering* não conseguiu inserir nos outros dez *clusters*.

Escolhemos a medida de similaridade *cosseno* para agrupar padrões em *clusters* por ser uma medida tradicional de similaridade vetorial (BAEZA-YATES; RIBEIRO-NETO, 1999). No entanto, outras medidas de similaridade poderiam ser utilizadas.

Após o processo de *clustering*, geramos a matriz de similaridade da matriz indexada de programas M_I a partir da base normalizada B_n. Essa matriz apenas contém os índices de

similaridades entre os vetores representantes dos programas da base dois a dois. Esse índice de similaridade é calculado também pela medida de similaridade *cosseno*. O modelo de matriz de similaridade M_{sim} gerado na Linha 5 do Algoritmo 1 é apresentado a seguir.

$$M_{sim} = \begin{bmatrix} I_{11} & I_{12} & I_{13} & & I_{1n} \\ I_{21} & I_{22} & I_{23} & \dots & I_{2n} \\ I_{31} & I_{32} & I_{33} & \dots & I_{3n} \\ I_{n1} & I_{n2} & I_{n3} & \dots & I_{nn} \end{bmatrix}.$$

Na Matriz M_{sim} cada Iij corresponde ao índice de similaridade entre o padrão i e o padrão j, representados por programas de computador normalizados.

Para cada *cluster*, realizamos a retirada dos *outliers*, a escolha dos padrões cujas notas serão atribuídas pelo professor e a predição de notas dos padrões a partir de um modelo de regressão linear múltipla ou de uma matriz de similaridade M_{sim}.

A retirada de *outliers* de cada *cluster* (linhas 8 e 9 do Algoritmo 1) é realizada comparando os padrões de um *cluster* em relação às características mais descritivas em $L_f(i)$, onde i é o número do *cluster*. Aqueles padrões que forem mais divergentes, isto é, que possuírem menores índices de similaridades em relação aos demais padrões do mesmo *cluster*, considerando as características descritivas do *cluster*, são retirados do *cluster* e inseridos na lista de *outliers* L_r. No caso de *clusters* muito heterogêneos, muitas amostras serão adicionadas à Lista L_r.

Uma vez formado o *clusterPuro*, iniciam-se os processos de escolha de padrões para o professor atribuir notas e para o NAD predizer notas.

Se o *cluster* contiver mais de três padrões, escolhem-se padrões representativos do *cluster*, isto é, os exemplos distintos dos padrões que formam um *cluster*, segundo as características mais descritivas de $L_f(i)$. Os padrões representativos selecionados para o professor atribuir notas formam o conjunto de treino, representado pela lista L_{tr} na Linha 11 do Algoritmo 1. Já os padrões a serem preditos automaticamente a partir do conjunto de treino, formam o conjunto de teste, representado pela lista L_{te}.

Uma vez selecionados os padrões de treino, as notas dos padrões de teste são preditas pelo seguinte Modelo de Regressão de Linear Múltipla:

$$Y_i = a_0 + a_1 \cdot X_1 + a_2 \cdot X_1 + \ldots + a_n \cdot X_n, \forall X_k \in L_f(i), 0 \leq n \leq n_t$$

Nesse modelo matemático, as variáveis independentes X_k representam as características de $L_f(i)$ quantificadas pela frequência de ocorrência em programas de computador normalizados da base B_n. O valor de n, que representa a quantidade dessas características, é menor que o número de padrões de treino n_t, de forma que a matriz $n \times n_t$ do modelo não tenha *rank* (posto) deficiente. Os coeficientes $a_k(k = 0, 2, \ldots, n)$ do modelo são estimados de acordo com as contribuições das características mais descritivas nas notas do conjunto de treino. A variável dependente Y_i, por sua vez, representa a nota a ser predita pelo modelo para padrões de um *Cluster i*.

Para predição das notas do conjunto de teste, utilizamos o *Software R* (TEAM, 2008). O algoritmo que desenvolvemos em Linguagem R para predizer as notas do conjunto *teste* de um *cluster* consiste essencialmente dos seguintes passos:

1. Definir a variável dependente Y e as variáveis independentes $X_k \in L_f(i)$.
2. Criar modelo de regressão linear múltipla a partir da variável dependente e das variáveis independentes definidas no Passo 1.
3. Simplificar o modelo criado no Passo 2 pelo algoritmo *stepwise* (ver Capítulo 5).
4. Predizer notas a partir do modelo simplificado gerado no Passo 3.

Nos *clusters* com dois ou três padrões, a predição de notas é realizada pelo índice de similaridade entre os padrões do *cluster*, conforme linhas 16 a 18 do Algoritmo 1. Para isso, selecionamos o melhor padrão do *cluster* para ser referência de predição para os outros padrões do *cluster*. O critério de escolha do padrão-referência é o número de ocorrências das características mais descritivas do *cluster* nesse padrão. A variável dependente $\hat{Y}_t$ é, dessa forma, predita segundo a relação a seguir:

$$\hat{Y}_t = n_{t_1} \cdot sim_{t_1} + \sum_{p=2}^{k} (n_{t_p} \cdot sim_{t_p} \cdot r_{p-1})$$

onde n_{tq} $(q = 1, 2, ..., p)$ é a nota de um padrão de treino e sim_{t_p} é o índice de similaridade entre o padrão de treino t_p e um padrão de teste. O valor de k é o número de padrões de treino e r_{p-1} é a diferença entre sim_{t_p} e $sim_{t_{p-1}}$, isto é, o quanto da variável dependente não foi explicado pela similaridade entre o padrão de teste e o padrão de treino anterior $(p-1)$.

Como exemplo de predição de nota, suponha ser $\hat{Y}_t$ a nota a ser predita para um padrão de teste a partir de dois exemplos de treino, t_1 e t_2. Sendo as similaridades entre o padrão de teste e os padrões de treino 0.8 e 0.6, respectivamente, a nota seria predita por:

$$\hat{Y}_t = (0.8) \cdot n_{t_1} + (0.6) \cdot (0.2) \cdot n_{t_2}$$

Escolhemos de dois ou três padrões aquele que possuisse menor soma dos valores das características mais descritivas. Isso porque entendemos que desenvolveu a melhor solução aquele aluno que escreveu o programa com menos operações e com menos recursos da linguagem de programação utilizada. O padrão escolhido como referência é do conjunto de *treino*. A partir dele, predizemos os demais padrões pelo índice de similaridade com o exemplo escolhido do *cluster*. Por exemplo, se a um padrão *treino* é atribuída nota 5.0, um padrão *teste* semelhante a ele em 94% obterá nota 4.7.

Nos *clusters* com apenas um padrão, adicionamos esse padrão à lista de *outliers* L_r, conforme Linha 22 do Algoritmo 1.

Após a predição de notas em cada *cluster*, é realizada, conforme a Linha 28 do Algoritmo 1, a predição de notas da lista de *outliers* L_r. Para predizer a nota de um padrão considerado *outlier*, através da matriz de similaridade M_{sim}, obtivemos os padrões mais próximos dos padrões que desejamos predizer através da seguinte equação:

$$Y = np_1 \cdot s_1 + np_2 \cdot s_2 \cdot r_1 + \ldots + np_n \cdot s_n \cdot r_{n-1}$$

Nessa equação, np_i é a nota do padrão-vizinho, s_i é o índice de similaridade entre o padrão a ser predito e o vizinho mais próximo. O valor de r_j é calculado por

$$r_j = 1 - s_i, \text{ onde } j = i - 1$$

Nesse caso, se os vizinhos mais próximos tivessem, por exemplo, índices de similaridade 0.71, 0.50 e 0.2 com o *outlier* e notas iguais a 3.5, 5.0 e 2.0, a nota Y_i seria calculada da seguinte forma:

$$Y_i = (3.5) \cdot (0.71) + (5.0) \cdot (0.5) \cdot (0.29) + (2.0) \cdot (0.2) \cdot (0.5) \rightarrow Y_i = 3.41$$

Os experimentos e resultados de experimentação do Algoritmo 1 em uma base real de exercícios de programação em Linguagem C são apresentados no Capítulo 8.

6.1.2 Mapeamento de perfis de alunos

O mapeamento de cada perfil de aluno é realizado a partir dos desempenhos dos alunos nas componentes de habilidades que representam o domínio de aprendizagem da programação. Essas componentes representam desempenhos em tarefas ou em atividades. Neste trabalho, as tarefas podem ser provas, listas de exercícios ou atividades avaliativas. Cada atividade de programação é um programa de computador desenvolvido em Linguagem C por um aluno.

Cada representação em componentes de habilidades é definida como um estado de perfil de aluno. Esse estado de perfil é armazenado em um repositório que contém estados de perfis de outros estudantes.

O mapeamento de perfis baseado em tarefas consiste em obter os desempenhos dos alunos em cada tarefa para diagnóstico dos progressos dos alunos ao longo de um curso.

O mapeamento de perfis baseado em atividades é aquele que representa o estado de aprendizagem dos alunos por um conjunto de componentes de habilidades. Essas componentes são variáveis $x_i(i = 1,2,3,\ldots k)$ que representam desempenhos no uso de estruturas de programação, palavras reservadas, símbolos e indicadores de execução, como a compilação e a execução. Esse modelo de representação de perfis é utilizado pelo módulo de avaliação semi-automática de exercícios de programação do NAD e pelo módulo de recomendação de atividades do NAF.

Para construir os gráficos de mapeamento de perfis utilizamos um algoritmo de *clustering* de abordagem hierárquica e com medida de similaridade de coeficiente de correlação.

Através dos mapas de perfis, o professor pode identificar classes de alunos que apresentam as mesmas características em relação às componentes de habilidades e reconhecer dificuldades e habilidades de aprendizagem entre os alunos de uma turma. São exemplos desses mapas as Figuras 6.5 e 6.6.

Além do mapeamento de perfis de alunos, aplicando a mesma estratégia de *clustering*, podemos realizar diagnósticos de questões, de tarefas e de provas para avaliar o nível de dificuldade das questões bem como apontar os alunos hábeis para resolvê-las.

Os gráficos de *recall* (R) e *precision* (P) (MANNING; RAGHAVAN; SCHUTZE, 2008) da Figura 6.5 são exemplos de mapeamento de perfis baseado em tarefas. O *recall* é uma medida que expressa a razão entre o número de acertos n_a e o total de exercícios n_t de uma tarefa. Já a medida *precision* informa a razão entre o número de acertos n_a e número n_r de exercícios resolvidos por um aluno conforme as equações a seguir:

$$R = \frac{n_a}{n_t}, P = \frac{n_a}{n_r}, \text{ para } (n_r \leq n_t)$$

Nos gráficos da Figura 6.5, as linhas representam os alunos e as colunas, cada tarefa aplicada ao longo de um curso de programação. As áreas em vermelho indicam desempenhos acima de 70%. Já as áreas verdes indicam ausência de desempenhos ou desempenhos inferiores a 70%.

Através dos gráficos da Figura 6.5, reconhecem-se diferentes classes de alunos, entre as quais citamos: as classes dos melhores alunos, alunos que se fazem as tarefas esporadicamente e os alunos com dificuldades em fazer provas.

Nos gráficos da Figura 6.5, os alunos que apresentam os melhores desempenhos nas tarefas são identificados pela letra A. Esses alunos fazem todas as tarefas com desempenhos acima de 70% e reproduzem os bons resultados nas provas práticas, que são as últimas colunas das faixas em vermelho, isto é, as colunas em vermelho antes do início da faixa verde (tarefas não avaliadas) e depois do seu final.

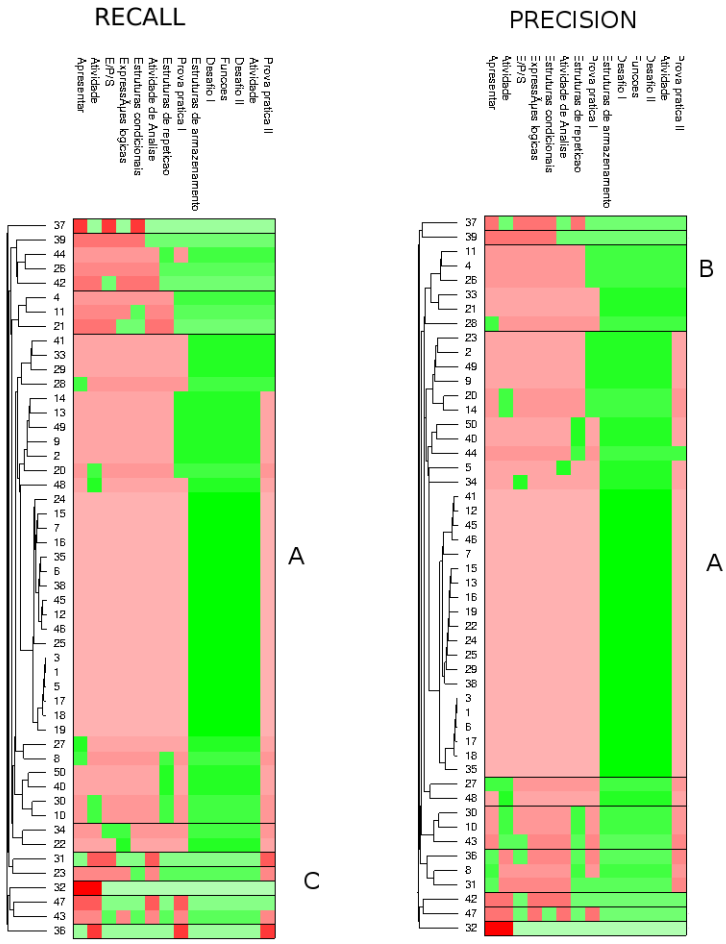

Figura 6.5: Perfil de aluno baseado em tarefas

No gráfico de *recall* da Figura 6.5, nas linhas da parte inferior estão os alunos identificados por C, que são aqueles alunos que fazem as tarefas esporadicamente. No gráfico de *precision*, nas linhas superiores, estão os alunos identificados pela letra B, que são os alunos que fazem bem todas as tarefas, mas não alcançam desempenhos satisfatórios nas provas práticas.

A Figura 6.6 é um mapa de diagnóstico dos alunos de uma turma em uma atividade de programação em Linguagem C. Nesse gráfico, as linhas representam os alunos de uma turma identificados por um número. As colunas são as componentes de habilidades representadas por indicadores de execução (*compile, run*), palavras reservadas da Linguagem C (*include, main, int, float, scanf, return, float, return, if, for, while, switch*) e operadores de atribuição (=), aritméticos (+, -, *,/, %), de comparação (==, >,<,>=,<=) e lógicos (&&, ||, !). As áreas

em vermelho apontam as dificuldades de aprendizagem nas componentes de habilidades.

Para indicar as dificuldades de aprendizagem nas áreas vermelhas do gráfico da Figura 6.6, foi estabelecida uma solução como gabarito. Cada componente de habilidade foi medida calculando-se a razão entre a sua frequência de ocorrência em soluções desenvolvidas por alunos e a sua frequência de ocorrência na solução-gabarito. Se essa razão resulta em um valor abaixo de 0.7, isto é de 70%, a componente de habilidade é sinalizada de vermelho indicando dificuldade de aprendizagem. Da mesma forma, se a medida da componente de habilidade exceder a 1, isto é, a 100%, a componente também é sinalizada de vermelho. Isso porque entende-se que o aluno usou instruções de programação acima do esperado em relação ao gabarito, o que caracteriza ineficiência no uso dos recursos da programação.

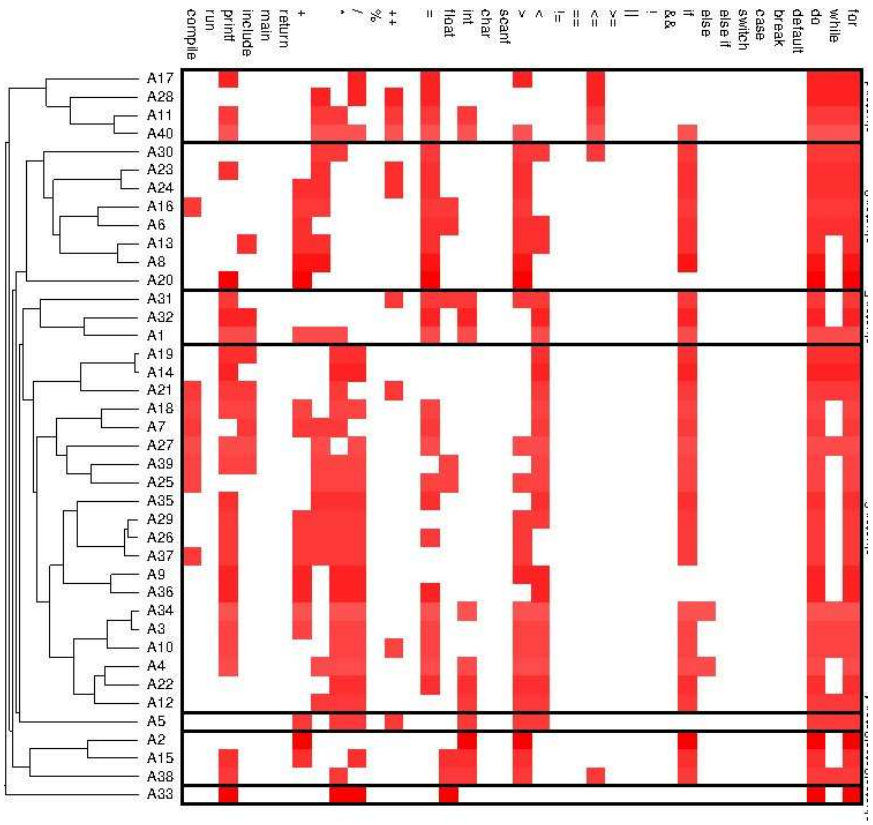

Figura 6.6: Perfil de aluno baseado em atividades

A representação de perfis da Figura 6.6 é utilizada no módulo de recomendação de atividades do NAF. Já o módulo de avaliação semi-automática de exercício do NAD recebe os perfis de alunos representados apenas pela frequência de ocorrência das palavras-chave, símbolos, operadores e outros *tokens* da Linguagem C.

6.2 Núcleo de Avaliação Formativa (NAF)

O Núcleo de Avaliação Formativa é formado por dois módulos: o Módulo de Controle de Estabilidade (MCE) e o Módulo de Recomendação de Atividades (MRA). O primeiro módulo tem a função de regular a recomendação de atividades realizada pelo segundo módulo determinando a continuação ou a interrupção do processo de recomendar atividades.

Para cada estado de perfil de aluno analisado, o MCE verifica se as variáveis avaliadas nesse perfil têm valores entre 0.7 e 1. Se alguma variável avaliada não tiver a sua medida entre esses valores, classes de atividades relacionadas a essa variável deverão ser recomendadas. Quando o aluno resolver as atividades recomendadas, as variáveis avaliadas são recalculadas e novamente analisadas. Se todas as variáveis avaliadas atingirem desempenhos satisfatórios, a recomendação de atividades é interrompida. Caso contrário, reinicia-se a recomendação de classes de atividades em concordância com as variáveis que estão com desempenhos insuficientes.

As variáveis de avaliação são, dessa forma, realimentadas continuamente através da recomendação de classes de atividades para alcançarem um estado de progresso. No entanto, se após um número *n* especificado de recomendações, as variáveis não indicarem progressos, encerra-se o processo de recomendação. Em seguida, uma mensagem é enviada para o professor pelo MCE para ele decidir como ajudar o aluno com dificuldades a melhorar os seus desempenhos na prática da programação.

No Módulo de Recomendação de Atividades do NAF, a recomendação é realizada seguindo a abordagem de recomendação por filtragem colaborativa porque considera a similaridade entre estados de perfis de alunos para recomendar-lhes atividades.

Considerando as similaridades entre perfis e tratando a recomendação de itens como um tarefa de classificação *multilabel*, podemos contemplar situações mais complexas em relação às recomendações. Um exemplo disso seria recomendar vários tipos de atividades para estudantes cujos perfis apontam para diferentes tipos de dificuldades de aprendizagem. Nesse caso, embora não haja qualquer relação entre os itens recomendados e as variáveis que caracterizam um perfil, o sistema de recomendação poderá aprender de outros exemplos

similares como as classes de atividades são associadas com os estados de perfis dos alunos.

A Figura 6.7 é uma ilustração no esquema *entrada-processamento-saída* do nosso modelo de recomendação de classes de atividades por classificação *multilabel*.

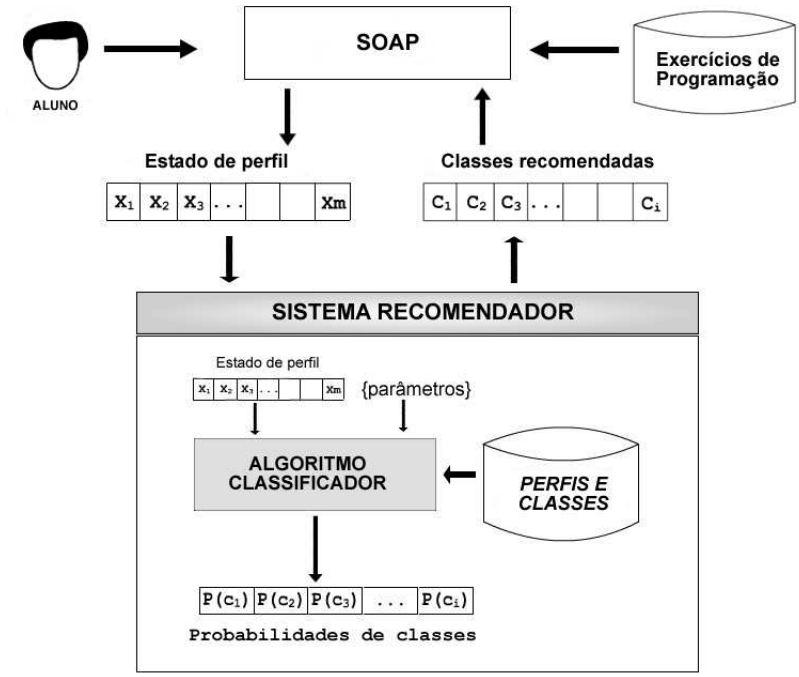

Figura 6.7: Modelo de recomendação de classes de atividades por classificação *MultiLabel*

No modelo de recomendação de classes de atividades na Figura 6.7, através do SOAP, os estudantes submetem soluções para os *exercícios de programação* disponibilizados por um professor.

De acordo com a Figura 6.7, o NAD mapeia os desempenhos de um aluno em uma atividade de programação em um perfil multidimensional ou *Estado de perfil*, onde cada dimensão representa uma componente de habilidade. Em nosso *sistema de recomendação*, o *Estado de perfil* e alguns *parâmetros* de entrada são enviados para o *algoritmo classificador*, que é o Algoritmo ML-kNN (ZHANG; ZHOU, 2007).

A base de *Perfis e classes* da Figura 6.7 é formada a partir de exemplos de outros perfis previamente associados a classes de atividades. A partir dessa base, calcula-se a probabilidade $P(c_i)$ de cada classe ser associada a um perfil recebido. Através da análise das *Probabilidades de classes*, o sistema recomendador seleciona as classes mais prováveis

(com probabilidades maiores ou iguais ao parâmetro de corte $\tau = 0.5$) de serem associadas ao perfil analisado e apresenta como saída um lista de *Classes recomendadas* para esse perfil.

6.2.1 Formalização do problema

A recomendação de atividades pode ser definida como uma tarefa de classificação *multilabel* em que cada perfil de aluno é associado a uma ou mais classes de atividades. Nesse caso, um perfil representa as performances de um estudante em diferentes variáveis de avaliação em um tempo específico para uma atividade de programação. Uma classe de atividades, por sua vez, representa um conjunto de atividades que possuem as mesmas características em relação a um ou mais conteúdos de uma linguagem de programação estudada durante um curso.

Formalizando o problema, seja D o domínio de perfis, $C = \{c_1, c_2, \ldots, c_{|C|}\}$ um conjunto de classes pré-definidas para os perfis e $\Omega = \{d_1, d_2, \ldots, d_{|\Omega|}\}$ um *corpus* inicial de perfis que são previamente classificados manualmente por um especialista. Um perfil $d_j = (x_1, x_2, x_3, ..., x_{|d_j|})$(para $j = 1, 2, 3, \ldots |D|$) é composto por um conjunto $|d_j|$ com variáveis de avaliação $x_q \geq 0$ (para $q = 1, 2, 3, \ldots, |d_j|$). Cada perfil d_j de Ω é previamente associado a um subconjunto de classes de atividades $c_i \in C$ (para $i = 1, 2, \ldots, |C|$).

Um sistema de recomendação de atividades, visto como um sistema de classificação *multilabel*, implementa a função $f : D \times C \rightarrow R$ que retorna um valor para cada par $(d_j, c_i) \in D \times C$, que é a evidência para o fato de que perfil de teste d_j deveria ser classificado na classe $c_i \in A_p$, onde $\bigcup_{p=1}^{m} A_p \subseteq C$, sendo m o número máximo de classes associadas a um perfil d_j. A função do atual valor de $f(.,.)$ pode ser transformada em uma função *ranking* $r(.,.)$, que é o mapeamento um-para-um em $1, 2, \ldots, |C|$ tal que se $f(d_j, c_1) > f(d_j, c_2)$, então $r(d_j, c_1) < r(d_j, c_2)$. Se A_p é o conjunto de classes apropriadas para o perfil de teste d_j, então um bom sistema de classificação tende a organizar as classes em A_p primeiro que as classes que não estão em A_p. Além disso, nós também usamos um parâmetro de corte para as classes que são organizadas acima do limite (ou *threshold*) τ(i.e., $c_k | f(d_j, c_k) \geq \tau$) para torná-las as únicas classes associadas a um perfil de teste.

Na Figura 6.8, por exemplo, de acordo com a descrição da formalização do problema,

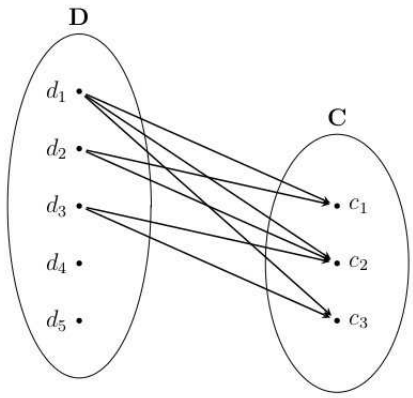

Figura 6.8: Formalização do problema

sendo $\Omega = \{d_1, d_2, d_3\} \subseteq D$, a tarefa é associar uma ou mais classes de C aos perfis d_4 e d_5 de D. Para isso, nós usamos padrões que foram identificados nos documentos d_1, d_2 e d_3 e previamente associados por um especialista humano ao conjunto de classes de C. Nesse caso, $|D| = 5$, $|\Omega| = 3$ e $|C| = 3$.

Neste trabalho, a formalização do problema foi aplicada como segue: D é o domínio de perfis obtidos a partir de atividades resolvidas por alunos. As classes de atividades $c_i \in C$ são representadas por tipos de conteúdos de uma linguagem de programação. Em um perfil d_j, cada valor x_q é o desempenho de um aluno em um item cognitivo q em uma atividade de programação. O item q pode ser um indicador de compreensão de conteúdos ou do uso apropriado de um operador aritmético, lógico ou relacional. Nós escolhemos o valor de 0.5 para o parâmetro de corte τ. Como um resultado, somente classes que obtiverem um valor de probabilidade maior ou igual a τ serão recomendadas para um perfil de teste.

6.3 Conclusão

Neste capítulo apresentamos o modelo de avaliação semi-automática da aprendizagem de programação proposto nesta tese. Esse modelo é constituído pelos núcleos NAD e NAF. O NAD contém os módulos de avaliação semi-automática de exercícios e de mapeamento de perfis e o NAF, os módulos de controle de estabilidade e de recomendação de atividades. Alunos e professores interagem com esses núcleos através da interface *web* do SOAP.

No módulo de avaliação semi-automática do NAD, os programas em Linguagem C submetidos por alunos como soluções de atividades de programação, são normalizados, indexados e clusterizados. Em seguida, através de uma estratégia combinando as técnicas de *clustering* e de regressão linear, selecionamos um conjunto de amostras mais representativas para um professor atribuir notas e, a partir delas, criamos um modelo para predizer as notas dos demais exercícios.

Para representar estados de aprendizagem, mapeamos perfis de alunos por tarefas ou por atividades. Nos perfis mapeados por tarefas, as componentes de habilidades são representadas por desempenhos em tarefas ao longo de um curso de programação. Nesse mapeamento por tarefas, obtêm-se informações sobre classes de perfis de alunos e conteúdos onde alunos mais apresentam dificuldades de aprendizagem.

No mapeamento de perfis por atividades, as componentes de habilidades são mapeadas em desempenhos representados pela frequência de ocorrência de palavras reservadas, símbolos, operadores e indicadores de execução da linguagem de programação C. Esses perfis são utilizados para avaliar como alunos utilizam os recursos da linguagem e quais dificuldades estão apresentando.

Os perfis mapeados por frequência de ocorrência de palavras-chave, símbolos e operadores da Linguagem C são utilizados pelo NAD para avaliação semi-automática de exercícios. No NAF, a frequência de ocorrência de termos é dividida pela frequência de ocorrência desses mesmos termos em uma solução-gabarito. Desse modelo, cada componente de habilidade de perfil no NAF é representada por um valor de 0 a 1. O valor 1 representa o melhor desempenho. Os valores acima de 0.7 indicam sucesso de aprendizagem. O valor zero indica ausência de desempenhos e os valores abaixo de 0.7 ou acima de 1 são indicadores de dificuldades de aprendizagem.

Uma vez reconhecendo as dificuldades de aprendizagem dos alunos a partir de seus perfis, o módulo de recomendação do NAF, recomenda classes de atividades conforme as recomendações realizadas para outros perfis semelhantes por um especialista humano.

As inovações do modelo de avaliação semi-automática de programação são as seguintes:

1. Combinamos técnicas de reconhecimento de padrões de abordagem supervisionada como a regressão linear e não-supervisionada como o *clustering* para predizer notas de exercícios conforme padrões de correção de professores.
2. Formulamos o problema de recomendação de atividades em um problema de classificação *multilabel*.
3. Criamos uma representação de perfis por tarefas por medidas de *recall* e *precision* nas atividades de cada tarefa para identificar classes de alunos e suas dificuldades.
4. Criamos uma representação de perfis por atividades que caracteriza a aprendizagem de programação a partir da frequência de uso dos recursos da Linguagem C.

O modelo de avaliação semi-automática da aprendizagem de programação tem como objetivos monitorar e controlar a prática da programação por avaliações diagnóstica e formativa, que são modelos de avaliação difíceis de serem praticados em turmas numerosas e que requerem extensa prática de exercícios.

As contribuições deste modelo para o domínio de aprendizagem da programação são reduzir esforços de professor na correção de exercícios e no acompanhamento individual de seus alunos e oferecer-lhes *feedbacks* mais rápidos, possibilitando-lhes uma verdadeira prática assistida da programação.

Capítulo 7

O Sistema SOAP

O SOAP é um sistema de internet de apoio à prática assistida de atividades de programação. Esse sistema foi desenvolvido com o propósito de ser para o aluno um instrumento de submissão de exercícios, de trabalhos e provas de programação e, para o professor, uma ferramenta de auxílio à avaliação e ao acompanhamento da aprendizagem dos alunos.

Para atender a esse propósito, o SOAP se apresenta em duas visões: a do aluno e a do professor. A visão do aluno possui as funcionalidades de submissão de programas em Linguagem C e visualização de desempenhos. A visão do professor possui funcionalidades para criar turmas, especificar tarefas, dar *feedbacks*, atribuir notas às atividades submetidas, gerar relatórios de desempenhos e mapear perfis de alunos.

A proposta do SOAP é fundamentada em princípios de avaliação diagnóstica e formativa (PERRENOUD, 1999) para treinamento *online* de programação. A princípio, o SOAP está sendo utilizado para submissão e avaliação de exercícios em Linguagem C, mas oferece possibilidades de ser estendido para outras linguagens de programação.

O SOAP interage com os núcleos NAD e NAF apresentados no Capítulo 6 gerando para eles as bases de dados necessárias para avaliação, mapeamento e gestão da aprendizagem dos alunos.

O SOAP é apresentado em detalhes nas próximas seções conforme a ordem a seguir. Na

Seção 7.1, explicamos o sistema SOAP mostrando as funcionalidades da visão do aluno e da visão do professor. Na Seção 7.2, descrevemos uma estratégia de controle do Núcleo Executor para execução eficiente e segura dos programas submetidos por alunos ao SOAP. Na Seção 7.3, concluimos com as considerações finais e as implementações futuras do SOAP.

7.1 O SOAP

O SOAP está sendo utilizado desde o segundo semestre do ano de 2011 por turmas de programação introdutória da Universidade Federal do Espírito Santo e está localizado no endereço: 200.137.66.62/soap.

O sistema SOAP foi construído através das linguagens de programação *PHP* e *Shell Script*. Para armazenamento dos dados, utilizamos o Sistema Gerenciador de Banco de Dados (SGBD) *MySql*.

Para acessar o SOAP é necessário passar por um processo de autenticação de usuários. Para realizar essa autenticação, na tela inicial do SOAP (Figura 7.1) deverão ser fornecidos *login* e senha de usuário. O usuário poderá ser do tipo aluno ou professor.

Figura 7.1: Autenticação de usuários

Caso um usuário não esteja cadastrado no SOAP, ele poderá fazer um cadastro para ter acesso às funcionalidades da visão do aluno ou da visão do professor no SOAP. Na tela de

cadastro, devem ser informados nome completo, localidade, *e-mail*, formação, *login* e senha de usuário. Ao ser pressionado o botão **Enviar**, os dados serão validados e, se estiverem corretos, serão armazenados no banco de dados do SOAP. Se estiveram incorretos, uma mensagem será exibida e será solicitado que o usuário preencha o cadastro mais uma vez. A tela de cadastro pode ser visualizada na Figura 7.2.

Figura 7.2: Cadastro de usuários

Após ser cadastrado, o usuário aluno ou professor poderá entrar no SOAP informando seu *login* e senha na tela inicial (Figura 7.1). Se o usuário for aluno, será apresentado a ele a visão do aluno, isto é, a interface com as funcionalidades de usuário aluno para matricular em turma, visualizar tarefas, submeter exercícios e visualizar desempenhos. Se for professor, é apresentada a interface de usuário para professor com as funcionalidades de criar turma, especificar tarefa, avaliar exercícios e emitir relatórios de desempenhos.

7.1.1 A visão do aluno

Ao acessar o SOAP após processo de autenticação, é apresentada ao aluno a tela com os *links* Alterar cadastro, Minhas turmas matriculadas e Fazer Matrícula em uma Turma conforme a tela da Figura 7.3. Acessando o *link* Alterar cadastro, o aluno poderá editar as

informações de seu cadastro e regravá-las. Através do Fazer Matrícula em uma Turma, o aluno acessa a tela da Figura 7.4. Nessa tela, ele deverá selecionar qual turma deseja se matricular. Caso o prazo de matrícula ainda esteja em vigor e o limite de alunos da turma escolhida não tenha excedido, o aluno será matriculado com êxito.

Figura 7.3: Tela inicial do usuário aluno

O SOAP dá ao aluno a possibilidade de se matricular em outras turmas, desde que sejam satisfeitas as condições de prazo de matrícula e de limite de alunos dessas turmas.

Uma vez que o aluno já esteja em uma ou mais turmas, ele poderá acessar as tarefas especificadas para as suas turmas através do *link* Minhas turmas matriculadas. Acessando esse *link*, será apresentada a tela de seleção de turmas. Ao selecionar uma das turmas em que está matriculado, o aluno tem acesso às tarefas especificadas pelo professor da turma selecionada, conforme a tela da Figura 7.5.

Ao acessar o *link* de uma tarefa especificada, o aluno é direcionado para a tela da Figura 7.6 que contém a descrição da tarefa acessada, o *link* Fale Conosco e o *link* Meus desempenhos. Nessa tela, o aluno poderá apertar o botão **Iniciar** para começar a fazer as atividades ou o botão **Sair** para abandonar a tarefa.

Caso tenha pressionado o botão **Iniciar** da tela de descrição da tarefa (Figura 7.6), o

Figura 7.4: Fazer matrícula em uma turma

aluno é conduzido para a tela de realização de atividades apresentada na Figura 7.7.

As atividades de programação disponibilizadas por um professor para uma tarefa são apresentadas aos alunos no esquema de sorteio. Quando o número de atividades é pequeno, todas as atividades especificadas pelo professor aparecem na forma de *links* no lado direito da tela de atividades. Dessa forma, o aluno pode acessar as atividades de acordo com o sorteio ou diretamente pelos *links* disponibilizados no lado direito da tela de atividades (Figura 7.7).

No lado esquerdo da tela de atividades, conforme a Figura 7.7, aparecem o número da tarefa, o número de submissão da questão sorteada, um *link* de acesso à especificação da questão e um *link* para o modelo de submissão da questão.

De acordo com o modelo de submissão de questão do SOAP, para cada questão, deve ser submetido o arquivo de *makefile*, isto é, o arquivo em que o aluno especifica como o seu programa será executado, e todos os arquivos necessários para a execução correta do programa submetido. Essencialmente o arquivo *makefile* segue o formato da Figura 7.8.

No *makefile* da Figura 7.8, a cláusula *all* executa os comandos das cláusulas *compila* e *executa*. A cláusula *compila* contém o comando de compilação do programa especificado nesse comando que, no modelo da Figura 7.8, é um programa *C*. A cláusula *executa* contém o comando de execução do programa executável gerado após execução da cláusula *compila*.

Figura 7.5: Tarefas especificadas

Na execução dessa cláusula a leitura de entrada de dados é redirecionada para o arquivo *arquivoentrada* e a saída, para o arquivo *arquivosaida*. Já a cláusula *clear* contém instruções de remoção dos arquivos gerados nos processos de compilação e execução das cláusulas *compila* e *executa*.

Supondo que um aluno escreveu um programa C e deseja submetê-lo como resposta a uma questão disponibilizada por um professor em uma tarefa, o aluno deverá submeter ao SOAP o *makefile* (Figura 7.8), o programa *.c* e o *arquivoentrada*, caso o programa *.c* receba entradas. Para um programa chamado *soma.c*, por exemplo, que calcula a soma de dois números, teríamos o *makefile*, o programa *soma.c* e o *arquivoentrada*, conforme apresentados na Figura 7.9.

Na Tela de Atividades (Figura 7.7), para submeter os arquivos que serão processados pelo SOAP, o aluno deverá pressionar o botão **Procurar** para localizar cada arquivo a ser submetido. Se um arquivo for selecionado por engano, o aluno deverá pressionar o botão **Delete**, que se apresenta ao lado do arquivo selecionado, para apagar da lista esse arquivo. Uma vez selecionados todos os arquivos para submissão, o aluno deverá pressionar o botão **Enviar** para concluir o processo de submissão da atividade. Concluído esse processo, o SOAP emitirá um relatório de confirmação de submissão, conforme a Figura 7.10.

O aluno poderá conferir as submissões realizadas através dos *links* disponibilizados em

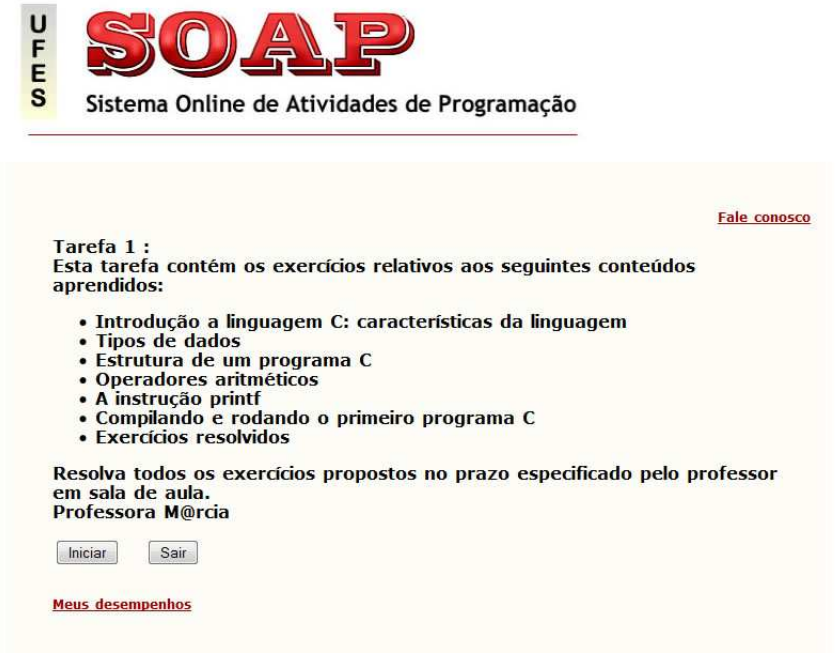

Figura 7.6: Especificação da tarefa

Atividades submetidas, no lado direito da tela de atividades (Figura 7.7). Acessando um desses *links*, o aluno visualiza as informações originais da questão submetida, isto é, o número da tarefa, o número da submissão e a especificação, e as informações de submissão, acessíveis através do *link* Relatório de submissão. Acessando esse *link*, o aluno tem a visualização dos arquivos submetidos, conforme apresentado na Figura 7.11. Caso perceba que houve algum erro na submissão de arquivos em uma questão, o aluno poderá realizar uma nova submissão dos arquivos dessa questão. Vale ressaltar, porém, que o SOAP faz o registro do número de submissões realizadas por um aluno para uma mesma questão. Essa informação é apresentada pelo SOAP ao professor no processo de avaliação de atividades.

Na tela de apresentação de tarefas (Figura 7.6), acessando o link Meus desempenhos, o aluno poderá visualizar um relatório de desempenhos (Figura 7.12) com as seguintes informações para cada questão submetida: código da questão com *link* para a sua especificação, situação (se fez a questão ou não), arquivos submetidos, saída gerada, nota e *feedbacks* do professor.

As dúvidas dos alunos poderão ser comunicadas ao professor ou ao suporte técnico do SOAP através do *link* Fale Conosco disponível na tela de apresentação de tarefas (Figura 7.6) e na tela de atividades 7.7. Ao acessar o Fale conosco, o aluno deverá informar nome, e-mail

Figura 7.7: Tela de atividades

```
all: compila executa

compila:
	gcc teste.c -o executavel > compilacao.txt

executa:
	./executavel < arquivoentrada  > arquivosaida

clear:
	rm executavel arquivosaida
```

Figura 7.8: Arquivo *makefile*

e escrever uma mensagem, conforme a janela mostrada na Figura 7.13. Ao ser pressionado o botão **Enviar**, a mensagem do aluno será encaminhada para o e-mail de suporte do SOAP.

Todas as atividades submetidas por alunos às tarefas especificadas são executadas nos servidores do SOAP e apresentadas em relatórios na interface de avaliação dos professores, conforme explicamos na subseção a seguir.

7.1.2 A visão do professor

Quando um usuário professor entra no sistema SOAP, após processo de autenticação, ele é direcionado para a tela que contém os links de acesso às principais funcionalidades para usuários professores: criar turma, cancelar turma, alterar turma, especificar tarefa, verificar

Arquivo *soma.c*

```c
#include <stdio.h>

// Este programa calcula a soma de dois números

int main()
{
        int num1=0,num2=0,soma=0;

        printf("Exercício de teste!! \n");

        scanf("%d %d", &num1,&num2);

        soma = num1 + num2;

        printf("A Soma de %d com %d é %d \n", num1,num2,soma);

        return 0;

}
```

Arquivo *makefile*

```
all: compila executa

compila:
        gcc teste.c -o executavel > compilacao.txt

executa:
        ./executavel < arquivoentrada  >  arquivosaida

clear:
        rm executavel arquivosaida
```

Arquivo *arquivoentrada*

```
20 31

```

Figura 7.9: Exemplo de submissão: programa soma.c, *makefile* e arquivoentrada

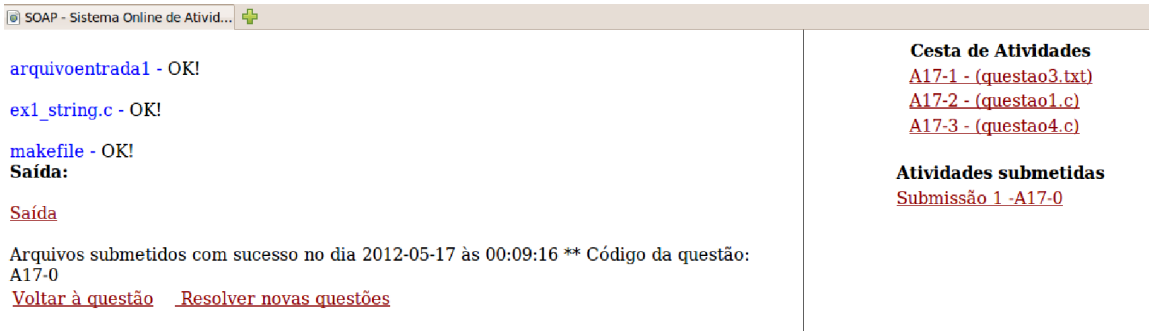

Figura 7.10: Relatório de confirmação de submissão

questões resolvidas e emitir relatórios de desempenhos. A tela com essas funcionalidades é mostrada na Figura 7.14.

Ao acessar Criar turma, o professor entrará na tela mostrada na Figura 7.15. Para efetuar a criação de uma turma, o professor deverá fornecer as seguintes informações: nome da turma, número de vagas, localidade, datas de início e término da matrícula e datas de início e término da turma que está sendo criada. Ao apertar o botão **Enviar**, as informações são validadas e, caso sejam válidas, a turma é criada e associada ao professor que realizou esse processo de criação de turma.

Para cancelar uma turma, o professor deverá acessar Cancelar turma e, em seguida, selecionar a turma que deseja cancelar. Para alterar as informações geradas na criação de uma turma, o professor deverá acessar o *link* Alterar turma na tela da Figura 7.14.

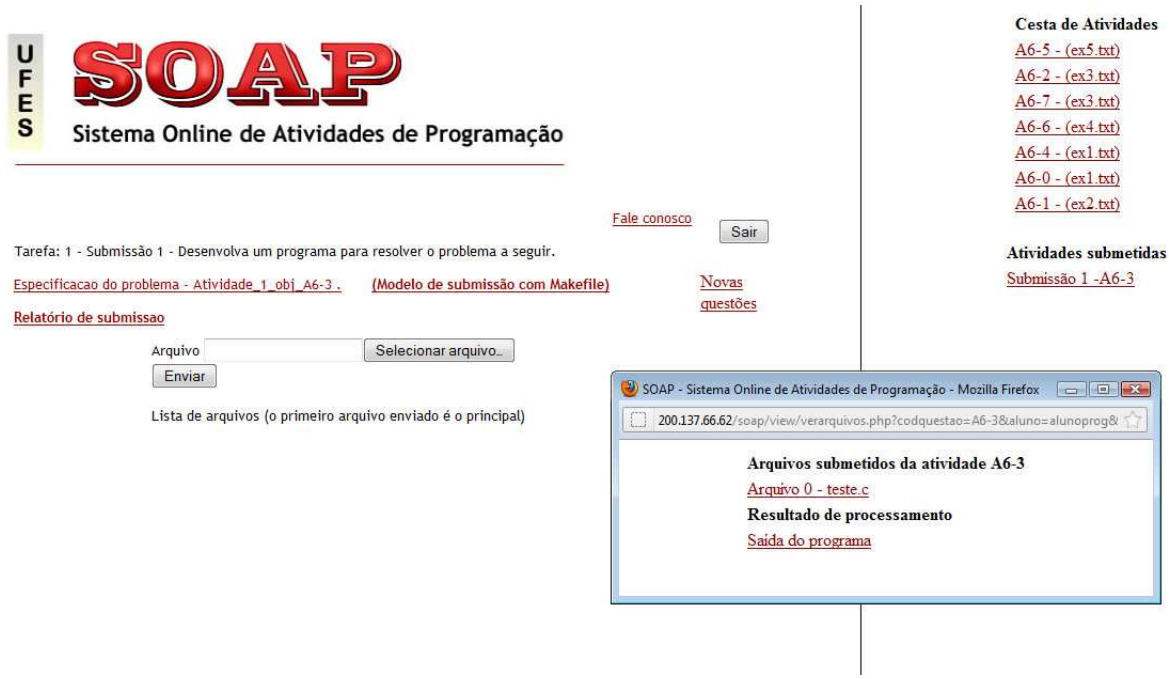

Figura 7.11: Relatório de submissão

Após criar uma turma, o professor poderá especificar tarefas através do *link* Especificar tarefa. Acessando esse *link*, o professor é direcionado para a tela da Figura 7.16. Para especificar uma tarefa, o professor deverá selecionar a turma a qual estará associada a tarefa, o título da tarefa e um texto de especificação da tarefa. Ao pressionar o botão **Enviar**, as informações da tarefa serão armazenadas no banco de dados do SOAP e apresentadas aos alunos na tela de descrição de tarefa da visão do aluno (Figura 7.5).

Ao especificar um tarefa, o professor deve selecionar de um diretório do computador onde está especificando a tarefa todos os arquivos que serão enunciados de questões. Vale ressaltar que o SOAP associa a cada enunciado um arquivo. Dessa forma, se um professor dispor mais de um arquivo para um enunciado, ele deve colocar esses arquivos em um pacote (.zip, .rar, etc.). Dois ou mais arquivos de enunciados podem ser selecionados ao mesmo tempo e enviados para o SOAP compor a lista de exercícios de uma tarefa.

Para ter uma visualização das atividades submetidas por seus alunos, um professor deverá acessar Verificar questões resolvidas, selecionar a turma e, em seguida, acessar a tarefa da qual deseja visualizar as atividades submetidas. Após selecionar a tarefa, será apresentada ao professor a tela da Figura 7.17.

Figura 7.12: Tela de apresentação de desempenhos

Figura 7.13: Fale conosco

Na tela da Figura 7.17, no lado esquerdo, estão os *links* das atividades das tarefas selecionadas para visualização bem como os *links* dos enunciados dessas atividades. Ao acessar o *link* de uma atividade, são apresentadas, no lado direito da tela da Figura 7.17, todas as submissões de alunos para essa atividade. Para cada submissão apresentada, são fornecidas as seguintes informações: data da submissão, número de tentativas, arquivos submetidos, se compilou, a saída gerada pelo SOAP após executar o programa submetido, a nota mais alta atribuída pelo professor e o número de *feedbacks*.

De posse das informações necessárias para avaliar as atividades dos alunos submetidas ao SOAP, na tela da Figura 7.17, o professor poderá, para cada atividade, atribuir nota e escrever *feedbacks* para os alunos quantas vezes achar necessário através do *link* Novo. Acessando esse *link*, será apresentada a janela da Figura 7.18 ao professor. Nessa janela, o professor,

Figura 7.14: Tela inicial do usuário professor

opcionalmente, informa a nota e escreve mensagens de *feedbacks*. Ao ser pressionado o botão **Enviar dados**, as informações são gravadas e disponibilizadas para visualização do aluno e do professor.

Para visualizar os *feedbacks*, o professor deverá acessar o *link* da quantidade de *feedbacks* na tela da Figura 7.17. Acessando esse *link*, são apresentados os *feedbacks* na ordem do mais recente para o mais antigo bem como as notas atribuídas. No entanto, para os cálculos de desempenhos realizados pelo SOAP e para a visualização de desempenhos, será apresentada para o professor (Figura 7.17) e para o aluno (Figura 7.12) a maior nota atribuída pelo professor à atividade submetida por esse aluno.

O *link* Recarregar, mostrado na tela da Figura 7.17, é utilizado para atualizar a página em que o professor está atribuindo notas e fornecendo *feedbacks* de forma que ele sempre visualize as modificações realizadas nesse processo de avaliação.

Para visualizar os relatórios de desempenhos dos alunos, o professor deverá acessar Relatório de desempenhos na tela da Figura 7.14, selecionar turma e acessar o *link* da tarefa da qual deseja visualizar os desempenhos dos alunos. Realizado esse processo, o professor será direcionado para a tela mostrada na Figura 7.19. Nessa tela são apresentados, para cada aluno, respectivamente, o percentual de exercícios submetidos, o percentual de exercícios

Figura 7.15: Criar turma

funcionais, isto é, que compilam e executam, e os desempenhos médios do aluno nas atividades que o professor atribuiu notas. Na tela da Figura 7.19, esses desempenhos são apresentados de forma a informar a média do aluno e o número de exercícios corrigidos do total de exercícios disponibilizados na tarefa analisada pelo professor.

O *link* Relatório comparativo da turma (.csv) na tela da Figura 7.19, disponibiliza para o professor um relatório detalhado em arquivo *.csv* de todas as atividades de uma tarefa submetidas por todos os alunos. As linhas são representadas pelos alunos e as colunas, por cada atividade da tarefa em análise. O valor 1 indica que o aluno fez a atividade e que seu programa submetido compilou e executou. Já o valor 0 informa que o aluno não fez uma atividade ou o programa submetido não funcionou. Um exemplo do arquivo *.csv* disponibilizado é mostrado na Figura 7.20.

Para abandonar a tela de apresentação da visão do professor (Figura 7.19), o professor deverá pressionar o botão **Sair**. Feito isso, o SOAP retorna à sua tela inicial (Figura 7.1).

Figura 7.16: Especificar tarefa

7.2 Avaliação dinâmica por rede de máquinas virtuais

No processo de avaliação dinâmica do SOAP, os programas submetidos por alunos são compilados e executados no Núcleo Executor e, em seguida, os relatórios de compilação e a saída gerada são disponibilizados para o professor. Esse processo é realizado a cada uma hora nos servidores do SOAP para todas as tarefas especificadas por professores. Cada submissão de exercício a essas tarefas, porém, é corrigida apenas uma vez. Isso porque o SOAP gera uma confirmação sempre que todo o processo de execução de um programa submetido é finalizado. Dessa forma, ao reiniciar o processo de correção, um programa já executado no SOAP não será executado novamente, salvo se houver uma nova submissão desse programa.

Uma vez que várias tarefas são especificadas por professores e, por conseguinte, vários exercícios são submetidos ao SOAP, demanda-se muito esforço computacional para o envio de *feedbacks* rápidos do processo de correção para os alunos. Além disso, há nesse processo os riscos de execução de códigos maliciosos que podem causar panes ou paralisações no servidor central do SOAP.

Atividades

1a. Prova
A76-1 questao2.zip
A76-0 questao1.zip

UFES SOAP
Sistema Online de Atividades de Programação

Avaliação de Atividades - Questão A76-0 Recarregar

Alunos	Data	Tentativas	Arquivos	Compilação	Saída	Notas/ *Feedbacks*
Aluno 1	21/03/2013	1	3	OK!	Saída	4.5 (1) Novo
Aluno 2	18/04/2013	1	3	OK!	Saída	0 (0) Novo
Aluno 3	21/03/2013	1	3	OK!	Saída	4.75 (1) Novo
Aluno 4	21/03/2013	1	3	OK!	Saída	4.75 (1) Novo
Aluno 5	21/03/2013	1	3	OK!	Saída	4 (1) Novo
Aluno 6	00/00/0000	0	0	NÃO!	***	0 (0) Novo
Aluno 7	21/03/2013	2	3	OK!	Saída	4 (1) Novo
Aluno 8	00/00/0000	0	0	NÃO!	***	0 (0) Novo
Aluno 9	21/03/2013	1	3	OK!	Saída	4 (1) Novo
Aluno 10	21/03/2013	1	3	OK!	Saída	3.75 (1) Novo

Figura 7.17: Visualizar questões resolvidas

Para atender às demandas de *feedbacks* rápidos para os alunos e de execução segura dos programas submetidos, desenvolvemos uma estratégia de utilizar máquinas virtuais paralelas para distribuir o processamento de correção de tarefas e isolar esse processo dos demais processos que rodam em uma máquina física. Dessa forma, como as máquinas virtuais funcionam em máquinas físicas como se fossem outras máquinas, executando programas de risco nas máquinas virtuais, protegemos as máquinas físicas dos problemas ocasionados pela execução desses códigos de risco.

Através do *Software Netkit* [1], construimos uma rede de máquinas virtuais que se iniciam em máquinas físicas de uma rede local quando é iniciado o processo de correção de exercícios. A tarefa de cada máquina virtual consiste em rodar os exercícios submetidos a uma tarefa e devolver os resultados de execução ao servidor central do SOAP.

O *Netkit* é um emulador de redes de computadores que representa, por *software*, dispositivos de *hardwares* necessários para o funcionamento de uma rede tais como roteadores, servidores, *switches*, e da criação dos enlaces. Além do *hardware*, esses equipamentos virtuais são inicializados com *softwares* reais.

[1] Software e documentação disponíveis em: http://www.netkit.org/labs.html

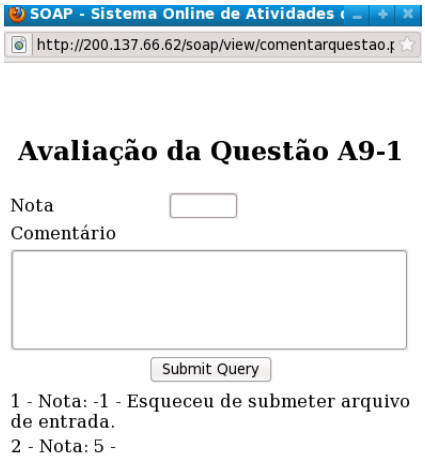

Figura 7.18: Janela de *feedbacks*

O *Netkit* utiliza *softwares* de código aberto para montar uma rede a partir da geração de máquinas virtuais. Cada uma dessas máquinas iniciadas pelo *Netkit* é um computador completo rodando uma distribuição monousuário da distribuição *Debian GNU/Linux*. A estrutura de arquivos de configurações e diretórios dessas máquinas virtuais formam um laboratório virtual.

As principais vantagens oferecidas pelo *Netkit* na construção de uma rede de máquinas virtuais são as seguintes:

- Realização de experimentos de rede a baixo custo e com pouco esforço, pois, através do *Netkit*, é possível montar uma rede inteira em uma única máquina física.
- Distribuição de processamento em várias máquinas virtuais remotas possibilitando a execução cooperativa de um trabalho.
- Proteção de máquinas físicas contra processamento de risco, uma vez que os processos da máquina virtual são executados isolados dos processos de uma máquina física.

  Aplicando a estratégia da rede de máquinas virtuais com o *software netkit*, o processo de avaliação dinâmica de exercícios do SOAP é realizado através dos seguintes passos:

  1. Iniciar processo de correção de tarefas
  2. Percorrer rede local para identificar máquinas ativas da rede que tenham o *software Netkit* instalado.

Avaliação dos Exercícios

Tarefa Capítulo 2

Relatório comparativo da turma - Tarefa Capítulo 2 (.CSV)

Alunos	Exercícios submetidos (%)	Exercícios funcionais (%)
Aluno 1	0	0
Aluno 2	0	0
Aluno 3	0	0
Aluno 4	90	90
Aluno 5	10	10
Aluno 6	100	100
Aluno 7	80	80
Aluno 8	100	100
Aluno 9	90	90
Aluno 10	100	100

Figura 7.19: Relatório de desempenhos

3. Nas máquinas ativas com o *Netkit* instalado, realizar o teste de funcionamento do *netkit* através de um *script* do próprio *Netkit*.
4. Para cada máquina em que o *Netkit* funciona normalmente, distribuir tarefas segundo o critério a seguir:

$$n = \frac{q_t}{q_m}, q_m > 0 \tag{7.1}$$

onde n é o número de tarefas a ser realizado por cada máquina, q_t é o número de tarefas especificadas por professores no SOAP e q_m é o número de máquinas com *Netkit* funcionando corretamente.

5. Iniciar máquina virtual em cada máquina que tenha o software *Netkit* funcionando corretamente. O *script* que inicia uma máquina virtual em uma máquina física da rede local é um arquivo *nomedamaquinavirtual.startup* que contém as instruções de definição de uma tarefa para uma máquina virtual escritas em *shell script*.
6. Após execução da tarefa definida no arquivo *nomemaquinavirtual.startup*, finalizar

	A	B	C	D	E	F	G
1		cap2ex3.jpg	aula3ex7.txt	cap2ex1.jpg	aula3ex3.pdf	cap2ex4.jpg	aula3ex5.txt
2	Aluno 1	1	0	0	0	0	1
3	Aluno 2	0	0	0	0	0	0
4	Aluno 3	0	0	0	0	0	0
5	Aluno 4	0	0	0	0	0	0
6	Aluno 5	0	0	0	0	0	0
7	Aluno 6	0	0	0	0	0	0
8	Aluno 7	0	0	0	0	0	0
9	Aluno 8	1	1	1	1	1	1
10	Aluno 9	1	1	1	1	0	1
11	Aluno 10	0	0	0	0	0	0
12	Aluno 11	1	1	1	1	0	1

Figura 7.20: Arquivo de desempenhos

a máquina virtual. As atividades a serem realizadas no processo de finalização devem ser especificadas no arquivo *nomemaquinavirtual.shutdown*

Os arquivos *nomemaquinavirtual.startup* e *nomemaquinavirtual.shutdown* da Figura 7.21 fazem parte de uma estrutura de representação de uma rede de máquinas virtuais. Além desses arquivos, existe em um laboratório virtual os arquivos *nomeLab.conf*, que contém a topologia da rede, e o arquivo *nomeLab.dep*, para executar arquivos *startups* paralelos. Um exemplo de laboratório virtual com esses arquivos é mostrado na Figura 7.21.

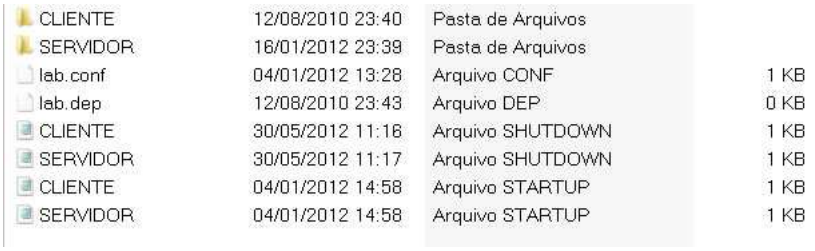

Figura 7.21: Exemplo de laboratório virtual

O laboratório virtual apresentado na Figura 7.21 é representado por uma rede de duas máquinas virtuais: a máquina cliente e a máquina servidora. Cada máquina adicionada ao laboratório virtual deverá conter, assim como as máquinas CLIENTE e SERVIDOR (Figura 7.21), a pasta de arquivos, arquivos *startups* e arquivos *shutdowns*.

Para a avaliação dinâmica dos exercícios submetidos às tarefas especificadas por professores, cada máquina virtual de uma rede recebe os exercícios de uma ou mais dessas tarefas e, para cada uma delas, executa o *script* definido pelo seguinte algoritmo:

1. Acessar diretório da tarefa

2. Para cada subdiretório da tarefa, que representa uma submissão de questão realizada por um aluno, executar, se não existir o arquivo *sucesso.txt* (que indica que os arquivos da submissão já foram rodados no Núcleo Executor), as instruções de compilação e execução contidas no arquivo *makefile* submetido como parte da questão.
3. Se a saída for gerada corretamente no Passo 2, criar arquivo *sucesso.txt* no subdiretório em que a questão foi corrigida, como indicador de que essa questão foi corrigida.

Através dessa estratégia de avaliação dinâmica por máquinas virtuais paralelas, retiramos do servidor central do SOAP o esforço computacional de rodar sequencialmente uma grande quantidade de programas submetidos por alunos e oferecemos uma estrutura de execução mais rápida e segura para esses programas.

7.3 Conclusão

O SOAP foi aplicado experimentalmente em algumas turmas de programação introdutória da Universidade Federal do Espírito Santo desde o segundo semestre do ano de 2011 para submissão de exercícios, trabalhos e provas. Nessa aplicação experimental do SOAP, o sistema é atualizado de acordo com os problemas reconhecidos no uso do sistema e com as demandas apontadas por alunos e professores.

Avaliando os problemas de uso do sistema e as demandas apresentadas por professores e alunos, identificamos as seguintes necessidades de atualizações no SOAP:

1. Desenvolver outros mecanismos de proteção contra a submissão de programas que causam riscos ao funcionamento normal dos servidores em que o SOAP está instalado.
2. Isolar e paralelizar o processo de execução dos programas submetidos para que os alunos recebam de forma mais rápida a resposta do sistema.
3. Desenvolver mecanismos para levantar indícios de plágio
4. Desenvolver uma estratégia de controle das versões de programas submetidos por um aluno em duas ou mais tentativas de submissão de uma questão.

5. Integrar o SOAP a ambientes de aprendizagem como o *Moodle*, por exemplo.

Para atender o Item 1, atualmente o SOAP oferece um mecanismo de proteção contra programas em *loop* para evitar sobrecarga do processador e escassez de memória, principalmente na geração de arquivos de saída. Nesse mecanismo desenvolvido, a princípio para cursos de programação introdutória, o SOAP avalia se um programa está em *loop* verificando se a execução desse programa ocorre em um tempo máximo determinado e se o arquivo de saída gerado excede o tamanho de um *megabyte*.

Entendemos, no entanto, que, mesmo em cursos de programação introdutória, há programas aplicados por professores que requerem tempo maior de execução e, em alguns casos, arquivos de saída acima de um *megabyte*. Por isso, há necessidade de uma interação entre o professor e o sistema ou uma *percepção* do sistema para se definir o tempo máximo de execução de um programa e a quantidade de memória requerida para a geração dos arquivos de saída dessa execução.

Para atender o Item 2, implementamos a estratégia de correção dinâmica de exercícios de programação proposta na Seção 7.2.

A necessidade de levantamento de indícios de plágio do Item 3 poderá ser atendida gerando uma matriz de similaridade de programas (OLIVEIRA; OLIVEIRA, 2008b) a partir dos códigos normalizados e indexados pelo NAD.

O número de tentativas de submissão de uma questão pode ser um indicador de dificuldades no desenvolvimento de um programa como solução de um problema. Dessa forma, para atender à necessidade apontada no Item 4, propomos que as versões submetidas de um programa por um aluno sejam comparadas uma contra a outra por medidas de similaridade. Isso seria realizado para avaliar como o aluno evoluiu na construção da solução e em que pontos as suas dificuldades persistem.

A integração do SOAP a um ambiente de aprendizagem como o *Moodle* indicada no Item 5 agregaria ao SOAP, como um repositório de dados, mais informações sobre a aprendizagem de programação dos alunos e, como um sistema de controle de aprendizagem, mais variáveis de avaliação que possibilitariam um mapeamento mais rico do processo de aprendizagem dos alunos. Isso porque, mais do que o produto de um processo de aprendizagem representado pela submissão de uma questão, obter-se-ia

o *tracing* do aluno no processo de construção de programas de computador.

Além das atualizações necessárias apresentadas, faz-se necessário realizar alguns melhoramentos de nível estético e funcional nas interfaces do SOAP de forma que professores e alunos tenham maior facilidade e rapidez na interação com o sistema.

Capítulo 8

Experimentos e Resultados do NAD

Um dos módulos do NAD é utilizado para avaliação semi-automática de exercícios de programação levando em consideração padrões de correção de professores. Para mostrar a eficácia de predição de notas deste módulo, mostramos neste capítulo os experimentos realizados para avaliação semi-automática de exercícios de programação em Linguagem C bem como os resultados alcançados.

Os experimentos foram realizados de forma progressiva. No primeiro experimento, realizamos a predição de notas aplicando apenas a técnica de regressão linear. Nesse experimento inicial, para cada base de exercícios de programação em Linguagem C, selecionamos dois terços da base como referência para a predição de notas das demais amostras dessa base.

No segundo experimento, por *clustering*, identificamos e removemos os *outliers*, escolhemos dentro de cada *cluster* dois terços das amostras e reunimos essas amostras em um conjunto de treino para a predição de notas de outras amostras presentes em um conjunto de teste.

No terceiro experimento, selecionamos como treino dois terços das amostras reunidas no maior *cluster* como treino e predizemos as notas das demais amostras de teste desse mesmo *cluster*.

No quarto e quinto experimentos, aplicamos o método desta tese utilizando o *clustering* não apenas para reunir as amostras em *clusters* e para retirar os *outliers*, mas também para selecionar as características que melhor descrevessem cada *cluster*. Em cada um

dos *clusters* formados, selecionamos as amostras mais representativas como treino e as características descritivas identificadas pelo algoritmo de *clustering* para gerar o modelo de regressão linear. O modelo gerado para cada *cluster* foi então aplicado para predição de notas das amostras de teste em cada *cluster* .

Os resultados dos experimentos realizados mostram que a utilização do *clustering* para retirada de *outliers* e para seleção de amostras e de características em um modelo de regressão linear melhoram os resultados de predição de notas em exercícios de programação. Dessa forma, podemos criar modelos de regressão linear para predição de notas com erro médio 6% a 13% da nota máxima, e com redução de até 70% do esforço manual de correção de um professor.

Este capítulo está organizado conforme a ordem a seguir. Na Seção 8.1, explicamos os métodos e materiais utilizados nos experimentos. Na Seção 8.2, mostramos a aplicação gradativa do nosso método, os procedimentos realizados e os resultados alcançados. Na Seção 8.3, apresentamos a aplicação do método proposto neste trabalho. Na Seção 8.4, discutimos os resultados e as inovações do nosso método. Na Seção 8.5, destacamos algumas sugestões de melhoramentos para dar continuidade às pesquisas. Na Seção 8.6, apresentamos algumas extensões do método de avaliação semi-automática de exercícios de programação. Na Seção 8.7, concluimos apresentando as contribuições e as considerações finais.

8.1 Métodos e materiais

Para a realização dos experimentos, primeiramente, realizamos o pré-processamento das soluções de questões desenvolvidas por alunos através dos procedimentos de normalização e de indexação de base descritos no Capítulo 6.

O procedimento de normalização da base foi realizado com os propósitos de retirar informações irrelevantes como comentários e *strings* bem como evitar ambiguidades no processamento automático de *tokens* e símbolos. Nesse procedimento de normalização também inserimos nos arquivos normalizados informações relevantes de execução do programa como, por exemplo, se um programa compila e se executa. A Tabela 8.1 contém um exemplo de um programa em Linguagem C e a sua versão normalizada.

Programa C

```
#include<stdio.h>
main()
{

int GP,GN,VF,VC,E,P,valc=0,valv=0,
    timec,timev,ntime=0;
do
{
   printf("No. gols positivos e negativos:\n");
   scanf("%d%d",&GP,&GN);
   printf("Vitorias dentro e fora de casa: \n");
   scanf("%d%d",&VF,&VC);
   printf("Numero de empates \n");
   scanf("%d",&E);
   P=5*GP - GN + 3*VF + 2*VC + E;

   ntime++;
   if(P>valc || P> valv)
    {
      if(P>valc && P>valv)
      {
          timev= timec;
          valv= valc;
          timec= ntime;
          valc= P;
       }
        if(P> valv && P< valc)
        {
          timev= ntime;
          valv= P;
        }
    }
}while(ntime<3);
printf("Campeao: %d com %d pontos \n",
timec,valc);
printf("Vice: %d com %d pontos \n",timev,valv);
}
```

Programa C normalizado

```
@compila  @funciona
#include<stdio.h>
main()
{
        int GP,GN,VF,VC,E,P,valc=0,valv=0,
        timec,timev,ntime=0;
        do
        {
        @imprimir( @textostr ,&GP,&GN);
        @imprimir( @textostr ,&VF,&VC);
        @imprimir( @textostr ,&E);
        P=5*GP - GN + 3*VF + 2*VC + E;

        ntime@inc;
        if(P>valc @ou P> valv)
        {
                if(P>valc @e P>valv)
                {
                timev= timec;
                valv= valc;
                timec= ntime;
                valc= P;
                }
                if(P> valv @e P< valc)
                {
                timev= ntime;
                valv= P;
                }
        }
        }while(ntime<3);
        @imprimir( @textostr ,timev,valv);
}
```

Tabela 8.1: Normalização de programas

Depois de normalizados, os programas da base foram indexados e reunidos em uma matriz de indexação, conforme o exemplo da Tabela 8.2. A primeira linha dessa matriz contém o número de linhas e o número de colunas. O número de linhas representa o número de amostras de exercícios de uma base e o número de colunas, o número de características (variáveis de avaliação ou componentes de habilidades).

As demais linhas da matriz de indexação representam as soluções dos alunos na forma vetorial, onde cada dimensão ou coluna é o valor numérico de uma característica representante de um programa.

Para representar as dimensões da matriz de indexação e as variáveis do modelo de regressão linear, escolhemos 60 características representativas do domínio de

100	60														
1	1	3	0	0	0	0	0	9	2	0	0	0	0	0	2
1	1	0	0	0	0	0	0	9	0	0	0	0	0	0	2
0	0	0	0	1	0	0	0	9	0	0	0	0	0	0	2
1	1	5	4	2	0	1	0	7	0	0	0	0	0	0	2
1	1	1	3	0	0	0	0	2	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	4	4	0	0	0	0	0	2
1	1	0	0	0	0	0	0	6	5	0	0	0	0	0	2

Tabela 8.2: Programa indexado

programação em Linguagem C apresentadas na Tabela 8.3. As características escolhidas foram uma variável que indica se o programa compila, outra que informa se o programa funciona, as palavras reservadas da linguagem C, o operador de atribuição, os operadores aritméticos e os operadores semânticos. Consideramos operadores semânticos os operadores utilizados em expressões lógicas e para atualização de variáveis de controle. Esses operadores são os operadores lógicos, os operadores relacionais e os operadores de incremento($++$) e decremento($--$) (SCHILDT, 1991).

Escolhemos os indicadores de execução, palavras reservadas, os operadores aritméticos e os operadores semânticos como variáveis de avaliação de exercícios de programação por entender que um programa de computador é uma combinação de linguagem, ações e lógica. Para representar essa combinação, quantificamos essas características pela frequência de ocorrência dos *tokens* e dos símbolos que as representam.

Vale ressaltar que o número de características representativas de uma base de exercícios pode ser reduzido conforme sua frequência de uso nos exercícios de uma base ou para melhorar a criação de modelos de predição de notas.

Para a predição das notas, combinamos as técnicas de *clustering* e de regressão linear múltipla, ambas descritas no Capítulo 5. Através dos agrupamentos formados pelo algoritmo de *clustering*, identificamos as possíveis classes que representam a variabilidade de soluções nos programas escritos pelos alunos. A Figura 8.1 é um gráfico gerado pelo algoritmo de *clustering* que mostra como os programas dos alunos foram agrupados em *clusters* pelos índices de similaridades entre eles.

Na Figura 8.1, as linhas representam vetorialmente os programas dos alunos e as colunas, as características escolhidas para descreverem esses programas. Nesse exemplo, os *clusters* foram numerados de 0 a 10. A cor branca em uma característica

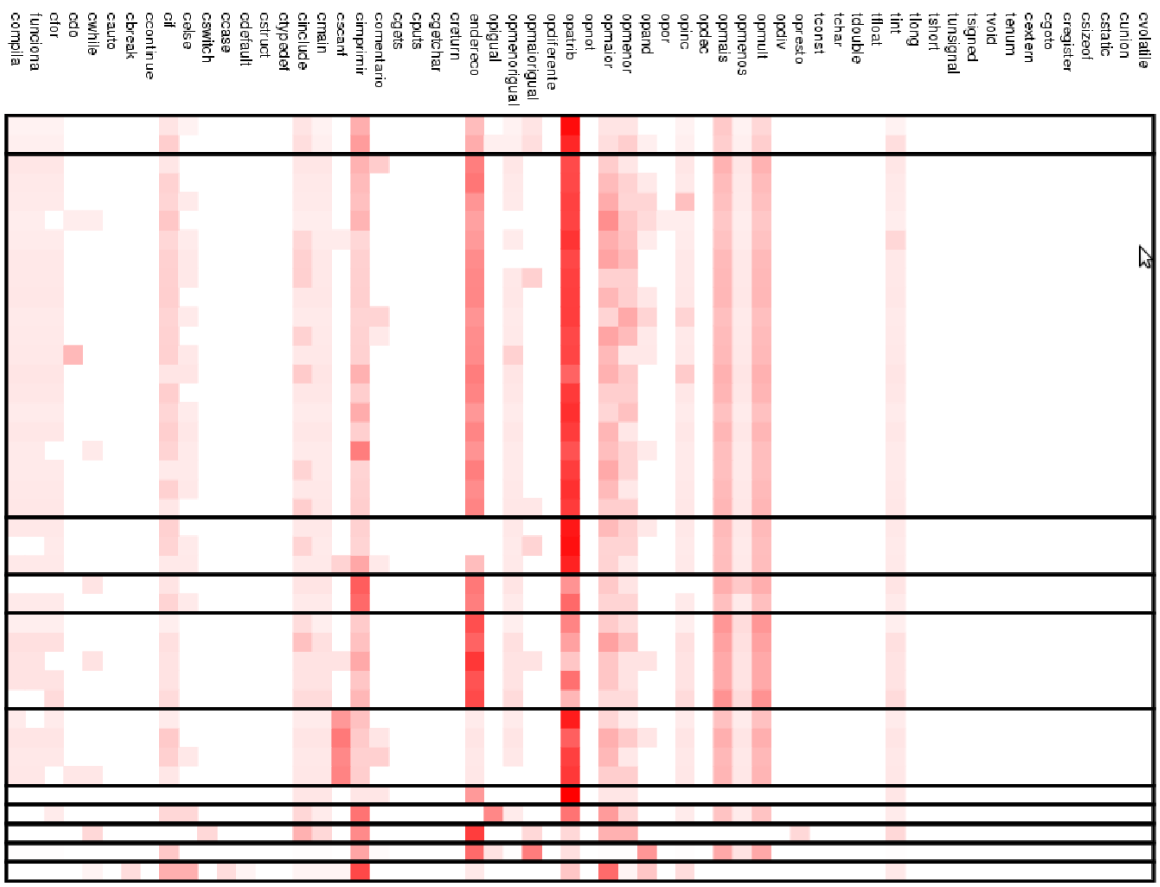

Figura 8.1: Agrupamento de programas

indica valor nulo. A tonalidade da cor indica quão relevante é uma característica no programa desenvolvido por um aluno. Dessa forma, quanto mais forte for a tonalidade de uma característica, mais relevante é essa característica para um programa. Da mesma forma, tonalidades mais fracas indicam pouca relevância das características. Essa relevância é medida, em geral, pelos valores numéricos dessas características em ordem crescente. Dessa forma, quanto maior for o valor numérico da característica, isto é, a dimensão do vetor representado por um programa, mais forte será a tonalidade de cor dessa característica nesse vetor.

Para gerar o gráfico da Figura 8.1, fornecemos como principais entradas do algoritmo de *clustering* o número de *clusters*, a medida de similaridade, os rótulos de cada característica e o método de agrupamento. Escolhemos a medida de similaridade *cosseno* e o método de partições por bissecções através do algoritmo *Bissecting K-means* (STEINBACH; KARYPIS; KUMAR, 2000; KARYPIS, 2003). Como as notas da questão utilizada nos experimentos variam de 0 a 5, escolhemos dividir a

base de soluções em onze *clusters* visando ter as notas distribuídas em dez classes representando possíveis intervalos de notas e reunir no décimo primeiro *cluster* os padrões *outliers*, isto é, os padrões que o algoritmo de *clustering* não conseguisse inserir em qualquer um dos dez *clusters*.

Uma vez descobertos os *clusters* (ou agrupamentos) de soluções semelhantes, selecionamos primeiro os maiores *clusters* para o método de predição de notas descrito no Capítulo 6. Para cada *cluster*, estimamos as notas com base nas notas atribuídas pelo professor nos exemplos indicados como representativos e nas características mais relevantes apontadas pelo algoritmo de *clustering* para esse *cluster*.

Para a predição de notas por regressão linear múltipla nos experimentos realizados, desenvolvemos um algoritmo em *Linguagem R* (TORGO, 2009), que executa os seguintes passos:

1. Obter os conjuntos de treino e de teste
2. Obter lista de variáveis independentes
3. Definir variável dependente a ser estimada
4. Criar modelo de regressão linear que relaciona a variável dependente às variáveis independentes conforme o conjunto de treino
5. Simplificar o modelo selecionando um número menor de variáveis independentes através do algoritmo *stepwise* (HAIR; TATHAM; BLACK, 1998)
6. Avaliar o modelo
7. Realizar predição de notas do conjunto de teste aplicando o modelo de regressão simplificado gerado no Passo 5
8. Limitar o intervalo de notas preditas para o intervalo definido pelo professor
9. Analisar os erros de predição

O modelo inicial de regressão linear que criamos tem como variável dependente a nota e como variáveis independentes a frequência de ocorrência das 60 características representantes de um programa C. Vale ressaltar que dessas 60 características, os algoritmos de *clustering* selecionam apenas as mais descritivas de cada cluster para compor um modelo de regressão linear de acordo com o número de amostras de cada *cluster*. Já o tamanho da amostra para cada *cluster* foi fixado em dois terços do tamanho

do *cluster*. Esse valor, porém, poderá ser reduzido selecionando-se apenas as amostras mais representativas de cada *cluster* como conjunto de treino para geração do modelo de regressão.

A Tabela 8.3 apresenta a variável dependente Y e as variáveis independentes $X_i(i = 1,2,...,60)$ do modelo de regressão linear criado.

Modelo de Regressão Linear

Y	X_i
NOTA	compila, funciona, cfor, cdo, cwhile, cauto, cbreak, ccontinue, cif, celse, cswitch, ccase, cdefault, cstruct, ctypedef, cinclude, cmain, cscanf, cimprimir, comentario, cgets, cputs, cgetchar, creturn, endereco, opigual, opmenorigual, opmaiorigual, opdiferente, opatrib, opnot, opmaior, opmenor, opand, opor, opinc, opdec, opmais, opmenos, opmult, opdiv, opresto, tconst, tchar, tdouble, tfloat, tint, tlong, tshort, tunsignal, tsigned, tvoid, tenum, cextern, cgoto, cregister, csizeof, cstatic, cunion, cvolatile

Tabela 8.3: Variáveis dependente e independentes

As etapas de criação, análise, predição e validação do modelo de regressão linear múltipla gerado em cada experimento são explicadas em detalhes no Capítulo 5.

8.1.1 As bases experimentais

Para a realização dos experimentos deste trabalho, utilizamos duas bases com amostras de soluções desenvolvidas por alunos como respostas a uma questão de programação. Na primeira base, chamada de *Base A*, com quarenta amostras, essa questão foi uma das questões de uma prova de programação introdutória. Essa prova foi aplicada em uma turma de Ciência da Computação da Universidade Federal do Espírito Santo. Escolhemos essa base de exercícios por conter amostras representativas da variabilidade de soluções e os principais construtos de um programa em Linguagem C. Entre esses construtos citamos as instruções de entrada e saída de dados, as operações lógicas e aritméticas e as estruturas de controle condicional e de repetição. Além disso, essas amostras foram obtidas em condições controladas, uma vez que os alunos resolveram essa questão de prova em um laboratório isolado, com limitação de tempo e com a inibição de qualquer tipo de consulta, seja a materiais, a pessoas ou à internet.

A segunda base experimental, chamada de *Base B*, reúne 100 amostras de soluções para

a questão utilizada na Base A. Porém, na Base B, a questão foi aplicada em provas, listas de exercícios e atividades de laboratório em diferentes turmas.

A especificação da questão utilizada nas Bases *A* e *B* consiste em desenvolver uma solução para obter o número de pontos *P* de três times em um campeonato de futebol, de acordo com a expressão matemática a seguir:

$$P = 5GP - GN + 3VF + 2VC + E$$

Nessa fórmula GP é o número de gols positivos, GN é o número de gols negativos, VF é o número de vitórias fora de casa, VC é o número de vitórias em casa e E é o número de empates. No final, o programa desenvolvido pelo aluno deveria mostrar, de acordo com o número de pontos obtidos por um time, o campeão e o vice-campeão do campeonato de futebol.

As amostras selecionadas das Bases *A* e *B* foram pontuadas pelo avaliador humano em uma escala de notas de 0 a 5, pois esta foi a faixa de valores utilizada pelo professor que avaliou a base de exercícios. Ressaltamos que o intervalo de notas deve ser fornecido para o algoritmo avaliador. Na Base *A*, sendo a questão aplicada em uma prova prática, o avaliador humano levou em consideração o funcionamento correto dos programas construídos pelos alunos bem como a escrita desses programas.

Na Base *B*, as correções das soluções dos alunos foram realizadas por um outro professor emulando o padrão de correção do professor que corrigiu a Base *A*.

8.2 Experimentos parciais

Os experimentos parciais são a aplicação gradativa do modelo de avaliação semi-automática de exercícios de programação proposto neste trabalho. Esses experimentos foram realizados para mostrar como evoluímos a ideia de predição de notas através da combinação das técnicas de reconhecimento de padrões de aprendizagem supervisionada e não-supervisionada.

Os experimentos parciais foram rotulados como I, II e III. No Experimento I, de uma forma mais simples, apenas criamos e aplicamos o modelo de regressão linear múltipla. Para isso, escolhemos dois terços das amostras da base experimental, que

é uma quantidade percentual acima dos 50% utilizados por Naude, Greyling e Vogts (2010). Em seguida, criamos um modelo de regressão linear com essas amostras e com as 60 características como variáveis independentes. A partir do modelo criado, predizemos a variável dependente nota.

No Experimento II, utilizamos o *clustering* para remover *outliers*, formar agrupamentos de amostras semelhantes e, de cada agrupamento formado, selecionar dois terços para formar um único conjunto de treino do modelo de regressão linear.

No Experimento III, aplicamos o *clustering* sobre as bases experimentais e, dos *clusters* formados, selecionamos o maior deles, isto é, o que contém o maior número de amostras. Do *cluster* selecionado, selecionamos dois terços como conjunto de treino do modelo de regressão para predizermos, a partir dele, as notas de um terço das amostras do *cluster* representando o conjunto de teste.

Os resultados dos experimentos I, II e III realizados nas Bases A e B demonstram que os resultados de geração do modelo de regressão linear e a predição de valores para a variável dependente *nota* podem ser melhorados utilizando-se a técnica de *clustering*. Porém, a eficácia de um modelo de predição pode ser afetada pelo grau de liberdade do modelo, isto é, pelo número de amostras e pelo número de características. Dessa forma, um valor pequeno do grau de liberdade do modelo indica um *ranking* deficiente e não confiável para aplicação do modelo de regressão nas amostras de teste. Por isso, é importante desenvolver estratégias que melhor selecionem não apenas as amostras, mas também o número de variáveis independentes do modelo.

8.2.1 Resultados

As tabelas 8.4 e 8.5 apresentam os resultados de geração do modelo de regressão linear para as bases A e B, respectivamente, nos experimentos I, II e III. A segunda coluna dessas tabelas mostra o tamanho da amostra utilizada como treino do modelo. A terceira coluna mostra a quantidade de variáveis selecionadas pelo algoritmo *stepwise*. A quarta coluna informa o Erro Padrão Residual (EPR), que é uma medida de variação do erro em torno da reta. O valor *GL* na quinta coluna é o valor do grau de liberdade do modelo, isto é, a diferença entre o número de amostras e o número de

características incrementado em um. Os valores R^2 e ${R_a}^2$ na quinta e sexta colunas são, respectivamente, o coeficiente de determinação e o coeficiente de determinação ajustado. A Estatística F e o Valor-p das colunas 8 e 9 são testes de significância estatística (HAIR; TATHAM; BLACK, 1998). Rejeita-se a hipótese nula quanto maior for o valor da Estatística F e menor o Valor-p.

Base A

Experimento	Tamanho da amostra	Qtd. variáveis	EPR	*GL*	R^2	${R_a}^2$	Estatística-F	Valor-p
I	27	17	0.3429	9	0.9816	0.9470	28.3	8.965×10^{-6}
II	23	19	0.1043	3	**0.9992**	**0.9944**	**204.9**	**0.0004869**
III	13	12	-	0	1	-	-	-

Tabela 8.4: Resultados do modelo de regressão (Base A)

Os resultados da Tabela 8.4 indicam, segundo os valores de EPR, R^2 e ${R_a}^2$, melhor ajuste com significância estatística para o modelo de regressão do Experimento II, em que se selecionam dois terços de cada *cluster* para criar o modelo. Porém, o valor de *GL* do Experimento II é pequeno, que pode indicar risco de superajuste do modelo, isto é, pouca capacidade de generalização de previsão. No Experimento III, foi criado um ajuste perfeito, conforme o valor de R^2, porém, com *ranking* deficiente, conforme o valor nulo de *GL*, que indica incapacidade de generalização de previsão.

Na Tabela 8.5, quando realizamos os experimentos I, II e III para a Base B, que é uma base maior do que a Base A, confirmamos que o Experimento II apresenta melhor ajuste. Mas, dessa vez, com um maior valor de *GL*, o que indica maior capacidade de generalização do modelo.

Na Tabela 8.5, o modelo do Experimento III, embora tenha um ajuste perfeito, conforme os valores de EPR, ${R_a}^2$ e ${R_a}^2$, tem baixa capacidade de generalização, conforme o valor de *GL*.

Base B

Experimento	Tamanho da amostra	Qtd. variáveis	EPR	*GL*	R^2	${R_a}^2$	Estatística-F	Valor-p
I	67	21	0.5901	45	0.7071	0.5705	5.174	1.924×10^{-6}
II	65	23	**0.5263**	**41**	**0.7903**	**0.6270**	**6.718**	$\mathbf{7.467 \times 10^{-8}}$
III	15	13	5.69×10^{-6}	1	1	1	1.861×10^{30}	5.738×10^{-16}

Tabela 8.5: Resultados do modelo de regressão - (Base B)

Uma importante observação nos Experimentos III das Tabelas 8.4 e 8.5 é que o tamanho da amostra e o número de características dos dois modelos de regressão são muito próximos, porém, o modelo do Experimento III da Tabela 8.5 possui o melhor ajuste e

com significância estatística, conforme os valores de R^2, ${R_a}^2$, *Estatística-F* e *Valor-p*. Isso sugere que, no Experimento III, as amostras da Base B e as características foram melhor selecionadas do que na Base A. Conforme pode ser observado no alto valor da Estatística-*F* no Experimento III da Tabela 8.5, houve alta significância na escolha dos coeficientes do modelo de regressão. O maior número de amostras da Base B possibilitou essa melhor seleção de amostras por *clustering* e características pelo algoritmo *stepwise* (HAIR; TATHAM; BLACK, 1998).

As tabelas 8.6 e 8.7 são as tabelas de ANOVA para análise de variância dos resíduos dos modelos de regressão dos experimentos I, II e III nas bases A e B. O valor *GL* é o grau de liberdade do modelo, *Soma SQ* é a soma dos quadrados dos resíduos e *Média SQ* é a média dos quadrados de resíduos. Quanto menores forem os valores de *Soma SQ* e *Média SQ* e maior o valor de *GL*, melhor é o modelo.

ANOVA (Resíduos) - Base A

Experimento	*GL*	Soma SQ	Média SQ
I	9	1.0584	0.1176
II	3	**0.0327**	**0.0109**
III	0	0	0

Tabela 8.6: Análise de variância - Base A

Na Tabela 8.6, o Experimento II sobre a Base A apresenta os melhores resultados, levando em consideração o valor de *GL*. Embora o Experimento III tenha obtido a menor soma de quadrados de resíduos e a menor média, o valor de *Gl* é nulo, o que indica incapacidade de generalização do modelo.

ANOVA (Resíduos) - Base B

Experimento	*GL*	Soma SQ	Média SQ
I	45	15.6707	0.3482
II	41	11.3579	0.2770
III	1	**0**	**0**

Tabela 8.7: Análise de variância - Base B

Por outro lado, na Tabela 8.7, o Experimento III apresentou os melhores resultados de variância dos resíduos, embora o valor de *GL* fosse pequeno. Vale ressaltar, no entanto, que os resultados do Experimento III mostrados na Tabela 8.5 para a Base B apontam significância estatística, mesmo sendo o valor de *GL* pequeno. Nesse caso, como o valor de similaridade interna do maior *cluster* da Base B é de 92.6%, não seria necessária tão alta capacidade de generalização, isto é, um alto valor de *GL*, para obter o melhor ajuste, uma vez que as amostras do *cluster* são muito semelhantes entre si.

Os gráficos da Figura 8.2 mostram os resultados de predição dos experimentos I, II e III analisando a relação linear entre os valores reais de notas atribuídas por um professor e os valores preditos pelo MRLM gerado em cada experimento para as bases A e B. Os gráficos (a) a (c) são os Experimentos I, II e III realizados, respectivamente, sobre a Base A. Os gráficos (d) a (f) são os experimentos I, II, III realizados, respectivamente, sobre Base B. Cada ponto dos gráficos representa o par (n_r, n_p), onde n_r é a nota real atribuída por um professor a uma amostra de exercício e n_p é o valor predito para essa nota por um modelo de regressão linear. A reta de regressão linear em cada gráfico representa a melhor relação linear entre a nota real e a nota predita.

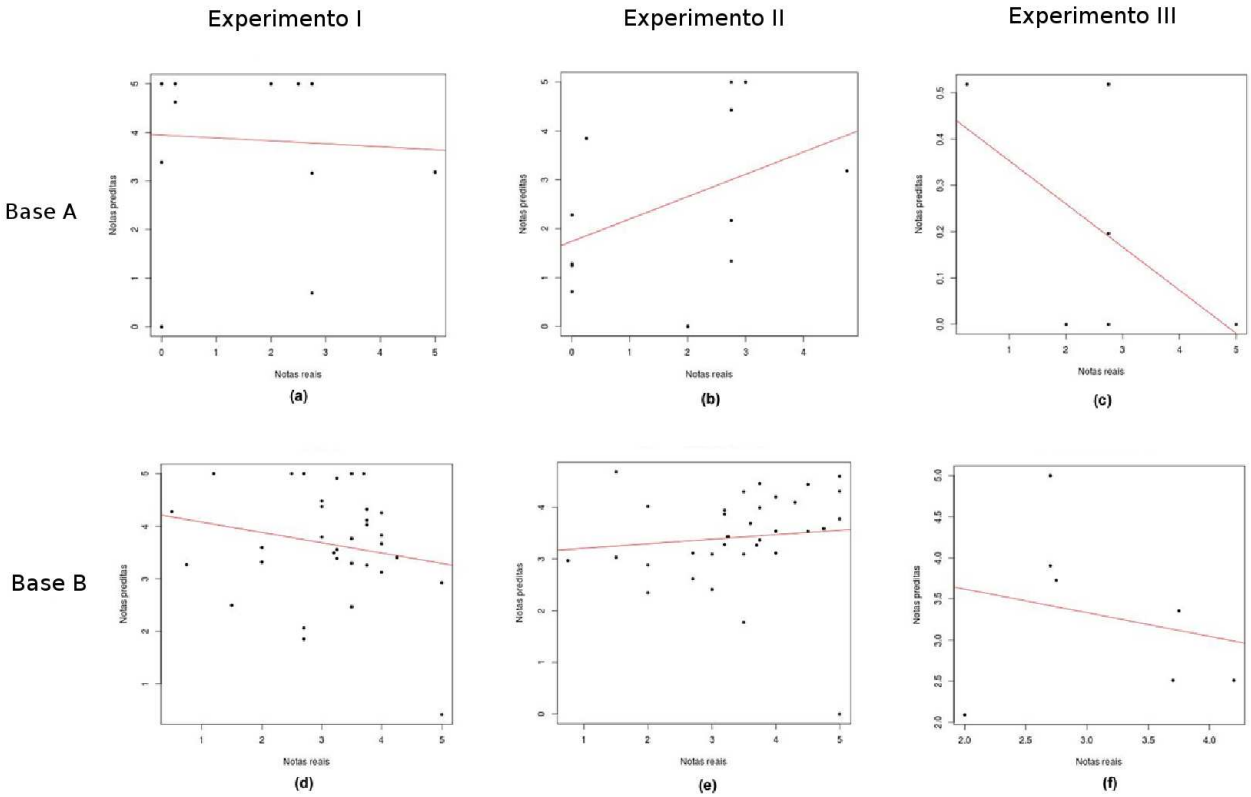

Figura 8.2: Resultados de predição

De acordo, com os gráficos da Figura 8.2, observamos que para o Experimento I (gráficos (a) e (d)) há maior dispersão dos pontos em relação à reta, indicando baixa relação entre a nota real e predita. Para a Base A, no Gráfico (a), isso é mais evidente. No entanto para a Base B, no Gráfico (d), a dispersão é menor, uma vez que o número de amostras é maior.

Nos gráficos (b) e (e) do Experimento II, quando utilizamos *clustering* para selecionar

as amostras de treino, observamos uma maior aproximação dos pontos em relação à reta de regressão do que nos gráficos (a) e (d) do Experimento I. Essa aproximação é mais nítida no Gráfico (e), quando utilizamos a Base B, que contém o maior número de amostras.

Os gráficos (e) e (f) do Experimento III, na Figura 8.2, indicam menor dispersão em relação à reta de regressão linear, o que pode ser confirmado pela análise de resíduos das Tabelas 8.8 e 8.9. A dispersão no Gráfico (f) também é menor em relação ao Gráfico (c), uma vez que o valor de *GL* do modelo é nulo para o Experimento III da Base A, conforme a Tabela 8.4.

As Tabelas 8.8 e 8.9 são as tabelas de análise de resíduos da predição de notas. Nessas tabelas informamos o erro mínimo (*Mínimo*), o erro máximo (*Máximo*) e o erro médio (*Médio*) de predição.

Resíduos - Base A

Experimento	Minimo	Maximo	Médio
I	0	5	2.8309
II	**0.5822**	**3.5900**	**1.716**
III	0.2790	5	2.467

Tabela 8.8: Análise de Resíduos - Base A

Na Tabela 8.8, o Experimento II apresenta os melhores resultados segundo a diferença entre o erro máximo e erro mínimo e o menor valor de erro médio. O Experimento I, mesmo com um número de amostras maior do que os Experimentos II e III (Tabela 8.4), apresentou os piores resultados. Isso reforça que a eficácia do modelo de regressão depende não somente da quantidade de amostras, mas também de uma seleção representativa dessas amostras, o que pode ser favorecido pelo aplicação do *clustering*. No Experimento III, a seleção de amostras e a predição de notas foram realizadas no mesmo *cluster*, porém, o número de amostras foi insuficiente para criar um modelo confiável de predição, conforme sugerem os resultados de predição no Experimento III.

Para uma base maior, como a Base B, os Experimentos II e III apresentaram os melhores resultados, conforme a Tabela 8.9. Isso confirma que a seleção de amostras e a predição de notas por *cluster* melhoram os resultados de regressão mesmo com um número menor de amostras de referências.

Os gráficos da Figura 8.3 ilustram a distribuição dos resíduos em relação ao valor

Resíduos - Base B			
Experimento	Minimo	Maximo	Médio
I	0.1367	4.5884	1.2578
II	0.0650	5	0.8609
III	**0.0800**	**2.300**	**1.12**

Tabela 8.9: Análise de Resíduos - Base B

de $\hat{Y}$ estimado. Os gráficos (a), (b) e (c) mostram a distribuição de resíduos nos experimentos I, II e III para a Base A, respectivamente. Os gráficos (d), (e) e (f) representam os três experimentos sobre a Base B. Quanto mais próximos da reta $(y = 0)$ estiverem os pontos que representam os resíduos, melhores os resultados.

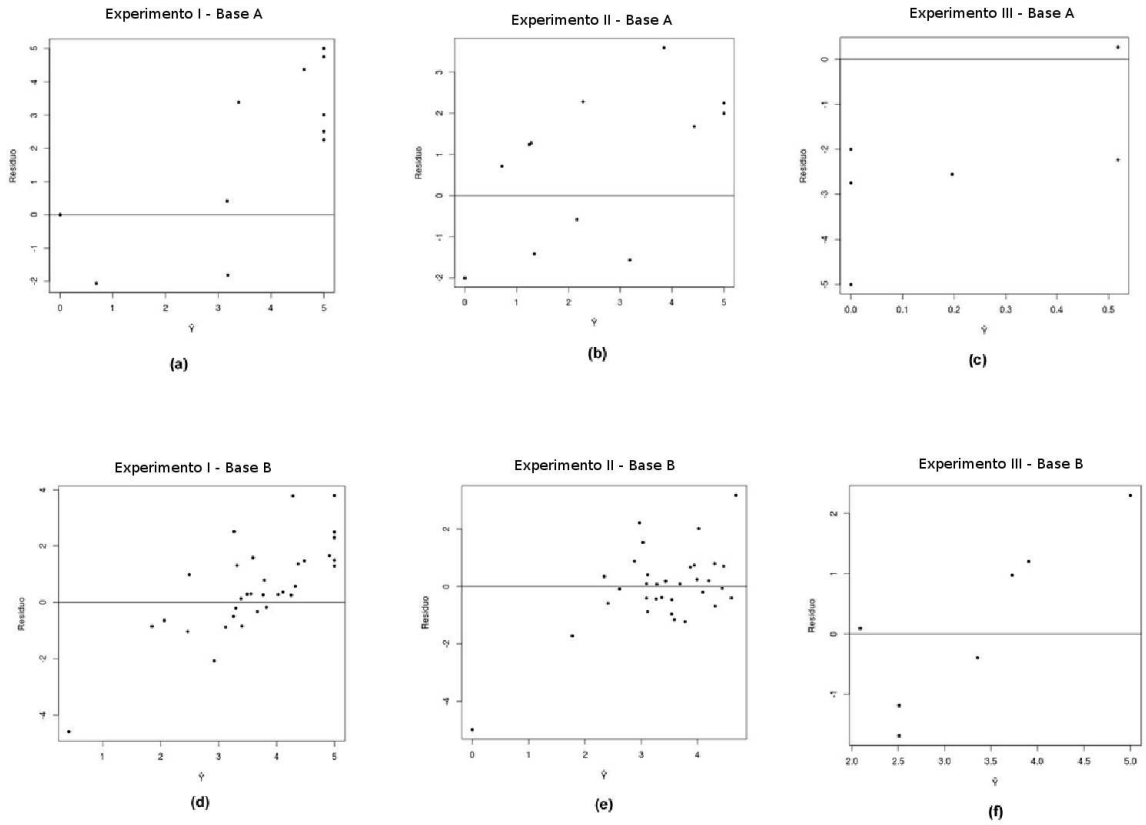

Figura 8.3: Análise de resíduos

Os gráficos (e) e (f) da Figura 8.3 confirmam a melhor distribuição dos resíduos em torno da reta nos experimentos II e III. O Experimento III, no entanto, possui uma faixa menor de distribuição dos resíduos, sugerindo maior confiança de predição de notas por *cluster*, conforme o Gráfico (f).

Uma importante observação nos gráficos dos Experimentos I, II e III na Figura 8.3 é que, em geral, os pontos de maior dispersão ocorrem mais nas notas mais altas

(próximas de 5.0) e nas mais baixas (0.0 a 1.0). Por outro lado, as notas medianas entre 2.0 e 4.0, apresentam menor taxa de erro, conforme os gráficos (d), (e) e (f). De acordo com Naude, Greyling e Vogts (2010), há maior dispersão entre padrões com notas mais baixas e maior aproximação entre os padrões com notas mais altas. Nesse caso, há uma necessidade dos modelos criados neste trabalho aprenderem melhor como distinguir melhor as notas altas, o que pode ser feito obtendo-se mais exemplos de amostras que caracterizem melhor essas notas e selecionando melhor as características que as descrevem.

Já para os padrões com notas mais baixas é mais difícil estabelecer um conjunto de características para descrevê-las pois em geral são muito divergentes entre si. Isso porque, para o domínio de programação, em geral, os programas avaliados com notas muito baixas têm um número muito pequeno ou excessivo de linhas de código. Dessa forma, o número de linhas pode ser uma possível variável a se considerar para caracterizar programas avaliados com notas muito baixas.

Com os resultados alcançados, concluímos, conforme os resultados dos Experimentos II e III, que o *clustering* melhora o ajuste e a predição de modelos de regressão linear. Embora haja perda na capacidade de generalização do modelo quando realizamos a predição por *cluster* devido ao valor pequeno do grau de liberdade do modelo, o *clustering* forma conjuntos mais homogêneos de amostras. Sendo os *clusters* mais homogêneos, a predição por *cluster* não demandará alta capacidade de generalização do modelo de regressão criado, uma vez que os exemplos a serem preditos são muito semelhantes aos exemplos aprendidos.

8.3 Experimentos de aplicação do método de tese

Os experimentos parciais realizados na seção anterior mostraram que o *clustering* melhora os resultados de predição por regressão linear porque seleciona um conjunto de amostras mais representativas como treino do modelo de regressão linear. Além disso, realizando a predição de notas de exercícios por *cluster*, temos a possibilidade de ter um conjunto de teste mais semelhante ao conjunto de treino, o que facilita a predição de notas sem que seja necessário um alto grau de liberdade do modelo de regresssão.

Nos experimentos desta seção, além de aplicar a seleção de amostras e a predição de notas por *cluster* realizadas nos experimentos parciais, reduzimos, para cada *cluster*, o número de características do modelo de predição a ser gerado. Essa redução foi realizada conforme as características mais descritivas de cada *cluster* apontadas pelos algoritmos de *clustering*.

Para aplicação do método desta tese, realizamos dois experimentos: o Experimento IV sobre a Base A e o Experimento V sobre a Base B. Para o Experimento IV, utilizamos inicialmente 34 características como variáveis independentes dos modelos de regressão gerados para cada *cluster*. Isso porque o número de amostras da Base A é pequeno e, realizando treino e predição por *cluster*, a predição poderia ser realizada por um *ranking* deficiente.

No Experimento IV, retiramos de cada *cluster* as amostras que eram muito diferentes das demais amostras e deixamos no *cluster* apenas as amostras mais semelhantes entre si. Dessas amostras, selecionamos um conjunto de treino e outro de teste e, em seguida, predizemos as notas das amostras de teste desse *cluster* tomando como referência as notas das amostras de treino. Se o número de amostras do *cluster* fosse menor que três e maior que 1, predizemos as notas dessas amostras conforme o índice de similaridade com as amostras selecionadas como treino. Se o número de amostras do *cluster* excedesse a três, predizemos as notas das amostras de teste por um modelo de regressão linear ajustado com um número de características menor ou igual ao número de amostras do *cluster*.

Para os experimentos IV e V, dividimos a base em onze *clusters* e selecionamos as características descritivas de cada *cluster*. Depois de gerar o modelo, através do algoritmo *stepwise*, realizamos uma nova seleção das características mais relevantes para o modelo.

No Experimento V, utilizando a Base B, que possui um maior número de amostras, realizamos o experimento treinando o modelo de cada *cluster* com dois terços de suas amostras e com um conjunto de características descritivas do *cluster*. O número de características descritivas para cada *cluster* é a metade do número de amostras de treino de cada *cluster*.

Como foi identificado apenas um *outlier* na Base B pelos algoritmos de *clustering*,

não realizamos no Experimento V a etapa de retirada de *outliers* dos *clusters*. Essa etapa, no entanto, conforme foi realizada no Experimento IV, deve ser executada caso não se formem *clusters* muito homemogêneos, isto é, *clusters* com baixo índice de similaridade interna.

Os procedimentos e resultados dos experimentos IV e V são apresentados em detalhes nas subseções a seguir.

8.3.1 O Experimento IV

Após a realização da etapa de *clustering* da Base A, os dois maiores *clusters* gerados continham dez amostras. Esses agrupamentos foram representados pelos *Clusters* 9 e 10 da Figura 8.4. O *Cluster* 10 é mais homogêneo do que o *Cluster* 9 por apresentar menos *outliers*, isto é, amostras muito distintas da maioria das amostras dentro de um mesmo *cluster*. Essa afirmação é confirmada pelos índices de entropia e de pureza do *Cluster* 10 apresentados nos resultados de *clustering* da Figura 8.9.

Como não temos um número grande de amostras que representem as variações dessa base para a predição de notas por regressão linear, realizamos essa predição com um número bem menor de características, menor ou igual ao número de padrões avaliados pelo professor. Nesse caso, o modelo seria bastante limitado em generalizar a predição de notas para outros exemplos (HAIR; TATHAM; BLACK, 1998). Porém, o algoritmo de *clustering*, ao agrupar as amostras mais semelhantes entre si em *clusters* e selecionar as características mais relevantes desses *clusters*, facilitou o processo de predição por regressão linear. Dessa forma, selecionando apenas algumas amostras de um *cluster* para que suas notas fossem atribuídas pelo professor, foi possível predizer as demais notas desse *cluster* com possibilidades de sucesso uma vez que eram muito similares às amostras pontuadas pelo professor.

Quando em um *cluster* há dois ou três padrões, e todos são idênticos nas características mais relevantes, escolhemos um deles para o professor atribuir a nota e repetimos essa nota para os demais padrões.

Para os demais *clusters*, formados por no máximo três padrões, devido à insuficiência de amostras para realizar predição por regressão linear, utilizamos apenas os índices de

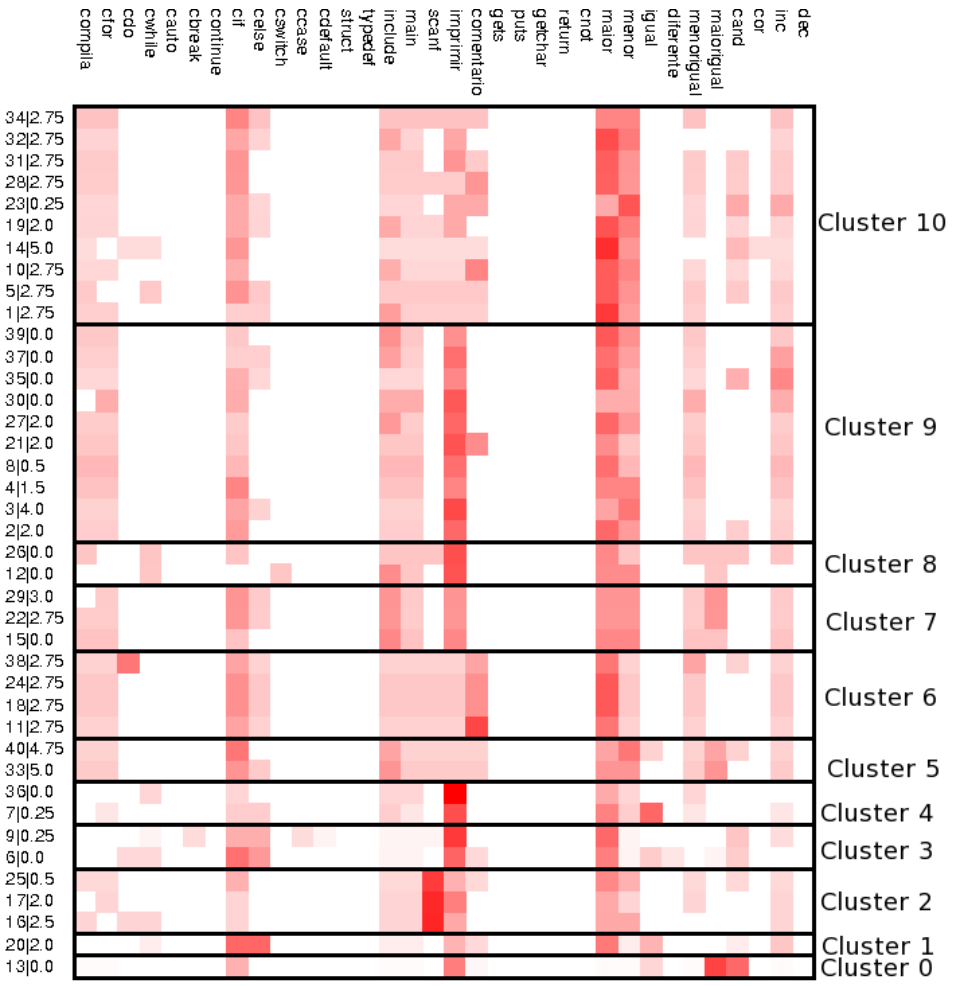

Figura 8.4: Agrupamento de programas

similaridades entre padrões mais próximos para estimar as notas.

No caso de um *cluster* conter apenas dois padrões, indicamos para o professor atribuir nota àquele padrão que, com base nas características mais relevantes do *cluster*, tem a possibilidade de ser a melhor solução. Sendo os padrões programas de computador, para duas soluções muito próximas, é considerada, em geral, a melhor solução aquela que foi construída com menos instruções e menos operações.

Um exemplo de como selecionamos a melhor solução de um *cluster* com dois padrões pode ser ilustrado através do *Cluster 5* (Figura 8.1). Para esse *cluster*, o algoritmo de *clustering* indica como as três características mais relevantes o sinal de $<$ (*menor*), o comando de seleção *if* (*cif*) e o sinal $\geq$ (*maiorigual*), conforme os resultados de *clustering* mostrados na Figura 8.5.

Na Figura 8.5, os padrões do *Cluster 5* são apresentados na forma vetorial com as três

Cluster 5

indice	compila	cfor	cdo	cwhile	**cif**	maior	**menor**	igual	diferente	menorigual	**maiorigual**	cand	cor	inc	dec
33	1	1	0	0	**2**	2	**2**	**0**	0	1	**2**	0	**0**	**1**	0
40	1	1	0	0	**3**	2	**3**	**1**	0	1	**2**	**1**	**0**	**1**	0

Figura 8.5: O *Cluster* 5 e as suas características mais relevantes

características mais relevantes em destaque. Observamos que o padrão de índice 33, para as características mais relevantes *cif* e *menor*, possui valores menores que os do padrão de índice 40. Isso nos leva a crer que o padrão 33 é a melhor solução, já que utilizou menos operações de comparações e menos instruções *if*. Dessa forma, deve-se solicitar ao professor que atribua nota a essa possível melhor solução. Em seguida, tendo essa nota o valor 0.97 como referência, estimamos a nota do segundo padrão proporcionalmente ao Índice de Similaridade Interna (ISIM) do *Cluster 5* (Figura 8.5). Nesse caso, sendo a nota 5.0 atribuída pelo professor à possível melhor solução, a nota 4.85 foi estimada para o segundo. Sabendo que a nota 4.75 era a nota real do segundo padrão, o erro de predição foi de 0.1.

Quando um *cluster* continha apenas um padrão, buscamos na matriz de similaridade entre todos os padrões de uma base os exemplos mais similares a esse padrão que já tenham sido avaliados por um professor ou mesmo automaticamente. Com base nos índices de similaridades entre esses exemplos e o padrão que desejamos avaliar, predizemos a nota deste último.

Como última etapa do processo de predição de notas, reunimos todas os *outliers* identificadas em cada *cluster* e tentamos predizê-las da mesma forma que realizamos as predições dos *clusters* com apenas um padrão.

Combinando as técnicas de *clustering* e de regressão linear para predizer parcialmente as notas da Base A, alcançamos resultados que confirmam que a nossa estratégia é viável para avaliar de forma semi-automática exercícios de programação.

Das 40 amostras da Base A, o nosso modelo conseguiu predizer 40% das notas com erro máximo de 0.5. Apenas 12.5% dessa base foi predita com erro acima de 0.5. Para chegar a esses resultados, 30% da base foi utilizada como treino de predição. Isso significa que um conjunto de exemplos foi selecionado pelo nosso modelo para o especialista humano (ou professor) atribuir notas e, a partir desses

exemplos, aprendendo o padrão de correção do professor, predizemos as notas das demais amostras. O gráfico da Figura 8.6 apresenta o percentual de treino, de predições certas e erradas e o percentual de *outliers* na Base A.

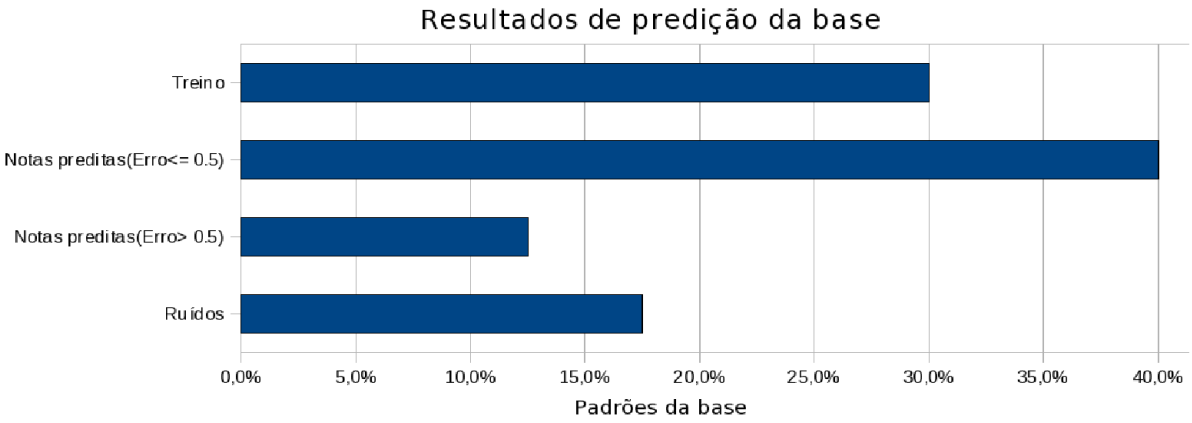

Figura 8.6: Resultados de predição

Os resultados indicam correlação de 81.7% entre as notas preditas pelo nosso modelo e as notas atribuídas pelo avaliador humano. O gráfico da Figura 8.7 mostra o gráfico dessa correlação.

De acordo com o gráfico da Figura 8.7, apenas cinco padrões estão distantes da linha de tendência das notas preditas com variações de erro ± 0.5. Esses últimos padrões possivelmente são *outliers* da base, padrões que foram agrupados de forma incorreta pelo algoritmo de *clustering* ou seguem um padrão de solução diferente dos demais de exercícios de programação da base.

Na Figura 8.8, mostramos a distribuição dos resíduos da Base A. Observa-se que a maior parte das amostras preditas apresentaram taxa de erro menor que 1.0, algumas estão muito próximas à reta $y = 0$ e outras sobre a própria reta, indicando taxa de erro igual a zero. Uma observação importante é que, ao contrário dos experimentos parciais deste trabalho e dos experimentos de Naude, Greyling e Vogts (2010), com a aplicação do método desta tese, alcançamos menor taxa de erros não só para as notas medianas, mas também para as notas baixas e altas.

Os resultados de *clustering* apresentados na Figura 8.9 revelam índices de entropia igual a 0.338 e de pureza, 0.725, que indicam homogeneidade dos *clusters* formados. A entropia mede o grau de confusão do *cluster*, ocasionada principalmente pela presença

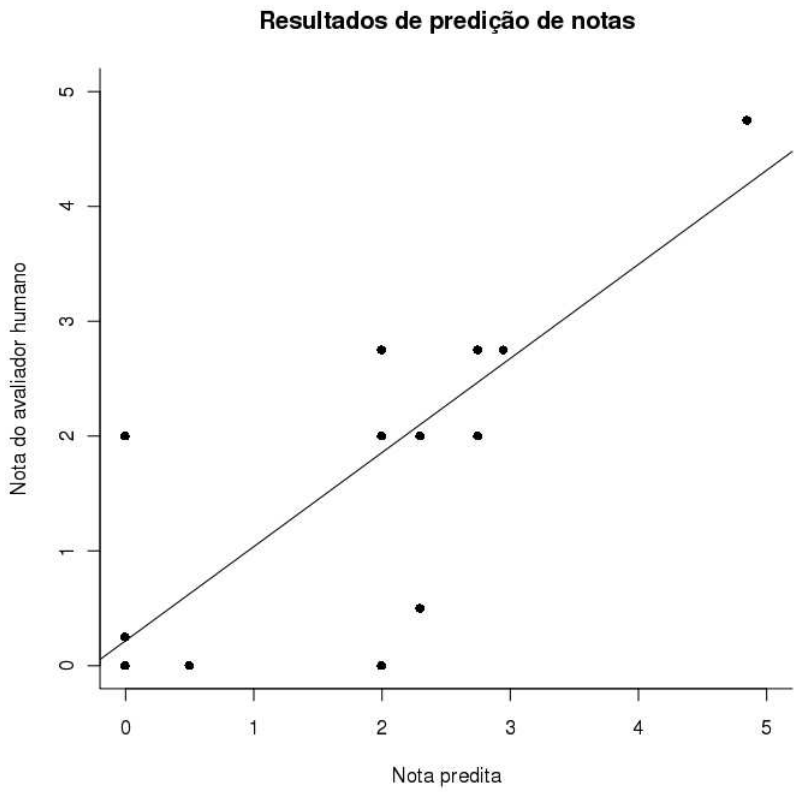

Figura 8.7: Gráfico de correlação das notas preditas

de *outliers*. Já a pureza indica quão homogêneo é um *cluster*. Esses índices são calculados avaliando se padrões de uma mesma classe, isto é, padrões com notas próximas, foram agrupados dentro do mesmo *cluster*. Dessa forma, valores de entropia baixos e valores de pureza altos indicam bons resultados de *clustering*. Isso também significa que o conjunto de características selecionadas para representar programas é uma boa combinação de variáveis para realizar predições de notas.

Cada *cluster* da Figura 8.9 é identificado por um rótulo chamado *CID*, que é um número. O valor *size* representa o número de padrões de um *cluster*. O valor ISIM é o índice de similaridade interna entre os padrões de um *cluster* e *ESim* é o índice de similaridade externa entre um *cluster* e os demais *clusters*. O valor *ISdev* e *ESdev* indicam, respectivamente, os desvios-padrão dos índices de similaridade interna e externa. Já *ENTPY* e *PURITY*, representam, nessa ordem, os índices de entropia e pureza de cada *cluster*. As colunas seguintes são as classes dos padrões representadas por notas diferenciadas em intervalos de 0.5 pontos. No *Cluster 6*, por exemplo, só

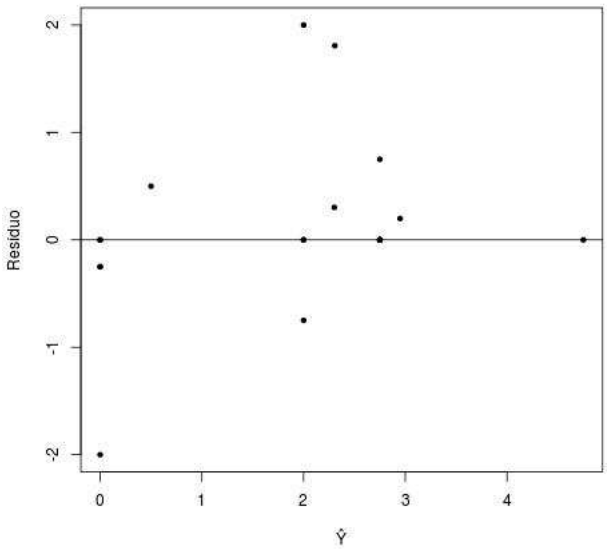

Figura 8.8: Análise de residuos

há padrões da classe 3, correspondente às notas de 2.5 a 3.0. Esse *cluster* é puro pois possui entropia 0 e pureza 1. O *Cluster 9*, por sua vez, é menos homogêneo porque contém exemplos distribuídos em diferentes classes de notas, conforme a Figura 8.9.

De acordo com a Figura 8.9, praticamente todos os *clusters* formados apresentam ISIM acima de 90%, o que informa que os padrões agrupados nos *clusters* são muito semelhantes entre si. Já os índices de similaridade externa (*ESim*) entre um *cluster* e os outros *clusters* não são tão altos, sugerindo menor distinção entre os *clusters*.

O algoritmo de *clustering* que aplicamos para reconhecer os agrupamentos de programas similares, o *Bissecting K-means* apresentado no Capítulo 6, após formar os *clusters*, também indica, em percentuais, quais características melhor descrevem e discrimam um *cluster*, conforme o exemplo da Figura 8.10.

De acordo com a Figura 8.10, as características que melhor descrevem o *Cluster 10* são o operador *maior*($>$), o operador *menor*($<$) e o *cif*, que é o comando de seleção *if*. Já as características que discriminam o *Cluster 10* são a instrução *imprimir* (função *printf* na Linguagem C), o operador *menor* ($<$) e o operador *maior ou igual* ($\geq$).

Para a predição de notas dos padrões de um *cluster*, selecionamos como características mais relevantes aquelas mais descritivas apontadas pelo algoritmo de *clustering* para

```
-------------------------------------------------------------------------------------------
11-way clustering: [I2=3.85e+01] [40 of 40], Entropy: 0.338, Purity: 0.725
-------------------------------------------------------------------------------------------
cid  Size  ISim  ISdev   ESim  ESdev  Entpy Purty |    3    2    4   15    0    5   25
-------------------------------------------------------------------------------------------
  0     1 +1.000 +0.000 +0.359 +0.000 0.000 1.000 |    0    0    0    0    1    0    0
  1     1 +1.000 +0.000 +0.626 +0.000 0.000 1.000 |    0    1    0    0    0    0    0
  2     3 +0.951 +0.002 +0.601 +0.060 0.565 0.333 |    0    1    0    0    1    0    1
  3     2 +0.947 +0.000 +0.663 +0.011 0.000 1.000 |    0    0    0    0    2    0    0
  4     2 +0.888 +0.000 +0.641 +0.017 0.000 1.000 |    0    0    0    0    2    0    0
  5     2 +0.970 +0.000 +0.750 +0.013 0.000 1.000 |    0    0    0    0    0    2    0
  6     4 +0.928 +0.035 +0.714 +0.056 0.000 1.000 |    4    0    0    0    0    0    0
  7     3 +0.970 +0.009 +0.769 +0.014 0.327 0.667 |    2    0    0    0    1    0    0
  8     2 +0.908 +0.000 +0.712 +0.074 0.000 1.000 |    0    0    0    0    2    0    0
  9    10 +0.922 +0.024 +0.760 +0.021 0.600 0.500 |    0    3    1    1    5    0    0
 10    10 +0.904 +0.021 +0.754 +0.037 0.483 0.700 |    7    1    0    0    1    1    0
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Figura 8.9: Resultados de Clustering

```
Cluster   9, Size:    10, ISim: 0.922, ESim: 0.760
 Descriptive:  imprimir 28.1%, maior 23.1%, menor 12.1%,
 Discriminating:  scanf 26.8%, comentario 14.5%, imprimir 13.2%,

Cluster  10, Size:    10, ISim: 0.904, ESim: 0.754
    Descriptive:  maior 34.5%, menor 19.2%, cif 11.8%
 Discriminating:  imprimir 56.3%, menor 13.1%, maiorigual  9.0%,
```

Figura 8.10: Identificação de características descritivas e discriminantes de clusters

cada *cluster*. Os resultados de predição obtidos com essa seleção de características indicam, na maioria das observações, que essas características explicam a atribuição de notas pelo professor aos padrões de um *cluster*. Dessa forma, as notas preditas para os demais padrões desse mesmo *cluster* muito se aproximam das notas atribuídas pelo professor, conforme apresentamos nos resultados a seguir.

Na Figura 8.11, são apresentados o número de erros e acertos de predição em cada um dos *clusters* formados (Figura 8.1). Consideramos erro de predição cada padrão cuja nota predita difere em mais de 0.5 da nota real atribuída pelo professor a esse padrão. Para predições com erro entre ± 0.5, consideramos que houve acerto de predição.

De acordo com os resultados apresentados na Figura 8.11, observamos que nos *clusters* 3, 4, 5, 7 e 8, um padrão foi predito com sucesso a partir de um padrão de treino. No *Cluster* 6, dois padrões foram preditos corretamente a partir de um. No *Cluster* 2, a partir de um padrão de treino, um padrão foi predito certo e outro errado. Os *clusters* 0 e 1 foram preditos por seus índices de similaridades com os demais padrões da base.

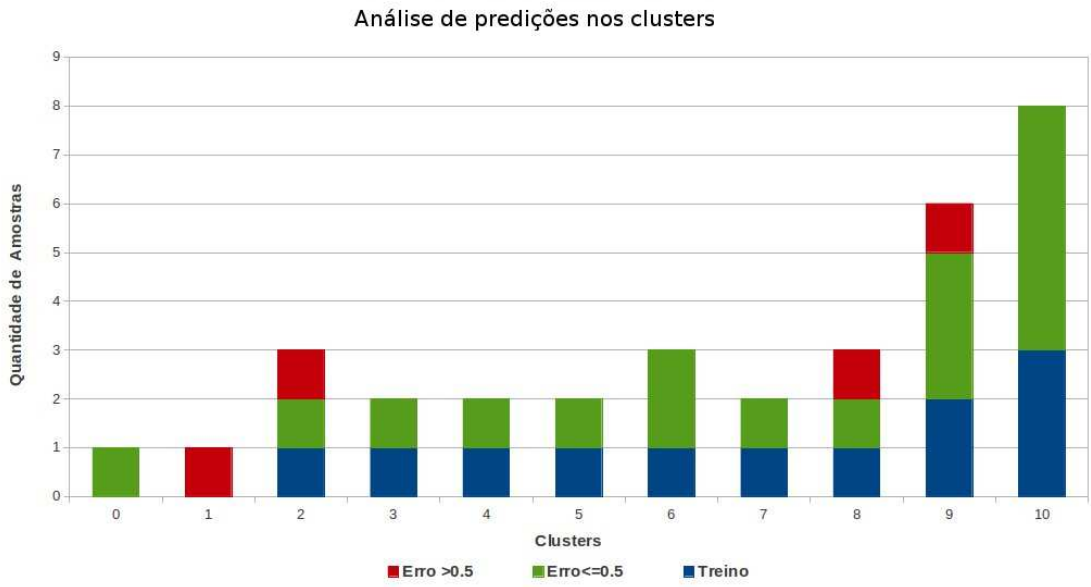

Figura 8.11: Análise de predições nos *clusters* da Base A

O padrão do *Cluster 0* foi predito com sucesso, mas o padrão do *Cluster* 1 foi predito incorretamente.

Os melhores resultados de predição, conforme a Figura 8.11, foram nos maiores *clusters*, isto é, nos *clusters 10 e 9*, respectivamente. No *Cluster* 10, com três padrões de treino, predizemos as notas de cinco padrões com apenas um erro. No *Cluster* 9, com dois padrões de treino, predizemos quatro notas com um erro. Os resultados do *Cluster* 10 foram melhores por ser esse *cluster* mais homogêneo que o *Cluster* 9, isto é, por conter menos *outliers*.

Dos vinte padrões da base de quarenta exercícios de programação que foram preditos pelo nosso modelo de avaliação semi-automática, 80% foram preditos com erro absoluto menor que 0.5.

Dos *clusters* com mais de três padrões preditos, o *Cluster* 10 foi o que apresentou menor erro médio, com valor de 0.15. A predição de maior erro, com valor de 2.0, foi do *Cluster 1*, que contém apenas um padrão.

A partir dos resultados de predição dos maiores *clusters*, temos indícios de que para uma base maior de exercícios de programação, poderemos alcançar melhores resultados. Isso porque se mais padrões entrarem nesses *clusters*, com uma base

mínima de treino, poderemos predizer notas de grande parte desses padrões e reduzir consideravelmente o esforço do professor em corrigir muitos exercícios.

Das amostras reconhecidas como *outliers*, conseguimos predizer as notas de 42% das amostras com erro máximo de 0.5 utilizando o critério de similaridade com o vizinho mais próximo, que pode ser uma amostra de treino ou um padrão já pontuado automaticamente. Esses *outliers* que conseguimos predizer as notas corretamente podem ter sido agrupadas incorretamente nos *clusters*.

Os resultados apresentados sugerem, portanto, que o agrupamento dos exercícios de programação mais semelhantes em *clusters* e a seleção das características mais relevantes de cada *cluster* realizadas pelo algoritmo de *clustering* facilitam a predição de notas por regressão linear e por índices de similaridades entre padrões.

8.3.2 O Experimento V

O Experimento V é a aplicação do método de tese apresentado no Capítulo 6 sobre a Base B. Nesse experimento, assim como no Experimento IV, a predição de notas foi realizada por *cluster*. Para isso, para cada *cluster*, foi gerada uma matriz de indexação cujas dimensões foram as características mais descritivas apontadas pelos algoritmos de *clustering* para cada *cluster*.

Uma vez divididos os padrões da matriz de indexação em treino e teste, criamos um Modelo de Regressão Linear Múltipla para cada *cluster* com número de amostras maior ou igual a quatro. Para os *clusters* com dois ou três padrões, realizamos a predição de nota de um padrão de teste conforme suas similaridades com os padrões de treino.

A Tabela 8.10 apresenta as informações dos modelos de regressão linear criados para cada *cluster*. A primeira coluna da Tabela 8.10 contém a identificação do *cluster*. A segunda coluna informa o número de padrões do *cluster*. A terceira coluna contém o ISIM entre os padrões de um mesmo *cluster*. Quanto maior o valor de ISIM, mais homogêneo é o *cluster*. O valor de *TA* na quarta coluna contém o número de amostras utilizadas para treino do modelo. A quinta coluna contém o número de variáveis selecionadas pelo algoritmo *stepwise*, a partir do conjunto já reduzido de características selecionadas para formarem o modelo de regressão. A sexta coluna indica o EPR. Na

sétima coluna, informa-se o valor de *GL*, isto é, o grau de liberdade do modelo. Os valores de R^2 e $R_a{}^2$ são, respectivamente, os valores do coeficiente de determinação e do coeficiente de determinação ajustado. As últimas duas colunas contêm os testes de significância estatística.

Modelos de Regressão - Experimento V

Cluster	*n*	ISIM	TA	Qtd.Variáveis	EPR	*GL*	R^2	$R_a{}^2$	Estatística-*F*	Valor-*p*
0	1	-	-	-	-	-	-	-	-	-
2	4	0.868	3	1	0.4818	1	0.5714	0.1429	1.333	0.4544
3	15	0.876	10	6	0.5974	3	0.8916	0.6747	4.112	0.1367
4	7	0.917	5	0	0.7416	4	-	-	-	-
6	6	0.877	4	0	0.8165	3	-	-	-	-
7	21	0.937	13	0	0.5685	13	-	-	-	-
8	14	0.911	9	1	0.4807	7	0.5208	0.4524	7.609	0.0281
9	10	0.891	7	0	1.398	6	-	-	-	-
10	17	0.892	11	3	0.4319	7	0.7024	0.5748	5.506	0.0293

Tabela 8.10: Resultados dos modelos de regressão

Na Tabela 8.10, o *Cluster* 0 contém apenas uma amostra, que, segundo a nossa estratégia, consideramos como sendo um *outlier*. Por conter apenas uma amostra, não foi criado um modelo de regressão de predição de notas para esse *cluster*. Nos *clusters* 4, 6, 7 e 9, apenas o coeficiente de intercepto β_0 foi estimado para compor o modelo de regressão, conforme a Tabela 8.11. Os padrões dos *clusters* 1 e 5 não aparecem na Tabela 8.10 porque as notas de seus padrões de teste foram preditas com base nas medidas de similaridade entre os padrões de treino, conforme resultados mostrados na Tabela 8.13

Os coeficientes de determinação R^2 e $R_a{}^2$, o tamanho da amostra, o valor do erro padrão, a quantidade de variáveis e os testes estatísticos da Tabela 8.10 apontam os melhores resultados de geração dos coeficientes do modelo de regressão para os *clusters* 3 e 10. O *Cluster* 8 embora tenha tamanho e número de amostras de treino próximos ao do *Cluster* 3, teve seu modelo formado apenas por uma variável independente. Dessa forma, mesmo tendo significância estatística maior do que dos *clusters* 3 e 10, segundo valores da Estatística-*F* e do Valor-*p*, no *Cluster 8*, a variável selecionada pelo algoritmo *stepwise* tem pouco poder de explicação do modelo, conforme os valores de R^2 e $R_a{}^2$. O mesmo acontece com o *Cluster* 2. Os valores altos dos erros máximo e médio de predição desses *clusters*, na Tabela 8.12, confirmam o baixo poder da variável selecionada do modelo para explicar a variável dependente nota.

Por outro lado, na Tabela 8.11, os modelos dos *clusters* 4, 6, 7 e 9, que não

foram explicados por um conjunto de variáveis, mas apenas pelos coeficientes β_0 de intercepto, apresentaram os menores valores de *Erro Padrão* e maior significância estatística, conforme o Valor-*t* e o Valor-*p* (Pr($>|t|$)) da Tabela 8.11. O Valor-*p* é a probabilidade de se obter um Valor-*t* maior ou igual aquele observado sob a hipótese nula. Um Valor-*p* muito baixo indica, dessa forma, alta significância estatística.

Os resultados da Tabela 8.11 sugerem, portanto, que o modelo gerado apenas pelo coeficiente de intercepto pode indicar uma mesma nota para todos os padrões de um *cluster* com probabilidade de erro em torno de 0.5.

Coeficientes β_0

Cluster	β_0	Erro padrão	Valor-*t*	Pr ($>|t|$)
4	2.4	0.3317	7.236	0.00194
6	3.0	0.4082	7.348	0.00521
7	3.4107	0.1519	22.45	8.8×10^{-12}
9	2.9714	0.5286	5.622	0.0013

Tabela 8.11: Coeficientes de intercepto estimados

A Tabela 8.12 apresenta os valores mínimo, máximo e médio dos resíduos, isto é, da variação do erro em cada *cluster*. O valor TT é o número de amostras de testes preditas. Conforme os resultados da Tabela 8.12, o *Cluster 5* obteve o melhor resultado de predição com erro máximo e médio de 0.17. A nota 4.17 predita para o padrão de teste do *Cluster* 5 foi obtida pelo seu indice de similaridade com os dois padrões de treino do mesmo *cluster*. Os *clusters* 3, 4 e 6 também apresentaram bons resultados de erro mínimo, máximo e médio, conforme pode ser observado na Tabela 8.12. Já os piores resultados foram obtidos pelos *clusters* 2 e 8 com erro médio maior ou igual a 1.

Resíduos - Experimento V

Cluster	TT	Mínimo	Máximo	Médio
0	1	-	-	-
1	1	0.96	0.96	0.96
2	1	1.00	1.00	1.00
3	5	0.37	0.63	0.42
4	2	0.30	0.60	0.45
5	1	**0.17**	**0.17**	**0.17**
6	2	0.25	0.50	0.38
7	7	0.09	1.66	0.72
8	5	0.16	2.39	1.14
9	3	0.27	0.97	0.59
10	6	0.03	1.22	0.67

Tabela 8.12: Resultados de predição

Embora a Tabela 8.12 aponte erro máximo de predição de 2.39 para o *Cluster* 8, apenas um padrão desse *cluster* atingiu esse valor máximo, conforme pode ser observado no

gráfico de resíduos da Base B na Figura 8.12. Os demais padrões dos *clusters* não excederam o valor de 1.66 de erro de predição.

O gráfico da Figura 8.12 apresenta os resíduos das notas de testes preditas. De acordo com esse gráfico, há valores de resíduos próximos de zero para notas baixas (abaixo de 2.5), medianas (entre 2.5 e 4.0) e altas (acima de 4.0). O *Cluster* 3, por exemplo, contém quatro amostras de teste com notas 5.0 e uma com nota 4.0. A nota predita para esses padrões foi 4.63, o que está muito próximo de seus valores reais. Isso confirma a eficácia do modelo para predição das notas mais altas, o que é justificado pelos valores dos coeficientes de determinação R^2 e ${R_a}^2$, pelo alto valor de ISIM e pela quantidade de variáveis preditoras do modelo gerado para o *Cluster* 3 na Tabela 8.10.

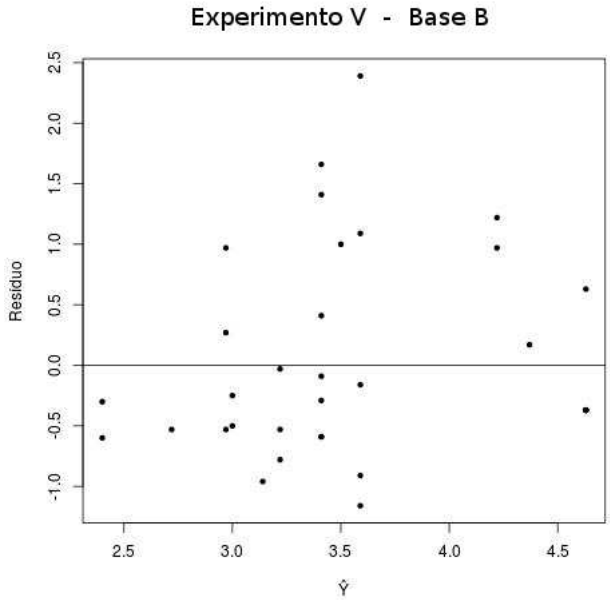

Figura 8.12: Análise de resíduos - Experimento V

A Tabela 8.13 apresenta os resultados de predição de notas para os *clusters* pequenos, isto é, para os *clusters* que contêm duas ou três amostras. O padrão de teste do *Cluster* 5, predito a partir das suas similaridades com as duas amostras de treino, obteve o menor erro de predição, que foi de 0.03. Esse resultado pode ser justificado pelo alto valor de similaridade SIM entre o padrão de treino Tr_1 e o padrão de teste Te_1 do *Cluster* 5 na Tabela 8.13, embora a similaridade entre Te_1 e Tr_2 seja menor.

As Tabelas 8.14 e 8.15 apresentam os resultados finais de predições de notas das amostras das bases A e B, respectivamente. Esses resultados mostram como a predição

Predição por similaridade

Cluster	TA	TT	SIM (Te_t, Tr_p)	Nota real	Nota predita	Erro
1	1	1	(Te_1, Tr_1): 0.7808	4.10	3.14	0.96
5	2	1	(Te_1, Tr_1): 0.8908	4.20	4.17	0.03
			(Te_1, Tr_2): 0.6399			

Tabela 8.13: Resultados de predição por similaridade - Base B

de notas por regressão linear evoluem com a utilização da técnica de *clustering* para retirada de *outliers* e para seleção de amostras e de características na formação de um modelo de regressão linear.

Análise de Resíduos - Base A

Experimento	Minimo	Maximo	Médio
I	0	5	2.8309
IV	**0.407**	**2.00**	**0.3326**

Tabela 8.14: Resultados finais - Base A

Na Tabela 8.14, observa-se que o valor do erro máximo de predição no Experimento IV cai em três pontos em relação ao Experimento I, em que apenas aplica-se um modelo de regressão linear que foi gerado sem critérios de seleção de amostras e de características. No Experimento IV para a Base A, retirando os *outliers* da base e dos *clusters*, selecionando amostras e características dos *clusters* e também predizendo as notas por *cluster*, alcançamos erro médio de 0.33 e erro máximo de 2.0 para a Base A.

Análise de Resíduos - Base B

Experimento	Minimo	Maximo	Médio
I	0.1367	4.5884	1.2578
V	**0.03**	**2.39**	**0.6809**

Tabela 8.15: Resultados finais - Base B

Para a Base B, conforme informa a Tabela 8.15, os valores de erro mínimo, máximo e médio também caíram em relação ao Experimento I, realizado apenas com a aplicação de um modelo de regressão linear.

Os resultados das Tabelas 8.14 e 8.15 confirmam, portanto, para uma base menor, como a Base A, e para uma base maior, como a Base B, que o *clustering* é uma estratégia eficaz para se obter melhores resultados de predição por regressão linear.

Para a Base B, com a aplicação do *clustering*, foram gerados *clusters* mais homogêneos com valor de ISIM acima de 86% e foi identificado apenas um *outlier* nessa base. Dessa forma, para amostras muito parecidas dentro de um *cluster*, mesmo com um grau de liberdade menor do modelo de regressão, foi possível predizer, para alguns *clusters*,

notas com pequena margem de erro apenas com a escolha e estimativa do coeficiente de intercepto na composição do modelo de regressão linear.

Na Base B, conseguimos acertar com erro máximo de 0.7, 66.6% das notas altas, 76% das notas medianas e nenhuma das quatro notas baixas existentes nessa base. Não conseguimos realizar predição com baixa taxa de erro para as amostras com notas baixas, pois na Base B havia poucas amostras representativas dessas notas e essas amostras ficaram distribuídas em diferentes *clusters*.

Conforme mostramos no Experimento IV, a seleção de amostras de treino, isto é, das amostras escolhidas para um professor corrigir, pode não ser aleatória e pode conter um número menor de amostras. Para isso, devem ser retirados dos *clusters*, quando estes são mais homogêneos, os *outliers* e escolher um número mínimo de amostras de cada exemplo diferente presente em um *cluster* para o professor atribuir nota.

Alcançamos um erro médio de predição de 0.68 para 33 amostras de teste. Para a Base B, apenas 18% dessas amostras ultrapassaram a taxa de erro de predição de 1.0 e apenas um padrão foi predito com erro acima de 2.0. Esses resultados confirmam, portanto, a eficácia do nosso método para melhorar resultados de predição de notas de exercícios de programação por modelos de regressão linear.

8.4 Discussões

O nosso modelo de avaliação semi-automática de exercícios de programação demonstrou precisão de 80% na predição de notas de 50% de uma base de 40 programas em Linguagem C. Embora contenha um número de amostras pequeno, essa base, obtida em condições controladas, possui um conjunto representativo da variabilidade de soluções para uma questão de prova. Para bases maiores, com um maior número de amostras de testes os resultados de aplicação do método podem apresentar melhores resultados.

O nosso modelo de predição de notas de exercícios de programação é inovador em relação às principais propostas de avaliação semi-automática no Brasil (MOREIRA; FAVERO, 2009) e no mundo (XU; CHEE, 2003; NAUDE; GREYLING; VOGTS, 2010) pelas seguintes razões:

1. Utilizamos uma representação numérica de programas de computador para processamento automático a partir das palavras reservadas e operadores semânticos que controlam estruturas condicionais e estruturas de repetição em programas escritos em Linguagem C.
2. A normalização que fizemos da base de exercícios de programação não compromete a integridade de informações relevantes do código-fonte de um programa de computador.
3. Tratamos a variabilidade da base de exercícios de programação agrupando os padrões semelhantes por técnicas de *clustering*
4. Realizamos a predição de notas por partes, isto é, por *clusters*, com maiores possibilidades de sucesso uma vez que os padrões já são semelhantes entre si.
5. Não temos soluções-modelo para predizer notas, mas, identificamos as características que as descrevem e, a partir dessas características, selecionamos as amostras a serem pontuadas por um professor e que são utilizadas para geração do nosso modelo preditor.
6. Conseguimos fazer predições precisas para uma base de poucas amostras porque, para cada *cluster*, selecionamos características relevantes e, com base nelas, escolhemos um conjunto pequeno de amostras representativas dos padrões similares de cada *cluster*.
7. O modelo realizou com sucesso predições de notas altas, médias e baixas na maioria das vezes.

Em relação ao Item 1, Naude, Greyling e Vogts (2010) e Xu e Chee (2003) utilizam representação de programas por AST. Essa representação torna-se complexa quando se leva em consideração a variabilidade de soluções desses programas. Para tratar esse problema da variabilidade, Naude, Greyling e Vogts (2010) propõem realizar normalizações nas variáveis, nas expressões lógicas e nas estruturas de controle condicional e de repetição. Porém, o esforço para fazer essas normalizações, principalmente das variáveis dos programas, é alto para programas extensos.

Já o trabalho de Moreira e Favero (2009), assim como o nosso modelo, utiliza uma representação numérica dos programas avaliados. Eles utilizam métricas de

software como a métrica de esforço de *Halstead* (HALSTEAD, 1977), a métrica de complexidade de *McCabe* (MCCABE, 1976), o número de linhas de código, o número de declarações e a quantidade de palavras reservadas de uma linguagem de programação. Porém, ao contrário de Moreira e Favero (2009), nós não resumimos as palavras reservadas de uma linguagem de programação a um único número, mas quantificamos a frequência de ocorrência de cada palavra reservada nos programas que desejamos avaliar. Isso porque entendemos que as palavras reservadas são as variáveis que nos dão mais informações sobre como um programa de computador foi construído assim como os operadores utilizados em estruturas de controle condicional e de repetição.

Com essa abordagem de quantificar palavras reservadas e operadores de uma linguagem, ficamos, porém, reféns do número de amostras da base, que muitas vezes não cobrem todas essas características, o que dificulta a predição de notas por um modelo de regressão linear. Podemos, no entanto, fazer uma seleção dessas palavras reservadas e dos operadores que mais explicam a atribuição de notas por um professor. Reduzindo esse espaço de características com essa seleção, tornamos possível e bem-sucedida a predição de notas por regressão linear.

Em relação ao Item 2, Naude, Greyling e Vogts (2010) propõem tratar a variabilidade dos exercícios de programação normalizando os códigos-fontes dos programas a uma forma comum. Nesse caso, estruturas *if* aninhadas e estruturas de escolha de opções (*switch*, em Linguagem C) seriam reduzidas a estruturas *if* mais simples. Da mesma forma, as diferentes formas de estruturas de controle de repetição como o *for* (para), o *while* (enquanto) e o *do-while* (faça-enquanto) seriam reduzidas à forma *while*.

Nessa abordagem de normalização de Naude, Greyling e Vogts (2010), porém, há possibilidades de haver perda de informações relevantes para a caracterização de um programa de computador. Para um professor, por exemplo, pode ser interessante saber se um aluno sabe usar as diferentes estruturas de controle de repetição. No caso das estruturas *if* aninhadas, desenvolveu uma solução mais eficiente o aluno que utilizou uma estrutura *if* do que o aluno que utilizou três estruturas *if*. Em nossa base de exercícios, por exemplo, o professor deu nota 5.0 a um aluno e nota 4.75 ao outro porque o primeiro utilizou menos estruturas *if*. Se o programa do primeiro aluno fosse

normalizado, segundo a proposta de Naude, Greyling e Vogts (2010), a estrutura *if* aninhada seria normalizada em duas estruturas *if*. Os dois alunos, dessa forma, seriam pontuados com nota 4.75, o que seria injusto para o aluno que desenvolveu uma solução mais eficiente.

As normalizações que realizamos em nossa base de exercícios de programação reduzem algumas ambiguidades de código e o tamanho do programa, uma vez que comentários e *strings* são retirados. Mas não perdemos informação relevante porque as instruções e símbolos normalizados apenas mudam de nome e as *strings* e comentários, embora retirados dos programas, são contados.

Em relação aos Itens 3 e 4, o trabalho de Naude, Greyling e Vogts (2010) trata a variabilidade por normalizações, a princípio, apenas normalizando as variáveis. Nós, entretanto, tratamos a diversidade de soluções separando-as em *clusters* que agrupam os padrões mais semelhantes entre si e realizamos a predição de notas por partes, isto é, por *clusters* tratando, assim, a variabilidade das soluções.

Em relação ao Item 5, os trabalhos de Naude, Greyling e Vogts (2010) e de Moreira e Favero (2009) realizam predições de notas a partir de uma base de treino por eles fornecida. O nosso modelo, porém, seleciona um conjunto mínimo de padrões para formar essa base de treino com base nas características mais relevantes para, dinamicamente, o professor atribuir notas. Temos, portanto, maiores possibilidades de sucesso porque informamos um número mínimo de amostras mais prováveis de representarem os gabaritos dos *clusters*.

Em relação ao Item 6, o nosso modelo, assim como os trabalhos de Naude, Greyling e Vogts (2010) e de Moreira e Favero (2009), é baseado em um modelo de regressão linear múltipla. Mas, mostramos que nosso modelo é menos dependente do número de amostras, uma vez que, utilizando a estratégia de selecionar as características mais relevantes para o modelo de predição de cada *cluster*, alcançamos bons resultados de predição.

Quanto ao Item 7, ao contrário do trabalho de Naude, Greyling e Vogts (2010) que apresentou menor índice de acerto para as notas mais baixas, o nosso modelo, para as notas que foram preditas, acertou com erro máximo de 0.5 as notas baixas, médias e altas com índices de 71,4%, 83% e 100%, respectivamente, conforme o gráfico da

Figura 8.13 relativo à Base A.

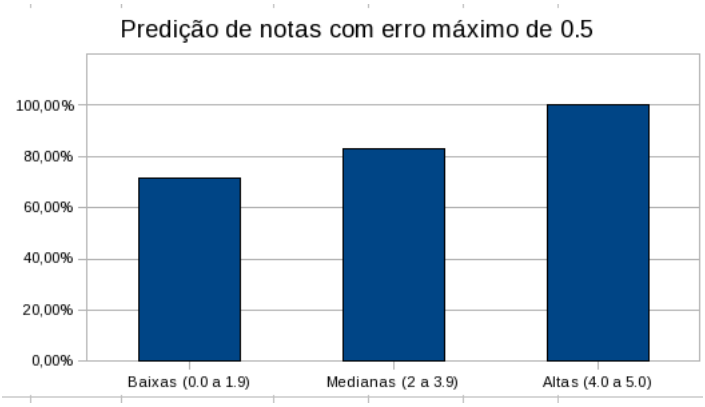

Figura 8.13: Resultados de predição de notas baixas, médias e altas

8.5 Melhoramentos

Este trabalho é um passo inicial nas pesquisas que estamos desenvolvendo para avaliação semi-automática de exercícios de programação. Destacamos nesta seção até onde chegamos, o que pode ser melhorado nos procedimentos de experimentação e como avançar a partir do que já foi desenvolvido.

O algoritmo de avaliação semi-automática de exercícios de programação apresentado no Capítulo 6 foi implementado praticamente em todos os seus passos, faltando apenas retirar a aleatoriedade na escolha das amostras de cada *cluster* bem como retirar, quando houver, os *outliers* de cada *cluster*. A implementação desses últimos passos possibilitará, conforme mostramos no Experimento IV de forma manual, melhoramentos na geração dos modelos de regressão para cada *cluster*. Isso será possível porque haverá uma melhor seleção das amostras de treino, isto é, das amostras escolhidas para o professor atribuir notas, para criar um modelo de regressão mais representativo dos exercícios reunidos em um mesmo *cluster*.

O primeiro ponto importante a ser considerado na realização dos experimentos de teste do método é o conjunto de características escolhido para representar o domínio da programação em Linguagem C. Neste trabalho, nós selecionamos sessenta características para representar esse domínio. Mas, para uma melhor seleção de características podemos também considerar, no conjunto dessas sessenta

características, a possibilidade de mesclar as características mais relacionadas entre si em novas variáveis, reduzindo assim o número de variáveis representativas de um domínio. Essa síntese poderia ser realizada através da análise fatorial, que é uma técnica estatística que pressupõe que um conjunto de variáveis observáveis pode ser explicado por um número menor de variáveis hipotéticas ou fatores (HAIR; TATHAM; BLACK, 1998). Esses fatores mapeariam, nesse caso, a causa do relacionamento entre as variáveis observáveis. A tarefa da análise fatorial seria, dessa forma, descobrir quais são esses fatores $F_i(i = 1,2,3,\ldots,n)$ para representar as variáveis independentes do modelo de regressão linear formado a partir da seguinte relação linear:

$$Y_i = a_0 + a_1 \cdot F_1 + a_2 \cdot F_1 + \ldots + a_n \cdot F_n$$

Na aplicação do método de tese foram escolhidas, dentre as características mais descritivas de cada *cluster*, dois terços do número de amostras de treino. Através do algoritmo *stepwise*, foi realizada uma nova seleção de características contendo apenas as características mais relacionadas à variável dependente nota, segundo o algoritmo *stepwise*. Em alguns *clusters*, porém, conforme a Tabela 8.10, o algoritmo *stepwise* não selecionou variáveis para compor o modelo de regressão, sendo as notas das amostras desses *clusters* estimadas a partir do coeficiente de intercepto β_0 (Tabela 8.11).

No caso dos *clusters* em que as notas das amostras de testes foram estimadas a partir do coeficiente de intercepto, deve-se avaliar se realmente seria necessário criar um modelo de regressão para estimar as notas.

A Tabela 8.16 apresenta as características mais descritivas selecionadas para cada *cluster* formado no Experimento V antes e depois da aplicação do algoritmo *stepwise*.

Características descritivas

Cluster	Antes do *stepwise*	Depois do *stepwise*
2	*opatrib, comentario*	*comentario*
3	*opatrib, opmais, cif, opmaior, endereco, cimprimir*	*opatrib, opmais, cif, opmaior, endereco, cimprimir*
4	*opmaior, opmult, opmais*	-
6	*cimprimir, opmaior, cif*	-
7	*endereco, opmaior, opmult, cimprimir, opmais, opatrib, cif, opand*	-
8	*opatrib, endereco, cimprimir, opmais, opmult, opmaior*	*endereco*
9	*cimprimir, endereco, opmult, opmais*	-
10	*opmaior, cimprimir, opatrib, opmult, cif, endereco, opmais*	*opmaior, cif, endereco*

Tabela 8.16: Variáveis descritivas dos *clusters*

De acordo com a Tabela 8.16 e com a Tabela 8.10, os *clusters* 3 e 10 consideraram um maior número de características selecionadas para compor o modelo de regressão

após aplicação do algoritmo *stepwise*. Por outro lado, para os *clusters* 4, 6, 7 e 9, nenhuma variável foi selecionada pelo *stepwise* para compor o modelo de regressão, sendo as notas das amostras de testes desses *clusters* estimadas apenas pelo valor do coeficiente de intercepto, conforme Tabela 8.11. No caso desses *clusters*, não foi gerado um modelo de regressão para estimação das notas. Dessa forma, deve-se realizar uma pesquisa para compreender por que, das características representantes desses *clusters*, não foi possível selecionar qualquer uma delas, mesmo que muitas dessas mesmas características tenham sido selecionadas para os *clusters* 3 e 10.

No estado atual das pesquisas, não conseguimos reconhecer sem aplicar o *stepwise* por que não foi possível selecionar um conjunto de características para alguns *clusters*. Por isso, torna-se necessário, desenvolver melhores estratégias para criar um corpo de características verdadeiramente mais descritivo de cada *cluster* ou descobrir através das próprias informações de *clustering* que não é possível ou não é necessário criar um modelo de regressão para predição de notas.

Em nosso método, a identificação das características mais descritivas de cada *cluster* foi realizada pelos algoritmos de *clustering*. O *ranking* das características mais descritivas foi formado em ordem decrescente do valor numérico dessas características. Dessa forma, as características que tinham maior frequência de ocorrência foram consideradas as mais relevantes. Esse critério, porém, em alguns casos, pode não refletir de fato as características que um professor considera relevantes na avaliação de um programa. Um exemplo disso é o professor considerar mais importante em um exercício o uso de uma estrutura de controle de repetição *for* do que dez operações de atribuição.

Na Tabela 8.17, por exemplo, observamos que as notas das amostras de testes dos *clusters* 4, 6, 7 e 9 poderiam ser preditas com melhores resultados calculando-se apenas a média das notas das amostras de treino do que estimando essas notas a partir de coeficientes de intercepto. Desse modo, sabendo o que caracteriza os padrões desses *clusters* de forma que se estimem notas pela média de notas dos padrões de treino, podemos realizar boas predições sem muito esforço computacional. Uma dessas características pode ser o alto valor de ISIM de um *cluster* como, por exemplo, os valores de ISIM dos *clusters* da Tabela 8.17.

Concluindo, para gerar modelos de regressão linear com maior confiança, sugere-se

Resúltados de predição - Experimento V

		Coeficiente de Intercepto			Média		
Clusters	ISIM	Mínimo	Máximo	Médio	Mínimo	Maximo	Médio
4	0.917	0.3	0.6	0.45	**0.01**	**0.31**	**0.16**
6	0.877	0.25	0.50	0.38	**0.21**	**0.46**	**0.335**
7	0.937	**0.09**	1.66	**0.72**	0.11	**1.14**	0.82
9	0.891	0.27	0.97	0.59	**0.058**	**0.858**	**0.3**

Tabela 8.17: Predição de notas por coeficiente de intercepto e por média

que se tenha, para cada característica, pelo menos quatro amostras no conjunto de treino utilizado para formar o modelo (HAIR; TATHAM; BLACK, 1998) antes da aplicação do algoritmo *stepwise*. Além disso, deve-se verificar se de fato as características do modelo são independentes. Para isso, deve-se verificar as correlações entre as variáveis que serão utilizadas para compor um modelo de regressão linear, descartando variáveis altamente correlacionadas com outras variáveis do modelo. Finalmente, deve-se analisar cuidadosamente as informações estatísticas dos modelos de regressão, como os valores de R^2, R_a e GL, para avaliar se realmente a aplicação de modelos de regressão linear é a solução mais viável para realizar a predição de notas em alguns *clusters*.

8.6 Extensões

O método de avaliação semi-automática de exercícios pode ser extendido a outras funcionalidades de apoio à gestão de aprendizagem de programação. Entre essas extensões, apresentamos, nas subseções a seguir, a identificação de exercícios que divergem da especificação do professor e a detecção automática de indícios de plágios nos códigos-fontes submetidos por alunos.

8.6.1 Identificação de soluções divergentes

Quando as soluções de exercícios desenvolvidas por alunos são reunidas em *clusters* na etapa *Clusterizar* do algoritmo de avaliação semi-automática apresentado no Capítulo 5, podemos identificar *outliers* nos *clusters* formados com apenas uma amostra. Esses *outliers* podem ser considerados soluções que divergem do que foi solicitado em atividades especificadas por professores ou soluções inéditas, uma vez que não foram

encontradas outras soluções semelhantes a elas. A Figura 8.14, por exemplo, apresenta as amostras de soluções da Base A reunidas em *clusters*, segundo as similaridades entre suas características. Esses *clusters* são identificados por valores de 0 a 10. Observa-se que os *clusters* numerados de 0 a 4 são formados por amostras que não se assemelham às demais amostras da coleção. Consideramos, portanto, as amostras desses *clusters* como *outliers*, isto é, como soluções que divergem da solução esperada segundo a especificação do professor.

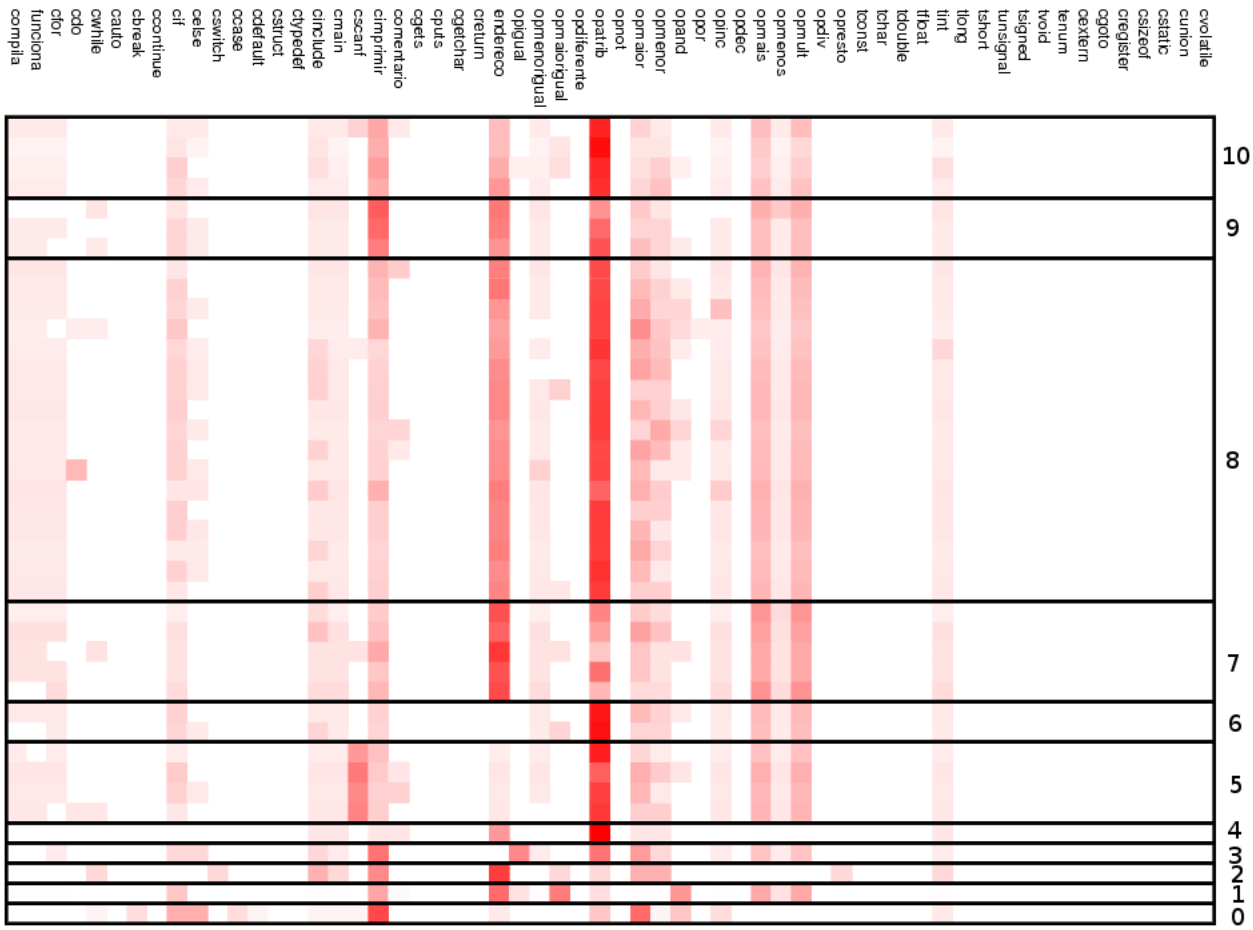

Figura 8.14: Soluções de exercícios reunidas em *clusters*

Analisando as amostras de programas em Linguagem C dos *clusters* com apenas uma amostra, observamos que quatro delas sequer compilam, conforme indica a primeira coluna do gráfico de *clustering* identificada pela característica *compila*.

Outras informações observadas nas amostras do gráfico de *clustering* da Figura 8.14 foram as seguintes:

1. A solução do *Cluster 0*, por exemplo, contém excesso de instruções, não compila

e não corresponde ao enunciado da questão. A nota atribuída pelo professor para essa solução foi 0.25 de 5.0.

2. A solução do *Cluster 1* contém excesso de instruções reunidas em um arranjo para fazer o programa funcionar de qualquer maneira, o que demonstra que os processos de desenvolvimento da solução foram forçados e não planejados adequadamente. O professor atribuiu para essa questão nota 0.0 de 5.0.
3. A solução do *Cluster 2* não compila e foi iniciada apenas com a declaração de variáveis e com a entrada de dados conforme a Figura 8.15. Em seguida foi colocada nessa solução uma estrutura *while* (*enquanto*) vazia e em *loop*. Isso indica possivelmente incompreensão do aluno no desenvolvimento da solução para resolver o problema especificado pelo professor. A nota atribuída pelo professor para essa solução foi 0.0 de 5.0.
4. A solução do *Cluster 3* não compila, está incompleta, contém estrutura *for* (*para*) mal construída, confunde operador de atribuição (=) com operador de igualdade (==) e diverge da especificação, conforme a Figura 8.16. A nota atribuída pelo professor a essa solução foi 0.25 de 5.0.
5. A solução do *Cluster 4* foi apenas iniciada e não compila. A nota atribuída pelo professor a essa solução foi 0.0 de 5.0.

```
#include <stdio.h>
#include <math.h>

main ()
{
int gp,gn,vf,vc,e,camp,vice,times=0;
printf("\ndigite numero de gols positivos:\n ");
printf("\ndigite numero de gols negativos:\n ");
printf("\ndigite numero de vitorias fora de casa:\n ");
printf("\ndigite numero de vitorias em casa:\n ");
printf("\ndigite numero de empates:\n ");
scanf("%d%d%d%d%d%d", &gp,%&gn&vf,&vc,&e);
while(times>=10)
{
switch
}
```

Figura 8.15: Exemplo de solução incompleta

A análise dos clusters 0 a 4 da Figura 8.14 nos leva a sugerir, portanto, que as amostras *outliers*, sendo soluções divergentes da especificação do professor, muito possivelmente

foram pontuadas com notas muito baixas, conforme os exemplos apresentados de cada *cluster* contendo apenas uma amostra.

```
        #include<stdio.h>
        #include<math.h>

        main()
{
        int p, gp, gn, vf, vc, e, i=1, n=0, n2=0, v=0, v2=0, t;

        for (i=1,i<=10,i++);
{
        printf("digite o número do time: \n");
        scanf("%d", "&t");
        printf("digite o GP: \n");
        scanf("%d", "&gp");
        printf("digite o GN: \n");
        scanf("%d", "&gn");
        printf("digite o VF: \n");
        scanf("%d", "&vf");
        printf("digite o VC: \n");
        scanf("%d", "&vc");
        printf("digite E: \n");
        scanf("%d", "&e");
        p=5*gp-gn+3*vf+2*vc+e;
{       if (p>n)
            (n==p);
        else (n>p>n2)
           (n2==p);
}{      if (n==p)
            (t==v);
        else (n2==p);
            (t==v2);
}}

}
```

Figura 8.16: Exemplo de solução divergente da especificação do professor

8.6.2 Detecção de indícios plágios

Uma outra possibilidade a ser realizada a partir do nosso algoritmo de avaliação semi-automática de exercícios de programação é a detecção de indícios de plágio em programas de computador desenvolvidos por alunos. Através da matriz de similaridade gerada pelo algoritmo de avaliação semi-automática, é possível identificar quão semelhantes são as soluções desenvolvidas por alunos.

Os índices de similaridade calculados pela medida de similaridade *cosseno* variam de 0 a 1. O valor 0 indica dissimilaridade e o valor 1, similaridade total. Um valor de similaridade de 0.80, por exemplo, indica que duas soluções possuem 80% de semelhança entre si.

A detecção de indícios de plágios em programas de computador pode ser menos

complexa do que em outros textos porque os programas são construídos segundo uma ordem lógica e em uma linguagem formal. Dessa forma aquele aluno que não conseguiu desenvolver uma solução e tenta copiá-la de um outro aluno, por não compreender a linguagem e a sequência lógica do programa, fará modificações apenas nas partes que compreende para não comprometer o funcionamento do programa. Essas partes podem ser a declaração de nomes de variáveis, as *strings* e os comentários. Como o nosso avaliador semi-automático retira as *strings* e os comentários e não considera o uso de variáveis na representação de programas, é possível identificar as soluções muito semelhantes pelos vetores que representam as soluções desenvolvidas pelos alunos. Isso pode ser realizado calculando os índices de similaridades entre esses vetores.

A Figura 8.17 apresenta a matriz de similaridade das soluções do *Cluster* 7 da Base B, que é o *cluster* com maior ISIM (Tabela 8.10). Aparecem em destaque na figura os índices de similaridade que indicam maior possibilidade de plágio, isto é, índices entre 0.98 e 1. Na Figura 8.18, mostramos os vetores que representam as soluções do *Cluster* 7 da Base B identificadas por l_i ($i = 1, 2, 3 \ldots 21$).

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	l1	l2	l3	l4	l5	l6	l7	l8	l9	l10	l11	l12	l13	l14	l15	l16	l17	l18	l19	l20	l21
l1	1	0,882	0,945	0,907	0,966	0,817	0,887	0,910	0,915	0,912	0,970	0,926	0,914	0,930	0,947	0,922	0,867	0,867	0,909	0,977	0,919
l2	0,882	1	0,921	0,956	0,923	0,945	0,941	0,960	0,966	0,954	0,864	0,888	0,951	0,916	0,931	0,900	0,951	0,951	0,955	0,891	0,906
l3	0,945	0,921	1	0,958	0,938	0,889	0,909	0,975	0,944	0,966	0,904	0,877	0,974	0,969	0,939	0,939	0,930	0,930	0,959	0,925	0,949
l4	0,907	0,956	0,958	1	0,942	0,923	0,951	0,983	0,952	0,998	0,886	0,888	0,970	0,943	0,908	0,915	0,963	0,963	0,999	0,912	0,929
l5	0,966	0,923	0,938	0,942	1	0,897	0,953	0,936	0,941	0,942	0,957	0,966	0,932	0,955	0,942	0,952	0,931	0,931	0,944	0,960	0,956
l6	0,817	0,945	0,889	0,923	0,897	1	0,975	0,944	0,962	0,922	0,841	0,894	0,931	0,925	0,881	0,913	0,985	0,985	0,924	0,849	0,930
l7	0,887	0,941	0,909	0,951	0,953	0,975	1	0,951	0,964	0,950	0,903	0,936	0,947	0,951	0,891	0,952	0,985	0,985	0,953	0,901	0,958
l8	0,910	0,960	0,975	0,983	0,936	0,944	0,951	1	0,959	0,987	0,887	0,879	0,989	0,953	0,925	0,924	0,975	0,975	0,984	0,917	0,940
l9	0,915	0,966	0,944	0,952	0,941	0,962	0,964	0,959	1	0,954	0,921	0,940	0,953	0,957	0,953	0,944	0,966	0,966	0,953	0,927	0,953
l10	0,912	0,954	0,966	0,998	0,942	0,922	0,950	0,987	0,954	1	0,887	0,885	0,975	0,947	0,911	0,920	0,964	0,964	0,999	0,913	0,935
l11	0,970	0,864	0,904	0,886	0,957	0,841	0,903	0,887	0,921	0,887	1	0,971	0,875	0,927	0,932	0,921	0,879	0,879	0,888	0,989	0,929
l12	0,926	0,888	0,877	0,888	0,966	0,894	0,936	0,879	0,940	0,885	0,971	1	0,867	0,928	0,928	0,933	0,906	0,906	0,889	0,953	0,941
l13	0,914	0,951	0,974	0,970	0,932	0,931	0,947	0,989	0,953	0,975	0,875	0,867	1	0,946	0,915	0,935	0,962	0,962	0,971	0,905	0,931
l14	0,930	0,916	0,969	0,943	0,955	0,925	0,951	0,953	0,957	0,947	0,927	0,928	0,946	1	0,922	0,979	0,949	0,949	0,941	0,917	0,996
l15	0,947	0,931	0,939	0,908	0,942	0,881	0,891	0,925	0,953	0,911	0,932	0,928	0,915	0,922	1	0,900	0,896	0,896	0,909	0,950	0,910
l16	0,922	0,900	0,939	0,915	0,952	0,913	0,952	0,924	0,944	0,920	0,921	0,933	0,935	0,979	0,900	1	0,933	0,933	0,916	0,903	0,981
l17	0,867	0,951	0,930	0,963	0,931	0,985	0,985	0,975	0,966	0,964	0,879	0,906	0,962	0,949	0,896	0,933	1	1	0,964	0,892	0,949
l18	0,867	0,951	0,930	0,963	0,931	0,985	0,985	0,975	0,966	0,964	0,879	0,906	0,962	0,949	0,896	0,933	1	1	0,964	0,892	0,949
l19	0,909	0,955	0,959	0,999	0,944	0,924	0,953	0,984	0,953	0,999	0,888	0,889	0,971	0,941	0,909	0,916	0,964	0,964	1	0,914	0,931
l20	0,977	0,891	0,925	0,912	0,960	0,849	0,901	0,917	0,927	0,913	0,989	0,953	0,905	0,917	0,950	0,903	0,892	0,892	0,914	1	0,910
l21	0,919	0,906	0,949	0,929	0,956	0,930	0,958	0,940	0,953	0,935	0,929	0,941	0,931	0,996	0,910	0,981	0,949	0,949	0,931	0,910	1

Figura 8.17: Identificação de indícios plágios por índices de similaridades

Na Figura 8.17, em amarelo, estão os vetores l_{17} e l_{18} que têm similaridade 1, isto é, de 100%, conforme a matriz de similaridade da Figura 8.16. Observa-se que as duas soluções são idênticas, o que pode ser caracterizado como plágio.

Os vetores l_{14} e l_{21}, marcados de rosa na Figura 8.18, possuem 99.8% de similaridade conforme a matriz de similaridade da Figura 8.17. Nesse caso, a solução l_{14} contém algumas pequenas diferenças em relação a l_{21}, não compila e não funciona, conforme os valores iguais a 0 nas primeiras duas colunas do vetor de l_{14}. Isso sugere que a cópia

foi mal sucedida, uma vez que o programa não funcionou. O mesmo acontece com a solução l_{10} em relação à solução l_{19}, ambas marcadas de azul. Conforme pode ser observado, solução l_{10} é praticamente idêntica à solução l_{19}, o que indica fortes indícios de plágio.

l9	1	1	0	0	0	0	0	0	6	0	0	0	0	0	0	2	1	0	9	1	0	0	0	1	15	0	0	0	0	7	0	10	6	6	0	1	0	9	3	9
l10	1	1	0	0	0	0	0	0	9	0	0	0	0	0	0	2	1	0	10	0	0	0	0	1	15	0	0	0	0	3	0	14	2	3	0	0	0	9	3	10
l11	1	1	0	0	0	0	0	0	0	2	2	0	0	0	0	0	1	1	0	6	0	0	0	0	1	15	0	0	0	0	6	0	4	1	1	0	0	0	9	3	9
l12	1	1	0	0	0	0	0	0	0	4	4	0	0	0	0	0	1	1	0	4	1	0	0	0	1	15	0	0	0	0	9	0	6	3	3	0	0	0	9	3	9
l13	1	1	0	0	0	0	0	0	0	6	0	0	0	0	0	0	2	1	3	13	3	0	0	0	1	15	0	0	0	0	3	0	14	2	6	0	0	0	9	3	10
l14	0	0	0	0	0	0	0	0	0	6	3	0	0	0	0	0	2	1	0	14	0	0	0	0	1	15	0	0	0	0	10	0	11	2	3	0	0	0	9	3	10
l15	1	1	0	0	0	0	0	0	0	4	4	0	0	0	0	0	2	1	0	9	0	0	0	0	1	15	0	0	3	0	3	0	6	6	5	2	0	0	9	3	9
l16	1	1	0	0	0	0	0	0	0	6	3	0	0	0	0	0	2	1	3	13	3	0	0	0	1	15	0	0	0	0	12	0	11	2	3	0	0	0	12	0	10
l17	1	1	0	2	0	0	0	0	0	6	0	0	0	0	0	0	2	1	0	9	0	0	0	0	1	15	0	0	0	0	8	0	16	2	8	0	0	0	9	3	10
l18	1	1	0	2	0	0	0	0	0	6	0	0	0	0	0	0	2	1	0	9	0	0	0	0	1	15	0	0	0	0	8	0	16	2	8	0	0	0	9	3	10
l19	1	1	0	0	0	0	0	0	0	9	0	0	0	0	0	0	2	1	0	9	0	0	0	0	1	15	0	0	0	0	3	0	14	2	3	0	0	0	9	3	10
l20	1	1	0	0	0	0	0	0	0	2	2	0	0	0	0	0	1	1	0	6	0	0	0	0	1	15	0	0	0	0	3	0	5	1	2	0	0	0	9	3	9
l21	1	1	0	0	0	0	0	0	0	6	3	0	0	0	0	0	2	1	0	13	0	0	0	0	1	15	0	0	0	0	12	0	11	2	3	0	0	0	9	3	10

Figura 8.18: Exemplos de exercícios com indícios de plagio

A Figura 8.19 apresenta os códigos dos programas l_{14} e l_{21}. Conforme previsto, no programa l_{14}, que não compila, o autor da solução fez alterações apenas na declaração de variáveis e nas *strings*, o que sugere fortes indícios de plágio.

Programa l21

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   int x=0, y1=0, y2=0, P, GP, GN, VF, VC, E;
   int T1=0, T2=0, T3=0;

    printf("TIME1: digite o numero de GP, GN, VF, VC, E ");
    scanf("%d %d %d %d %d", &GP, &GN, &VF, &VC, &E);
    P = 5*GP - GN + 3*VF + 3*VC + E;
    T1 = P;

    printf("TIME2: digite o numero de GP, GN, VF, VC, E ");
    scanf("%d %d %d %d %d", &GP, &GN, &VF, &VC, &E);
    P = 5*GP - GN + 3*VF + 3*VC + E;
    T2 = P;

    printf("TIME3: digite o numero de GP, GN, VF, VC, E ");
    scanf("%d %d %d %d %d", &GP, &GN, &VF, &VC, &E);
    P = 5*GP - GN + 3*VF + 3*VC + E;
    T3 = P;

    printf("Time1:%d  , Time2:%d , Time3:%d \n", T1, T2, T3);

    if((T1>T2) && (T1>T3))
    {
        printf(" campeao: T1");
        if(T2>T3){
                printf(" vice-campeao: T2");
        }
        else{
                printf("vice-campeao: T3");
            }
```

Programa l14

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int time1, time2, time3, P=0, GP, GN, VF, VC, E, T1=0, T2=0, T3=0;

  printf("time1:Digite o numero de:\nGP:\nGN:\nVF:\nVC:\nE:\n\n");
  scanf("%d %d %d %d %d", &GP,&GN,&VF,&VC,&E);
  P = 5*GP - GN + 3*VF + 2*VC + E;
  T1=P;

  printf("time2:Digite o numero de:\nGP:\nGN:\nVF:\nVC:\nE:\n\n");
  scanf("%d %d %d %d %d", &GP,&GN,&VF,&VC,&E);
  P = 5*GP - GN + 3*VF + 2*VC + E;
  T2=P;

  printf("time3:Digite o numero de:\nGP:\nGN:\nVF:\nVC:\nE:\n\n");
  scanf("%d %d %d %d %d", &GP,&GN,&VF,&VC,&E);
  P = 5*GP - GN + 3*VF + 2*VC + E;
  T3=P;

  printf("time1;%d  ,time2:%d   ,time3:%d", T1,T2,T3);

  if((T1>T2) &&(T1>T3)){
            printf("campeao:T1");
            if(T2>T3){
                      printf("o vice campeao:T2");
                      }
                      else{
                           printf("vice campeao:T3");
```

Figura 8.19: Exemplos de plágio

8.7 Conclusão

Neste capítulo apresentamos a aplicação do nosso método de forma gradativa e completa. Nos experimentos parciais, mostramos que um número pequeno de amostras e a falta de critério para selecionar um conjunto representativo de amostras afetam os resultados de predição por modelos de regressão linear. Por outro lado, utilizando algoritmos de *clustering* para auxiliar a remoção de *outliers*, a seleção de amostras mais representativas e a predição de notas por *cluster*, podemos alcançar melhores resultados na aplicação de um modelo de regressão linear múltipla.

No experimento de tese, mostramos que selecionando também as características ou variáveis de um modelo de regressão para cada *cluster*, isto é, identificando as características descritivas de cada *cluster*, podemos utilizar um conjunto de treino menor. Reduzir o conjunto de treino implica em reduzir o número de exercícios a serem corrigidos por professores. Essa foi uma vantagem do nosso método porque podemos reduzir em até 70% o esforço de correção manual de um professor.

Uma outra importante vantagem do nosso método é que, mesmo com poucas amostras reunidas em *clusters*, conseguimos reduzir o erro de predição tanto em uma base pequena quanto em uma base grande de amostras. Isso porque ter *clusters* mais homogêneos pode reduzir a dependência de uma alta capacidade de generalização de um modelo de predição de notas.

Nos experimentos parciais, os modelos gerados indicaram menor erro de predição para notas medianas e maior erro para as notas altas e baixas. Mas os resultados de aplicação do método de tese mostram resultados com erro de até 10% em 71% das notas baixas, 83% das notas medianas e 100% das notas altas.

Os resultados demonstram, portanto, que combinando a técnica de aprendizagem não-supervisionada de *clustering* e de aprendizagem supervisionada de regressão linear, podemos aumentar a confiança de predição de um modelo e também reduzir a dependência de um número alto de amostras, desde que formem-se *clusters* mais homogêneos de forma que seja possível selecionar amostras de testes mais semelhantes às amostras de treino.

A representação vetorial e o *clustering*, como partes do processo de avaliação

semi-automática de exercícios, podem também ser utilizados para reconhecimento de soluções submetidas que sejam divergentes da especificação do professor e para a detecção automática de indícios de plágios. Essas extensões do método de tese possibilitam ao professor ter uma visão dos processos de desenvolvimento de programas em Linguagem C bem como compará-los para avaliar a qualidade das soluções e analisar suspeitas de plágios.

Para dar continuidade às pesquisas, apresentamos na Seção 8.5 algumas considerações sobre número de amostras e de características que podem ajudar a melhorar os modelos de predição por regressão linear ou nos levar a utilizar outras técnicas que ofereçam mais eficácia e menos esforço na predição de notas.

Capítulo 9

Experimentos e Resultados do NAF

A estratégia do nosso sistema de recomendação é tratar a recomendação de atividades como uma tarefa de classificação *multilabel*, que consiste em associar uma ou mais classes a uma instância (ZHANG; ZHOU, 2007). Para isso, o sistema analisa perfis multidimensionais de alunos e, com base em um histórico de recomendações feitas manualmente, recomenda-lhes classes de atividades conforme as similaridades apontadas pelas variáveis dos perfis que já receberam recomendações.

Para avaliar essa estratégia de recomendação de classes de atividades, nós obtivemos, a partir de atividades de programação resolvidas por alunos do curso de Engenharia Elétrica da Universidade Federal do Espírito Santo, as indicações realizadas por um professor para uma turma de programação em Linguagem C. Dessas recomendações uma base de exercícios foi gerada, a Base *ds-FAR* (*Formative Assessment Recommendation* - Recomendações de Avaliação Formativa), composta por 1784 amostras de programas em Linguagem C. Esses programas são soluções desenvolvidas por cerca de 50 alunos para 39 atividades propostas por um professor e que foram distribuídas em sete tarefas e uma prova.

Os resultados obtidos, sob diferentes métricas de avaliação, confirmam a eficácia do algoritmo ML-kNN em recomendar classes de atividades. Os experimentos indicam que o ML-kNN imitou corretamente as decisões humanas na recomendação de classes de atividades em 96% das vezes com 89% de precisão média.

Este capítulo está organizado conforme a ordem a seguir. Na Seção 9.1, explicamos

a base experimental *ds-FAR*. Na Seção 9.2, descrevemos os procedimentos de experimentação realizados. Na Seção 9.3, apresentamos os resultados, os novos significados das métricas de classificação *multilabel* e as discussões. Na Seção 9.4, concluimos com as considerações finais.

9.1 A Base *ds-FAR*

Para avaliar as dificuldades de aprendizagem dos estudantes e remediá-las, um professor selecionou de cada exercício aplicado, um modelo de solução, isto é, a melhor solução desenvolvida por um aluno para ser o gabarito de outras soluções. Cada solução desenvolvida, incluindo a solução-gabarito, foi então representada por um conjunto de variáveis de avaliação (ou dimensões), cujos valores foram a frequência de ocorrência de palavras reservadas da Linguagem C, símbolos e operadores e *flags* indicadores de compilação e execução correta. Os textos de comentários, os identificadores de variáveis e os textos de *strings* presentes nas soluções dos alunos não foram considerados no processo de avaliação.

Os desempenhos em cada variável de avaliação foram calculados dividindo-se o valor da variável pela sua correspondente na solução-gabarito. Quando os desempenhos nas variáveis escolhidas pelo professor para avaliar uma atividade estavam abaixo de 0.7, o professor recomendava as classes de atividades referentes aos conteúdos dessas variáveis de avaliação. Se em relação às variáveis do gabarito excedessem a 1, classes de atividades de desenvolvimento eficiente de programas eram recomendadas.

Em resumo, através do SOAP, obtivemos 1784 programas em Linguagem C submetidos por alunos. Esses programas foram representados por 37 variáveis de avaliação e associados a 19 classes de atividades de programação. A Figura 9.1 é um exemplo de como três perfis de alunos foram mapeados nessas variáveis e como as classes de atividades foram recomendadas pelo professor.

Na Figura 9.1, os três estados de perfis $A1$, $A2$ e $A3$ são representados pelos desempenhos dos alunos em uma atividade de programação. Essa atividade-exemplo é descrita conforme o enunciado:

Escreva um programa C que leia o voto de uma pessoa em uma urna automática. Leia o número do candidato votado pela pessoa,

assessment variables (attributes)																		
Profiles	compile	run	printf	include	main	return	+	-	*	/	%	++	--	=	float	int	char	scanf
A1	1	1	0.3	1	1	1	0	0	0	0.1	0	0	0	0	0	0.5	0	0
A2	1	1	2.4	1	1	1	0	1	0	0.2	0	0	0	0	0	1.5	0	0
A3	1	1	0.3	3	1	1	0	0	0	0	0	0	0	0	0	1	0	2

<table>
<tr><th>Profiles</th><th>></th><th><</th><th>!=</th><th>==</th><th><=</th><th>>=</th><th>||</th><th>!</th><th>&&</th><th>if</th><th>else</th><th>else if</th><th>switch</th><th>case</th><th>break</th><th>default</th><th>do</th><th>while</th><th>for</th></tr>
<tr><td>A1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0.5</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>A2</td><td>1</td><td>1</td><td>0</td><td>5</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>2.5</td><td>2.5</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>A3</td><td>3</td><td>3</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0.5</td><td>0.5</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

(A)

classes of activities																			
Profiles	nothing	structure	output	arithmetic operators	types	assignment	input	comparison operators	logic operators	if structure	if-ladder	switch structure	while structure	do-while structure	for structure	understanding	depuration	execution	efficiency
A1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
A3	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0

(B)

Figura 9.1: (A) Mapeamentos de perfis em variáveis de avaliação (B) Classes de Atividades (1 = Recomendada; 0 = Não-recomendada)

conforme legenda, e mostre na tela o nome e o número do candidato escolhido. Em seguida, o programa deve perguntar se a pessoa confirma o voto. Se sim, deve aparecer a mensagem: ”Voto computado!” Senão, deve aparecer a mensagem: ”Voto não confirmado!”.

Nessa atividade, o professor avaliou o uso das variáveis *compile, run, if, else, switch, case, break* e *default*. As duas primeiras variáveis indicam se um programa compilou e executou, respectivamente. As demais variáveis representam a presença de *tokens* da Linguagem C que caracterizam estruturas de controle condicionais.

De acordo com essas variáveis de avaliação, as principais classes recomendadas foram as seguintes: *if structure* e *switch structure*. Quando o aluno apresentava desempenhos abaixo de 0.7 em *if* ou *else*, a classe *if structure* foi recomendada. Quando os desempenhos estavam abaixo de 0.7 nas variáveis *switch, case, break* ou *default*, foi recomendada a classe *switch structure*. Caso o aluno obtivesse desempenhos abaixo de 0.7 em todas as variáveis avaliadas ou se o seu programa não compilasse, a classe *understanding* era recomendada. Se em todas as variáveis o aluno obtivesse desempenhos maiores que 0.7, a classe *nothing* era recomendada, significando que o aluno não precisaria fazer atividades relativas a esses conteúdos. Quando o programa do aluno não compilava, além da classe *understanding*, era recomendada a classe *depuration*. Se não executasse corretamente, recomendava-se a classe *execution*. Por último, se os desempenhos do aluno em qualquer uma das variáveis avaliadas fosse

Base	C	A	I	DC	CD	MNC	RC
ds-FAR	19	37	1784	54.93%	2.03	8	31.58%
Legenda:	C: número de classes A: número de atributos I: número de instâncias DC: percentual de instâncias pertencentes a mais de uma classe CD: número médio de classes por instância MNC: número máximo de classes associadas a uma instância RC: percentual de classes raras (contendo menos que 5% das instâncias na base).						

Tabela 9.1: Informações sobre o *ds-FAR*

Bases	C	A	Conjunto de treino DC	CD	*RC**	Conjunto de teste DC	CD	*RC**
Business & Economy	30	438	42.20%	1.590	50.00%	41.93%	1.586	43.33%
Recreation & Sports	22	606	30.20%	1.414	18.18%	31.20%	1.429	18.18%
	*RC**: percentual de classes raros (contendo menos de 1% das instâncias da base).							

Tabela 9.2: Bases de páginas web do Yahoo.com (ZHANG; ZHOU, 2007)

acima de 1, recomendava-se a classe *efficiency*, entendendo-se que o aluno utilizou instruções em excesso para desenvolver uma simples solução.

A Tabela 9.1 contém as principais estatísticas das características da base de exercícios *ds-FAR*. Conforme pode ser observado, mais da metade das instâncias no *ds-FAR* são rotuladas com mais de uma classe. Em média, cada amostra é rotulada com mais de duas classes, e pelo menos uma instância é associada a oito classes.

Para melhor compreender o nível de dificuldade do *ds-FAR*, nós escolhemos duas das bases de páginas *web* do *Yahoo.com* (ZHANG; ZHOU, 2007) para comparar suas características com as características do *ds-FAR* e para avaliar a performance do algoritmo ML-kNN. A Tabela 9.2 apresenta essas duas bases. Para cada um delas, o conjunto de treino contém 2000 amostras de documentos e o de teste, 3000 amostras (ZHANG; ZHOU, 2007).

De acordo com as Tabelas 9.1 e 9.2, verifica-se que as bases de páginas web do Yahoo.com são maiores em relação ao número de classes, de atributos, de classes raras e do percentual de classes raras do que a base de exercícios do *ds-FAR*. No entanto, o *ds-FAR* tem um percentual maior de instâncias associadas a mais de uma classe e um valor médio de classes por instâncias mais alto.

9.2 Procedimentos

Para identificar e automaticamente recomendar classes de atividades de acordo com as variáveis de avaliação que precisam ser melhoradas, o classificador ML-kNN foi aplicado no *ds-FAR*. Nós descrevemos a seguir o procedimento que foi usado para ajustar o algoritmo e testar sua performance.

Inicialmente, a base foi dividida em duas partes: o conjunto de treino/validação, que contém 1338 instâncias (75% da base), e um conjunto de teste contendo 446 instâncias (25% da base).

As performances do algoritmo ML-kNN foram avaliadas usando diferentes valores para o número *k* de vizinhos mais próximos, e a métrica usada nessa avaliação foi o *one-error*. O menor valor de *one-error* foi a melhar performance obtida.

O conjunto de treino foi divido em um conjunto de 1004 instâncias, que foi usado para construir o classificador, e um conjunto de 334 instâncias de validação, que foi usada para avaliar a performance do classificador para cada combinação dos parâmetros.

Depois desse passo, um teste foi realizado vinte vezes utilizando partições randômicas para treino e teste. Para avaliação, as métricas utilizadas foram *hamming loss*, *one-error*, *coverage*, *ranking loss* e *average precision*. Essas métricas foram explicadas em detalhes no Capítulo 5.

9.3 Resultados

A Tabela 9.3 mostra os resultados de avaliação do algoritmo ML-kNN sob diferentes métricas de avaliação de classificação *multilabel* tais como *hamming loss, one-error, coverage, ranking loss* e *average precision*, quando aplicadas ao *ds-FAR* e as bases de páginas *web Business & Economy* e *Recreation & Sports* do *Yahoo.com* (ZHANG; ZHOU, 2007). Sendo as bases do *Yahoo.com* uma referência no domínio da classificação de documentos, nosso objetivo foi executar o mesmo algoritmo nessas duas bases e no *ds-FAR* para avaliar os resultados do algoritmo no contexto da recomendação de atividades reformulada como um problema de classificação.

Os resultados da métrica *hamming loss* no *ds-FAR*, de acordo com a Tabela 9.3, indicam

Algoritmo ML-kNN					
Bases	*Hamming Loss*	*One-error*	*Coverage*	*Ranking Loss*	*Average Precision*
ds-FAR	0.04	0.11	2.3	0.05	0.89
Business & Economy	0.0269	0.1213	2.1840	0.0373	0.8798
Recreation & Sports	0.0620	0.7057	5.1010	0.1913	0.4552

Tabela 9.3: Resultados do algoritmo ML-kNN

que o algoritmo ML-kNN alcança 96% de acurácia ao sugerir as classes corretas de atividades para os perfis de testes. Além disso, o algoritmo também tem, conforme o valor de *one-error*, apenas 11% de probabilidade de indicar incorretamente como a principal uma classe pertinente para cada instância. O valor 2.3 para *coverage* indica que, para o *ds-FAR*, na maioria das vezes, o classificador sequencia entre as três classes de atividades aquelas que mais provavelmente um professor recomendaria para um perfil de aluno. Como a maioria dos perfis do *ds-FAR* são associadas a mais de uma classe, há alta precisão nessa tarefa de recomendação de atividades.

De acordo com a Tabela 9.3, nos resultados de performance do algoritmo ML-kNN no *ds-FAR*, a maioria das métricas estão entre os valores alcançados pelas mesmas métricas nas bases do *Yahoo.com* que têm um número muito maior de instâncias, classes, atributos e percentual de classes raras (ZHANG; ZHOU, 2007). Os valores de *one-error* e *average precision* para o *ds-FAR*, apesar de estarem fora da faixa de resultados para as bases do *Yahoo.com*, estão muito próximos aos valores dessas métricas para a base *Business & Economy*. Esses resultados mostram que o *ds-FAR* comporta-se de forma similar a outras bases multirrotuladas na classificação de documentos. Isso demonstra que a recomendação de atividades pode ser reformulada como um problema de classificação *multilabel*.

O gráfico da Figura 9.2 ilustra quão difícil é analisar o *ds-FAR* e as bases do *Yahoo.com*. Cada métrica na Figura 9.2 é representada por um raio que inicia-se no centro do círculo. Seu valor varia de 0.0, no centro do círculo, a 1.0 nas bordas no círculo. O resultado alcançado por um algoritmo em relação à métrica dada, é então plotado sobre os correspondentes raios. Os menores valores das métricas *hamming loss*, *ranking loss*, *one-error*, e *coverage* apontam para os melhores resultados.

A normalização de *coverage* foi realizada para que seu valor fosse ajustado para um valor entre 0 e 1 na Figura 9.2 dividindo o atual valor de *coverage* por $|C| - 1$.

Da mesma forma, para desenhar os resultados de *average precision*, o gráfico foi desenhado com o valor de *average precision* normalizado para 1− *average precision*.

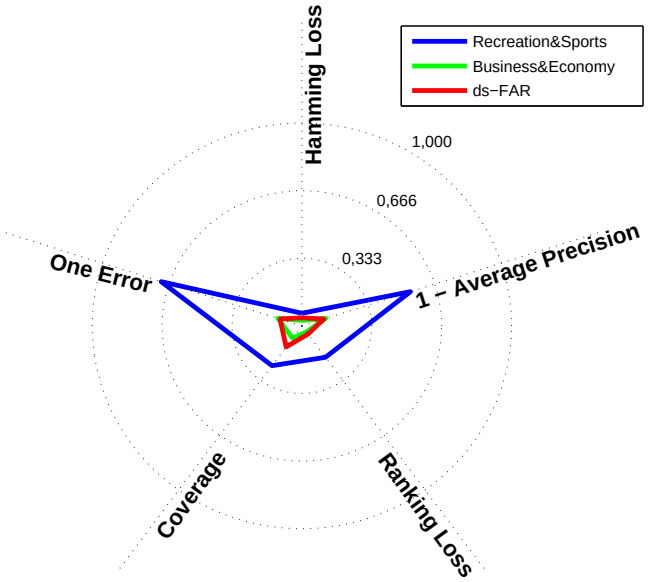

Figura 9.2: Análise de dificuldade da base *ds-FAR*

Na Figura 9.2, quanto maior a área do desenho geométrico, mais difícil é a base. Dessas três bases, a base *Recreation & Sports* é a mais difícil. O *ds-FAR*, embora tenha menos instâncias, classes e atributos que as bases do *Yahoo.com*, apresenta um nível de dificuldade comparável à base *Business & Economy*, que é também considerada uma base difícil, segundo (ZHANG; ZHOU, 2007).

O gráfico da Figura 9.2 e as características do *ds-FAR* na Tabela 9.1 comparadas às características das bases de páginas *web* do *Yahoo.com* na Tabela 9.2 indicam que o *ds-FAR* não é uma base menos complexa no contexto da classificação de documentos. Além disso, os resultados mostram que o algoritmo ML-kNN é eficaz tanto para a classificação de documentos quanto para a recomendação de atividades de acordo com os perfis de alunos.

9.3.1 As métricas de avaliação de classificação *multilabel* como instrumentos de avaliação formativa

Os resultados das métricas de avaliação de classificação *multilabel* podem ter novos significados quando consideradas como instrumentos de avaliação que auxiliam professores no monitoramento da aprendizagem de seus alunos e no melhoramento da prática de ensinar programação.

A métrica *hamming loss*, por exemplo, é um indicador que ajuda um professor a avaliar a eficácia da remediação do processo de aprendizagem através da recomendação de atividades. Um baixo valor de *hamming loss* indica a eficácia do sistema em recomendar de fato conforme as características de cada tipo de perfil de aluno. Isso significa que os *remédios* corretos são sugeridos para cada tipo de *paciente* conforme os *sintomas* apresentados. Dessa forma, sabendo por *hamming loss* que as classes de atividades foram corretamente recomendadas, o professor poderá visualizar as características de perfis para compreender por que essas classes foram recomendadas, isto é, quais as possíveis causas das dificuldades de aprendizagem de seus alunos.

Um exemplo de como um professor pode compreender as dificuldades de aprendizagem de seus alunos é mostrado nos gráficos da Figura 9.3. No gráfico de *Perfis de estudantes*, cada estudante A, B, C, D, E ou F, é representado por um conjunto de características, conforme gráfico da Figura 9.1.

No gráfico de classes de atividades recomendadas da Figura 9.3, para cada uma das classes de estudantes, um conjunto de dezenove classes de atividades são mostradas. No primeiro gráfico, o de perfis de estudantes (*Students' Profile*), quanto mais forte for a tonalidade de vermelho em uma coluna, mais significante é a característica dessa coluna em um perfil. Da mesma forma, quanto mais fracas são as tonalidades de vermelho, menores são os desempenhos de um aluno. Já, no gráfico de classes recomendadas (*Recommended Classes*), a cor vermelha em uma coluna, que representa uma classe de atividades, sinaliza que essa classe foi recomendada para um perfil de aluno. A cor verde indica ausência de desempenhos em uma característica no primeiro gráfico e a cor branca, no segundo gráfico, indica valor nulo para uma classe, indicando que esta não foi recomendada para um perfil.

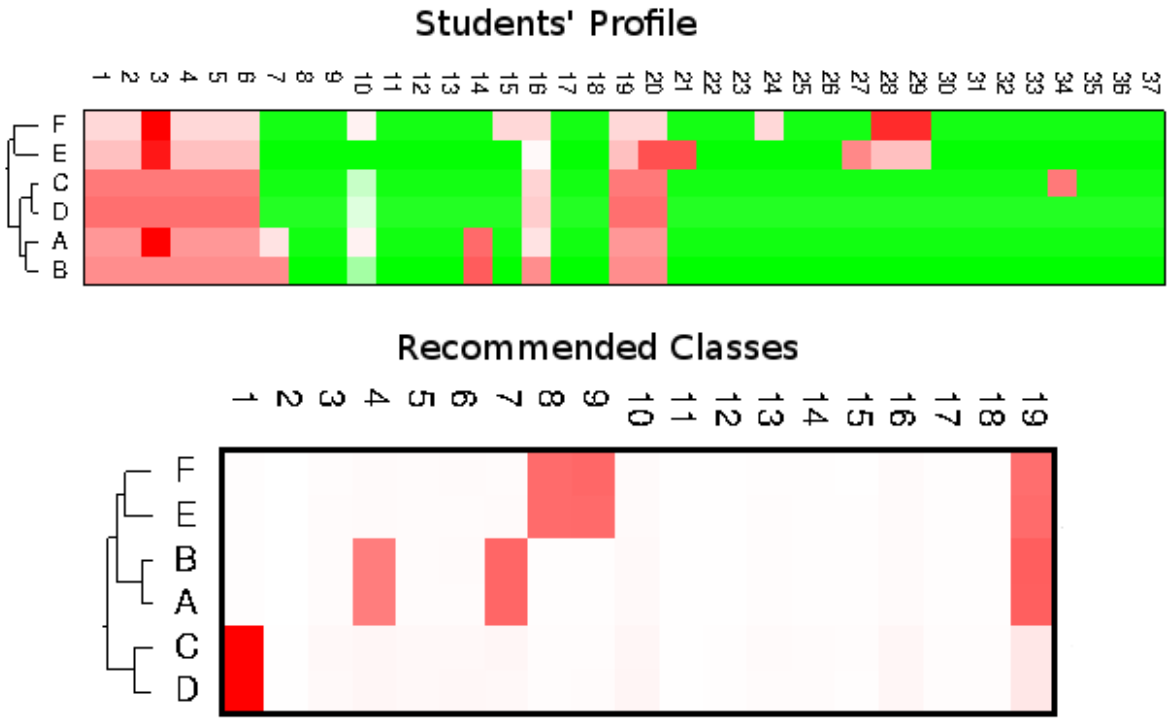

Figura 9.3: Um exemplo de classes de atividades recomendadas para alguns perfis

Conforme a hierarquia de estudantes nos dois gráficos da Figura 9.3, observa-se que, para cada par de estudantes com têm perfis similares (F/E, C/D, A/B) no primeiro gráfico, o mesmo conjunto de classes de atividades é recomendado no segundo gráfico. Por exemplo, as classes 8, 9 e 19 são recomendadas para os estudantes F e E. Da mesma forma, as classes 4, 7 e 19 são recomendadas para os estudantes A e B. Para compreender por que essas classes são recomendadas, devem ser observadas as características coincidentes (em vermelho) em cada par de estudantes com perfis semelhantes no primeiro gráfico da Figura 9.3.

Como o valor de *hamming loss* para o *ds-FAR* é 0.04 (Table 9.3), isto é, um valor baixo, mostramos a corretude de recomendação de classes de atividades e que as características dos perfis associados a essas classes explicam as possíveis causas das dificuldades de aprendizagem para cada tipo de perfil de estudante na maioria das vezes.

A métrica *one-error*, por sua vez, por ser utilizada nas ações de avaliação formativa de um professor porque ela fornece uma indicação dos conteúdos que certos grupos de estudantes estão apresentando mais dificuldades. Essa indicação ocorre porque um baixo valor de *one-error* significa que o classificador escolhe com mais confiança a

classe de atividades mais provável para cada estado de perfil de estudante. Desse modo, se uma classe é altamente recomendada para etudantes com estados de perfis semelhantes a esse estado de perfil, tal como a classe c_1 mostrada na Figura 9.4, então muito provavelmente essa classe aponta para os conteúdos que estudantes com esse mesmo perfil mais têm dificuldades. Essa informação ajuda, portanto, um professor a identificar os conteúdos que ele precisa dar mais ênfase, reorientando a prática de seu ensino para melhorar a aprendizagem de seus alunos.

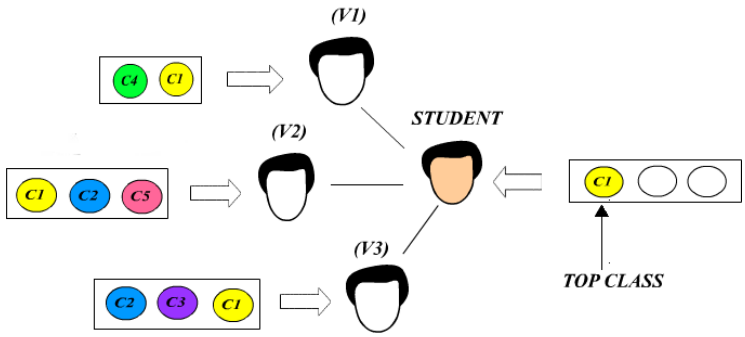

Figura 9.4: A recomendação da principal classe

Na Figura 9.4, a classe c_1 é a classe do topo do *ranking* recomendada para um estudante. Essa classe foi a mais recomendada para os vizinhos V_1, V_2 e V_3. Nesse caso, o conteúdo de atividades associadas à classe c_1 pode ser melhor assistido por um professor.

Um alto valor para a métrica *average precision* estende a interpretação da métrica *one-error* ao indicar que as classes recomendadas mais próximas do topo do *ranking* são aquelas mais apropriadas para estudantes com perfis similares e cujos conteúdos precisam receber maior ênfase de ensino pelo professor.

O valor de *coverage* para o *ds-FAR* na Tabela 9.3 indica que o sistema recomendador desce aproximadamente duas classes no *ranking* para recomendar todas as classes pertinentes para um perfil de estudante. Como o número médio de classes de atividades, por exemplo, no *ds-FAR*, é 2.3 (Tabela 9.1), o sistema recomendador na maioria das vezes recomenda primeiro as classes de atividades mais importantes para um perfil de estudante. Em seguida, recomendam-se classes que são menos importantes ou algumas vezes não-pertinentes. O valor obtido para *coverage* sugere, portanto, que o sistema recomendador, em geral, não recomenda classes de atividades menos relevantes antes das classes que, de fato, seriam sugeridas por um professor.

Perfis	Classes pertinentes	*rankings* de classes recomendadas
1	c_3	c_3
2	c_7, c_4	c_4, c_7
3	c_{19}, c_7	c_7, c_8, c_{19}
4	c_{19}, c_7	c_7, c_{19}
5	c_7	c_7
6	$c_8, c_{19}, c_{12}, c_{13}, c_9$	c_8, c_9, c_{10}, c_{19}
7	c_6, c_9	c_6, c_9
8	c_1	c_1

Tabela 9.4: *Rankings* de recomendações

A Tabela 9.4 mostra como, em nosso sistema, algumas classes foram recomendadas para alguns perfis de estudantes. O valor 2.3 para *coverage* na Tabela 9.3 no *ds-FAR* nos leva a entender que, para todos os perfis da Tabela 9.4, é necessário descer pelo menos três classes no *ranking* de recomendações para recomendar cada uma das classes pertinentes para cada perfil.

O baixo valor 0.05 para *ranking loss* no *ds-FAR* (Tabela 9.3) confirma que, na maioria das vezes, as classes pertinentes não são recomendadas abaixo das classes não-pertinentes. Na Tabela 9.4, por exemplo, somente os Perfis 3 e 6 têm classes não-pertinentes sendo recomendadas acima de classes pertinentes no seu *ranking* de recomendação. No Perfil 3, a classe c_8, que é não-pertinente, é recomendada acima de c_{19}, que é uma classe pertinente. No Perfil 6, c_{10}, uma classe não-pertinente, foi recomendada acima da classe pertinente c_{19}.

Os valores de *one-error*, *average precision* e *coverage* sugerem, portanto, que as primeiras classes recomendadas em um *ranking* para cada perfil de estudante mais provavelmente representam os conteúdos de aprendizagem que precisam de maior atenção do professor para remediação do processo de aprendizagem de programação. Já os baixos valores de *coverage* e *ranking loss* indicam que classes de atividades não relevantes para um tipo de perfil de aluno não são recomendadas na frente das classes mais relevantes, o que sugere precisão de recomendação do sistema recomendador.

9.3.2 Discussões

A classificação *multilabel* é uma estratégia viável para sistemas recomendadores baseados em similaridades de itens (KATAKIS; TSOUMAKAS; VLAHAVAS, 2008;

SONG; ZHANG; GILES, 2011; LÓPEZ et al., 2012) e de perfis. Neste estudo, nós demonstramos a viabilidade dessa última abordagem através do algoritmo ML-kNN.

Os experimentos de recomendação por classificação *multilabel* através do algoritmo ML-kNN na base *ds-FAR* obtiveram resultados bem superiores aos resultados do mesmo algoritmo na classificação das bases do *Yahoo.com* (ZHANG; ZHOU, 2007). Esses resultados poderiam sugerir que o *ds-FAR* é uma base com baixo nível de dificuldade. No entanto, o vocabulário das bases de páginas *web* do *Yahoo.com* é muito mais amplo do que o vocabulário do *ds-FAR*, que é composto de programas de computador em Linguagem C, uma linguagem de programação com vocabulário mais reduzido.

Apesar disso, de acordo com os resultados do gráfico da Figura 9.2, nosso modelo de representação de perfis combina documentos que são representados pela frequência de ocorrência de termos em documentos (ZHANG; ZHOU, 2007) e perfis representados pelos desempenhos baseados em variáveis de avaliação (BAYLARI; MONTAZER, 2009). Nossa representação de perfis é, portanto, um meio termo entre a representação de perfis para recomendação (BAYLARI; MONTAZER, 2009) e a representação de documentos para classificação (ZHANG; ZHOU, 2007).

Embora haja várias bases disponíveis para experimentação em sistemas recomendadores, incluindo sistemas recomendadores TEL (MANOUSELIS et al., 2011), nós tivemos dificuldades em obter bases que fossem formadas por um representativo número de amostras com recomendações realizadas pelo especialista humano para os estados de perfis de estudantes com diferentes tipos de dificuldades de aprendizagem. O *ds-FAR*, conforme descrito na Seção 9.1, é uma base adequada para esse tipo de problema.

Pelo número de amostras, pela referência do especialista humano nas recomendações e pela representação de estados de perfis baseados em desempenhos calculados a partir dos próprios programas do alunos, o *ds-FAR* pode ser considerado uma importante contribuição para a área de sistemas recomendadores. O *ds-FAR* pode, desse modo, ser utilizado por outros pesquisadores para reproduzir novos experimentos no domínio de sistemas de recomendação.

O *ds-FAR* e as bases de páginas *web* do *Yahoo.com* caem na mesma formalização de

problema apresentada no Capítulo 6, embora sejam aplicadas em diferentes domínios de conhecimento. Conclui-se então que o ML-kNN demonstra boa performance tanto para problemas de dificuldade mediana como a recomendação de classes de atividades quanto para níveis mais complexos de classificação de documentos, conforme confirmam os resultados da Tabela 9.3 e da Figura 9.2.

Os resultados das métricas de avaliação de classificação *multilabel* da Tabela 9.3 sugerem que as principais classes de atividades são recomendadas de acordo com as dificuldades de aprendizagem de cada estado de perfil de estudante e que essas recomendações geralmente estão em concordância com as recomendações realizadas por um professor. Esses resultados nos fornecem indicadores de que o *ranking* de classes recomendadas para cada perfil representa o conjunto de atividades mais importantes para estudantes melhorarem a sua aprendizagem no domínio da programação. Recomendando, portanto, as principais as classes de atividades para cada perfil de aluno, estendemos a proposta de recomendação sequenciada de atividades desenvolvida por (KLASNJA-MILICEVIC et al., 2011).

Neste estudo, assumimos que atividades de programação já eram pré-classificadas nas classes que foram recomendadas. No entanto, a classificação *multilabel* pode também ser utilizada para classificar descrições das atividades da mesma forma que documentos são classificados em uma ou mais classes (ZHANG; ZHOU, 2007)..

Embora tenhamos utilizado neste trabalho o algoritmo ML-kNN para a tarefa de recomendação de itens por classificação *multilabel*, outros algoritmos, de equivalente ou melhor performance que o ML-kNN, poderiam ser utilizados para os mesmos propósitos (ZHANG; ZHOU, 2007; TSOUMAKAS; KATAKIS; VLAHAVAS, 2011).

9.4 Conclusão

Neste capítulo, mostramos como a recomendação de atividades baseada em perfis de estudantes pode ser tratada como uma tarefa de classificação *multilabel* em que uma ou mais classes de atividades são associadas a cada perfil de estudante. Um perfil é caracterizado pelos desempenhos de um estudante em uma atividade de programação. Para realizar a recomendação de classes de atividades, nós utilizamos o algoritmo

ML-kNN (ZHANG; ZHOU, 2007). Em nosso contexto de recomendação, o ML-kNN indica as classes de atividades mais prováveis associadas a um perfil de estudante conforme as classes que foram associadas aos *k* vizinhos mais próximos desse perfil.

Nosso modelo de recomendação aplica princípios de avaliação formativa para remediar a prática da programação, indicando classes de atividades que um estudante deve resolver para melhorar a sua performance em diferentes variáveis de avaliação. Com um sistema recomendador, nós temos como objetivo transformar estados de perfis promovendo êxitos nas variáveis de avaliação e desenvolver competências em programação de computadores.

Com a nossa estratégia de recomendação, que é orientada a perfis multidimensionais representados por desempenhos em atividades de programação, nós oferecemos um modelo de recomendação mais compreensível baseado nos atuais estados de aprendizagem dos alunos porque diagnosticamos os seus possíveis problemas de aprendizagem (ADOMAVICIUS; TUZHILIN, 2005; BAYLARI; MONTAZER, 2009).

O nosso sistema de recomendação alcança 96% de acurácia em suas recomendações e 89% de precisão média. Esses e outros resultados que foram obtidos sob diferentes métricas de avaliação para classificação *multilabel* demonstram a eficácia da nossa estratégia na tarefa de recomendar classes de atividades para estudantes que tenham problemas de aprendizagem no domínio da programação de computadores.

Neste trabalho, também damos novos significados às métricas de classificação *multilabel* no contexto da recomendação de atividades. A métrica *hamming loss* indica a eficácia da remediação do processo de aprendizagem por recomendação de atividades informando como os *remédios* corretos são sugeridos para cada tipo de *paciente* conforme os *sintomas*. A métrica *one-error* informa a professores sobre a principal classe recomendada sinalizando os conteúdos que os alunos mais apresentam dificuldades. As demais métricas, por sua vez, informam se de fato as principais classes de atividades foram recomendadas conforme perfis dos alunos.

Embora as métricas indiquem que, na maioria das vezes, as classes mais importantes de atividades são recomendadas para estudantes, essas classes não são apresentadas em uma sequência. Portanto, como trabalho futuro a partir deste, nós propomos sequenciar as classes de atividades recomendadas para estudantes a fim de guiá-los em um caminho

de aprendizagem traçado por uma sequência de atividades mais adequadas que os ajudem a melhorar os seus desempenhos e, por conseguinte, a sua aprendizagem.

Capítulo 10

Considerações Finais

Este trabalho apresentou uma estratégia computacional de avaliações diagnóstica e formativa para regulação da aprendizagem de programação. Os experimentos de avaliação semi-automática de exercícios, de mapeamento de perfis e de recomendação de atividades demonstraram a viabilidade de implementação dos métodos propostos. Os resultados alcançados confirmam a contribuição deste trabalho para os domínios da aprendizagem da programação e de reconhecimento de padrões.

Para o desenvolvimento da metodologia de regulação de aprendizagem de programação, realizamos pesquisas nas áreas de Ciência da Computação (SCHILDT, 1991), da Educação (PERRENOUD, 1999), da Psicologia Cognitiva (ANDERSON, 2000), da Estatística (HAIR; TATHAM; BLACK, 1998), de Reconhecimento de Padrões (DUDA; HART; STORK, 2001) e da Teoria Geral dos Sistemas (L.BERTALANFY; VON, 1973). Por isso, este trabalho pode ser considerado multidisciplinar.

Para desenvolver o módulo de avaliação semi-automática de exercícios de programação, realizamos profundo estudo de como programas de computador são construídos, principalmente em Linguagem C, uma linguagem segundo o paradigma estruturado de programação (SCHILDT; MAYER, 2006). Esse conhecimento foi relevante para escolher um conjunto de características que melhor representassem um programa de computador e para reconhecer possíveis critérios de avaliação de professores.

Para o desenvolvimento da metodologia dos núcleos de avaliação, utilizamos os conceitos de avaliação diagnóstica e formativa da área de Educação (PERRENOUD, 1999). Essas estratégias de avaliação, embora sejam praticamente inviáveis de aplicação no modelo de ensino presencial e em turmas numerosas, podem ser muito eficazes para promover êxitos coletivos de aprendizagem se implementadas em tecnologias computacionais (ANDERSON, 2000; PACHECO, 2005; MAZZA; DIMITROVA, 2007; CASTELLANO et al., 2007; OLIVEIRA; OLIVEIRA, 2008a).

Alguns conhecimentos da Psicologia Cognitiva foram relevantes para o planejamento das componentes de habilidades, que são as nossas variáveis de avaliação para monitoramento e controle da aprendizagem da programação (ANDERSON, 2000; PEA; KURLAND, 1984). Uma vez conhecendo os efeitos cognitivos da programação e as habilidades envolvidas na prática da programação, planejamos tarefas para contemplar esses efeitos e para desenvolver habilidades fundamentais para a prática da programação (PEA; KURLAND, 1984). A ideia é, dessa forma, utilizar os próprios conteúdos dos cursos de programação na forma de tarefas como instrumentos para trabalhar habilidades de programação como compreender, planejar, criar e depurar (ANTUNES, 2001; PEA; KURLAND, 1984).

Das áreas de Reconhecimento de Padrões e Estatística, combinamos técnicas de aprendizagem supervisionada e não-supervisionada para predizer notas, classificar alunos e para recomendar atividades apropriadas para alunos com dificuldades de aprendizagem (DUDA; HART; STORK, 2001; HAIR; TATHAM; BLACK, 1998; JAIN; MURTY; FLYNN, 1999; MANNING; RAGHAVAN; SCHUTZE, 2008). As técnicas aplicadas neste trabalho foram o *clustering*, que é de aprendizagem não-supervisionada, a regressão linear e o algoritmo ML-kNN, ambos de aprendizagem supervisionada.

Para predizer notas, utilizamos a técnica de *clustering* e a regressão linear múltipla. O *clustering* facilitou a predição ao selecionar conjuntos mais homogêneos de amostras de exercícios de programação bem como as características que melhor os representam. De cada um desses conjuntos, foi escolhido um subconjunto mínimo de amostras para o professor atribuir notas. A partir dessas, as demais amostras de um conjunto são preditas por um modelo de regressão linear múltipla ou simplesmente pelos índices de

similaridades entre padrões, quando um conjunto de amostras é muito pequeno.

Alguns princípios da Teoria Geral dos Sistemas como a *realimentação* também foram aplicados para o desenvolvimento do NAF em conjunto com os conceitos de avaliação diagnóstica e formativa (MAYA; LEONARDI, 2010). A ideia foi tratar o processo de aprendizagem como um sistema que recebe instruções como entrada, gera desempenhos como saída e é realimentado pela avaliação. Dessa forma, mapeamos desempenhos em componentes de habilidades para que fossem controlados e regulados até atingir um estado-objetivo, que é a aprendizagem (ALLAL; SAADA-ROBERT, 1992; KIRCHNER; STOLZ, 2008). E para que esse objetivo fosse alcançado, esses estados precisariam ser continuamente realimentados. Essa realimentação ou regulação poderia ser a recomendação de atividades apropriadas para os alunos que estão em estados de aprendizagem insatisfatórios. O objetivo da abordagem sistêmica da avaliação seria, portanto, no domínio da programação, que a prática de exercícios fosse planejada e controlada de forma a promover êxitos de aprendizagem (ANDERSON, 2000).

Para apresentar a viabilidade deste trabalho, destacamos nas seções a seguir como foram as pesquisas de tese e quais os trabalhos futuros a partir desta pesquisa. Na Seção 10.1, reforçamos as contribuições deste trabalho para a aprendizagem de programação e para a área de Reconhecimento de Padrões. Na Seção 10.2, apresentamos os trabalhos futuros a serem realizados a partir deste. Na Seção 10.3, concluimos com as considerações finais.

10.1 Contribuições

Uma vez conhecidas as metodologias NAD e do NAF, bem como os resultados alcançados nos experimentos de avaliação semi-automática e de recomendação de atividades, explicamos as contribuições deste trabalho mencionadas no Capítulo 1.

A nossa metodologia de avaliação semi-automática e de recomendação de atividades de programação é inovadora pelas seguintes razões:

1. O nosso modelo de avaliação aponta para uma avaliação fina, do tipo clínica e dinâmica para captar a multidimensionalidade do objeto aprendido para se coletar

uma pluraridade de informações (RAPHAEL; CARRARA, 2002).

2. Desenvolvemos uma estratégia inovadora em relação a outras propostas de avaliação semi-automática de exercícios de programação.
3. Desenvolvemos uma estratégia semi-automática de recomendação de atividades que transforma a tarefa de recomendação em uma tarefa de classificação *multilabel* tendo como referência recomendações de um especialista humano (OLIVEIRA; CIARELLI; OLIVEIRA, 2013).
4. Oferecemos uma estratégia de avaliação semi-automática que possibilita a prática assistida da programação com *feedbacks* mais rápidos em turmas com grande quantidade de alunos e onde se aplicam grandes quantidades de exercícios.
5. Combinamos técnicas de reconhecimento de padrões de abordagens não-supervisionada e supervisionada para predição de notas e recomendação de atividades de programação em conformidade com padrões de professores.
6. Formamos uma ampla base de exercícios de programação resolvidos por alunos e avaliados por professores para realização de experimentos de avaliação e de recomendação semi-automática de atividades.

Em relação à Contribuição 1, através do mapeamento de perfis do NAD, podemos capturar informações de uma questão, de uma tarefa e de um aluno comparando todos os alunos de uma turma e os classificando por perfis. Podemos, dessa forma, através do mapa de uma questão comparar os programas desenvolvidos por todos os alunos de uma turma e capturar as dificuldades de aprendizagem. Da mesma forma, os mapas de tarefas e dos desempenhos dos alunos, nos mostram através de um conjunto de variáveis como os alunos progridem ao longo de um curso.

Na Contribuição 2, desenvolvemos uma estratégia inovadora de avaliação semi-automática de exercícios que utiliza a técnica de *clustering* combinada à técnica de regressão linear para melhor predizer notas de exercícios de programação conforme padrões de correção de um professor. Realizando as tarefas de selecionar amostras e características mais representativas para compor um modelo de regressão linear, o *clustering* favoreceu a redução dos erros na predição de notas e, principalmente, possibilitou a redução de esforço de correção do professor em até 70% .

Conforme a Contribuição 3, através do algoritmo ML-kNN recomendamos classes de atividades de programação conforme as dificuldades de aprendizagem reconhecidas nas componentes de habilidades dos perfis de alunos. Cada perfil foi representado por desempenhos em cada componente de habilidade, que representa o uso correto dos recursos, isto é, palavras-reservadas, operadores e indicadores de funcionamento de programas em Linguagem C. Os resultados mostraram que conseguimos imitar em até 90% as decisões de recomendação de atividades do especialista humano.

Em relação à Contribuição 4, a avaliação semi-automática de exercícios agiliza o *feedback* para o aluno e a recomendação de atividades para regulação de aprendizagem o torna praticamente imediato. A avaliação semi-automática reduz o esforço de correção do professor. No caso das atividades sugeridas, sendo elas pontuadas, não seria necessário solicitar ao professor uma base referência de predição. O sistema realizaria, portanto, a correção automática das atividades sugeridas entregues pelos alunos com retorno imediato.

Sobre a prática ser assistida, o sistema tem como reconhecer onde estão as dificuldades de aprendizagem dos alunos e, por isso, pode intervir recomendando atividades que podem ajudar o aluno a tratar essas dificuldades e melhorar os seus desempenhos. Como um guia do aluno, o sistema realiza essa recomendação de atividades de forma contínua ao longo de um curso.

De acordo com a Contribuição 5, através dessa combinação, mostramos que a técnica de *clustering*, ao formar grupos homogêneos e indicar as características descritivas desses grupos, facilita a predição de notas mesmo para bases com poucas amostras. Isso mostra que o nosso modelo atende tanto ao professor, que reaplica exercícios e tem uma ampla base de exemplos como referência de predição, como também aquele professor que não tem turmas grandes ou não repete exercícios.

Conforme a Contribuição 6, formamos uma ampla base de exercícios de programação resolvidos e pontuados para realização de experimentos bem como para ter uma base referência para predição de notas de outros exercícios de programação.

A formação dessa base foi realizada através da aplicação do sistema SOAP em seis turmas de programação. Apenas entre o segundo semestre de 2011 e o primeiro semestre de 2012, foram resolvidas 3.255 questões de programação. Formamos

também a base *ds-FAR* que contém 1784 amostras soluções de exercícios de programação com as recomendações do especialista humano como referência.

10.2 Trabalhos futuros

Os trabalhos propostos nesta tese podem ser estendidos para outros domínios desde que se definam as componentes de habilidades que representem a aprendizagem de um domínio e que essas componentes possam ser quantificadas.

Como trabalhos futuros a partir deste trabalho, sugerimos que se desenvolvam mais pesquisas em relação à representação de perfis de forma que realmente reflitam um modelo do aprendiz e os seus estados de aprendizagem. Esse modelo ajudaria o sistema de recomendação de atividades a traçar uma rota de aprendizagem através da prática de exercícios conforme as condições de aprendizagem do aluno.

Outra sugestão é que se desenvolvam mais pesquisas sobre a combinação de parâmetros para ajustes de algoritmos de *clustering* para que estes produzam *clusters* mais homogêneos.

É importante também que o Núcleo Executor do SOAP seja integrado a outras interfaces de submissão de exercícios de programação, de forma que seja possível rodar programas submetidos a partir de e-mails, de ambientes de aprendizagem como o *Moodle*, de páginas pessoais de professores e de outros meios interativos da *web*.

Por último, sugerimos um estudo aprofundado sobre aprendizagem de programação a partir de exercícios de programação desenvolvidos por alunos. As bases de exercícios que levantamos ao longo desses quatro anos através do Sistema SOAP representam um material rico do domínio de aprendizagem de programação. Essas bases contribuem, dessa forma, para melhor compreender como as pessoas aprendem programação e como desenvolver estratégias para que elas tornem-se *experts* nesse domínio.

10.3 Conclusão

Este trabalho é um passo inicial e relevante para a avaliação da aprendizagem de programação. Através da nossa metodologia de avaliação oferecemos uma

possibilidade de professores reduzirem esforços na árdua tarefa de corrigir exercícios de programação em turmas com grande quantidade de alunos. Para o aluno, oferecemos as possibilidades de ter *feedbacks* mais rápidos e de ter um acompanhamento contínuo do seu processo de aprendizagem.

Como partes das pesquisas, levantamos um estudo profundo da programação de computadores e do estado da arte da avaliação automática no Brasil e no mundo, colocamos o sistema SOAP em funcionamento em turmas de programação da Universidade Federal do Espírito Santo e realizamos, sob conhecimento dos alunos envolvidos, experimentos de avaliação semi-automática de exercícios de programação, de mapeamento de perfis através do NAD e de recomendação de atividades de programação através do NAF.

O maior objetivo deste trabalho foi criar condições de êxitos de aprendizagem para o domínio da programação de computadores, que é considerada um conhecimento complexo. Mas, sabendo que a programação é um conhecimento que envolve tantas habilidades, entendemos que nossa metodologia poderá trabalhar a partir das causas das dificuldades de aprendizagem, monitorando e regulando as componentes de habilidades para que um estado-objetivo de aprendizagem seja alcançado, que a contribuição principal deste trabalho seja favorecer, através da tecnologia, o desenvolvimento das habilidades e a formação de programadores mais competentes.

Referências Bibliográficas

ADOMAVICIUS, G.; TUZHILIN, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 17, n. 6, p. 734–749, jun. 2005.

ALA-MUTKA, K. M. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, v. 15, n. 2, p. 83–102, 2005.

ALLAL, L.; SAADA-ROBERT, M. La métacoginition: cadre conceptuel pour l'étude des régulations en situation scolaire. *Archives de Psychologie*, n. 60, p. 265–296, 1992.

AMATRIAIN, X. et al. Data Mining Methods for Recommender Systems. In: RICCI, F. et al. (Ed.). *Recommender Systems Handbook*. [S.l.]: Springer US, 2011. p. 39 – 71.

ANDERSON, E. F.; MCLOUGHLIN, L. Critters in the classroom: a 3d computer-game–like tool for teaching programming to computer animation students. In: *ACM SIGGRAPH 2007 educators program*. New York, NY, USA: ACM, 2007. (SIGGRAPH07).

ANDERSON, J. R. *Cognitive psychology and its implications*. New York and Basingstoke: Worth Publishers, 2000. 181-183 p.

ANTUNES, C. *Trabalhando habilidades: construindo idéias*. São Paulo, SP: Scipione, 2001.

BAEZA-YATES, R. A.; RIBEIRO-NETO, B. *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

BALLESTER, M. *Avaliação como apoio à aprendizagem*. Porto Alegre, RS: Artmed, 2003. Trad. Valério Campos.

BARBOSA, L.; OLIVEIRA, M. Metodologia anea para avaliação online de lógica de programação. In: *XVII Workshop de Informática na Educação (WIE 2011), SBC -*

XXII SBIE/XVII WIE. Aracaju, SE: SBC, 2011.

BAYLARI, A.; MONTAZER, G. Design a Personalized E-learning System Based on Item Response Theory and Artificial Neural Network Approach. *Expert Systems with Applications*, v. 36, n. 4, p. 8013 – 8021, 2009.

BECERRA, C.; ASTUDILLO, H.; MENDOZA, M. Improving Learning Objects Recommendation Processes by Using Domain Description Models. In: *LACLO 2012 – Séptima Conferencia Latinoamericana de Objetos y Tecnologias de Aprendizaje*. Guayaquil, Ecuador: [s.n.], 2012. v. 3, n. 1.

BELL, R. M.; KOREN, Y. Lessons from the Netflix prize challenge. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 9, n. 2, p. 75–79, dez. 2007. ISSN 1931-0145.

BENFORD, S. D. et al. The Ceilidh system for the automatic grading of students on programming courses. In: *Proceedings of the 33rd annual on Southeast regional conference*. New York, NY, USA: ACM, 1995. (ACM-SE 33), p. 176–182. ISBN 0-89791747-2.

BERRY, R. E.; MEEKINGS, B. A. A style analysis of C programs. *Commun. ACM*, ACM, New York, NY, USA, v. 28, p. 80–88, 1985.

BLOOM, B.; HASTINGS, J.; MADAUS, G. *Evaluación del Aprendizage*. [S.l.: s.n.], 1975. Buenos Aires, Troquel.

BLUMENSTEIN, M. M. et al. Game: A generic automated marking environment for programming assessment. 2004.

BOBADILLA, J.; SERRADILLA, F.; HERNANDO, A. Collaborative Filtering Adapted to Recommender Systems of E-learning. *Knowledge-Based Systems*, v. 22, n. 4, p. 261 – 265, 2009.

BRUSILOVSKY, P.; KOBSA, A.; NEJDL, W. *The Adaptive Web*. [S.l.]: Springer Berlin, Heidelberg, 2007.

BURKE, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, Springer Netherlands, v. 12, p. 331–370, 2002.

CAMPOS, C.; FERREIRA, C. *BOCA: um sistema de apoio para competicões de programacão*. Salvador, BA: [s.n.], 2004.

CARRILLO, D.; LÓPEZ, V.; MORENO, M. Multi-label classification for recommender systems. In: *Trends in Practical Applications of Agents and Multiagent Systems*. [S.l.]: Springer International Publishing, 2013, (Advances in Intelligent Systems and Computing, v. 221). p. 181–188.

CASTELLANO, M. et al. Neural techniques to improve the formative evaluation procedure in intelligent Tutoring Systems. In: *IEEE International Conference on Computational Intelligence for Measurement Systems and Application (CIMSA2007)*. Ostuni, Italy: [s.n.], 2007. p. 27–29.

CHARNET, R. et al. *Análise de modelos de regressão linear com aplicações*. [S.l.]: Campinas, São Paulo, Unicamp, 1999.

CHEN, C.-M.; CHEN, Y.-Y. Learning performance assessment approach using learning portfolio for e-learning systems. In: *ICALT05: Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies*. Washington, DC, USA: IEEE Computer Society, 2005. p. 557–561. ISBN 0-7695-2338-2.

CHEN, C.-M.; LEE, H.-M.; CHEN, Y.-H. Personalized E-learning System Using Item Response Theory. *Comput. Educ.*, Elsevier Science Ltd., Oxford, UK, UK, v. 44, n. 3, p. 237–255, abr. 2005.

CHEN, R.-C. et al. A Recommendation System Based on Domain Ontology and SWRL for Anti-diabetic Drugs Selection. *Expert Systems with Applications*, v. 39, n. 4, p. 3995 – 4006, 2012.

CORRAR, L. J.; PAULO, E.; FILHO, J. M. D. *Análise multivariada: para os cursos de administração, ciências contábeis e economia*. [S.l.]: São Paulo: Atlas, 2007.

CURTIS, B. et al. Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 2, p. 96–104, 1979.

DIAS, P. *Avaliação Automática de Exercícios em SQL*. Dissertação (Mestrado) — Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2001.

DOUCE, C.; LIVINGSTONE, D.; ORWELL, J. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, ACM, New York, NY, USA, v. 5, September 2005. ISSN 1531-4278.

DRACHSLER, H.; HUMMEL, H. G. K.; KOPER, R. Identifying the Goal, User model and Conditions of Recommender Systems for Formal and Informal Learning. *J. Digit. Inf.*, v. 10, n. 2, 2009.

DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. Second edition. New York: Wiley-Interscience, 2001.

EBRAHIMI, A. Novice programmer errors: language constructs and plan composition. *International Journal of Human-Computer Studies*, v. 41, n. 4, p. 457 – 480, 1994.

FAHIM, A. et al. An efficient enhanced K-means clustering algorithm. *Journal of Zhejiang University SCIENCE A*, v. 7, n. 10, p. 1626–1633, 2006. ISSN 1673-565X.

FERNEDA, E.; SMIT, J. *Introdução aos modelos computacionais de recuperação de informação*. [S.l.]: Ciência moderna, 2012. ISBN 9788539901883.

FONSECA, J. S. d. Martins, gilberto de andrade e toledo, geraldo luciano. Estatística aplicada. São Paulo: Atlas, 1995.

FRANCA, A. et al. Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programacão com integracão ao ambiente moodle. *RENOTE - Revista Novas Tecnologias na Educaccão*, v. 9, n. 1, 2011.

GERDES, A.; JEURING, J. T.; HEEREN, B. J. Using strategies for assessment of programming exercises. In: *Proceedings of the 41st ACM technical symposium on Computer science education*. New York, NY, USA: ACM, 2010. (SIGCSE '10), p. 441–445.

GHAUTH, K.; ABDULLAH, N. The Effect of Incorporating Good Learners' Ratings in e-Learning Content-Based Recommender System. *Educational Technology & Society*, v. 14, n. 2, p. 248 – 257, 2011.

GODOY, D.; AMANDI, A. Link Recommendation in E-learning Systems Based on Content-Based Student Profiles. In: C. ROMERO AND S. VENTURA AND M. PECHENIZKIY AND R.S.J.D BAKER. Boca Raton, FL: CRC Press, 2010. (Handbook of Educational Data Mining), p. 273–284.

GUJARATI, D. N. *Econometria básica*. [S.l.]: Elsevier, 2006.

GUO, R. et al. Intelligent diagnostic feedback for online multiple-choice questions. *Artificial Intelligence Review*, Springer Netherlands, p. 1–15, 2013. ISSN 0269-2821.

HAIR, J.; TATHAM, R. A. R.; BLACK, W. *Multivariate Data Analysis 5th ed.* [S.l.]: Prentice Hall, 1998.

HALSTEAD, M. H. *Elements of Software Science (Operating and programming systems series)*. New York, NY, USA: Elsevier Science Inc., 1977.

HAMALAINEN, W.; VINNI, M. Classifiers for Educational Datamining. In: C. ROMERO AND S. VENTURA AND M. PECHENIZKIY AND R.S.J.D BAKER. Boca Raton, FL: CRC Press, 2010. (Handbook of Educational Data Mining), p. 467–479.

HARB, M. et al. Avaliação automática de consultas sql em ambiente virtual de ensino-aprendizagem. In: *2a. Conferência Ibérica de Sistemas e Tecnologias de Informação*. Porto, Portugal: [s.n.], 2007.

HAYDT, R. *Avaliação do processo ensino-aprendizagem*. São Paulo: Ática, 2002.

HERLOCKER, J. L. et al. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Inf. Syst.*, ACM, New York, NY, USA, v. 22, n. 1, p. 5–53, jan. 2004.

HOLLINGSWORTH, J. Automatic graders for programming classes. *Commun. ACM*, ACM, New York, NY, USA, v. 3, n. 10, p. 528–529, out. 1960.

HUNG, S.-l.; KWOK, L.-f.; CHUNG, A. New metrics for automated programming assessment. In: *Proceedings of the IFIP WG3.4/SEARCC (SRIG on Education and Training) Working Conference on Software Engineering Education*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993. p. 233–243.

HUNTER, G. et al. Learn programming++: The design, implementation and deployment of an intelligent environment for the teaching and learning of computer programming. In: *Intelligent Environments (IE), 2013 9th International Conference on*. [S.l.: s.n.], 2013. p. 129–136.

IHANTOLA, P. et al. Review of recent systems for automatic assessment of programming assignments. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. New York, NY, USA: ACM, 2010. (Koli Calling '10), p. 86–93.

JACKSON, D.; USHER, M. Grading student programs using assyst. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 29, n. 1, p. 335–339, mar. 1997.

JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 31, n. 3, p. 264–323, 1999. ISSN 0360-0300.

JONES, K. S. *Information Retrieval Experiment*. Newton, MA, USA: Butterworth-Heinemann, 1981. ISBN 0408106484.

KARYPIS, G. *CLUTO - A clustering toolkit*. 2003. Dept. of Computer Science, University of Minnesota.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. Multilabel Text Classification for Automated Tag Suggestion. In: *In: Proceedings of the ECML/PKDD-08 Workshop on Discovery Challenge*. [S.l.: s.n.], 2008.

KIRCHNER, R.; STOLZ, T. Tomada de consciência e conhecimento metacognitivo. Artigo: As funções das regulações cognitivas e metacognitivas na prática das atividades complexas do adulto: questões e propostas para um ensaio conclusivo (Maria Helena Fávero). Curitiba,Brasil, p. 321–342, 2008.

KLASNJA-MILICEVIC, A. et al. E-Learning Personalization Based on Hybrid Recommendation Strategy and Learning Style Identification. *Computers & Education*, v. 56, n. 3, p. 885 – 899, 2011.

L.BERTALANFY; VON, L. *Teoria geral dos sistemas*. [S.l.]: Petrópolis: Vozes, 1973.

LINDEN, G.; SMITH, B.; YORK, J. Amazon.com Recommendations: Item-to-item Collaborative Filtering. *Internet Computing, IEEE*, v. 7, n. 1, p. 76 – 80, jan/feb 2003.

LOOKS, M. et al. Streaming Hierarchical Clustering for Concept Mining. *Aerospace Conference, 2007 IEEE*, p. 1–12, 2007. ISSN 1095-323X.

LÓPEZ, V. F. et al. A Model for Multi-label Classification and Ranking of Learning Objects. *Expert Systems with Applications*, v. 39, n. 10, p. 8878 – 8884, 2012. ISSN 0957-4174.

LOPS, P.; GEMMIS, M.; SEMERARO, G. Content-based Recommender Systems: State of the Art and Trends. In: RICCI, F. et al. (Ed.). *Recommender Systems Handbook*. [S.l.]: Springer US, 2011. p. 73–105.

MALMI, L.; KORHONEN, A.; SAIKKONEN, R. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In: *Proceedings of the 7th annual conference on Innovation and technology in computer science*

education. New York, NY, USA: ACM, 2002. (ITiCSE '02), p. 55–59. ISBN 1-58113-499-1.

MANNING, C.; RAGHAVAN, P.; SCHUTZE, H. *Introduction to information retrieval*. Cambridge University Press: Editora Thomson, 2008.

MANOUSELIS, N.; COSTOPOULOU, C. Analysis and Classification of Multi-Criteria Recommender Systems. *World Wide Web*, Springer Netherlands, v. 10, p. 415–441, 2007.

MANOUSELIS, N. et al. Recommender Systems in Technology Enhanced Learning. In: RICCI, F. et al. (Ed.). *Recommender Systems Handbook*. [S.l.]: Springer US, 2011. p. 387–415.

MANOUSELIS, N.; VUORIKARI, R.; ASSCHE, F. V. Collaborative Recommendation of E-learning Resources: an Experimental Investigation. *Journal of Computer Assisted Learning*, Blackwell Publishing Ltd, v. 26, n. 4, p. 227–242, 2010.

MARINAGI, C.; KABURLASOS, V. Work in progress: practical computerized adaptive assessment based on bayesian decision theory. In: *ASEE/IEEE Frontiers in Education Conference, 36th Annual*. San Diego, CA: Frontiers in Education Conference, 36th Annual, 2006. p. 23–24.

MAVRIKIS, M. Machine–learning assessment of students{' behavior within interactive learning environments. In: ROMERO, C., VENTURA, S., PECHENIZKIY, M. AND BAKER, R.S.J.D. Boca Raton, FL: CRC Press, 2010. (Handbook of Educational Data Mining), p. 441–449.

MAYA, P.; LEONARDI, F. *CONTROLE ESSENCIAL*. [S.l.]: PEARSON BRASIL, 2010.

MAZZA, R.; DIMITROVA, V. CourseVis: A graphical student monitoring tool for supporting instructors in web–based distance courses. In: *International Journal of Human–Computer Studies*. London, ROYAUME-UNI: Elsevier, 2007. v. 65, p. 125–139.

MCCABE, T. A complexity measure. *Software Engineering, IEEE Transactions on*, SE-2, n. 4, p. 308 – 320, dec. 1976.

MCCLAVE, J. T.; BENSON, P. G.; SINCICH, T. *Estatística para administração e economia*. [S.l.]: Tradução de Fabrício Pereira Soares e Fernando, 2008.

MCNEE, S. M.; RIEDL, J.; KONSTAN, J. A. Making Recommendations Better: an Analytic Model for Human-recommender Interaction. In: *CHI06 Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2006. (CHI EA06), p. 1103–1108.

MEISAMSHABANPOOR; MAHDAVI, M. Implementation of a Recommender System on Medical Recognition and Treatment. *International Journal of e-Education, e-Business, e-Management and e-Learning*, v. 2, n. 4, p. 315–318, 2012.

MENEZES, C. de et al. Computer supported co-operative systems to support the problem solving - a case study of learning computer programming. *Frontiers in Education, Annual*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. S2H–5–S2H–10, 2008.

MIZUKAMI, M. *Ensino: as abordagens do processo*. São Paulo: EPU, 1986.

MOREIRA, M. P.; FAVERO, E. L. Um ambiente para ensino de programação com feedback automático de exercícios. *Workshop Sobre Educação em Computação*, p. 429–438, 2009.

NAUDE, K. A.; GREYLING, J. H.; VOGTS, D. Marking student programs using graph similarity. *Computers Education*, v. 54, n. 2, p. 545 – 561, 2010.

NEVADO, R.; CARVALHO, M.; MENEZES, C. *Aprendizagem em rede na formação à distância*. Porto Alegre: Ricardo Lenz, 2007.

NUNES, I.; LISBOA, M. *Testador Automático e Método de Avaliacão de Programas em Java*. Porto Alegre, RS: [s.n.], 2004.

OLIVEIRA, M. *Avaliações Online para Nivelamento e Formação de Classificadores Humanos*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, Vitória, ES, 2009.

OLIVEIRA, M.; OLIVEIRA, E. Avaliar para nivelar e formar: um sistema online de avaliação formativa para alunos de biblioteconomia. In: *Anais do XIX Simpósio Brasileiro de Informátiva na Educação (SBIE 2008)*. Fortaleza: SBC, 2008.

OLIVEIRA, M.; OLIVEIRA, E. Uma metodologia para detecção automática de plágios em ambientes de educação a distância. In: *Anais do V Congresso Brasileiro de Ensino Superior a Distância e 6o Seminário Nacional de Educação a Distância*. Gramado, RS: [s.n.], 2008.

OLIVEIRA, M.; ZANDONADE, E.; OLIVEIRA, E. Uma metodologia para avaliação formativa em um ambiente de ensino e aprendizagem de classificação em biblioteconomia. In: *Anais do IX ENANCIB : Encontro Nacional de Pesquisa em Ciência da Informação*. São Paulo: ENANCIB, 2008.

OLIVEIRA, M. G. D.; CIARELLI, P. M.; OLIVEIRA, E. Recommendation of programming activities by multi-label classification for a formative assessment of students. *Expert Systems with Applications*, Elsevier, 2013.

ORCERO, D. S. The code analyser lclint. *Linux J.*, Belltown Media, Houston, TX, v. 2000, n. 73es, maio 2000. ISSN 1075-3583.

PACHECO, R. *Avaliação formativa continuada do processo educativo em engenharia usando mapas cognitivos difusos*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2005.

PARK, D. H. et al. A Literature Review and Classification of Recommender Systems Research. *Expert Systems with Applications*, v. 39, n. 11, p. 10059 – 10072, 2012.

PEA, R. D.; KURLAND, D. On the cognitive effects of learning computer programming. *New Ideas in Psychology*, v. 2, n. 2, p. 137 – 168, 1984.

PERRAUDEAU-DELBRIEL, M. *Estratégias de aprendizagem: como acompanhar os alunos na aquisição dos saberes*. [S.l.]: ARTMED, 2009.

PERRENOUD, P. *L'Évaluation des Élèves. De la Fabrication de l'Excellence à la Régulation des Apprentissages*. Bruxelles: De Boeck, 1998.

PERRENOUD, P. *Avaliação: da excelência à regulação das aprendizagens – Entre Duas Lógicas*. Porto Alegre, RS: Artmed Editora, 1999.

PERRENOUD, P. *A Pedagogia na escola das diferenças: fragmentos de uma sociologia do fracasso*. Porto Alegre, RS: Artmed, 2001.

PILLAY, N. Developing intelligent programming tutors for novice programmers. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 35, n. 2, p. 78–82, jun. 2003.

PIMENTEL, E. et al. Avaliações adaptativas baseadas no nível de aquisição de conhecimentos do aprendiz. In: *Anais do XVIII Simpósio Brasileiro de Informática na Educação*. São Paulo: Editora e Gráfica Vida, 2007. p. 566–575.

PIMENTEL, E.; FRANCA, V.; OMAR, N. A caminho de um ambiente de avaliação e acompanhamento contínuo de aprendizagem em programação de computadores. In: *II Workshop de Educação em Computação e Informática do Estado de Minas Gerais (WEIMIG 2003)*. Poços de Caldas, MG: Anais do II WEIMIG, 2003. p. 212–213.

POWERS, K. et al. Tools for teaching introductory programming: what works? *SIGCSE Bull.*, ACM, New York, NY, USA, v. 38, n. 1, p. 560–561, mar. 2006.

PU, P.; CHEN, L.; HU, R. Evaluating Recommender Systems from the User's Perspective: Survey of the State of the Art. *User Modeling and User-Adapted Interaction*, Springer Netherlands, v. 22, p. 317–355, 2012.

RAHMAN, K. A. et al. The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique. 2008.

RAPHAEL, H.; CARRARA, K. *Avaliação sob exame*. Campinas: Autores Associados, 2002.

REEK, K. A. The try system -or- how to avoid testing student programs. In: *Proceedings of the twentieth SIGCSE technical symposium on Computer science education*. New York, NY, USA: ACM, 1989. (SIGCSE '89), p. 112–116. ISBN 0-89791-298-5.

REES, M. J. Automatic assessment aids for pascal programs. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 17, p. 33–42, October 1982.

REYNOLDS, G.; STAIR, R. *Prinćpios de Sistemas de Informação*. [S.l.]: Cengage Learning, 2010.

ROMLI, R.; SULAIMAN, S.; ZAMLI, K. Automatic programming assessment and test data generation a review on its approaches. In: *Information Technology (ITSim), 2010 International Symposium in*. [S.l.: s.n.], 2010. v. 3, p. 1186 –1192.

SAIKKONEN, R.; MALMI, L.; KORHONEN, A. Fully automatic assessment of programming exercises. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 33, p. 133–136, June 2001. ISSN 0097-8418.

SANT, J. A. "mailing it in": email-centric automated assessment. In: *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*. New York, NY, USA: ACM, 2009. (ITiCSE '09), p. 308–312.

SANTOS, L. Auto-avaliação regulada: por quê, o quê e como? In: *Avaliação das aprendizagens. Das concepções às práticas. Paulo Abrantes e Filomena Araújo (Orgs.)*. Lisboa: Ministério da educação, Departamento do Ensino Básico, 2002.

SCHAFER, J. et al. Collaborative Filtering Recommender Systems. In: BRUSILOVSKY, P.; KOBSA, A.; NEJDL, W. (Ed.). *The Adaptive Web*. [S.l.]: Springer Berlin / Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4321). p. 291–324.

SCHAFER, J. B.; KONSTAN, J.; RIEDI, J. Recommender Systems in E-commerce. In: *Proceedings of the 1st ACM conference on Electronic commerce*. New York, NY, USA: ACM, 1999. (EC'99), p. 158–166.

SCHAPIRE, R. E.; SINGER, Y. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, Springer Netherlands, v. 37, p. 297–336, 1999.

SCHILDT, H. *C completo e total*. Makron, 1991. Disponível em: <http://books.google.com.br/books?id=AAD4pwAACAAJ>.

SCHILDT, H.; MAYER, R. C. *C completo e total*. [S.l.: s.n.], 2006.

SOARES, J. et al. Instrumentação computacional e realimentação no processo de avaliação para o ensino de matemática: o conhecimento de função real como estudo de caso. In: *Anais do XIX Simpósio Brasileiro de Informática na Educação*. Fortaleza: SBC, 2008.

SONG, Y.; ZHANG, L.; GILES, C. L. Automatic Tag Recommendation Algorithms for Social Recommender Systems. *ACM Trans. Web*, ACM, New York, NY, USA, v. 5, n. 1, p. 4:1–4:31, fev. 2011.

SOUZA, D. de; MALDONADO, J.; BARBOSA, E. Progtest: An environment for the submission and evaluation of programming assignments based on testing activities. In: *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*. [S.l.: s.n.], 2011. p. 1 –10.

STEINBACH, M.; KARYPIS, G.; KUMAR, V. A comparison of document clustering techniques. In: . [S.l.]: KDD workshop on text mining, 2000.

SULEMAN, H. Automatic marking with sakai. In: *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*. New York, NY, USA: ACM, 2008. (SAICSIT '08), p. 229–236.

TANG, Y. Y. C.; POON, C. An approach towards automatic testing of student programs using token patterns. In: *Proceedings of the 17th International Conference on Computers in Education*. Hong Kong: [s.n.], 2009. p. 118–190.

TARDIF, J. *Le transfert de compétences analysé à travers la formation de professionnels*. Lyon: Centre Regional de Documentation Pédagogique de l'Académie de Lyon: In: MEIRIEU, Ph.; DEVELAY, M.; DURAND, C., 1996.

TEAM, R. D. C. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2008.

TORGO, L. A linguagem r-programação para análise de dados. *Lisboa: Escolar Editora*, 2009.

TRUONG, N.; ROE, P.; BANCROFT, P. Static analysis of students' java programs. In: *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2004. (ACE '04), p. 317–325.

TSOUMAKAS, G.; KATAKIS, I.; VLAHAVAS, I. Random k-Labelsets for Multilabel Classification. *Knowledge and Data Engineering, IEEE Transactions on*, v. 23, n. 7, p. 1079 –1089, july 2011.

WANG, T. et al. Ability-training-oriented automated assessment in introductory programming course. *Comput. Educ.*, Elsevier Science Ltd., Oxford, UK, UK, v. 56, n. 1, p. 220–226, jan. 2011.

WU, W. et al. AnalyseC: A Framework for Assessing Students' Programs at Structural and Semantic Level. In: *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*. [S.l.: s.n.], 2007. p. 742 –747.

XU, S.; CHEE, Y. S. Transformation-based diagnosis of student programs for programming tutoring systems. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 29, p. 360–384, 2003. ISSN 0098-5589.

ZAIANE, O. Building a Recommender Agent for E-learning Systems. In: *Computers in Education, 2002. Proceedings. International Conference on*. [S.l.: s.n.], 2002. v. 1, p. 55 – 59.

ZHANG, M.-L.; ZHOU, Z.-H. ML-KNN: A Lazy Learning Approach to Multi-label Learning. *Pattern Recognition*, v. 40, n. 7, p. 2038 – 2048, 2007.

ZIN, A.; FOXLEY, E. Analyse:an automatic program assessment system. *Malaysian Journal of Computer Science*, v. 7, p. 123–142, 1994.