FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Algoritmos incrementais para previsão de variáveis quantitativas usando dados de chamadas móveis

Marta Carolina Madeira Bebiano

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Professor João Pedro Carvalho Leal Mendes Moreira

21 de julho de 2015

Algoritmos incrementais para previsão de variáveis quantitativas usando dados de chamadas móveis.

Marta Carolina Madeira Bebiano

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Professor João Pascoal Faria

Vogal Externo: Professor José Moreira

Orientador: Professor João Pedro Carvalho Leal Mendes Moreira

_____

21 de julho de 2015

Resumo

O fluxo de informação gerado e que circula hoje em dia nas redes de dados locais e transnacionais é enorme. Essa informação tem origem, por exemplo, nos meios de comunicação e resulta da atividade quotidiana dos utilizadores. O registo em massa de informação em bases de dados diversas, de dimensão muitas vezes colossal, e a um ritmo permanente, cria nas organizações dificuldades crescentes de gestão dessa informação, mas que ao mesmo tempo encerra um potencial de valor oculto, muitas vezes mal compreendido e mal explorado. Com o aparecimento deste fenómeno, crescente acumulação de dados, emergiram novos problemas e desafios. Como descobrir, no meio de dados aparentemente irrelevantes, os dados significantes, a informação útil, e os padrões de valor?

Com este trabalho desenvolvem-se algoritmos *online*. Como fonte de dados, as empresas de telecomunicações dispõem de milhões de registos que poderiam utilizar, para prestar novos serviços aos clientes, se encontrassem uma forma clara de o fazer. Assim, com essa informação dominada, poderiam realizar novas tarefas e dar maior rentabilidade ao seu negócio. Nesta tese, como caso de estudo, utilizam-se dados de chamadas telefónicas, para fazer a previsão da duração das chamadas quando estas se iniciam.

As grandes dificuldades de investigação nos domínios, *Data Stream e Data Mining*, resultam portanto da análise dum imenso volume de dados, dos seus elevandos fluxos e da necessidade de processar essa informação. Assim, o principal objetivo deste projeto passa por estudar, implementar e comparar algoritmos incrementais, e pela análise dos resultados obtidos, de modo a poder concluir qual ou quais apresentam o melhor desempenho.

Para lidar com o problema colocado, constroem-se processos de aprendizagem supervisionada, nos quais se utilizam técnicas de regressão. Implementaram-se métodos baseados em distâncias, *k-Nearest Neighbor*, métodos baseados em procura - árvores de decisão, VFDT - *Very Fast Decision Tree*, e métodos de *ensemble homogéneo e ensemble heterogéneo*, onde se combinam vários modelos de modo a ser possível tomar as melhores decisões. Também é usado nesse processo o método agregação de modelos especializados, onde se atribuiem um peso à previsão de cada um dos modelos, efetuando-se depois uma média ponderada no cálculo da nova previsão.

Nesta investigação prática usam-se métodos de avaliação que comparam a eficiência dos algoritmos desenvolvidos, tendo-se concluído que comparativamente o método de agregação é o que apresenta melhor desempenho Apesar disso, a diferença encontrada entre o método de agregação e o método VFDT é praticamente irrelevante.

Palavras-chave: *Data Mining, Big Data, Data Stream, M*étodos de regressão, *Very Fast Decioson Tree, Ensemble Homogéneo e Eensemble Heterogéneo, k-Nearest Neighbor, Aaprendizagem automática,* Agregação de modelos especializados.

Abstract

The information flow that circulates nowadays in both local and transnational data networks is huge. That information originates, for example, in the media or as the result of users' everyday activities. The mass storage of information in massive databases, and at a increasing rate, creates growing difficulties for the organizations in how this information should be handled, but at the same time, it contains an hidden potential, often misunderstood and poorly acknowledged. With the emergence of this phenomenon of the growing accumulation of data, new problems and challenges have also arisen. How can one identify significant data, useful information and patterns of value amongst seemingly irrelevant information?

With this work we have developed online algorithms. Telecommunication enterprises in particular have at their disposal millions of records of precious information which they could use to develop new services for their clients, that is, if they could find a clear way to use it properly. Therefore, having that information under control, they could undertake new tasks and improve the profitability of their businesses. In this case study we used phone calls data to predict the duration of a phone call from the moment it begins.

The difficulties of investigating *Data Stream* and *Data Mining* lie in the analysis of a large amount of data, its flow and the need to process that information. Therefore the main goal of this project became the study, implementation and comparison of incremental algorithms. Another goal is the analysis of the results, in order to conclude which one presented the best performance.

To deal with the problem, we built supervised learning processes which use regression techniques. We implemented distance based methods, *k-Nearest Neighbor* methods, search based methods - decision trees, VFDT - Very Fast Decision Tree methods and methods for heterogeneous *ensemble* and homogeneous ensemble, where several models are combined to make the best decisions. We also used the aggregation method of specialized models which attributes a weight to each model predicted, performing afterwards a weighted average in the calculation of the new forecast.

During the practical study we used assessment methods that compare the efficiency of the developed algorithms. We concluded by comparison that the aggregation method is the best one.
In practice, there is almost no difference between the aggregation method and the VFDT methos.

Keywords: *Data Mining, Big Data, Data Stream, Methods Regression*, *Very Fast Decision Tree, Homogeneous Ensemble, Heterogeneous Ensembles, k-Nearest Neighbor, Machine Learning, Aggregation method of specialized models.*

Agradecimentos

Esta dissertação de Mestrado é seguramente o resultado de um estudo aturado e consolidado ao longo do tempo, e ao longo de todo um percurso académico. Sistematiza resultados, conselhos, experiências, reflexões e orientações de docentes e colegas. É um produto de análise crítica, mas também o cruzamento de ideias e de vivências pessoais e interpessoais, de aquisições múltiplas, que nos dizem que o trabalho é sempre o reflexo de uma partilha. Faz por isso sentido mencionar aqui quem me ajudou nesse percurso.

Em primeiro lugar, quero agradecer ao meu orientador, Professor João Pedro Carvalho Leal Mendes Moreira, pedra angular deste trabalho, a sua atenção, a sua disponibilidade, compreensão, e à forma competente como lidou com as minhas dificuldades, desde o primeiro até ao último dia. Sem a sua ajuda, sem os seus conselhos empenhados teria sido difícil dar cumprimento aos objetivos propostos para esta dissertação. Estes agradecimentos estendem-se também ao Professor Carlos Soares pelos seus conselhos e sugestões.

Agradeço também a toda a equipa do projeto wedo pelos incentivos, conhecimentos e experiência que me transmitiram. Aos amigos e colegas que me acompanharam ao longo deste percurso e que contribuiram também para o meu amadurecimento e enriquecimento pessoal.

Finalmente, a toda a minha família, principalmente aos meus pais, que sempre estiveram presentes ao longo desta caminhada, nos bons e nos maus momentos, prestando sempre um apoio incondicional.

A todos um Bem Hajam.

Marta Carolina Madeira Bebiano

“Machines take me by surprise with great frequency.”
Alan Turing

Conteúdo

CONTEÚDO

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE TABELAS

Abreviaturas e Símbolos

BD	Big Data
DCBD	Descoberta de conhecimento em base de dados (Knowledge Discovery in Database)
DM	Data Mining
IA	Inteligência artificial
IFN	Information Network
K-NN	K-Nearest Neighbor
MAD	Mean Absolute Deviation
MOA	Massive Online Analysis
MSE	Mean Squared devision
VFDT	Very Fast Decision Tree
WEKA	Waikato Environment for Knowledge Analysis

Capítulo 1

Introdução

Neste capítulo apresentam-se as bases e o contexto motivacional em que decorreu a investigação "Algoritmos incrementais para previsão de variáveis quantitativas usando dados de chamadas móveis" sobre algoritmos incrementais. Aplicado ao caso de estudo − previsão de duração de chamadas − enumeram-se os objetivos visados na dissertação. Finalmente, relata-se o trabalho desenvolvido e descreve-se a estrutura do documento.

1.1 Contexto/Enquadramento

A velocidade a que as sociedades atuais produzem informação, de forma praticamente contínua, colocam desafios enormes às organizações, designadamente: ao nível do tratamento dessa informação, sua gestão e transformação em conhecimento**.**

Atendendo ao facto deste trabalho, "Algoritmos incrementais para previsão de variáveis quantitativas usando dados de chamadas móveis", usar um grande volume de dados, os dados serem diferenciados, e não haver uma base determinística de avaliação, foram usadas técnicas de Data Mining/aprendizagem automática[1], designadamente, algoritmos de aprendizagem incremental adequadas a problemas de BIG DATA (referente, neste caso, à velocidade a que os dados chegam para tratamento).

O termo DM pode ser mencionado como forma de extração de conhecimento dos dados, como arqueologia de dados, colheita de informação e *Data dreadging* [Fayyad 1996]. DM é também aplicação de algoritmos, envolvendo inteligência artificial (*IA*), com o objetivo de reconhecer padrões ocultos nos dados, sem a execução dos passos adicionais do processo DCBD[2] (tais como a inclusão de conhecimento anterior e interpretação correta dos resultados).

O *DM* é um campo interdisciplinar, que emergiu da interseção entre várias áreas vários saberes, principalmente aprendizagem máquina (uma subárea da inteligência artificial), estatística e tratamento de base de dados.

Existem em *DM*, principalmente, dois tipos de estratégia [Berry00], que são: o modelo de verificação hipótese (verificar a hipótese colocada pelo investigador/utilizador) e o modelo de reconhecimento de padrões (procurar, descobrir, novos padrões ocultos no intrincado de dados).

[1] *Machine Learning*, termo usado em inglês. Com o desenvolvimento da IA (Inteligência Artificial) incorporaram-se nas tecnologias computacionais metodologias provenientes da análise estatística clássica, que se traduziram em avanços significativos nos processos preditivos de aprendizagem máquina, também conhecidos por aprendizagem automática.
[2] Encontrar, descobrir tendências e associações escondidas num grande volume de dados, de forma a retirar conhecimento útil.

Neste trabalho pretendeu-se apresentar algoritmos online de modo a poder tratar os dados em tempo útil.

Muitas empresas, em particular as de telecomunicações, dispõem de milhões de registos com informação preciosa que poderiam utilizar na rentabilização dos seus negócios. Infelizmente essa informação útil, está envolta em muita informação irrelevante que importa extrair [Fayyad96]. Por outro lado a informação com valor não é a mesma para empresas diferentes, nem a forma de a tratar e valorizar é igual.

Como identificar informação útil que revele tendências nos hábitos dos clientes, permitindo à empresa, eventualmente, prestar novos serviços aos clientes? Como tratar a informação de forma a adequar os serviços já existentes a novas necessidades? Como definir campanhas de marketing e identificar estratégias mais lucrativas? Estas e outras perguntas podem ser retiradas dos dados, bem como as suas respostas, se a metodologia adequada for encontrada. No entanto, atendendo ao volume de dados, pode não ser fácil e demorar muito tempo a obter respostas a estas perguntas.

Assim, com esse propósito, utilizou-se este caso de estudo para fazer a previsão da duração de uma chamada quando esta se inicia.

1.2 Motivação

Abordagens de DM, Big Data (BD) e Data Stream, através de IA, são temas onde muita investigação tem sido efetuada nos últimos anos mas onde os desafios continuam a ser enormes e a maior parte do trabalho parece estar ainda por fazer. Fez sentido portanto colocar o esforço numa área promissora, em desenvolvimento, multidisciplinar e com considerável potencial no tratamento de grande quantidade de informação. Daí decorreu um trabalho sobre algoritmos especializados no tratamento de grande quantidade de dados, tratamento incremental e no reconhecimento de padrões que conduziram à resolução de problemas na previsão à priori da duração de chamadas telefônicas.

Uma das áreas onde são geradas maiores quantidades de informação, são, sem dúvida, nas empresas de telecomunicações, setor que atingiu enorme crescimento e importância nos últimos vinte anos [Sferra03]. Sendo a concorrência entre empresas desse setor muito grande e onde se procura, intensamente, formas de cativar os clientes, nomeadamente através da criação de pacotes supostamente vantajosos para o utente, o conhecimento do perfil do cliente pode assumir alguma importância para as empresas. Apresentar condições que cativem atuais ou potenciais clientes, sem prejudicar os interesses da empresa, é uma tarefa difícil que pode ser melhorada com boas previsões sobre consumos do cliente. Vem assim, no seguimento destas considerações, o caso de estudo sobre a previsão da duração de chamada móvel ao ser iniciada.

1.3 Problema e Objetivos

Com a elaboração desta tese pretendeu-se contribuir para o conhecimento do modo como transformar uma grande base de dados em informação. Informação que seja relevante, que acrescente mais valia aos dados e que traga mais rentabilidade aos operadores.

No que se refere à metodologia, importa em primeiro lugar caracterizar o conceito de "Aprendizagem" entendido como aquisição de nova informação, de forma estruturada, e que permita aplicação a novas situações de conhecimento. Numa aprendizagem baseada em exemplos, utiliza-se o conhecimento das previsões e do valor verdadeiro de outras instâncias para efetuar uma previsão para a instância atual. Este é o objetivo do DM, processo computacional de reconhecimento de padrões em extensos conjuntos de dados, utilizando aprendizagem automática, estatística, sistemas de bases de dados e algoritmia computacional. Esses padrões permitem fazer a previsão de dados futuros (previsão).

Na previsão[3] os problemas podem ser tratados como processo envolvendo classificação[4] dos dados ou processos de regressão. Neste trabalho aplicou-se um processo de regressão à determinação da variável dependente. Estes algoritmos seguem o paradigma da aprendizagem supervisionada[5]. As tarefas supervisionadas distinguem-se pelo tipo de rótulos aplicados aos dados: discreto[6], no caso de classificação; e contínuo[7], no caso de regressão Figura 1.1.

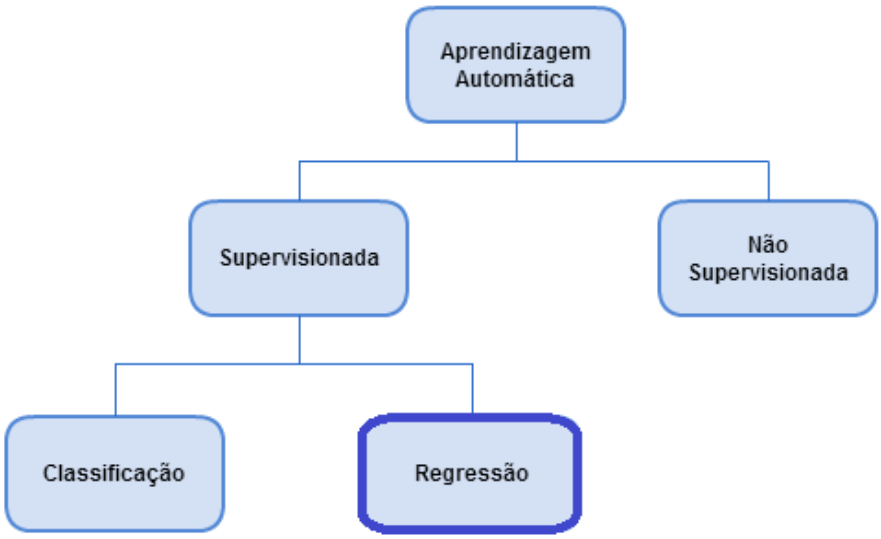

Figura 1.1: Processos de Aprendizagem automática

[3] Procura de padrões que permitam prever o futuro
[4] Procura-se uma função que faça o mapeamento dos dados em classes pré-definidas
[5] Paradigma de aprendizagem automática, no qual os exemplos são associados a classes ou a uma variável dependente. Em oposição à aprendizagem não supervisionada, os exemplos não são associados a classes ou a uma variável dependente, como por exemplo na segmentação.
[6] São dados que podem assumir apenas um número finito ou infinito contável de valores (valores inteiros). Geralmente são o resultado de contagens.
[7] São dados que podem corresponder a qualquer valor num intervalo de números reais.

O principal objetivo deste projeto passou por estudar, implementar e comparar algoritmos incrementais. Estes algoritmos foram aplicados ao caso de estudo previsão da duração de chamadas, quando esta se inicia. Identificaram-se os melhores algoritmos que se podiam adaptar a este problema de regressão.

1.4 Estrutura do documento

O relatório apresenta cinco capítulos. O primeiro capítulo de introdução, capítulo 1, faz uma apresentação do trabalho, define e contextualiza o problema e traça os objetivos. O capítulo 2, apresenta e descreve um conjunto de algoritmos incrementais, focando o respectivo estado da arte em que se encontram. Neste capítulo também se referem as ferramentas que podem ser usadas com esses algoritmos. No capítulo 3 descreve-se a tipologia dos dados de trabalho e as metodologias usadas no desenvolvimento desta Dissertação. Este capítulo termina com o detalhe das tecnologias utilizadas. O capítulo 4 apresenta os métodos de avaliação utilizados na comparação dos algoritmos, os resultados encontrados e as conclusões retiradas da investigação levada a cabo ao longo desta tese. O capítulo 5 apresenta uma síntese das conclusões finais da Dissertação e o trabalho que se pode perspectivar para o futuro.

Capitulo 2

Revisão Bibliográfica sobre algoritmos incrementais

Neste capítulo é descrito o estado da arte em que se encontram os algoritmos utilizados nesta investigação. São apresentados algoritmos de regressão e também descritas as ferramentas onde podem ser usados.

2.1 Algoritmos Incrementais

2.1.1 Definição e objetivos

Uma das características principais dum algoritmo incremental, é que é capaz de actualizar o modelo preditivo sempre que existam novos dados disponíveis. Um exemplo simples de algoritmo incremental pode ser dado para o "cálculo da média de n valores". Este caso está retratado, na Figura 2.1, com um algoritmo não incremental, e na Figura 2.2, com um algoritmo incremental.

No algoritmo não incremental, a média só fica disponível depois de serem lidos todos os valores.

```
// Algoritmo não incremental para o cálculo da média de x_1, x_2, ..., x_n .
soma = 0;
para i=1 até n faça {
        ler(x_i);
        soma = soma+x_i;
}
media = soma/n;
escrever(media);
```

Figura 2.1: Exemplo de um algoritmo não incremental

No algoritmo incremental a média incorpora os valores sucessivamente adicionados.

```
// Algoritmo incremental para o cálculo da média de x_1, x_2, ..., x_n
media = 0;
i = 0;
repita {
        i=i+1;
        ler(x_i);
        media = (media*(i-1) + x_i/i)i;
        parar = testar_paragem();
} até (parar=sim);
escrever(media);
```

Figura 2.2: Exemplo de um algoritmo incremental

No algoritmo incremental tem-se, em qualquer iteração, um valor disponível para a média, sendo essa média calculada a partir da média da iteração anterior com o novo valor lido. Em cada iteração testa-se uma condição de paragem para terminar o ciclo.

Para que um algoritmo se possa considerar incremental, este tem de ter pelo menos três restrições: a primeira impõe que se possa interromper o processo para responder a qualquer questão, a segunda exige que o tempo de processamento de cada novo caso se mantenha constante, independentemente do número de casos que o algoritmo já analisou e como última regra, destinada aos algoritmos que não retenham os dados, manterem em memória um resumo completo desses dados ou das suas possíveis alternativas [Langley95].

Os algoritmos incrementais são especialmente vantajosos quando se trabalha com grande volume de dados, com muitas instâncias, frequentemente milhões de instâncias.
Admitindo que se retirava uma amostra de mil instâncias poder-se-ía trabalhar sobre essa amostra. De seguida retirava-se outra amostra de mil instâncias, depois outra e assim sucessivamente. Este processo é impraticável quando se dispõem de milhões de instâncias. No entanto, esta amostragem, pode ser implementada através de uma técnica conhecida por Sliding Windows.
Esta técnica consiste na definição de um número de observações N, que representa o número de observações que são analisadas de cada vez que surge uma nova observação. Assim, para a observação de ordem N, O_N representa os elementos da amostra $-$ O_1, O_2,..., O_N, quando surge a observação seguinte, de ordem N+1, $O_{(N+1)}$ será constituída pelos elementos O_2, O_3,..., O_N, O_{N+1} ignorando-se a observação O_1. De seguida, ao surgir a observação de ordem N+2, $O_{(N+2)}$, seguindo o mesmo critério, usam-se as observações O_3, O_4,..., O_{N+1}, O_{N+2} e assim sucessivamente, ignorando sempre a observação mais antiga na Sliding Windows [Babcock02].

2.1.2 Vantagens e Desvantagens

A principal vantagem dos algoritmos incrementais é ter o resultado disponível em qualquer nível de iteração pretendida. Não se impõem restrições de análise aos novos dados (quer em tempo, quer em número) e, como se disse, a disponibilidade do algoritmo é permanente [Langley95]. Podem enumerar-se outras vantagens, como: aproximar-se da forma de aprendizagem humana, ser rápido na resposta e poupar memória de computação. A qualidade do algoritmo pode medir-se em função da precisão obtida a partir dum menor número de valores lidos.

A principal desvantagem destes algoritmos incrementais resulta do facto do valor final encontrado para o modelo ser apenas uma aproximação, não consistindo num cálculo riguroso dos coeficientes de regressão. Alem disso, a ordem pela qual aparecem as instâncias influencia os resultados produzidos pelos algoritmos incrementais [Pinto05].

2.2 Métodos regressão

Os métodos de regressão têm por objetivo fazer uma previsão duma variável contínua, variável dependente, conhecidos os valores assumidos por outras variáveis, variáveis independentes [Berk05].
Nas subseções seguintes apresentam-se e descrevem-se com algum detalhe os métodos de regressão estudados.

2.2.1 Algoritmo K-NN (K-Nearest Neighbour)

K-Nearest Neighbour ou k-Vizinhos mais próximos é um método baseado em distâncias.

K-Nearest Neighbour funciona da seguinte forma:

1. Calcula a distância entre o exemplo de teste e cada exemplo de treino.
2. Escolhe os k objetos do conjunto de treino mais próximos do ponto teste x_t, sendo F(x_t) a previsão ao objeto teste, Figura 2.3.
3. Na regressão há duas formas de agregar, dependo da função custo usada.

   Se a função custo for para minimizar [Kramer11] o:

   - Erro quadrático, então F(x_t) <- média(f(x_1),f(x_2),...,f(x_k))
   - Desvio absoluto, então F(x_t) <-mediana(f(x_1),f(x_2),...,f(x_k))

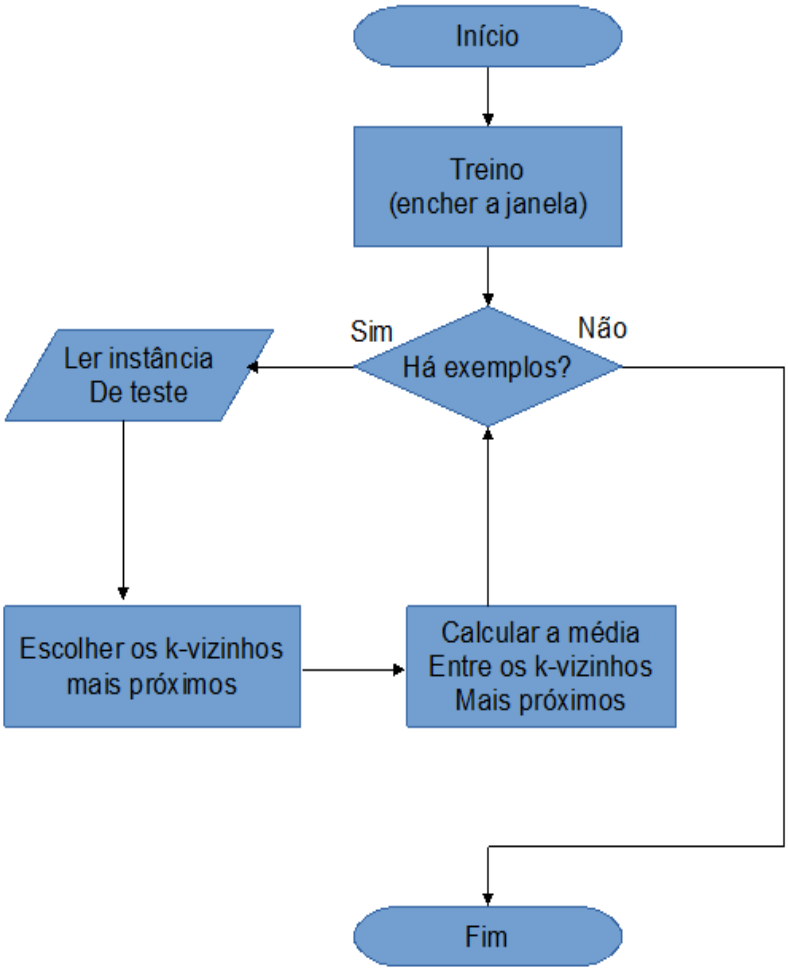

Figura 2.3 Fluxograma do algoritmo KNN

A média pode ser ponderada associando um peso à contribuição de cada vizinho. Esta contribuição é pesada de forma inversamente proporcional à distância ao ponto de teste.

$$F(x_t) = \frac{1}{s}\sum_{i=1}^{k} w_i \, f(x_i) \text{ sendo } s=\sum_{i=1}^{k} w_i \text{ com } w_i = \frac{1}{d(x_t,x_i)} \qquad (2.1)$$

onde w_i é peso associado e k é um parâmetro do algoritmo, sendo $f(x_i)$ o valor verdadeiro para a instância x_i.

Existem várias métricas para o cálculo da distância entre dois pontos.
São frequentemente usados casos particulares da distância de Minkowski.

Um desses casos é a distância Euclidiana, cuja distância entre dois pontos (x_i e y_i) corresponde à raíz quadrada do somatório dos quadrados das diferenças entre valores de x_i e y_i para todas as variáveis independentes i= (1, 2,3,4,...,k).

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \quad (2.2)$$

Outro caso particular é a distância Manhattan, cuja distância entre dois elementos (x_i e y_i) corresponde à soma dos valores absolutos das diferenças entre os valores das variáveis (i = 1, 2... k).

$$\sum_{i=1}^{k}|x_i - y_i| \quad (2.3)$$

Em geral, na distância Minkowski, a distância entre dois elementos (x_i e y_i) é a soma dos valores absolutos das diferenças entre os valores das variáveis (i = 1, 2... k), sendo q um parâmetro dessa distância.

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q} \quad (2.4)$$

2.2.2 VFDT (Very Fast Decision Tree)

O algoritmo VFDT (*Very Fast Decision Tree*) baseia a construção da árvore de decisão num processo de aprendizagem, onde utiliza um pequeno conjunto de exemplos para fazer o teste de selecção que permite a divisão da árvore em determinado nó de decisão.
Este algoritmo recebe sequencialmente cada instância de treino.
Uma árvore de decisão é construída recursivamente substituindo as folhas por nós de decisão [Gama10].
Cada folha armazena as estatísticas suficientes sobre os valores dos atributos.

Estas estatísticas são as requeridas para estimar a função de avaliação que calcula o mérito dos testes de divisão.

Cada novo caso de teste atravessa a árvore, partindo da raíz, escolhendo o ramo que corresponde ao valor do seu atributo, e, atingindo uma folha, actualiza os dados estatísticos nessa folha.
Quando essa folha acumula informação suficiente sobre um número mínimo de exemplos, é avaliada com base no valor dos atributos cada possível condição. Se os dados existentes na folha favorecerem um determinado teste, então essa folha é transformada num nó de decisão do referido teste com um ramo descendente para cada um dos possíveis valores do atributo analisado [Yang11].

Os nós de decisão apenas mantêm a informação sobre o teste de divisão instalado.
O que carateriza o algoritmo VFDT, fundamentalmente, é o modo como decide a incorporação de um novo atributo num nó da árvore, a forma como decide qual é esse novo atributo, e qual o número suficiente de exemplos a analisar.
O algoritmo usa o limitador de Hoeffding pela equação 2.5, onde R é a amplitude.

$$\varepsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2n}} \quad (2.5)$$

- ε representa o limite de Hoeffding
- $1 - \delta$ é a probabilidade de se escolher corretamente o melhor atributo
- n é a quantidade de exemplos da amostra

Seja H uma função de avaliação, at_a o atributo com maior H, at_b o segundo maior H e ΔH = H(at_a)-H(at_b) a diferença entre os dois melhores atributos. Então $\Delta H > \varepsilon$ diz que at_a é o melhor atributo, o que implica que a folha da árvore seja transformada em nó de decisão, o que divide o exemplo pelo valor de at_a. Caso contrário o algoritmo prossegue processando mais exemplos.

A figura 2.4 mostra o fluxograma do algoritmo VFDT. No diagrama, xi representa atributos independentes da instância i, yi caracteriza a duração da instância i, f é a folha, H(a1) a heurística do melhor atributo, H(a2) a heurística do segundo melhor atributo e HT a Hoeffding Tree.

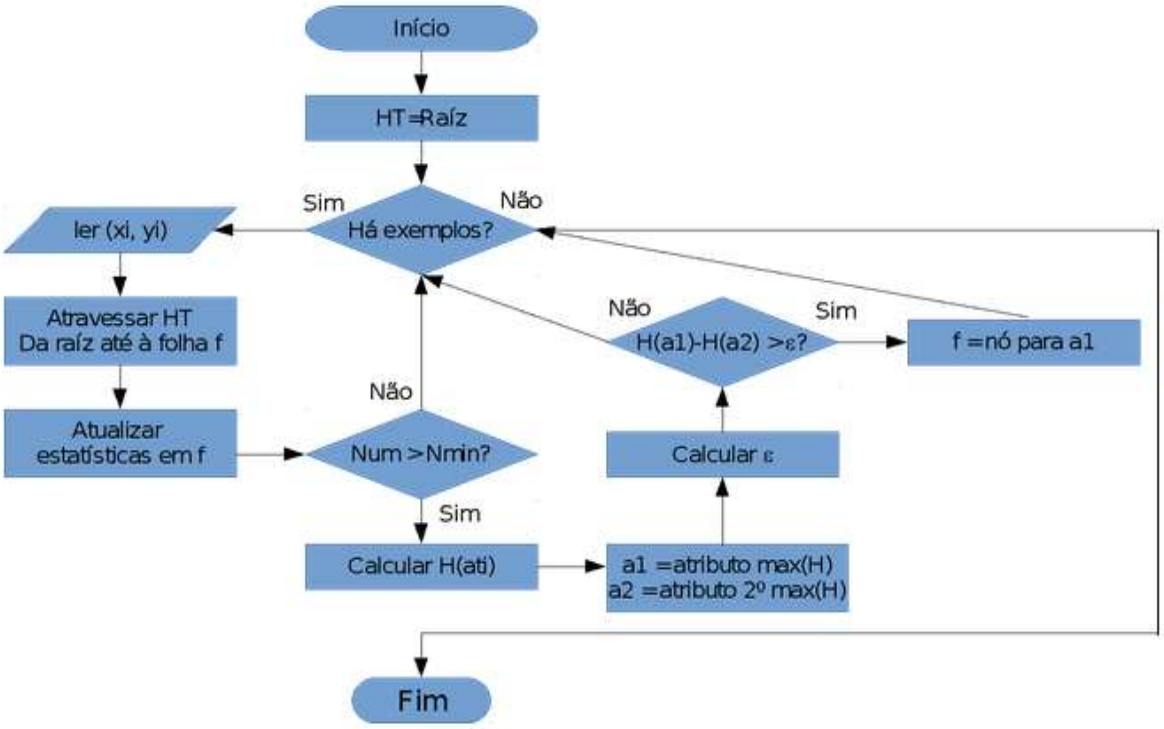

Figura 2.4 Fluxograma do algoritmo VFDT.

2.2.3 OLIN

O algoritmo Information Network – IFN, é um método com árvores de decisão usado para minimizar o número de atributos utilizados em previsão. O algoritmo OLIN é uma generalização deste método para fluxo de dados contínuo e dinâmico. Utiliza métodos de decisão incremental, permitindo a atualização da árvore de decisão a partir apenas das novas instâncias, sem ter que processar por inteiro todo o conjunto de dados [Gama10].

2.2.4 Ensemble Learning

Na aprendizagem em conjunto, o termo modelos múltiplos é utilizado para identificar um conjunto de preditores cujas decisões individuais são combinadas ou agregadas de alguma forma para efetuar previsões em novos exemplos.

2.2.4.1 Ensemble Homogéneo

Os algoritmos de bagging referidos, fazem parte de um método mais geral conhecido como ensemble. Usam vários métodos para efetuar previsão e apresentam os resultados, com base nas previsões obtidas em cada um dos outros métodos. Quando esses métodos dizem respeito todos ao mesmo algoritmo, fala-se em ensemble homogéneo.

Bagging Batch

A técnica denominada *Bootstrap Aggregating* [Breiman96]. *Bagging* produz replicações do conjunto de treino por amostragem com reposição. Isto é, gera vários conjuntos a partir de um conjunto único, por processo de aprendizagem. Os novos conjuntos geralmente possuem o mesmo tamanho, incorporando alguns elementos repetidos, enquanto outros podem desaparecer.

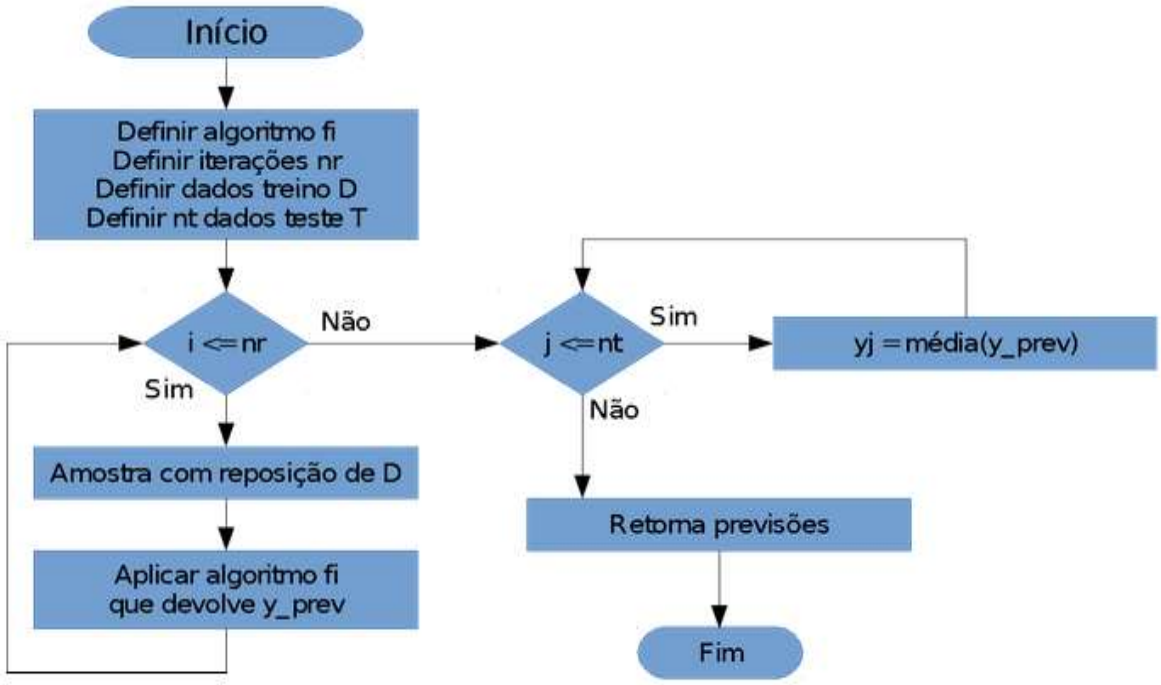

Figura 2.5 Fluxograma do algoritmo Bagging.

Na figura 2.5, *nr* designa o número de vezes a aplicar o processo repetitivo e nt o número de instâncias de teste. O algoritmo produz réplicas e aplica um algoritmo incremental até ao final das iterações. Em seguida passa para a fase de teste onde faz a média dos novos conjuntos e retorna as previsões.

Bagging Online

No Bagging existe um conjunto de treino T com n instâncias. Este cria M modelos cada um com n instâncias. Escolhe-se uma amostra de tamanho n, com reposição, sobre n instâncias [Oza05].
Considerando a variável aleatória X: "número de vezes que a instância x_i, a instância a testar, foi escolhida, na amostra". Sabe-se que X segue uma distribuição binomial, X ~ Bi(n ,$\frac{1}{n}$). No entanto, para valores elevados de n, isto é quanto n tende para +∞ , X ~ Poisson(λ=n×$\frac{1}{n}$) , ou seja, X ~ Poisson(λ=1) [Wang&Pineau03].
Assim, no Bagging Online, ao testar a instância x_i, gera-se aleatoriamente, baseado na distribuição de Poisson de parâmetro 1, o número de vezes que essa instância será escolhida em cada um dos modelos.

Na Figura 2.6 está um pseudocódigo retirado do artigo Online Bagging and Boosting [Oza05], onde h é o conjunto de M modelos e d é o exemplo mais recente a ser testado e L_o são as previsões.

OnlineBagging$(\mathbf{h}, d)$

For each base model h_m, $(m \in \{1, 2, \ldots, M\})$ in the ensemble,

- Set k according to $Poisson(1)$.
- Do k times
  $h_m = L_o(h_m, d)$

Figura 2.6: Pseudocódigo do algoritmo Bagging Online retirado do artigo Online Bagging and Boosting [Oza05].

2.2.4.2 Ensemble Heterogéneo

Ao contrário do ensemble homogéneo, o ensemble heterogéneo combina modelos gerados por diferentes algoritmos para aumentar a diversidade e a precisão [Qiang10].

2.2.5 Agregação de modelos especializados

O modelo de agregação de modelos especializados é proposto por Devaine, Gaillard, Goude e Stoltz [Gaillard14].

Considere-se *m* o número de modelos.

Para cada instante t, e para cada modelo j∈{1,. .. ,m} existe uma previsão y_t^j, e existe um peso w_t^j.

Os pesos dependem do instante t e considere-se s_t a soma dos pesos para o instante t. A previsão agregada é a média ponderada,

$$\hat{y}_t = \frac{1}{S_t} \sum_{j=1}^{m} w_t^j \hat{y}_t^j \quad (2.6)$$

Para todo o instante t, y_t é o valor verdadeiro da variável dependente [Devaine12] e define-se ainda,

$$q(t,j) = (\hat{y}_t^j - y_t)^2 \quad (2.7)$$

No instante t=1 todos os pesos são iguais a $\frac{1}{m}$. Isto é o algoritmo começa com um vetor de pesos convexos uniforme [Oliveira14].

Para cada instante t>1, define-se

$$w_{t+1}^j = w_t^j e^{-\eta q(t,j)} \frac{S_t}{\sum_{k=1}^{m} w_t^k e^{-\eta q(t,k)}} \quad (2.8)$$

Onde o parâmetro η introduzido pelo utilizador tem de ser maior que zero.

2.3 Análise das Ferramentas que podem ser usadas com os algoritmos

2.3.1 R

R é um ambiente de programação para estatística e análise de dados. Está disponível para plataformas Unix/Linux, MacOs e Windows. Com um leque variados de pacotes e a possibilidade de integrar programas escritos noutras liguagens, R, facilita ao utilizador a manipulação, cálculo e representação de dados. É um projecto GNU e foi criado na Universidade de Auckland na Nova Zelândia. Trata-se de um projecto open source que pode ser encontrado em http://www.r-project.org/.

2.3.2 Cubist

A ferramenta Cubist é frequentemente utilizada para gerar modelos baseados em regras para problemas de regressão. É uma ferramenta de fácil utilização e na compreensão dos modelos gerados, além de suportar a análise de Bases de Dados com milhares de registos [Research12].

2.3.3 Weka (*Waikato Environment for Knowledge Analysis)*

Weka é uma ferramenta open-source implementada em Java, que possui diversas técnicas de aprendizagem de máquina já implementadas [Waikato13]. Disponibiliza também, diversos algoritmos de pré-processamento de dados, bem como de análise resultados.

Foi criada na Universidade de Waikato na Nova Zelândia [Witten05].
Este software pode ser utilizado para problemas de regressão e permite que sejam desenvolvidos modelos de aprendizagem máquina.

2.3.4 Moa (Massive Online Analysis)

Moa é uma framework para data streams, desenvolvido pela Universidade de Waikato na Nova Zelândia. Inclui ferramentas para avaliação e uma coleção de algoritmos de aprendizagem de máquina. Está relacionado com o projecto WEKA, que também está implementado em Java [Bifet10].
A Figura 2.7 mostra o separador da regressão da interface gráfica do moa.

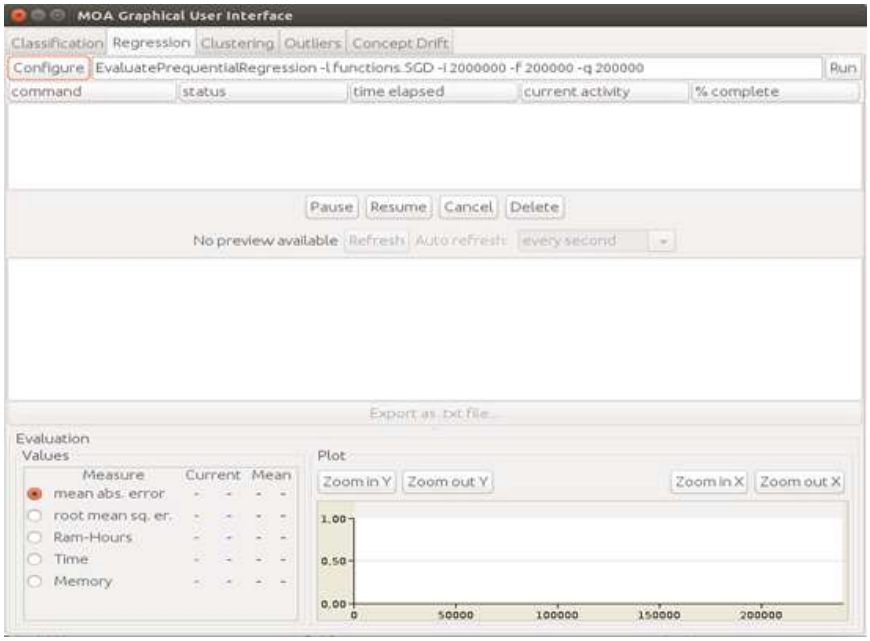

Figura 2.7: Separador da regressão da interface gráfica do moa

Capitulo 3

Implementação

Na implementação utilizou-se sliding windows, dispondo os dados na forma de séries temporais. Trabalhou-se sobre *dados* previamente armazenados numa base de dados, ligada ao projeto WEDO.
Implementaram-se e aplicaram-se os algoritmos K-NN, VFDT, ensemble homogêneo e ensemble heterogéneo e usou-se o método agregação de modelos especializados. Para avaliar os diferentes algoritmos usou-se uma métrica de erro. Calculou-se o erro quadrático médio, MSE, e a distância absoluta média, MAD, para cada um dos algoritmos e dessa forma criou-se uma base objetiva para a comparação da eficiência dos algoritmos.
Para fundamentar estatisticamente as diferenças obtidas entre os erros dos vários algoritmos usou-se o teste não paramétrico de Friedman

3.1 Dados

Os dados utilizados são provenientes do projeto WEDO, e chegaram ao trabalho completamente anonimizados. Os dados obedecem a uma estrutura, representada na tabela 3.1, e que assenta num conjunto de campos.

Tabela 3.1: Estrutura de Dados

Campo	Dados	Opções	Observações
1	DATE	Formato YYMMDD	Data da chamada
2	TIME	Formato HHMMSS	Hora da chamada
3	DURATION	Formato SS	Duração em segundos
4	MSISDN	Formato N	Número codificado
5	DIRECTION	Formato I/O	
6	OTHER_MSISDN	Formato N	Interlocutor
7	CONTRACT_ID	Formato N	Identificação contrato/ “Código cliente”
8	OTHER_CONTRACT_ID	Formato N	Outro operador
9	DESTINATION	Formato L/I	Local/Internacional

10	DROPPED_CALL	Formato Y/N	Chamada caiu (Sim/Não)
11	CALL_TYPE	Formato FI/MO/ON/SV/OT	Fixo/móvel/ON-Net/voicemail ou serviços valor acrescentado/restantes serviços
12	START_CELL_ID	Formato N	Identifica Célula de Origem chamada
13	END_CELL_ID	Formato N	Identifica Célula de Destino chamada
14	VOICMAIL	Formato Y/N	Vocemail (Sim/Nâo)

A estrutura enquadra os campos representados na tabela e têm a seguinte descrição: campo **DATE,** data da chamada com o formato de "YYMMDD" sendo o "YY" ano, "MM" mês e "DD" dia, o campo **TIME,** a hora a que foi efetuada a chamada, seguindo o formato de "HHMMSS", sendo "HH" horas, "MM" minutos e "SS" segundos, o campo **DURATION,** a duração da chamada em segundos, o campo **MSISDN,** número (anonimizado) do telefone "em causa" (devido a efeitos legais), o formato deste campo é N ou seja é númerico. Se a chamada for de *Incoming*, o data valor corresponde ao número que recebe a chamada, se for de *Outgoing*, corresponde ao número que está a iniciar a chamada. A informação sobre se a chamada, campo **DIRECTION,** é de entrada – I (*incoming*) ou de saída O – (*outcoming*) deve ficar registada, bem como o campo **OTHER_MSISDN**, número do "*outro*" cliente - o originador da chamada (no caso de Incoming) ou destinatário (no caso de Outgoing). O campo **CONTRACT_ID**, "*Código Cliente*", identifica o contrato *id*, ou campo **OTHER_CONTRACT_ID,** identifica o caso do interlocutor não pertencer ao operador. Outro dado de chamada importante na análise é o campo **DESTINATION,** que regista o destino, como sendo local (L) ou internacional (I). Outras componentes do vector de dados de chamada indicam se a chamada caiu, campo **DROPPED_CALL** (Yes or No) e se foi para voicemail ou não (Yes or No) é o campo **VOICMAIL**. O tipo de chamada, campo **CALL_TYPE**, identifica se a chamada em que a outra parte pertence a uma rede fixa **FI**, rede móvel **MO**, VoiceMail ou serviços de valor acrescentado **SV**, restantes serviços **OT** e on-net **ON** (chamadas em que o originador e destino são ambos da rede deste operador).
Para além deste dados ainda existe o campo **START_CELL_ID** e o campo **END_CELL_ID** que correspondem ao originador e ao receptor da chamada, respectivamente.

Os dados originais estão em ficheiros com formato txt, organizados por dias, ou seja com o mesmo valor em todas as instâncias por dados definidos. Como o *moa* só consegue ler em formato *arff* procedeu-se à conversão dos ficheiros da extensão *txt* para a extensão *arff* [Bifet12].

Essa conversão processa-se na classe Converter. Primeiro lêem-se os dados do ficheiro antigo e guardam-se num novo ficheiro. Assim, primeiro guardam-se as variáveis independentes (data, hora, anónimo, outro, cliente, destino, caiu, VoiceMail) e por último a variável dependente (duração).
O Moa obriga a ter as primeiras linhas com os nomes dos campos e de seguida os dados, como se indica na Figura 3.1:

```
@relation Dia_1
@attribute Data numeric
@attribute Hora numeric
@attribute Anonimo numeric
@attribute Outro numeric
@attribute Cliente numeric
@attribute Sentido numeric
@attribute Tipo numeric
@attribute Destino numeric
@attribute Caiu numeric
@attribute VoiceMail numeric
@attribute Duracao numeric
@data
```

Figura 3.1: Formato obrigatório do moa

Os dados surgem após a palavra-chave "@data". Na conversão foram alterados alguns campos para formatos apropriados. A data foi passada para o dia do ano respectivo, entre 1 e 366 dias, a hora passou a minutos, isto é o valor das horas foi multiplicada por 60, e somados os minutos que lá estavam, o sentido da chamada efectuada passou a 0 e a chamada recebida passou a 1, o tipo FI passou a 1, o MO passou a 2, o ON passou a 3, o OT passou a 4 e o SV passou a 5. No destino, local passou a 0 e internacional passou a 1. Nos campos caiu e voicemail passou a 0 se for não e 1 se for sim.

3.1.1Setup experimental -Sliding windows

Sliding windows é uma técnica para treinar o modelo com dados na forma de séries temporais. A janela corresponde ao período temporal utilizado para cada caso de treino.
Na implementação do *sliding windows* usou-se o método **capacidadeJanela()** onde se retorna o número máximo de instâncias que podem existir na janela e o método **tamanhoJanela()** que retorna o número de instâncias na janela atual.
O tamanho da janela utilizada foi de mil instâncias tendo um salto de uma instância de cada vez.
Resumidamente o objectivo do *sliding windows* é encontrar uma dependência, $y=f(x)$, que possa estimar o valor de y quando for dado o valor de x, sendo y igual à duração da chamada e x o vetor com o conjunto de atributos registados [Babcock02].

3.2 Algoritmos Implementados

O código está implementado de maneira a poder ser aplicado a problemas gerais de regressão para os algoritmos incrementais.

3.2.1 KNN

Um algoritmo utilizado foi o *K-NN*, pelo facto do algoritmo de treino ser simples, já que consiste em memorizar as instâncias de treino, e também por poder ser usado em problemas de natureza complexa. É um algoritmo incremental uma vez que incorpora na memória os novos exemplos de treino, quando estes são disponibilizados, incrementando a solução com este caso.

Este algoritmo está implementado na classe ***SubKNN***. Esta classe usa como parâmetros: o tamanho janela − n, o número vizinhos − k, e o algoritmo da distância Minkovski − p. Quando o parâmetro *p* é igual a 1 corresponde à distância Manhattan, quando *p* é igual a dois corresponde à distância Euclidiana e para outros valores usa-se a distância Minkowski.

O método ***trainOnInstance()*** é o método onde se treina o modelo. O método ***getVotesForInstanceImpl(Instance instância)*** é o método que faz a previsão para cada instância de teste.

3.2.2 VFDT

Outro algoritmo utilizado foi o VFDT. Relativamente a outros algoritmos que usam árvores de decisão, o VFDT tem a vantagem de gerar árvores mais rápidas de construir devido à utilização dum limitante − *Hoeffding bound* [Yang11].

Este método está definido na classe ***SubFIMTDD*** que faz uma extensão à classe ***FIMTDD*** do *API* do moa. A classe herda o método ***getVotesForInstance()*** da classe ***FIMTDD*** do *API* do moa e o método ***trainOnInstance()*** da classe **AbstractClassifier**.

Todos os regressores devem extender a classe **AbstractClassifier**. No caso particular dos regressores devem ainda implementar o interface Regressor.

3.2.3 Ensemble Learning

3.2.3.1 Ensemble homogéneo

O *Ensemble* é um método que efetua previsão usando vários modelos. Quando esses modelos dizem respeito ao mesmo algoritmo diz-se que o *ensemble* é homogéneo, sendo heterogéneo no caso contrário.

Usa-se Ensemble online pela necessidade de utilizar um algoritmo incremental sobre grande quantidade de dados.

Neste trabalho testou-se um ensemble sob modelos de algoritmo KNN, fazendo variar os parâmetros relativos ao número de vizinhos e à distância utilizada.

3.2.3.2 Ensemble heterogêneo

No Ensemble Heterogénio definiu-se o número de modelos, M igual a quatro. Cada chamada telefónica corresponde a uma instância e as instâncias são tratadas sequencialmente pela ordem que aparecem no ficheiro de dados. Quando uma instância é lida ela é tratada independentemente por cada um dos algoritmos.

A classe **TestaEnsembleHeterogenio** faz a gestão da sucessiva leitura de cada instância enviando-a para a classe **EnsembleHeterogenio**, que usa cada um dos M modelos para tratar a instância. Há dois tratamentos fundamentais: treinar e testar. Treinar corresponde a guardar informação sobre a instância para vir a ser usada nas instâncias seguintes e testar é fazer uma previsão da variável dependente para essa instância. Nas primeiras instâncias, até encher a janela usada, apenas se treina, e a partir daí, primeiro testa-se e depois a instância é também usada para treinar.

O método **trainOnInstance()** da classe EnsembleHeterogenio chama os correspondentes métodos para treinar cada um dos modelos, das classes **subFIMTDD** e **subKNN**. O método **getVotesForInstance**() da classe EnsembleHeterogenio chama os correspondentes métodos para testar essa instância em cada um dos modelos, das classes **subFIMTDD** e **subKNN.**

No método que faz a previsão é apresentada, além da previsão para cada modelo, uma previsão do conjunto dos modelos que é uma média ponderada das previsões de cada modelo. O peso a atribuir a cada previsão é definido por um valor, k, gerado aleatoriamente baseado na distribuição de Poisson(1), e o valor de k representa o número de vezes que essa instância seria escolhida numa amostragem com reposição efetuada para esse modelo.

Como são conhecidos os valores verdadeiros da duração de cada chamada é possível comparar cada uma das previsões, y_i^m, com o valor real, y_i^v. O erro cometido em cada instância vai sendo guardado de modo que, ao fim de T instâncias testadas é possível, para cada modelo m, o erro quadrático médio $mse_m = \frac{1}{T}\sum_{i=1}^{T}(y_i^m - y_i^v)^2$. Também é calculado o erro médio absoluto $mad_m = \frac{1}{T}\sum_{i=1}^{T}|y_i^m - y_i^v|$.

Assim, ao chegar a uma dada instância os valores disponíveis para os erros refletem todos os erros cometidos até aí.

3.2.4 Agregação de modelos especializados

Incorporando vários modelos no mesmo processo pode determinar-se uma previsão agregada atribuindo pesos diversos a cada um dos modelos. A atribuição dos pesos é dinâmica, sendo atualizados de instância a instância, através de um algoritmo de agregação de modelos especializados [Oliveira14].

O algoritmo começa com N igual a cinco, sendo N o número finito de métodos de previsão.

O algoritmo de agregação começa com os pesos iguais, sendo o vetor de pesos convexos uniforme (w_1) , isto é, igual a $\frac{1}{5}$.

3.3 Tecnologia usada

Big Data é um termo usado em ciências computacionais aplicado a dados de grande volume, alta velocidade, alta variabilidade, valor e complexidade, que colocam grandes desafios ao nível da sua análise, processamento e armazenamento, é uma área de investigação em rápido crescimento. Depois de uma análise optou-se pela tecnologia *moa*. Usou-se bibliotecas já implementadas no moa, mas isso implicou a necessidade dos dados estarem no formato, arff, exigido pelo moa tal como mencionado anteriormente. Como o *moa* está implementado em *java* usou-se essa linguagem.

O projeto foi desenvolvido em eclipse Luna Service Release 1a (4.4.1) Figura 3.2.

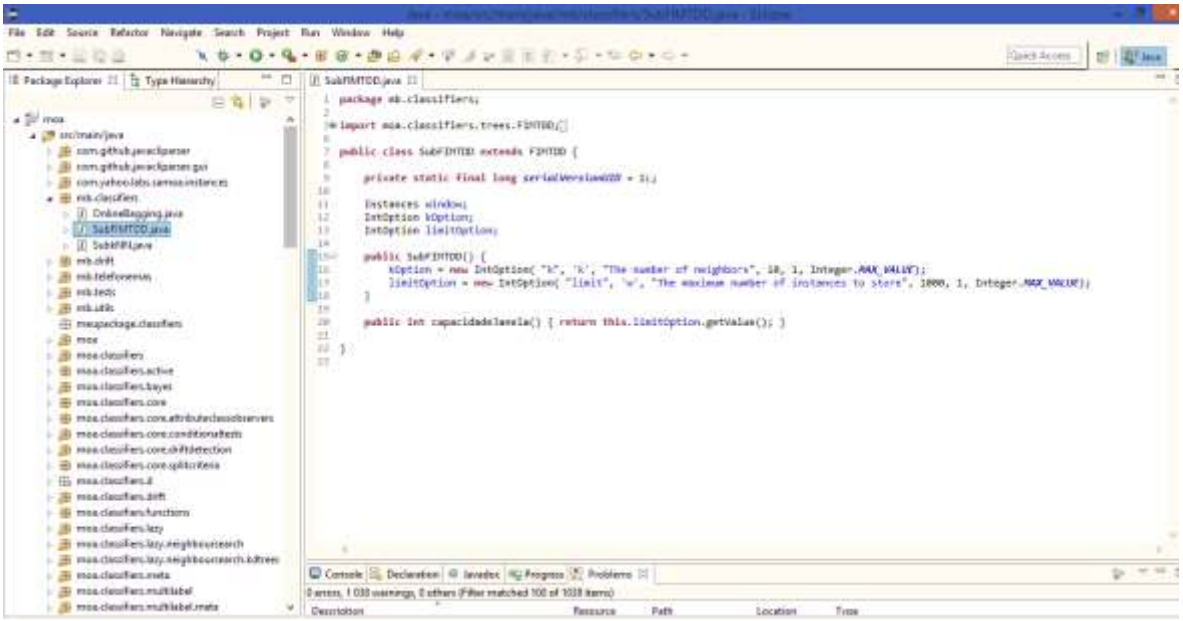

Figura 3.2: Ambiente de desenvolvimento, eclipse.

Capitulo 4

Avaliação, Testes e Resultados

Neste capítulo descreve-se como foi efetuada a avaliação dos algoritmos, os resultados obtidos, e os testes que foram realizados para avaliar a qualidade desses resultados.

4.1 Avaliação

4.1.1 MSE e MAD

Para comparar os diferentes algoritmos usam-se métricas de erro. Considerou-se a distância entre o valor y_i conhecido e o valor preditivo $h(x_i)$ gerado pelo modelo, calculando-se assim o erro da hipótese h, onde n é o número de objetos.
As medidas de erro para problemas de regressão são o erro quadrático médio MSE (mean squared error) e distância absoluta média MAD (mean absolute distance) [Monard03].

O MSE é calculado pela fórmula:

$$\text{mse-err}(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2 \tag{4.1}$$

O MAD é calculado pela seguinte expressão:

$$\text{mad-err}(h) = \frac{1}{n}\sum_{i=1}^{n}|y_i - h(x_i)| \tag{4.2}$$

O MSE e MAD são sempre não negativos. Para ambas as medidas, valores mais baixos correspondem a erros menores, logo maior capacidade preditiva, e portanto melhores modelos.
Estas métricas estão implementadas nas funções errosMAD() e errosMSE() correspondendo ao MAD e MSE respetivamente.

4.1.2 Friedman test

Para cada instância tem se uma previsão para cada algoritmo. Assim, ao comparar previsões de diferentes algoritmos, está-se a comparar previsões para a mesma instância, isto é, as amostras são emparelhadas.
Para decidir se há diferenças entre as previsões de cada algoritmo usa-se o teste de Friedman. Este teste é uma alternativa não paramétrica ao teste ANOVA, para amostra emparelhadas.
Para cada instância atribui-se uma ordem (rank) à previsão de cada algoritmo, r_{ij} . Atribui-se rank 1 ao algoritmo com menor erro, rank 2 ao algoritmo com erro seguinte, e assim sucessivamente. Para cada grupo de N instâncias faz-se a média do ranks atribuídos a cada algoritmo, R_j.
A estatística de teste calcula-se por $F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}$, sendo $\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right]$ [Demsar06], com N é o número de instâncias de cada grupo, k o número de algoritmos, R_j^2 o quadrado do total de ordens para cada situação e o $\sum_j R_j^2$ é a soma dos quadrados dos totais das ordens para cada situação.

As hipóteses sujeitas a teste são:

H0: Todos os algoritmos tem a mesma mediana
H1: Há pelo menos um algoritmo com mediana diferente.

O teste de Friedman segue uma distribuição F, com k-1 graus de liberdade no numerador e (k-1)(N-1) graus de liberdade no denominador, definindo um valor critico, VC [Demsar06].
Se F_F >VC então rejeita-se H0 e assume-se que há diferenças entre algoritmos.
Nesse caso testa-se a diferença entre os algoritmos dois a dois.

As hipóteses de teste colocadas são:

H0: Não há diferença entre os algoritmos i e j.
H1: Há diferença entre os algoritmos i e j.

A estatística de teste é:

$$z = (R_i - R_j)/\sqrt{\frac{k(k+1)}{6N}} \qquad (4.3)$$

A estatística do teste segue uma distribuição normal padronizada [Demsar06].
O Friedman test está implementado na classe Friedman.

4.2 Testes

Para confirmar os valores das previsões geradas pelos algoritmos desenvolvidos foram feitos testes usando uma folha de cálculo. Foram testadas com sucesso, para algumas instâncias, as previsões dos algoritmos: KNN, ensemble homogéneo e ensemble heterogéneo e do método agregação de modelos especializados.
Também foram feitos testes aos erros quadráticos médios e erros da distância absoluta média de todos os modelos.

4.3 Resultados

4.3.1 Resultados com MSE e MAD

Os programas desenvolvidos criaram um conjunto de resultados que foram registados em ficheiros.

Esses resultados usados como teste apresentam-se organizados da seguinte forma: na primeira coluna encontram-se os dados reais (previsão real de cada instância, Real), nas colunas seguintes encontram as previsões de cada um dos algoritmos e a sua média. Por fim temos os erros.

São apresentados resultados para dados relativos ao primeiro e segundo dia de dezembro de 2012, onde os algoritmos foram comparados ao nível do desempenho da previsão por eles realizada.
No primeiro teste foi usado o algoritmo KNN_1 com k=10, p=2, um KNN_2 com k=20, p=2 e outro KNN_3 com k=10 e p=1, onde k é o número de vizinhos e p o método da distância.
Usou-se também um ensemble heterogéneo, com estes três KNN e com um algoritmo VFDT. O método agregado (Agregação de modelos especializados) utilizou o parâmetro $\eta = 0{,}0000001$.
As primeiras mil instâncias testadas originaram os gráficos das Figuras 4.1, 4.2, 4.3, onde o primeiro KNN é representado pela linha vermelha, o segundo KNN pela linha rosa, o terceiro KNN pela linha azul, o algoritmo VFDT pela linha verde, ensemble heterogéneo pela linha amarela e agredada pela linha cinzenta.

Na figura 4.1 o gráfico representa a média das previsões da duração das chamadas em segundos de cada algoritmo nas primeiras mil instâncias do segundo dia.

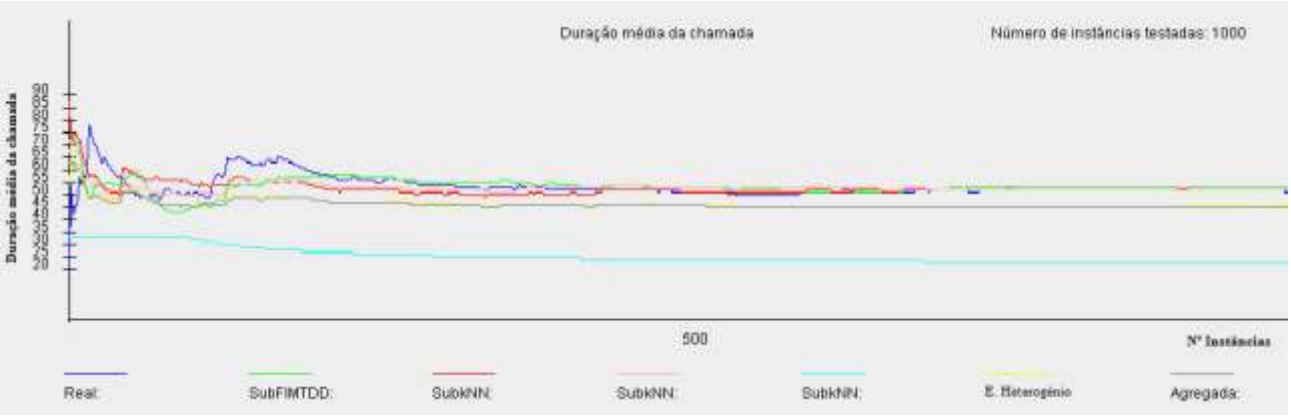

Figura 4.1: Gráfico da média das previsões de cada algoritmo durante as primeiras 1000 instâncias.

Na figura 4.2 o gráfico representa raiz do erro quadrático médio em segundos nas primeiras mil instâncias do segundo dia.

Figura 4.2: Gráfico da raiz do erro MSE de cada algoritmo nas primeiras 1000 instâncias.

Nas primeiras instâncias existe maior irregularidade. Isto pode ser justificado pelo facto dos algoritmos seguirem um processo adaptativo necessitando de uma grande quantidade de dados antes de estabilizarem.

Na figura 4.3 o gráfico representa a o erro absoluto médio médio nas primeiras mil instâncias do segundo dia.

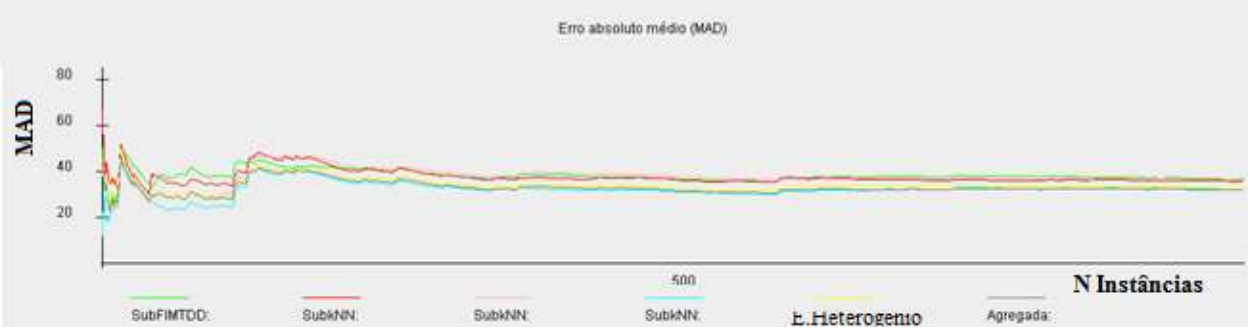

Figura 4.3: Gráfico do MAD de cada algoritmo durante as primeiras 1000 instâncias.

Em comparação com gráfico da raiz do MSE da Figura 4.2, no gráfico do erro absoluto médio, os resultados têm valores menores.

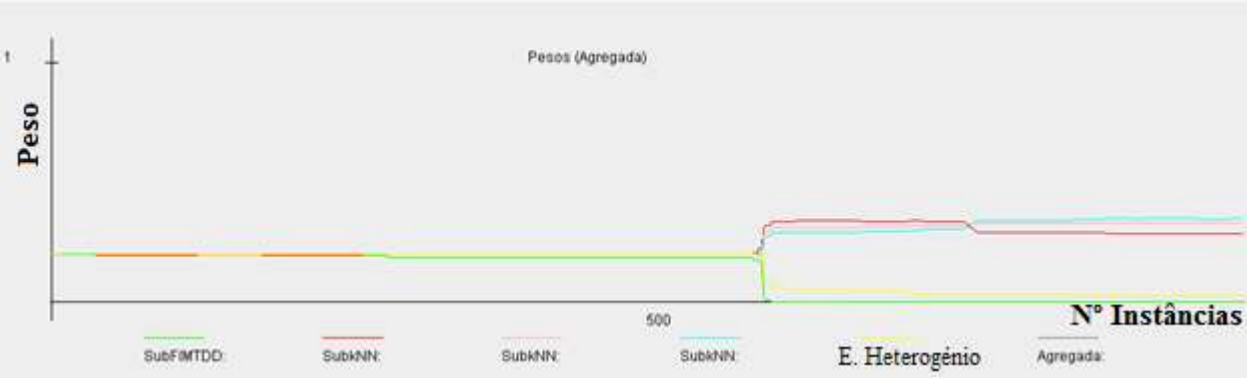

Figura 4.4: Gráfico dos pesos nas primeiras 1000 instâncias com $\eta = 0{,}0000001$.

Para a previsão agregada faz-se uma média ponderada e começa-se com o mesmo peso para todos os modelos Figura 4.4. O valor do parâmetro η tem influência na rapidez com que os pesos variam. Para o valor usado não se nota diferença significativa nos pesos ao longo das primeiras 600 instâncias, mantendo-se todos os pesos aproximadamente iguais a 0.2.
Se fosse usado $\eta = 0{,}0001$ a variação dos pesos seria muito mais rápida como se pode observar na Figura 4.5.

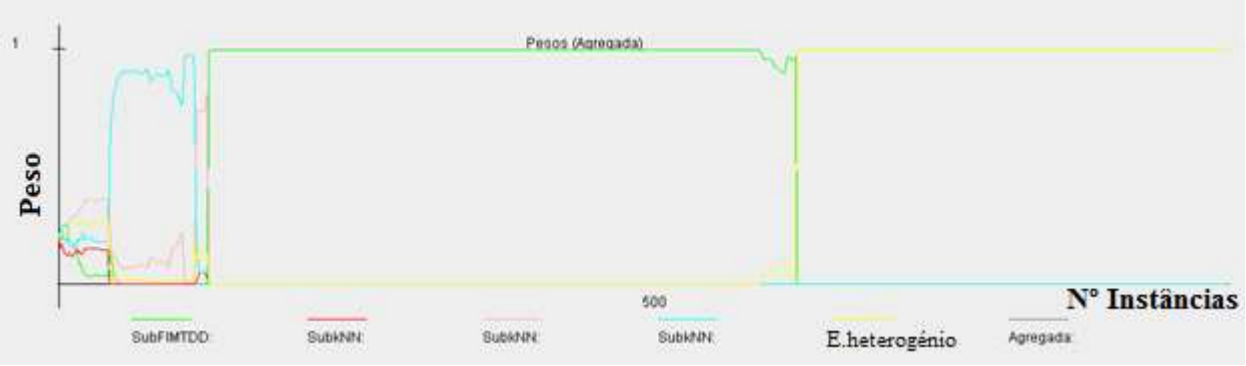

Figura 4.5: Gráfico dos pesos nas primeiras 1000 instâncias para η =0,0001.

Para este valor de η a variação dos pesos é demasiado rápida recaindo o peso de toda a previsão apenas num modelo. Depois de várias experiências optou-se por $\eta = 0{,}0000001$.

Nas tabelas seguintes apresentam-se os resultados para as últimas instâncias, isto é, as últimas chamadas no ficheiro de dados.

Na Tabela 4.1 apresenta-se os valores verdadeiros da duração da chamada, em segundos, bem como as previsões dos modelos, para a duração da chamada nas últimas oito instâncias testadas do primeiro dia.

Tabela 4.1: Valores verdadeiros e previsões dos modelos do dia um.

Real	VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
56	96,63	74,5	72,6	98	88,33	96,63
35	64,86	61,6	77,55	98	74,69	64,86
13	64,86	60,5	70,95	98	74,34	64,86
9	64,87	58	57,45	98	68,01	64,87
20	62,94	46,4	48,9	98	73,03	62,94
36	66,00	40,5	78,95	98	70,86	66,00
46	66,02	183,9	119,6	98	108,89	66,02
35	66,10	93,1	71,35	98	75,48	66,10

Na Tabela 4.2 apresentam-se as médias dos valores verdadeiros da duração da chamada em segundos e dos valores da previsão da duração da chamada em segundos em cada um dos modelos, do dia um, para as últimas oito instâncias testadas.

Tabela 4.2: Médias dos modelos do dia um

Real	VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
67,81660	67,83396	67,74986	67,78624	68,47801	67,95996	67,84746
67,81660	67,83396	67,74986	67,78625	68,47801	67,95996	67,84746
67,81659	67,83396	67,74986	67,78625	68,47801	67,95996	67,84746
67,81659	67,83396	67,74986	67,78624	68,47802	67,95996	67,84746
67,81659	67,83396	67,74986	67,78624	68,47802	67,95996	67,84746
67,81658	67,83396	67,74986	67,78624	68,47802	67,95996	67,84746
67,81658	67,83396	67,74987	67,78625	68,47802	67,95996	67,84746
67,81658	67,83396	67,74987	67,78625	68,47803	67,95996	67,84746

Nas últimas instâncias o modelo que tem média mais próxima da média verdadeira é VFDT, no entanto isto não garante que este seja o melhor modelo.
A escolha do melhor modelo foi feita pelo que apresentou menor erro quadrático médio, como se pode verificar na tabela 4.3, e que recaiu no modelo agregado.

Na tabela 4.3 apresenta-se o erro quadrático médio das previsões para cada modelo, em segundos quadrados, para as últimas oito instâncias. O erro quadrático médio calculado em cada instância reflecte todos os erros cometidos nas instâncias anteriores. Assim o erro quadrático médio calculado na última instância representa a qualidade desse modelo.

Tabela 4.3: Erros quadrático médio dos modelos do dia um.

VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
15463,587	16955,825	16427,291	18920,524	16139,606	15460,542
15463,586	16955,823	16427,290	18920,523	16139,605	15460,541
15463,585	16955,822	16427,289	18920,522	16139,604	15460,539
15463,584	16955,821	16427,288	18920,521	16139,603	15460,538
15463,582	16955,820	16427,286	18920,520	16139,602	15460,537
15463,581	16955,818	16427,285	18920,518	16139,601	15460,536
15463,580	16955,818	16427,284	18920,517	16139,600	15460,535
15463,579	16955,817	16427,283	18920,516	16139,598	15460,534

Na tabela 4.4 apresenta-se a raiz do erro quadrático médio, em segundos para as últimas oito instâncias, para cada um dos modelos, para melhor clareza dos resultados.

Tabela 4.4: Raiz dos erros quadrático médio dos modelos do dia um.

VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
124,35267	130,21453	128,169	137,5519	127,0417	124,3404
124,35267	130,21453	128,169	137,5519	127,0417	124,3404
124,35266	130,21452	128,169	137,5519	127,0417	124,3404
124,35266	130,21452	128,169	137,5519	127,0417	124,3404
124,35265	130,21452	128,169	137,5519	127,0417	124,3404
124,35265	130,21451	128,169	137,5519	127,0417	124,3404
124,35264	130,21451	128,169	137,5519	127,0417	124,3404
124,35264	130,2145	128,169	137,5519	127,0417	124,3404

O modelo que tem menor erro quadrático médio, ao fim de testadas todas as instâncias, é o Agregado. O erro neste modelo é praticamente igual ao erro do VFDT porque a partir de certa altura, no modelo agregado, o peso reduz-se a um para o VFDT e zero para todos os outros modelos. Logo na última instância a média agregada é baseada únicamente no VDFT. Podemos também observar, comparando os KNN que o KNN_1 e KNN_2 que usam a distância Minkowski, com p igual a dois (euclidiana), comporta-se melhor do que o KNN_3 quando usa o p igual a um (distância *manhattan*).

Na Figura 4.6 observam-se os resultados para as últimas instâncias, dos dados registados na tabela 4.3, onde no eixo vertical está representado o erro quadrático médio e no eixo horizontal o número da instância testada.

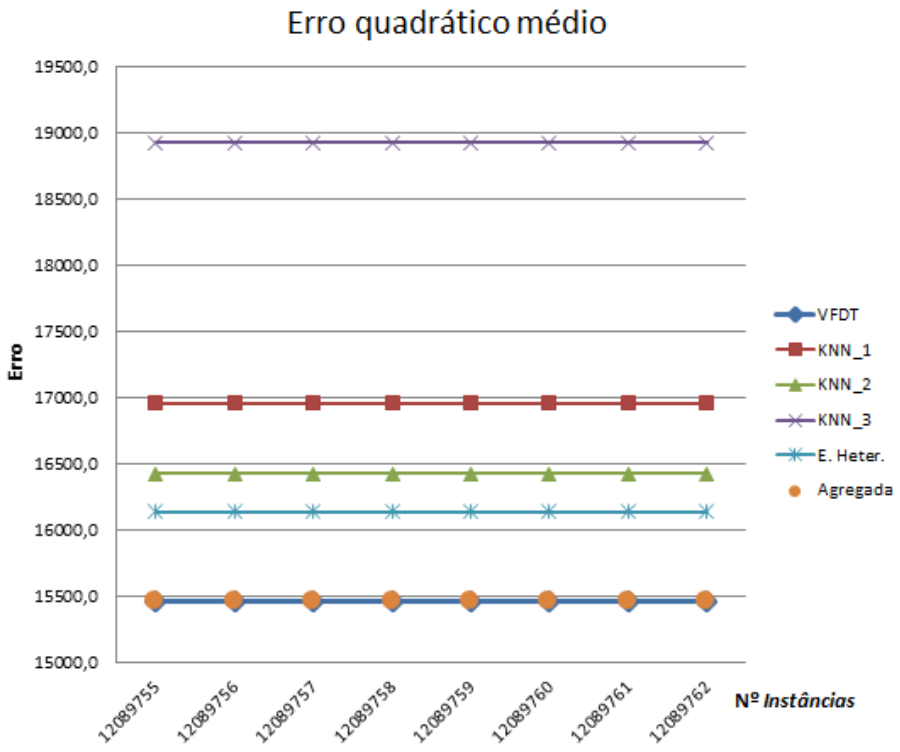

Figura 4.6: Gráfico MSE para as últimas instâncias do primeiro dia.

Através da tabela 4.5 que representa o peso atribuído à previsão de cada modelo, é possível verificar-se que a partir de determinado momento existe um modelo com o peso total do conjunto, VFDT.

Tabela 4.5 Pesos do primeiro dia no final das últimas oito instâncias

VFDT	KNN_1	KNN_2	KNN_3	E. Heter.
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0

Fizeram-se mais testes usando o segundo dia, os mesmos algoritmos, com os mesmos parâmetros, que confirmam os resultados anteriores, e verificando-se, mais uma vez, que o modelo agregado é o método melhor, Tabela 4.6.

Tabela 4.6 Erro quadrático médio em segundos quadrados do segundo dia.

VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
22456,121	24218,257	23464,060	27104,560	23025,019	22430,564
22474,203	24243,053	23486,337	27139,784	23047,808	22448,706
22482,424	24255,127	23496,957	27155,697	23058,729	22456,992
22488,539	24262,231	23504,052	27163,551	23065,663	22463,119
22502,170	24276,429	23517,954	27176,591	23079,318	22476,756
22497,571	24272,838	23513,940	27169,549	23074,641	22472,172
22496,788	24272,430	23513,004	27165,819	23073,856	22471,413
22495,614	24273,809	23514,185	27163,221	23073,555	22470,299

Na tabela 4.7 apresenta-se a raiz do erro quadrático médio, em segundos para as últimas oito instâncias do segundo dia, para cada um dos modelos, para melhor clareza dos resultados.

Tabela 4.7: Raiz dos erros quadrático médio em segundos do segundo dia

VFDT	KNN_1	KNN_2	KNN_3	E. Heter.	Agregado
149,8537	155,6222	153,1798	164,6346	151,7400	149,7684
149,914	155,7018	153,2525	164,7416	151,8150	149,8289
149,9414	155,7406	153,2872	164,7899	151,8510	149,8566
149,9618	155,7634	153,3103	164,8137	151,8738	149,8770
150,0072	155,809	153,3557	164,8532	151,9188	149,9225
149,9919	155,7974	153,3426	164,8319	151,9034	149,9072
149,9893	155,7961	153,3395	164,8206	151,9008	149,9047
149,9854	155,8005	153,3434	164,8127	151,8998	149,9010

Na Figura 4.7, observam-se os resultados do erro quadrático médio para algumas instâncias do segundo dia que reproduzem dados equivalentes à tabela 4.6. Na figura 4.7, o eixo vertical representa o erro e o eixo horizontal o número da instância testada.

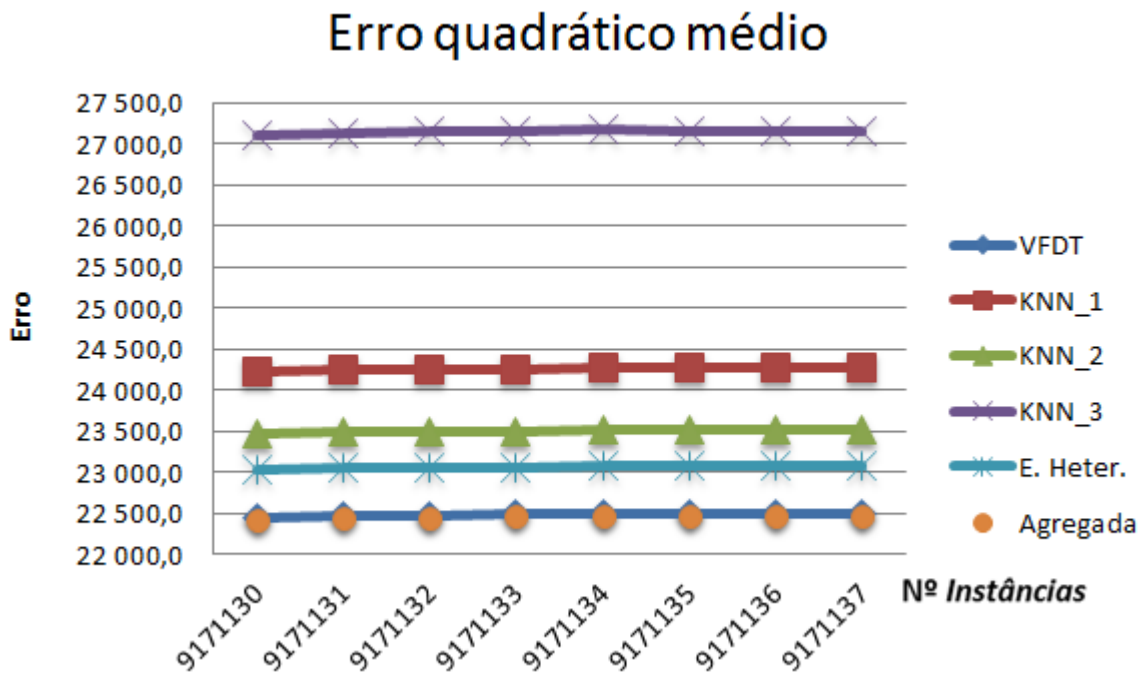

Figura 4.7: Gráfico MSE para as últimas instâncias do segundo dia.

Fizeram-se testes usando o ensemble homogéneo utilizando os parâmetros: KNN1 com k=10, p=2, KNN2 com k=20 e p=2, KNN3 com k=10 e p=1, KNN4 com k=20 e p=3 e KNN5 com k=100 e p=2 onde k é o número de vizinhos e p o método da distância. Na tabela 4.8 apresenta os valores verdadeiros da duração da chamada em segundos bem como as previsões dos modelos para as últimas oito instâncias testadas do primeiro dia.

Tabela 4.8: Valores reais e previsões usando o ensemble homogêneo.

Real	KNN1	KNN2	KNN3	KNN4	KNN5	E.Homog	Agregado
56,00	74,50	72,60	98,00	147,75	81,28	108,98	81,28
35,00	61,60	77,55	98,00	64,60	73,48	70,40	73,48
13,00	60,50	70,95	98,00	66,30	73,32	60,50	73,32
9,00	58,00	57,45	98,00	65,30	73,40	65,30	73,40
20,00	46,40	48,90	98,00	80,40	46,60	72,20	46,60
36,00	40,50	78,95	98,00	78,35	62,11	59,73	62,11
46,00	183,90	119,60	98,00	122,60	97,56	107,30	97,56
35,00	93,10	71,35	98,00	45,75	71,65	78,50	71,65

Na Tabela 4.9 apresenta as médias dos valores verdadeiros bem como das previsões dos vários modelos para as últimas oito instâncias do primeiro dia em segundos.

Tabela 4.9 Médias dos valores verdadeiros e média das previsões ensemble homogêneo.

REAL	KNN1	KNN2	KNN3	KNN4	KNN5	E. Homog.	Agregado
67,81660	67,74986	67,78624	68,47801	67,84408	67,83568	67,94604	67,83710
67,81660	67,74986	67,78625	68,47801	67,84408	67,83568	67,94604	67,83710
67,81659	67,74986	67,78625	68,47801	67,84408	67,83568	67,94604	67,83710
67,81659	67,74986	67,78624	68,47802	67,84408	67,83568	67,94604	67,83710
67,81659	67,74986	67,78624	68,47802	67,84408	67,83568	67,94604	67,83710
67,81658	67,74986	67,78624	68,47802	67,84408	67,83568	67,94604	67,83710
67,81658	67,74987	67,78625	68,47802	67,84409	67,83568	67,94604	67,83710
67,81658	67,74987	67,78625	68,47803	67,84409	67,83568	67,94604	67,83710

Como a média esta a ser calculada desde a primeira instância e no fim do ficheiro há cerca de 12000000 de instâncias, o peso de cada instância na média é muito pequeno não sendo suficiente para alterar as médias nesta últimas instâncias.
O modelo que tem média mais próxima da média verdadeira é o método do agregado, e a segunda mais próxima é o KNN_2.

Na tabela 4.10 apresenta-se o erro quadrático médio para as últimas oito instâncias para cada um dos modelos do primeiro dia em segundos quadrados.

Tabela 4.10 Erro quadrático médio usando o ensemble homogéneo.

KNN1	KNN2	KNN3	KNN4	KNN5	E.Homog	Agregado
16955,8233	16427,2897	18920,5225	16781,1523	15930,5944	16160,3836	15928,8509
16955,8221	16427,2887	18920,5216	16781,1512	15930,5934	16160,3825	15928,8499
16955,8209	16427,2875	18920,5207	16781,1501	15930,5924	16160,3814	15928,8490
16955,8196	16427,2862	18920,5196	16781,1490	15930,5911	16160,3803	15928,8477
16955,8182	16427,2850	18920,5183	16781,1477	15930,5899	16160,3790	15928,8464
16955,8183	16427,2841	18920,5170	16781,1468	15930,5888	16160,3780	15928,8453
16955,8172	16427,2828	18920,5158	16781,1455	15930,5876	16160,3768	15928,8441
16955,8233	16427,2897	18920,5225	16781,1523	15930,5944	16160,3836	15928,8509

Na Figura 4.8, observam-se os resultados do erro quadrático médio para algumas instâncias do primeiro dia usando ensemble homogéneo que reproduzem dados equivalentes à tabela 4.10.

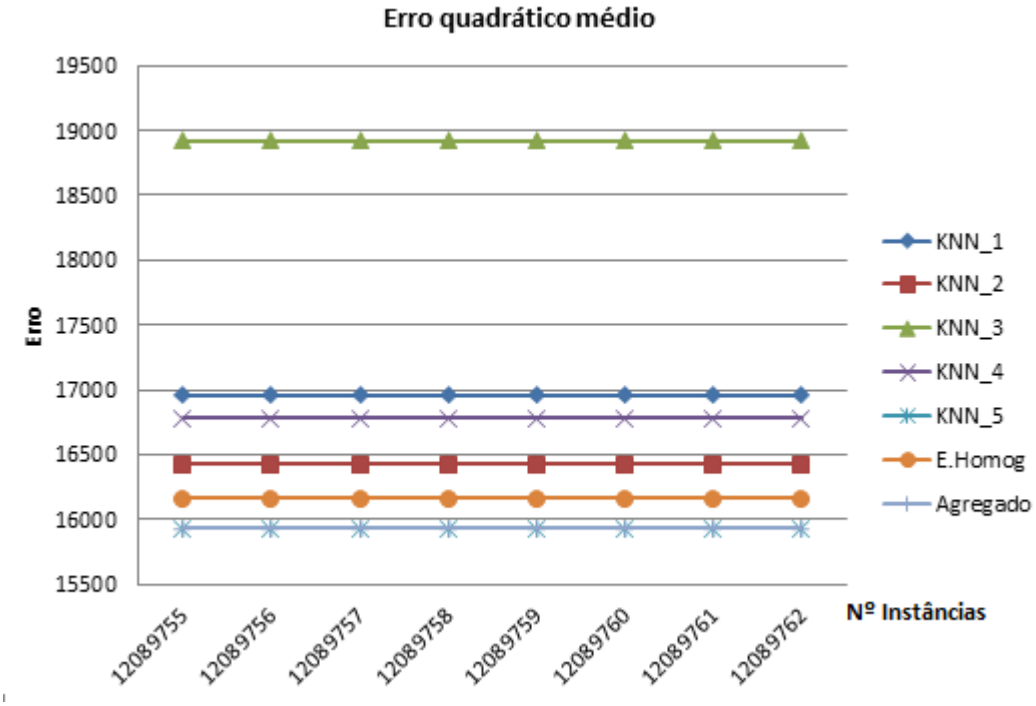

Figura 4.8: Gráfico MSE para as últimas instâncias do primeiro dia para os testes com ensemble homogéneo.

Na tabela 4.11 apresenta-se o MAD para as últimas oito instâncias para cada um dos modelos do primeiro dia.

Tabela 4.11 MAD usando o ensemble homogéneo.

KNN1	KNN2	KNN3	KNN4	KNN5	E.Homog	Agregado
54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342

Verifica-se que o modelo que apresenta menor erro quadrático médio, depois de testadas todas as instâncias, é o agregado, sendo o segundo melhor método o ensemble homogéneo.

4.3.2 Resultados com o Friedman test

Para calcular a estatística Friedman procedeu-se do seguinte modo.
As posições de cada algoritmo foram calculados para cada instância e fez-se a média dos *ranks* para grupos de 500000 instâncias do primeiro dia. Obtiveram-se os resultados da tabela 4.12.

Tabela 4.12: Ranking de cada algoritmo por análise de dados do primeiro dia

	SubFIMTDD	**SubkNN_1**	**SubkNN_2**	**SubkNN_3**	**E. Heterogeneo**	**Agregada**
	3,70223	4,555234	3,330582	5,727514	2,65843	1,02601
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
	2	5	4	6	3	1
Soma	49,70223	119,55523	95,33058	143,72751	71,65843	24,02601
Media R_j	2,07097	4,98147	3,97211	5,98865	2,98577	1,00108
Media quadrado R_j^2	4,28874	24,81502	15,77764	35,86389	8,91481	1,00217

Observando a tabela 4.12 conclui-se que o melhor algoritmo é o agregado, o que se segue é o VFDT, em seguinda e Ensemble Heterogéneo por último os KNN. Sendo o pior o que usou a distância de manhattan, KNN_3.

Obteve-se F_f=1168,75601.

Para testar as hipoteses seguintes usou-se sempre um nivel de significancia de 3%.

No trabalho, usou-se k=6, pois é o número de algoritmos e N= 24 correspondente ao número de grupos de instâncias. Assim o valor crítico é $F_{5,115} = 2{,}58$ (Anexo C) [Webster07].

Como $F_f > F_{5,115}$ então rejeita-se H0, todos os algoritmos tem a mesma mediana, e admite-se há diferenças entre os algoritmos.
Decorre da conclusão que se devem comprar os algoritmos dois a dois.

Tabela 4.13: Diferenças entre algoritmos

	VFDT	KNN_1	KNN_2	KNN_3	E.heterogéneo	Agregada
VFDT	0	-5,39	-3.52	-7.25	-1,69	1.98
KNN_1		0	1,87	-1.86	3.70	7.37
KNN_2			0	-3,73	1,83	5,50
KNN_3				0	5.56	9.23
E.heterogéneo					0	3.67
Agregada						0

A diagonal é toda zero porque está-se a comprar um algoritmo com ele próprio, logo não há diferença observável.
A tabela encontra-se preenchida apenas acima da diagonal dado que os valores abaixo desta são os simétricos dos valores acima.

A estatística segue uma distribuição normal padronizada logo o valor critico é 2,17 (Anexo C)[Webster07].

Rejeita-se H0, quando a estatística do teste não pretencer ao intervalo [-2,17;2,17], que são os casos sombreados (cor amarela) na tabela 4.13.

Assim conclui-se que o algoritmo VFDT não apresenta grandes diferenças quando comparado com o Ensemble Heterogéneo e com o Agregado.
Conclui-se também que o KNN_1 não tem diferenças substanciais quando comprado com os KNN_2 e KNN_3, e que o KNN_2 é semelhante ao Ensemble Heterogéneo.
Apesar de termos visto que o método agregada era o que tinha menor erro esta analise estatistica indica que as diferencas entre o método agregada, método Ensemble heterogéneo e método VFDT não tem significado estatístico.

Capitulo 5

Conclusões e Trabalho Futuro

Neste capítulo apresentam-se as conclusões decorrentes do trabalho desenvolvido e explicitado ao longo desta tese de dissertação e as perspectivas para um trabalho futuro.

5.1 Conclusões

Durante os trabalhos de desenvolvimento desta tese de dissertação foram estudados e implementados algoritmos incrementais, testados e aplicados na previsão da duração de chamadas telefónicas, quando estas se iniciam. Os algoritmos desenvolvidos foram o K-NN, VFDT, ensemble homogéneo, ensemble heterogéneo e o método de agregação de modelos especializados.

No que se refere às características dos algoritmos, o k-NN é um algoritmo simples que apresenta uma boa taxa de acerto preditiva em vários conjuntos de dados. O valor de k e a medida de distância utilizada influenciam o desempenho do algoritmo. Os modelos múltiplos são usados para identificar um conjunto de preditores cuja decisões individuais são combinadas ou agregadas de forma a efetuar previsões em novos exemplos.

Depois de efectuados os testes e feita a análise de resultados, concluíu-se que o método de agregação de modelos especializados apresentava o melhor comportamento, uma vez que mostra resultados com erro quadrático médio inferior aos restantes modelos. A vantagem do modelo de agregação resulta de, através de ponderação, incorporar e escolher o melhor modelo individual no seu funcionamento.
O valor do MSE para este modelo agregado é muito similar ao do VFDT. Uma vez que apartir de um certo número de instâncias o modelo VFDT fica com um e o restantes modelos com zero.

Com o teste Friedman provamos que o VFDT e o método de agregação têm erro similiar, e também provamos que KNN_2 é equivalente ao Ensemble Heterogéneo e o KNN_1 é similar aos outros dois KNN. Pelo *ranking* estabelecido a primeira posição, e portanto o melhor modelo é o agregado, a que se seguem o VFDT, Ensemble Heterogéneo e por fim os KNN. Verificou-se ainda, que noutros testes adicionais efectuados, com o método homogéneo, KNN e agregado, levavam à mesma conclusão que o *agregado* é o melhor método.

5.2 Trabalho Futuro

Um trabalho de dissertação nunca fica totalmente concluído. No decorrer da investigação e do estudo realizado vão-se abrindo novas perspectivas e novas ideias no modo e na forma como poderiam ter sido abordados e explorados diversos conceitos. Uma dessas àreas de abordagem não realizada seria a implementação por *Drifting Concepts*, que poderia trazer melhorias na previsão das chamadas. O *Drifting Concepts* é um método de aprendizagem online em que o conceito muda ao longo do tempo [Lazarescu03].

Os fluxos contínuos de dados evoluem ao longo do tempo com possíveis derivas desconhecidas e imprevisíveis. Pode acontecer que não haja uma única distribuição que modele todo o nosso conjunto de dados. É também possível admitir que uma distribuição modele bem durante um período de tempo mas que noutro contexto e noutro período de tempo já seja outra distribuição.

Assim, a utilização de modelos dinâmicos, conceitos referidos neste trabalho, poderiam ser aprofundados e incorporados e regulados por *Drifting Concepts* de modo a produzir mais e melhores resultados na previsão de dados.

Referências

[Altshuler11] Yaniv Altshuler, Nadav Aharony, Alex ("Sandy"). Incremental Learning with Accuracy Prediction of Social and Individual Properties from Mobile-Phone Data, MIT Media Lab , Cambridge, 2011

[Babcock02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom. Models and Issues in Data Stream Systems, Stanford, 2002

[Berk05] Richard A. Berk. Data Mining and Knowledge Discovery Handbook Springer, 2005

[Berry00] Michael J. A. Berry, Gordon Linoff. Mastering Data Mining: The Art and Science of Customer Relationship Management; John Wiley & Sons, 2000

[Bifet10] Albert Bifet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer. MOA: Massive Online Analysis, pages 1601-1604, Department of Computer Science, University of Waikato Hamilton, New Zealand, 2010.
URL: http://www.jmlr.org/papers/volume11/bifet10a/bifet10a.pdf,

[Bifet12] Albert Bifet, Richard Kirkby, Philipp Kranen, Peter Reutemann. Massive Online Analysis Manual, The university of waikato, 2012

[Breiman96] Leo Breiman. Bagging predictors, Machine Learning, 24, pages 123-140, University of California, 1996

[Demsar06] Demsar Janez. Statistical Comparisons of Classifiers over Multiple Data Sets, pages 1–30, Faculty of Computer and Information Science, Trzaska 25, Ljubljana, Slovenia, 2006

[Fayyad1996] Fayyad, Usama M…; Haussler, David; Stolorz, Paul. Mining Scientific Data, Communications of the ACM, 1996

[Fayyad96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery in Databases, 1996

[Friedman40] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings, pages 86-92, Annals of Mathematical Statistics, 1940

Referências

[Gaillard14] Pierre Gaillard, Yanning Goude. Forecasting electricity consumption by aggregating specialized experts, how to design a good set of experts, EDF R&D, Clamart, France, 2014

[Gama10] João Gama. Knowledge Discovery from Data Streams, Chapman & Hall/CRC Press, 2010

[Kalsing12] André Cristiano Kalsing. Uma abordagem incremental para Mineração de Processos de Negócio, Universidade Federal do Rio Grande do Sul, 2012

[Kramer11] Olivier Kramer. Unsurpervised K-Nearest Neightbor Regression, Universitat Oldenburg, Oldenburg, Germany, 2011

[Kuhn13] Max Kuhn, Kjell Johnson. Applied Predictive Modeling, Springer, 2013

[Langley95] P. Langley. Learning in humans and machines: Towards an interdisciplinary learning science, chapter Order Effects in Incremental Learning, pages 154-165, P.Reimann & H. Spada, Oxford, 1995

[Lazarescu03] Mihai M. Lazarescu, Svetha Venkatesh, Hung H. Bui. Using Multiple Windows To Track Concept Drift, Faculty of Computer Science, Curtin University, 2003

[Monard03] Maria Carolina Monard, José Augusto Baranauskas. Sistemas Inteligentes-Fundamentos e Aplicações, capítulo Conceitos de aprendizado de máquina, páginas 89-114, Manole Ltda, 2003

[Oliveira14] Liliana Sousa Oliveira. Modelos de Agregação de Previsões Aplicados à Previsão de energia Eólica, Faculdade de Economia da Universidade do Porto, 2014

[Oza05] Nikunj C. Oza, Stuart Russell. Online Bagging and Boosting, Univerty of California, Berkeley, 2005

[Pinto05] Carlos Manuel Silva Pinto, Algoritmos Incrementais para Aprendizagem Bayesiana, Faculdade de Economia da Universidade do Porto, 2005

[Research12] RuleQuest Research. Data mining with Cubist, 2012
URL: http://www.rulequest.com/cubist-info.html, Accessed: 2014

Referências

[Sferra03] Heloisa Helena Sferra, Ângela M. C. Jorge Corrêa. Conceitos e Aplicações de Data Mining, Universidade Metodista de Piracicaba, 2003

[Waikato13] Waikato, Weka 3: Data Mining Software in Java, 2013
URL: http://www.cs.waikato.ac.nz/ml/weka/, Accessed: 2014

[Wang03] Haixun Wang, Wei Fan, Philip SYu, Jiawei Han. Mining Concept Drifting Data Streams using Ensemble Classifiers, ACM, 2003

[Wang&Pineau03] Boyu Wang, Joelle Pineau. Online Bagging and Boosting for Imbalanced Data Streams, School of Computer Science, McGill University, Montreal, Canada 30 Oct 2003

[Webster07] Allen L. Webster. Estatística Aplicada à administração e Economia, McGrawHill, 2007

[Witten05] Ian H. Witten, Eibe Frank, Mark A. Hall. "Data Mining – Practical Machine Learning Tools and Techniques", Elsevier, 2005.

[Yang11] Hang Yang, Simon Fong. Moderated VFDT in Stream Mining Using Adaptive Tie Threshold and Incremental Pruning, University of Macau, Macau, China, 2011

[Zang14] Wenyu Zang, Peng Zhang, Chuan Zhou, Li Guo**.** Comparative study between incremental and ensemble Learning on data streams: Case study, Springer, 2014
URL: http://www.journalofbigdata.com/content/1/1/5, Accessed: 2014

[Zhao10] Qiang Li Zhao, Yan Huang Jiang, Ming Xu. Incremental Learning by Heterogeneous Bagging Ensemble, School of Computer Science, National University of Defense Technology, Changsha, China, 2010

Anexos A

Código

Neste anexo apresenta-se o código desenvolvido agrupados segundo nos diferentes *package* de classes implementadas.

A.1 Package mb.classifiers

Neste *package* estão as classes relativas aos modelos usados.

classe subKNN

```
package mb.classifiers;

import java.util.ArrayList;
import com.yahoo.labs.samoa.instances.Instance;
import moa.classifiers.lazy.kNN;
import mb.utils.Vizinho;

public class SubkNN extends kNN {

	private static final long serialVersionUID = 1L;

	private int p;	// distância de Minkovski: d(X,Y) = (soma(xi-yi)^p)^(1/p)
	private boolean ponderada;

	public SubkNN() {
		this.p=2; // p=2 é a distância euclidiana
		// valores pré-definidos em kNN: k=10 e n=1000
	}
	public SubkNN(boolean ponderada) {
		this.p=2; // p=2 é a distância euclidiana
		this.ponderada = ponderada;
		// valores pré-definidos em kNN: k=10 e n=1000
	}
	public SubkNN(int k, int n, int p, boolean ponderada) {
		super.kOption.setValue(k); super.limitOption.setValue(n); this.p=p;
this.ponderada=ponderada;
	}
	// Métodos
	public int getk() { return super.kOption.getValue(); }
	public int tamanhoJanela() {
```

```
                try { return super.window.numInstances(); }
                catch(Exception e) { return 0; } }
        public int capacidadeJanela() { return super.limitOption.getValue(); }

        public double getVotesForInstanceImpl(Instance instância) {
                double previsao=0;
                int ene = tamanhoJanela();                              // Tamanho da janela
                int kapa = super.kOption.getValue();                    // Número de vizinhos
                int num = instância.numAttributes();                      // Número de atributos,
incluindo o y
                ArrayList<Vizinho> kproximos = new ArrayList<Vizinho>();     // k-vizinhos mais
próximos
                ArrayList<Double> distkproximos = new ArrayList<Double>();
                for(int i=0; i<ene; i++) {
                        int instAcrescentada=0;
                        double d = distancia(instância, super.window.instance(i));
                        Double D = new Double(d);
                        Vizinho vizinho = new Vizinho(i,super.window.instance(i),d);
                        for(int k=0; k<kproximos.size(); k++)
                                if(d<kproximos.get(k).getDistance()) {
                                        kproximos.add(k,vizinho);
                                        distkproximos.add(k,D);
                                        instAcrescentada=1;
                                        if(kproximos.size()>kapa) {
                                                kproximos.remove(kapa);
                                                distkproximos.remove(kapa);
                                        }
                                        break;
                                }
                        if(kproximos.size()<kapa && instAcrescentada==0) {
                                kproximos.add(kproximos.size(),vizinho);
                                distkproximos.add(distkproximos.size(),D);
                        }
                }
                previsao=0;
                double somad=0, retorno=0, d=0;
                boolean umIgual=false;
                int k=0;
                do {
                        d = distkproximos.get(k).doubleValue();
                        if(ponderada) {
                                if(d==0) { umIgual=true; previsao =
kproximos.get(k).getInstance().value(num-1); }
                                else {
                                        previsao = previsao+kproximos.get(k).getInstance().value(num-
1)/d;
                                        somad = somad + 1.0/d;
                                }
                        } else previsao = previsao+kproximos.get(k).getInstance().value(num-1);
                        k++;
```

```
                } while(k<kapa);
                if(ponderada) {
                        if(umIgual) retorno = previsao;
                        else retorno = previsao/somad;
                } else retorno = previsao/kapa;
                return retorno;
        }

        public double distancia(Instance X, Instance Y) {
                double valor = 0;
                for(int i=0; i<X.numAttributes()-1; i++) {
                        double a = Math.pow(X.value(i)-Y.value(i),this.p);
                        valor = valor + a;
                }
                valor = Math.pow(valor,1.0/this.p);
                return valor;
        }

}
```

classe subFIMTDD

```
package mb.classifiers;

import moa.classifiers.trees.FIMTDD;
import com.github.javacliparser.IntOption;
import com.yahoo.labs.samoa.instances.Instances;

public class SubFIMTDD extends FIMTDD {

        private static final long serialVersionUID = 1L;

        Instances window;
        IntOption kOption;
        IntOption limitOption;

        public SubFIMTDD() {
                kOption = new IntOption( "k", 'k', "The number of neighbors", 10, 1, Integer.MAX_VALUE);
                limitOption = new IntOption( "limit", 'w', "The maximum number of instances to store", 1000,
1, Integer.MAX_VALUE);
        }

        public int capacidadeJanela() { return this.limitOption.getValue(); }

}
```

classe EnsembleHeterogenio

```
package mb.classifiers;

import java.util.ArrayList;
import java.io.FileWriter;
import java.io.IOException;
import com.yahoo.labs.samoa.instances.Instance;
import mb.utils.Modelo;
import mb.utils.PrevisaoDependente;
import mb.utils.FundoGrafico;

public class EnsembleHeterogenio {
	// Variáveis
	private int M;                                          // Número de modelos
	private int N;                                          // Capacidade da janela
	private ArrayList<Modelo> modelos = new ArrayList<Modelo>();         // Lista dos vários
modelos em Bagging
	private ArrayList<Double> somas = new ArrayList<Double>();
	private ArrayList<Double> somasMAD = new ArrayList<Double>();
	private ArrayList<Double> somasQ = new ArrayList<Double>();
	private double somasBagging, somasQBagging;
	private double somasBaggingMAD;
	private long numExemplos;                  // Número de instâncias testadas
	private double somasV;
	private double somasB;
	private double somasA;
	private double somasAgregada;
	private double somasQAgregada;
	private double somasAgregadaMAD;
	private PrevisaoDependente anterior = null;
	int tipoErro;
	int tipoBagging;           // 1. Homogéneo; 2. Heterogéneo
	double eta;                                     // Para a atualização dos pesos na previsão agregada
	private ArrayList<Double> somasY = new ArrayList<Double>();
	private ArrayList<ArrayList<Double>> valoresY = new ArrayList<ArrayList<Double>>();
	private ArrayList<ArrayList<Double>> valoresSegundo = new
ArrayList<ArrayList<Double>>();
	private FundoGrafico fundoGraficos = new
FundoGrafico(valoresY,valoresSegundo,modelos,tipoErro);
	// Construtores
	public EnsembleHeterogenio(int tipoErro,int tipoBagging, int p, int k, double eta) {
		this.tipoErro = tipoErro; this.tipoBagging = tipoBagging; this.eta = eta;
		if(tipoBagging==1) this.M=3;
		else this.M=4;
		if(tipoBagging==2) {
		SubFIMTDD fimtdd = new SubFIMTDD(); fimtdd.prepareForUse();
```

```
                Modelo modelo1 = new Modelo("SubFIMTDD",fimtdd);
                modelos.add(modelo1);
                this.N = fimtdd.capacidadeJanela(); }
                SubkNN knn1 = new SubkNN(); Modelo modelo2 = new Modelo("SubkNN",knn1);
                modelos.add(modelo2);
                if(tipoBagging==1) this.N = knn1.capacidadeJanela();
                SubkNN knn2 = new SubkNN(20,N,2,false); Modelo modelo3 = new
Modelo("SubkNN",knn2);
                modelos.add(modelo3);
                SubkNN knn3 = new SubkNN(k,N,p,false); Modelo modelo4 = new
Modelo("SubkNN",knn3);
                modelos.add(modelo4);

                Double inicial = new Double(0.0);
                for(int i=0; i<this.M; i++) { somas.add(inicial); somasQ.add(inicial);
somasY.add(inicial); somasMAD.add(inicial); }
                somasBagging=0.0; somasQBagging=0.0; this.numExemplos=0L; somasV=0.0;
somasB=0.0; somasA=0.0;
                somasBaggingMAD=0.0; somasAgregada=0.0; somasQAgregada=0.0;
somasBaggingMAD=0.0;
        }
        public EnsembleHeterogenio(ArrayList<Modelo> modelos) {
                this.modelos = modelos;
                this.M = modelos.size();
                if(modelos.get(0).getTipo().equals("SubFIMTDD"))
                        this.N = modelos.get(0).getFimtdd().capacidadeJanela();
                else this.N = modelos.get(0).getKnn().tamanhoJanela();
                Double inicial = new Double(0.0);
                for(int i=0; i<this.M; i++) { somas.add(inicial); somasQ.add(inicial);
somasY.add(inicial); somasMAD.add(inicial); }
                somasBagging=0.0; somasQBagging=0.0; this.numExemplos=0L; somasV=0;
somasB=0; somasA=0.0;
                somasBaggingMAD=0.0; somasAgregada=0.0; somasQAgregada=0.0;
somasBaggingMAD=0.0; }
        // Métodos
        public int tamanhoJanela() { return this.N; }
        public void trainOnInstance(Instance inst) {
                for(int i=0; i<this.M; i++) {
                        if(modelos.get(i).getTipo().equals("SubFIMTDD"))
                                modelos.get(i).getFimtdd().trainOnInstance(inst);
                        if(modelos.get(i).getTipo().equals("SubkNN"))
                                modelos.get(i).getKnn().trainOnInstance(inst);
                }
        }
        public double[] getVotesForInstance(Instance inst) {
                double valores[] = new double[this.modelos.size()];
                for(int i=0; i<this.M; i++) {
                        if(modelos.get(i).getTipo().equals("SubFIMTDD"))
                                valores[i] = modelos.get(i).getFimtdd().getVotesForInstance(inst)[0];
                        if(modelos.get(i).getTipo().equals("SubkNN"))
```

```
                                valores[i] = modelos.get(i).getKnn().getVotesForInstanceImpl(inst);
                }
                this.numExemplos++;
                return valores;
        }
        public void atualiza(double yVerdadeiro, double yPrevisto[], FileWriter writer) throws
IOException {
                ArrayList<Double> paraGrafico = new ArrayList<Double>();
                ArrayList<Double> paraGraficoErros = new ArrayList<Double>();
                PrevisaoDependente previsao = new
PrevisaoDependente(yPrevisto,this.anterior,this.eta);
                for(int i=0; i<this.M; i++) {
                        Double novoSoma = new Double((yPrevisto[i]-
yVerdadeiro)+somas.get(i).doubleValue());
                        somas.set(i,novoSoma);
                        Double novoSomaQ = new Double(Math.pow(yPrevisto[i]-yVerdadeiro,
2)+somasQ.get(i).doubleValue());
                        somasQ.set(i,novoSomaQ);
                        Double novoSomaY = new
Double(yPrevisto[i])+somasY.get(i).doubleValue();
                        somasY.set(i,novoSomaY);
                        Double novoSomaM = new Double(Math.abs(yPrevisto[i]-
yVerdadeiro)+somasMAD.get(i).doubleValue());
                        somasMAD.set(i,novoSomaM);
                }
                double yB = previsao.mediaPonderada(yVerdadeiro);
                double yA = previsao.mediaAgregada(yVerdadeiro, yB);
                somasV = somasV+yVerdadeiro;
                somasB = somasB+yB;
                somasA = somasA+yA;
                somasBagging = yB-yVerdadeiro+somasBagging;
                somasAgregada = yA-yVerdadeiro+somasAgregada;
                somasQAgregada = Math.pow(yA-yVerdadeiro,2)+somasQAgregada;
                somasAgregadaMAD=Math.abs(yA-yVerdadeiro)+somasAgregadaMAD;

                somasQBagging = Math.pow(yB-yVerdadeiro,2)+somasQBagging;
                somasBaggingMAD=Math.abs(yB-yVerdadeiro)+somasBaggingMAD;
                paraGrafico.add((double)somasV/numExemplos);
                double grafico2[] = new double[10]; int num2=0;
                switch(tipoErro) {
                case 1: grafico2 = errosMSE(); num2=this.M+2; break;
                case 2: grafico2 = errosMAD(); num2=this.M+2; break;
                case 3: grafico2 = previsao.getPesos(); num2=this.M+1; break;
                }
                for(int i=0; i<this.M; i++) paraGrafico.add((double)somasY.get(i)/numExemplos);
                paraGrafico.add((double)somasB/numExemplos);
                paraGrafico.add((double)somasA/numExemplos);
                for(int i=0; i<num2; i++) paraGraficoErros.add((double)grafico2[i]);
                if(numExemplos>1000) { valoresY.remove(0); valoresSegundo.remove(0); }
                valoresY.add(paraGrafico);
```

```
            valoresSegundo.add(paraGraficoErros);
            fundoGraficos = null;
            fundoGraficos = new FundoGrafico(valoresY, valoresSegundo,modelos,tipoErro);
            if(writer!=null) this.gravaDados(writer,yVerdadeiro,yPrevisto,yB,yA,previsao);
            anterior = previsao;
        }
        public long getNumExemplos() { return this.numExemplos; }
        public double getSomasV() { return this.somasV; }
        public double getSomasB() { return this.somasB; }
        public ArrayList<Double> getSomasY() { return this.somasY; }
        public FundoGrafico getPanel() { return fundoGraficos; }
        public double[] errosMSE() {
            double erros[] = new double[this.M+2];
            long num = this.numExemplos;
            for(int i=0; i<this.M; i++) {
                    double valor = this.somas.get(i).doubleValue();
                    double valorQ = this.somasQ.get(i).doubleValue();
                    erros[i]=valorQ/num-Math.pow(valor/num, 2);
            }
            erros[this.M]=this.somasQBagging/num-Math.pow(this.somasBagging/num, 2);
            erros[this.M+1]=this.somasQAgregada/num-Math.pow(this.somasAgregada/num, 2);
            return erros;
        }
        public double[] errosMAD() {
            double erros[] = new double[this.M+2];
            long num = this.numExemplos;
            for(int i=0; i<this.M; i++) {
                    erros[i]=somasMAD.get(i).doubleValue()/num;
            }
            erros[this.M]=this.somasBaggingMAD/num;
            erros[this.M+1]=this.somasAgregadaMAD/num;
            return erros;
        }
        public void imprimirErros() {
            int num = this.M;
            double erros[] = errosMSE();
            for(int i=0;i<num; i++) {
                    System.out.println("Erro de "+modelos.get(i).getTipo()+": "+erros[i]);
            }
            System.out.println("Erro de Bagging: "+erros[num]);
        }
        public void imprimirDados() {
            int num = this.M;
            double erros[] = errosMSE();
            for(int i=0;i<num; i++) {
                    System.out.println("Erro de "+modelos.get(i).getTipo()+": "+erros[i]);
            }
            System.out.println("Erro de Bagging: "+erros[num]);
            for(int i=0; i<num; i++) {
                    double yBarraM = somasY.get(i).doubleValue()/numExemplos;
```

```
                        System.out.println("Média das previsões por "+modelos.get(i).getTipo()+":
"+yBarraM);
                }
                double yBarraB = somasB/numExemplos;
                System.out.println("Média das previsões por Bagging: "+yBarraB);
                double yBarraV = somasV/numExemplos;
                System.out.println("Média dos verdadeiros: "+yBarraV);
        }
        public void gravaTitulos(FileWriter escreve) throws IOException {
                int num = this.M;
                String str = new String("");
                str = str + "Ordem\t"; str = str + "yTrue\t";
                for(int i=0; i<num; i++) str = str + "y" + modelos.get(i).getTipo() + "\t";
                str = str + "yBagging\t";
                str = str + "yAgregada\t";
                str = str + "mediaTrue\t";
                for(int i=0; i<num; i++) str = str + "media" + modelos.get(i).getTipo() + "\t";
                str = str + "mediaBagging\t";
                str = str + "mediaAgregada\t";
                for(int i=0; i<num; i++) str = str + "MSE" + modelos.get(i).getTipo() + "\t";
                str = str + "MSEBagging\t";
                str = str + "MSEAgregada\t";
                for(int i=0; i<num; i++) str = str + "MAD" + modelos.get(i).getTipo() + "\t";
                str = str + "MADBagging\t";
                str = str + "MADAgregada\t";
                for(int i=0; i<num; i++) str = str + "Peso" + modelos.get(i).getTipo() + "\t";
                str = str + "PesoBagging\t";
                escreve.write(str+"\n");
        }
        public void gravaDados(FileWriter escreve, double yVerdadeiro, double yPrevisto[], double
yB, double yA,
                        PrevisaoDependente previsao) throws IOException {
                double errosMSE[] = errosMSE();
                double errosMAD[] = errosMAD();
                int num = this.M;
                String str = new String("");
                // Previsões
                str = str + numExemplos + "\t";
                str = str + virgula(yVerdadeiro) + "\t";
                for(int i=0; i<num; i++) str = str + virgula(yPrevisto[i]) + "\t";
                str = str + virgula(yB)+"\t";
                str = str + virgula(yA)+"\t";
                // Médias
                str = str + virgula(somasV/numExemplos)+"\t";
                for(int i=0; i<num; i++) str = str +
virgula(somasY.get(i).doubleValue()/numExemplos) + "\t";
                str = str + virgula(somasB/numExemplos)+ "\t";
                str = str + virgula(somasA/numExemplos)+ "\t";
                // Erros
                for(int i=0; i<=num+1; i++) str = str + virgula(errosMAD[i]) + "\t";
```

```
                for(int i=0; i<=num+1; i++) str = str + virgula(errosMSE[i]) + "\t";
                for(int i=0; i<=num; i++) str = str + virgula(previsao.getPesos()[i]) + "\t";
                escreve.write(str+"\n");
        }
        public String virgula(double numero) {
                String str = new String("");
                str = Double.toString(numero).replace('.', ',');
                return str;
        }
}
```

A.2 Package mb.telefonemas

Neste package estão as classes específicas para o problema das chamadas telefónicas. Classe que faz a conversão de um ficheiro de texto para o formato específico do moa.

classe Converter

```
package mb.telefonemas;
import java.io.IOException;
import java.io.FileReader;
import java.io.FileWriter;

public class Converter {
        public static void main(String args[]) throws IOException {
                String fileOrigem = new String("C:/moa-master/moa-master/moa/2012dez02.txt");
                String fileFinal = new String("C:/moa-master/moa-master/moa/2012dez02.arff");
                FileReader oldfile = new FileReader(fileOrigem);          // objeto para aceder ao ficheiro com os dados
                FileWriter newfile = new FileWriter(fileFinal);   // objeto para aceder ao ficheiro novo
                // Variáveis auxiliares para a leitura
                Character letra = new Character(' ');
                int intLetra=0;
                String campo = new String("");
                int numCampo = 0;
                int linha=0;
                // Campos existentes no ficheiro com os dados
                String Texto = new String("");
                String date = new String("");
                String time = new String("");
                String duracao = new String("");
                String anonimo = new String("");
                String outro = new String("");
                String cliente = new String("");
                String inexistente = new String("");
                String originador = new String("");
                String recetor = new String("");
```

```
                String sentido = new String("");
                String tipo = new String("");
                String destino = new String("");
                String caiu = new String("");
                String voicemail = new String("");
                // Escrever o cabeçalho do ficheiro novo
                // O Moa obriga a ter uma lista dos campos, um por linha

                Texto = new String("@relation telefones");      // guardar o que se quer escrever num objeto String
                for(int i=0; i<Texto.length(); i++)             // percorrer o texto caráter a caráter
                        newfile.write(Texto.codePointAt(i));    // Texto.codePointAt(i) retorna o código ASCII do
caráter da posição i
                newfile.write(10);                               // imprime no ficheiro o caráter de mudança de linha
                // Os campos restantes são impressos de forma análoga
                Texto = new String("@attribute Data numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Hora numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Anonimo numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Outro numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Cliente numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Sentido numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Tipo numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Destino numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Caiu numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute VoiceMail numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@attribute Duracao numeric");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);
                Texto = new String("@data");
                for(int i=0; i<Texto.length(); i++) newfile.write(Texto.codePointAt(i)); newfile.write(10);

                // Percorrer o ficheiro com os dados
                long numRegisto=0;
                do {
                        intLetra = oldfile.read();              // a função read() lê um caráter do ficheiro antigo,
retorna
                        letra = new Character((char) intLetra);         // o código desse caráter e faz-se os cast para
caráter
                        if(intLetra==10) {                              // se intLetra==10 é porque é o fim da linha
                                linha++;                                // e incrementa-se a variável que conta o
número de linhas
                                if(linha!=1) {                  // A primeira linha não interessa, tem os nomes dos
```

```
campos
                                        voicemail = new String(campo);          // Em cada linha o último campo é voicemail
                                        Antigo antigo = new Antigo(date, time, duracao, anonimo, outro, cliente, inexistente,
originador, recetor, sentido, tipo, destino, caiu, voicemail);                          // Cria-se um objeto, antigo, da classe
Antigo
                                                Novo novo = new Novo(antigo);  // Cria-se, a partir do antigo, um objeto, novo,
da classe Novo
                                                // Gravar os dados novos, no ficheiro novo
                                                acrescenta(novo,newfile);
                                                numRegisto++;
                                                System.out.println("Número de registo: "+numRegisto);
                                        }
                                        numCampo=0;  // preparar a variável que conta o número de campos, para a próxima
linha
                                        campo = new String("");             // preparar o objeto String onde se vai ler o valor de
cada campo
                                } else {
                                        // No ficheiro antigo, o caráter '|' separa os campos, por isso, enquanto o caráter lido for
diferente
                                        if(letra!='|') campo = campo+letra.toString();       // de '|' acrescenta-se o caráter à String
campo
                                        else {
                                                // quando é lido '|', acabou o campo e incrementa-se a variável com o número de
campos
                                                        numCampo++;
                                                switch(numCampo) {
                                                        // dependendo do número do campo, preenche-se a String que irá ser
usada
                                                        // no construtor da classe Antigo
                                                        case 1: date = new String(campo); break;
                                                        case 2: time = new String(campo); break;
                                                        case 3: duracao = new String(campo); break;
                                                        case 4: anonimo = new String(campo); break;
                                                        case 5: outro = new String(campo); break;
                                                        case 6: cliente = new String(campo); break;
                                                        case 7: inexistente = new String(campo); break;
                                                        case 8: originador = new String(campo); break;
                                                        case 9: recetor = new String(campo); break;
                                                        case 10: sentido = new String(campo); break;
                                                        case 11: tipo = new String(campo); break;
                                                        case 12: destino = new String(campo); break;
                                                        case 13: caiu = new String(campo); break;
                                                }
                                                campo = new String("");                  // preparar a String campo para o
próximo campo
                                        }
                                }

                } while(intLetra!=-1);              // quando intLetra==-1 chegou ao fim do ficheiro
```

```
                // Fechar os ficheiros
                newfile.close();
                oldfile.close();
        }

        static void acrescenta(Novo novo, FileWriter arquivo) throws IOException {
                // No objeto novo estão os novos campos, que irão ser escritos no ficheiro novo
                String escreve = new String("");
                int i;
                // Pormenores do primeiro campo
                // o valor do campo novo é um inteiro mas usa-se toString() para converter para String
                escreve = Integer.toString(novo.getData());
                for(i=0;i<escreve.length();i++)                 // percorre-se a String com o valor do novo campo
                        arquivo.write(escreve.codePointAt(i));  // e escreve-se, no ficheiro novo, caráter a caráter
                arquivo.write(44);                                      // no fim do campo escreve-se uma vírgula,
código ASCII 44
                escreve = Integer.toString(novo.getHora());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                //escreve = Integer.toString(novo.getAnonimo());
                escreve = Double.toString(novo.getAnonimo());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                //escreve = Integer.toString(novo.getOutro());
                escreve = Double.toString(novo.getOutro());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                //escreve = Integer.toString(novo.getCliente());
                //for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getSentido());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getTipo());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getDestino());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getCaiu());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getVoiceMail());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(44);
                escreve = Integer.toString(novo.getDuracao());
                for(i=0;i<escreve.length();i++) arquivo.write(escreve.codePointAt(i)); arquivo.write(10);
                // depois do último campo não se escreve uma vírgula mas sim o caráter de mudança de linha, código
ASCII 10
        }
}

class Novo {
        // Variáveis de instância, os únicos campos que interessam
        private Integer Data;           // Dia do ano: 1 a 366
        private Integer Hora;           // Minuto do dia: 0 a 24*60
        //private Integer Anonimo;      // Número (anonimizado) do telefone "em causa".
        //private Integer Outro;        // Número do "outro", o originador da chamada (Incoming) ou destinatário
(Outgoing).
```

```
        private Double Anonimo;
        private Double Outro;
        //private Integer Cliente;          // "Código cliente". Autónomos dos MSISDN
        private Integer   Sentido;          // Sentido: 0 - chamada efetuada; 1 - chamada recebida
        private Integer Tipo;               // Tipo: 1-FI; 2-MO; 3-ON; 4-OT; 5-SV;
        private Integer   Destino;          // Destino: 0-Local; 1-Internacional;
        private Integer   Caiu;             // Caiu: 0-Não; 1-Sim;
        private Integer   VoiceMail;        // VoiceMail: 0-Não; 1-Sim;
        private Integer Duracao;// Duração: em segundos
        // Construtores
        Novo() { }
        Novo(Antigo antigo) {
                // Construtor para criar um objeto da classe Novo, a partir de um objeto da classe Antigo
                this.Data = new Integer(diaAno(antigo.getDate().substring(4,8)));           // diaAno() retorna o número
do dia no ano
                this.Hora = new Integer(minDia(antigo.getTime()));                  // minDia() retorna o número do minuto
no dia

                //this.Anonimo = new Integer(antigo.getAnonimo());
                //this.Outro = new Integer(antigo.getOutro());
                this.Anonimo = new Double(antigo.getAnonimo()); this.Anonimo = new
Double(this.Anonimo.doubleValue()/10000);
                this.Outro = new Double(antigo.getOutro()); this.Outro = new Double(this.Outro.doubleValue()/10000);
                //this.Cliente = new Integer(antigo.getCliente());

                // cria o inteiro com o sentido da chamada mediante convenção (ver variável Sentido)
                if(antigo.getDirection().equals("O")) this.Sentido = new Integer(0); else this.Sentido = new Integer(1);
                // cria o inteiro com o tipo da chamada mediante convenção (ver variável Tipo)
                if(antigo.getCallType().equals("FI")) this.Tipo = new Integer(1);
                else if(antigo.getCallType().equals("MO")) this.Tipo = new Integer(2);
                else if(antigo.getCallType().equals("ON")) this.Tipo = new Integer(3);
                else if(antigo.getCallType().equals("OT")) this.Tipo = new Integer(4);
                else this.Tipo = new Integer(5);
                // cria o inteiro com o destino da chamada mediante convenção (ver variável Destino)
                if(antigo.getDestination().equals("L")) this.Destino = new Integer(0); else this.Destino = new Integer(1);
                // cria o inteiro mediante convenção (ver variável Caiu)
                if(antigo.getDroppedCall().equals("N")) this.Caiu = new Integer(0); else this.Caiu = new Integer(1);
                // cria o inteiro mediante convenção (ver variável VoiceMail)
                if(antigo.getVoicemail().equals("N")) this.VoiceMail = new Integer(0); else this.VoiceMail = new
Integer(1);
                this.Duracao = new Integer(antigo.getDuration());                           // cria o inteiro com a duração,
em segundos
        }
        // Get's
        public Integer getData() { return this.Data; }
        public Integer getHora() { return this.Hora; }
        //public Integer getAnonimo() { return this.Anonimo; }
        //public Integer getOutro() { return this.Outro; }
        public Double getAnonimo() { return this.Anonimo; }
        public Double getOutro() { return this.Outro; }
```

```
//public Integer getCliente() { return this.Cliente; }
public Integer getSentido() { return this.Sentido; }
public Integer getTipo() { return this.Tipo; }
public Integer getDestino() { return this.Destino; }
public Integer getCaiu() { return this.Caiu; }
public Integer getVoiceMail() { return this.VoiceMail; }
public Integer getDuracao() { return this.Duracao; }

// Funções
int diaAno(String mesdia) {
	// Recebe uma String com o mês e o dia e retorna o número de ordem do dia no ano
	// Exemplo: "1021", corresponde ao dia 21 de Outubro, então, é preciso somar o número de dias de
	// todos os meses de Janeiro a Setembro, 31+29+31+30+31+30+31+31+30, depois somar ainda 21
	// Os dados, para já, dizem respeito a 2012, um ano bissexto

	Integer Mes, Dia;
	int mes, dia, numDias=0;
	Mes = new Integer(mesdia.substring(0,2)); mes = Mes.intValue();
	Dia = new Integer(mesdia.substring(2,4)); dia = Dia.intValue();
	switch(mes) {
		case 1: numDias = 0; break;
		case 2: numDias = 31; break;
		case 3: numDias = 31+29; break;
		case 4: numDias = 31+29+31; break;
		case 5: numDias = 31+29+31+30; break;
		case 6: numDias = 31+29+31+30+31; break;
		case 7: numDias = 31+29+31+30+31+30; break;
		case 8: numDias = 31+29+31+30+31+30+31; break;
		case 9: numDias = 31+29+31+30+31+30+31+31; break;
		case 10: numDias = 31+29+31+30+31+30+31+31+30; break;
		case 11: numDias = 31+29+31+30+31+30+31+31+30+31; break;
		case 12: numDias = 31+29+31+30+31+30+31+31+30+31+30; break;
	}
	numDias = numDias+dia;
	return numDias;
}

int minDia(String horario) {
	// Recebe uma String com o horário a que se iniciou a chamada e retorna a ordem do minuto no dia
	// Exemplo: "173047" são 17 horas, 30 minutos e 47 segundo por isso é o minuto: 17*60+30
	// arredondando os segundos ao minuto é o minuto 17*60+31
	Integer Hora, Minuto, Segundo;
	int hora, minuto, segundo, numMin=0;
	if(horario.length()==6) {
		// se a String tem 6 carateres é hhmmss
		Hora = new Integer(horario.substring(0,2)); hora = Hora.intValue();
		Minuto = new Integer(horario.substring(2,4)); minuto = Minuto.intValue();
		Segundo = new Integer(horario.substring(4,6)); segundo = Segundo.intValue();
	} else if(horario.length()==4) {
		// se a String tem 4 carateres é mmss, logo hora = 0
```

```
                        hora = 0;
                        Minuto = new Integer(horario.substring(0,2)); minuto = Minuto.intValue();
                        Segundo = new Integer(horario.substring(2,4)); segundo = Segundo.intValue();
                } else if(horario.length()==2) {
                        // se a String tem apenas 2 carateres é ss, logo hora = 0 e minuto = 0
                        hora = 0; minuto = 0;
                        Segundo = new Integer(horario.substring(0,2)); segundo = Segundo.intValue();
                } else {
                        hora = 0; minuto = 0; segundo = 0;
                }
                numMin=60*hora+minuto;
                if(segundo>=30) numMin++;
                return numMin;
        }
}

class Antigo {
        // Variáveis de instância
        private String DATE;                    // Data chamada
        private String TIME;                    // Hora a que foi efetuada a chamada
        private String DURATION;                        // Duração da chamada
        private String MSISDN;                  // Número (anonimizado) do telefone "em causa".
                                                // Se for Incoming, é o nº que recebe a chamada,
                                                // se for Outgoing, é o que está a iniciar a chamada.
        private String OTHER_MSISDN;                    // Número do "outro", o originador da chamada (Incoming) ou
destinatário (Outgoing).
        private String CONTRACT_ID;             // "Código cliente". Autónomos dos MSISDN
        private String OTHER_CONTRACT_ID;       // Nem todos os registos tem porque não pertecem ao operador
        private String START_CELL_ID;                   // Deveria corresponder ao originador da chamada,
                                                // pelo que deveria estar preenchido quando o registo é Outgoing.
                                                // Mas não acontece. Não fiaveis
        private String END_CELL_ID;             // Deveria corresponder ao receptor, e por isso preenchido nos registos
Incoming.
                                                // Mas não acontece Não fiaveis
        private String DIRECTION;                       // Indica se o registo indica a recepção de uma chamada
("I"ncoming) ou
                                                // a realização de uma chamada para outro número ("O"utgoing).
        private String CALL_TYPE;                       // Tipo de chamada
        private String DESTINATION_TYPE;  // Se destino é local (L) ou internacional (I)
        private String DROPPED_CALL;                    // Chamadas caiu (Yes or No)
        private String VOICEMAIL;                       // Chamada foi para voicemail (Yes or No)
        // Construtores
        Antigo() { }
        Antigo(String date, String time, String duracao, String anonimo, String outro, String cliente, String inexistente,
String originador, String recetor, String sentido, String tipo, String destino, String caiu, String voicemail) {
                // Construtor para criar um objeto da classe Antigo, portanto com os campos existentes no ficheiro de
dados
                this.DATE = date;
                this.TIME = time;
                this.DURATION = duracao;
```

```
                this.MSISDN = anonimo;
                this.OTHER_MSISDN = outro;
                this.CONTRACT_ID = cliente;
                this.OTHER_CONTRACT_ID = originador;
                this.END_CELL_ID = recetor;
                this.DIRECTION = sentido;
                this.CALL_TYPE = tipo;
                this.DESTINATION_TYPE = destino;
                this.DROPPED_CALL = caiu;
                this.VOICEMAIL = voicemail;
        }
        // Get's
        public String getDate() { return this.DATE; }
        public String getTime() { return this.TIME; }
        public String getAnonimo() { return this.MSISDN; }
        public String getOutro() { return this.OTHER_MSISDN; }
        public String getCliente() { return this.CONTRACT_ID; }
        public String getDuration() { return this.DURATION; }
        public String getDirection() { return this.DIRECTION; }
        public String getCallType() { return this.CALL_TYPE; }
        public String getDestination() { return this.DESTINATION_TYPE; }
        public String getDroppedCall() { return this.DROPPED_CALL; }
        public String getVoicemail() { return this.VOICEMAIL; }
}
```

A.3 Package mb.tests

Neste package estão as classes específicas usadas para aplicar os modelos.

classe TestaEnsembleHeterogenio

```
package mb.tests;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import mb.classifiers.EnsembleHeterogenio;
import mb.utils.GraficoDuplo;
import com.yahoo.labs.samoa.instances.Instance;
import com.yahoo.labs.samoa.instances.Instances;
import javax.swing.SwingUtilities;

public class TestaEnsembleHeterogenio {
        public static void main(String args[]) throws IOException {
                int tipoErro=1;
```

```
		int tipoBagging=1;
		int p=1, k=10;
		double eta=0.0001;
		int numAtributos = 11; long numExemplos, ene=1000;
		FileReader reader; FileWriter writer;
		Instances instâncias;
		long nTeste = 9000000;
		String msg;
		Scanner leitor = new Scanner(System.in);
		msg = new String("1. Bagging Homogéneo;\n2. Bagging Heterogéneo;\nQual o tipo de
Bagging? ");
		System.out.print(msg); tipoBagging=leitor.nextInt();
		msg = new String("\nQual o valor de p, no último modelo kNN? ");
		System.out.print(msg); p=leitor.nextInt();
		msg = new String("\nQual o valor de k, no último modelo kNN? ");
		System.out.print(msg); k=leitor.nextInt();
		msg = new String("\nQual o valor de eta? ");
		System.out.print(msg); eta=leitor.nextDouble();
		msg = new String("\n1. MSE;\n2. MAD;\n3. Peso\nQue quer ver no 2º gráfico? ");

		System.out.print(msg); tipoErro=leitor.nextInt();
		msg = new String("\nQuantas instâncias quer visualizar? ");
		System.out.print(msg); ene = leitor.nextInt();
		final EnsembleHeterogenio bagging = new
EnsembleHeterogenio(tipoErro,tipoBagging,p,k,eta);
		leitor.close();

		try {
			writer = new FileWriter("C:/moa-master/moa-master/moa/2012dez02.csv");
			reader = new FileReader("C:/moa-master/moa-master/moa/2012dez02.arff");
			instâncias = new Instances(reader,0,numAtributos);
		} catch(Exception e) { System.out.println("Erro ao abrir o ficheiro."); return; }
		// Caso tenha lido o ficheiro com sucesso, começa por encher a janela
		int i = 0;
		while(i<bagging.tamanhoJanela()) {
			instâncias.readInstance(reader);
			Instance inst = instâncias.get(i);
			bagging.trainOnInstance(inst); // Enquanto está a encher a janela não faz
previsão, apenas treina
			i++; }

		numExemplos = 0;
		// Guardar alguns valores
		//if(numExemplos>nTeste-1000)
		bagging.gravaTitulos(writer);
		// Depois de já ter a janela cheia passa fazer previsões

		final GraficoDuplo gd = new GraficoDuplo();
		SwingUtilities.invokeLater(new Runnable() {
			@Override public void run() { gd.inicializacao(); }
```

```
                });
                while(instâncias.readInstance(reader) && numExemplos<ene) {
                // Depois de acrescentar uma nova instância, para manter o tamanho da janela, retira-se
a primeira instância
                        instâncias.delete(0);
                        // A previsão será feita sobre a última instância acrescentada
                        final Instance inst = instâncias.get(bagging.tamanhoJanela()-1);
                        // Testar, portanto, fazer a previsão
                        final double yVerdadeiro = inst.value(numAtributos-1);
                        final double yPrevisto[] = bagging.getVotesForInstance(inst);
                        bagging.atualiza(yVerdadeiro,yPrevisto,writer);
                        SwingUtilities.invokeLater(new Runnable() {
                                @Override public void run() {
                                        try { gd.getContentPane().add(bagging.getPanel(),0);
gd.repaint(); }
                                        catch (NullPointerException e) { }
                                }
                        });
                        // Treinar, portanto, acrescentar à janela
                        bagging.trainOnInstance(inst);
                        numExemplos++;
                }
                reader.close();
                writer.close();

        }

}
```

classe TestaSubFIMTDD

```
package mb.tests;

import java.io.FileReader;
import com.yahoo.labs.samoa.instances.Instances;
import mb.classifiers.SubFIMTDD;

public class TestaSubFIMTDD {
        public static void main(String[] args) {
                SubFIMTDD fimtdd = new SubFIMTDD();
                fimtdd.prepareForUse();
                int N=fimtdd.capacidadeJanela();
                int T=10;
                int numInstâncias = N+T;
                int numAtributos = 11;
                int numExemplos;
                Instances instâncias;
                FileReader arff;
```

```
                try {
                        arff = new FileReader("C:/moa-master/moa-master/moa/2012dez01.arff");
                        instâncias = new Instances(arff,numInstâncias,numAtributos);
                        for(int i=0; i<numInstâncias; i++) instâncias.readInstance(arff);
                } catch(Exception e) { System.out.println("Erro ao abrir o ficheiro."); return; }


                // Caso tenha lido o ficheiro com sucesso
                numExemplos = 0;
                while(numExemplos<N) { fimtdd.trainOnInstance(instâncias.get(numExemplos));
numExemplos++; }
                while(numExemplos<N+T) {
                        // Testar, portanto, fazer a previsão
                        double previsao[] =
fimtdd.getVotesForInstance(instâncias.get(numExemplos));
                        // para explorar: knn.getPredictionForInstance(instâncias.get(numExemplos));
                        double yVerdadeiro = instâncias.get(numExemplos).value(numAtributos-1);
                        System.out.println("y="+yVerdadeiro+"\tf(X)="+previsao[0]+".");
                        // Treinar, portanto, acrescentar à janela
                        fimtdd.trainOnInstance(instâncias.get(numExemplos));
                        numExemplos++;
                }


        }

}
```

classe TestaSubKNN

```
package mb.tests;

import java.io.FileReader;
import com.yahoo.labs.samoa.instances.Instances;
import mb.classifiers.SubkNN;

public class TestaSubkNN {
        public static void main(String[] args) {
                SubkNN knn = new SubkNN(true);
                int N=knn.capacidadeJanela();
                int T=50;
                int numInstâncias = N+T;
                int numAtributos = 11;
                int numExemplos;
                Instances instâncias;
                FileReader arff;
```

```
                try {
                        arff = new FileReader("C:/moa-master/moa-master/moa/2012dez01.arff");
                        instâncias = new Instances(arff,numInstâncias,numAtributos);
                        for(int i=0; i<numInstâncias; i++) instâncias.readInstance(arff);
                } catch(Exception e) { System.out.println("Erro ao abrir o ficheiro."); return; }
                // Caso tenha lido o ficheiro com sucesso
                // Encher a janela
                numExemplos = 0;
                while(numExemplos<N) { knn.trainOnInstance(instâncias.get(numExemplos));
numExemplos++; }
                while(numExemplos<N+T) {
                        // Testar, portanto, fazer a previsão
                        double previsao =
knn.getVotesForInstanceImpl(instâncias.get(numExemplos));
                        // para explorar: knn.getPredictionForInstance(instâncias.get(numExemplos));
                        double yVerdadeiro = instâncias.get(numExemplos).value(numAtributos-1);
                        System.out.println("y="+yVerdadeiro+"\tf(X)="+previsao+".");
                        // Treinar, portanto, acrescentar à janela
                        knn.trainOnInstance(instâncias.get(numExemplos));
                        numExemplos++;
                }


        }

}
```

classe TestaFriedman

```
package mb.tests;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import mb.utils.Friedman;

public class TestaFriedman {
        public static void main(String args[]) throws IOException {
                FileReader amostra = new FileReader("C:/moa-master/moa-
master/moa/diaInteiro.csv");
                FileWriter stats = new FileWriter("C:/moa-master/moa-master/moa/ranks.csv");
                FileWriter multiplas = new FileWriter("C:/moa-master/moa-
master/moa/multiplas.csv");
                String campos[] = new String[6];
                campos[0] = new String("MSESubFIMTDD"); campos[1] = new
String("MSESubkNN"); campos[2] = new String("MSESubkNN");
                campos[3] = new String("MSESubkNN"); campos[4] = new String("MSEBagging");
```

```
campos[5] = new String("MSEAgregada");

        stats.write("MSESubFIMTDD\tMSESubkNN\tMSESubkNN\tMSESubkNN\tMSEBagging\tM
SEAgregada\n");
                Friedman friedman = new Friedman(amostra,campos,500000);
                long i=0;
                boolean haIndividuos = true;
                while(haIndividuos) {
                        haIndividuos = friedman.leSujeito();
                        i++;
                        if(i%500000==0) System.out.println("Já foram analisados "+i+" indíviduos.");
                }
                System.out.println("TERMINOU: Foram lidos "+(i-1)+" indivíduos");
                System.out.println("Estatística do teste: "+friedman.estatistica(stats,multiplas));
                amostra.close();
                stats.close();
                multiplas.close();
        }

}
```

classe AlteraMSE

```
package mb.tests;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class AlteraMSE {
        public static void main(String args[]) throws IOException {
                FileReader leitura = new FileReader("C:/moa-master/moa-
master/moa/2012dez01.csv");
                FileWriter escrita = new FileWriter("C:/moa-master/moa-master/moa/diaInteiro.csv");
                Scanner leLinha = new Scanner(leitura), leCampo;
                boolean haLinha=true, haCampo=true; Double raizQ;
                String linha = new String(), campo = new String(), novaLinha = new String();
                long numLinhas=0;
                int numCampo=0;
                try { linha=leLinha.nextLine(); }
                catch(Exception e) { haLinha=false; }
                // Linha dos títulos
                String titulos = new String("Ordem\t");   // 1
                titulos +=
"yTrue\tySubFIMTDD\tySubkNN\tySubkNN\tySubkNN\tyBagging\tyAgregada\t";
                titulos +=
"mediaTrue\tmediaSubFIMTDD\tmediaSubkNN\tmediaSubkNN\tmediaSubkNN\tmediaBagging\tmedi
```

```
aAgregada\t";
                titulos +=
"MSESubFIMTDD\tMSESubkNN\tMSESubkNN\tMSESubkNN\tMSEBagging\tMSEAgregada\t";

                titulos +=
"MADSubFIMTDD\tMADSubkNN\tMADSubkNN\tMADSubkNN\tMADBagging\tMADAgregada\t";

                titulos +=
"PesoSubFIMTDD\tPesoSubkNN\tPesoSubkNN\tPesoSubkNN\tPesoBagging\t\n";          /
                escrita.write(titulos);
                try { linha=leLinha.nextLine(); }
                catch(Exception e) { haLinha=false; }
                while(haLinha) {
                        leCampo = new Scanner(linha); haCampo=true;
                        numCampo=0; novaLinha = new String();
                        while(haCampo) {
                                try { campo=leCampo.next(); }
                                catch(Exception e) { haCampo=false; }
                                if(haCampo) {
                                        if(numCampo>=15 && numCampo<=20) {
                                                raizQ = new
Double(Math.sqrt(Double.parseDouble(campo.replace(',', '.'))));
                                                novaLinha = novaLinha + raizQ.toString().replace('.',
',') +"\t";
                                        } else novaLinha = novaLinha+campo+"\t";
                                        numCampo++;
                                }
                        }
                        novaLinha = novaLinha+"\n";
                        escrita.write(novaLinha);
                        if(numLinhas%500000 ==0) System.out.println("Já foram alteradas
"+numLinhas+" instâncias");
                        try { linha=leLinha.nextLine(); }
                        catch(Exception e) { haLinha=false; }
                        if(haLinha) numLinhas++;
                }
                leLinha.close();
                leitura.close();
                escrita.close();
                System.out.println("\n\n=== Terminado. ===\n\n");
        }
}
```

classe TestaDiaInteiro

```
package mb.tests;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

import mb.classifiers.EnsembleHeterogenio;
import mb.utils.GraficoDuplo;

import com.yahoo.labs.samoa.instances.Instance;
import com.yahoo.labs.samoa.instances.Instances;
import javax.swing.SwingUtilities;

public class TestaEnsembleHeterogenio {
	public static void main(String args[]) throws IOException {
		int tipoErro=1;
		int tipoBagging=1;
		int p=1, k=10;
		double eta=0.0001;
		int numAtributos = 11; long numExemplos, ene=1000;
		FileReader reader; FileWriter writer;
		Instances instâncias;
		long nTeste = 9000000;
		String msg;
		Scanner leitor = new Scanner(System.in);
		msg = new String("1. Bagging Homogéneo;\n2. Bagging Heterogéneo;\nQual o tipo de
Bagging? ");
		System.out.print(msg); tipoBagging=leitor.nextInt();
		msg = new String("\nQual o valor de p, no último modelo kNN? ");
		System.out.print(msg); p=leitor.nextInt();
		msg = new String("\nQual o valor de k, no último modelo kNN? ");
		System.out.print(msg); k=leitor.nextInt();
		msg = new String("\nQual o valor de eta? ");
		System.out.print(msg); eta=leitor.nextDouble();
		msg = new String("\n1. MSE;\n2. MAD;\n3. Peso\nQue quer ver no 2º gráfico? ");

		System.out.print(msg); tipoErro=leitor.nextInt();
		msg = new String("\nQuantas instâncias quer visualizar? ");
		System.out.print(msg); ene = leitor.nextInt();
		final EnsembleHeterogenio bagging = new
EnsembleHeterogenio(tipoErro,tipoBagging,p,k,eta);
		leitor.close();
```

```
		try {
			writer = new FileWriter("C:/moa-master/moa-master/moa/2012dez01.csv");
			reader = new FileReader("C:/moa-master/moa-master/moa/2012dez01.arff");
			instâncias = new Instances(reader,0,numAtributos);
		} catch(Exception e) { System.out.println("Erro ao abrir o ficheiro."); return; }
		// Caso tenha lido o ficheiro com sucesso, começa por encher a janela
		int i = 0;
		while(i<bagging.tamanhoJanela()) {
			instâncias.readInstance(reader);
			Instance inst = instâncias.get(i);
			bagging.trainOnInstance(inst); // Enquanto está a encher a janela não faz
previsão, apenas treina
			i++; }

		numExemplos = 0;
		// Guardar alguns valores
		//if(numExemplos>nTeste-1000)
		bagging.gravaTitulos(writer);
		// Depois de já ter a janela cheia passa fazer previsões

		final GraficoDuplo gd = new GraficoDuplo();
		SwingUtilities.invokeLater(new Runnable() {
			@Override public void run() { gd.inicializacao(); }
		});
		while(instâncias.readInstance(reader) && numExemplos<ene) {
		// Depois de acrescentar uma nova instância, para manter o tamanho da janela, retira-se
a primeira instância
			instâncias.delete(0);
			// A previsão será feita sobre a última instância acrescentada
			final Instance inst = instâncias.get(bagging.tamanhoJanela()-1);
			// Testar, portanto, fazer a previsão
			final double yVerdadeiro = inst.value(numAtributos-1);
			final double yPrevisto[] = bagging.getVotesForInstance(inst);
			bagging.atualiza(yVerdadeiro,yPrevisto,writer);
			SwingUtilities.invokeLater(new Runnable() {
				@Override public void run() {
					try { gd.getContentPane().add(bagging.getPanel(),0);
gd.repaint(); }
					catch (NullPointerException e) { }
				}
			});
			// Treinar, portanto, acrescentar à janela
			bagging.trainOnInstance(inst);
			numExemplos++;

		}
		reader.close();
		writer.close();
```

```
	}
}
```

A.4 Package mb.utils

Classes auxiliares

classe comparaModelos

```
package mb.utils;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import com.yahoo.labs.samoa.instances.Instance;
import com.yahoo.labs.samoa.instances.Instances;

import mb.classifiers.EnsembleHeterogenio;
import mb.classifiers.SubFIMTDD;
import mb.classifiers.SubkNN;
import mb.utils.Modelo;

public class ComparaModelos extends JFrame {
	// Variáveis
	private static final long serialVersionUID = 1L;
	static EnsembleHeterogenio bagging;
	ArrayList<Modelo> modelos = new ArrayList<Modelo>();
	ArrayList<ArrayList<Double>> valoresY = new ArrayList<ArrayList<Double>>();
	ArrayList<ArrayList<Double>> valoresErro = new ArrayList<ArrayList<Double>>();
	FundoGrafico fundo;
	// Construtor
	public ComparaModelos() {
		SubFIMTDD fimtdd = new SubFIMTDD();
		Modelo modelo1 = new Modelo("SubFIMTDD",fimtdd); modelos.add(modelo1);
		SubkNN knn1 = new SubkNN();
		Modelo modelo2 = new Modelo("SubkNN",knn1); modelos.add(modelo2);
		SubkNN knn2 = new SubkNN(20,fimtdd.capacidadeJanela(),2,false);
		Modelo modelo3 = new Modelo("SubkNN",knn2); modelos.add(modelo3);
		SubkNN knn3 = new SubkNN(10,fimtdd.capacidadeJanela(),1,false);
		Modelo modelo4 = new Modelo("SubkNN",knn3); modelos.add(modelo4);
		bagging = new EnsembleHeterogenio(modelos);
```

```
		//Modelo modelo5 = new Modelo("EnsembleHeterogenio",bagging);
modelos.add(modelo5);

		this.setTitle("Compara Modelos");
		this.setSize(1100,700);
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);
	}

	public static void main(String[] args) throws IOException {

				int numAtributos = 11;
				long numExemplos;
				FileReader reader;
				FileWriter writer;
				Instances instâncias;
				int tam=0;
				try {
					writer = new FileWriter("C:/moa-master/moa-
master/moa/2012dez01.csv");
					reader = new FileReader("C:/moa-master/moa-
master/moa/2012dez01.arff");
					instâncias = new Instances(reader,0,numAtributos);
				} catch(Exception e) { System.out.println("Erro ao abrir o ficheiro.");
return; }
				int i = 0;
				//while(i<janela.bagging.tamanhoJanela()) {
				try { tam = bagging.tamanhoJanela(); } catch(Exception e) {
System.out.println("Erro janela."); return; }
				while(i<tam) {
					instâncias.readInstance(reader);
					Instance inst = instâncias.get(i);
					//janela.bagging.trainOnInstance(inst);
					bagging.trainOnInstance(inst);
					i++; }
				writer.close();
				reader.close();

	}
	void addValores(long numExemplos) {
		ArrayList<Double> paraGrafico = new ArrayList<Double>();
		ArrayList<Double> paraGraficoErros = new ArrayList<Double>();

		paraGrafico.add((double)bagging.getSomasV()/numExemplos);
		double erros[] = bagging.errosMSE();
		for(int i=0; i<modelos.size(); i++) {
			paraGrafico.add((double) bagging.getSomasY().get(i)/numExemplos);
			paraGraficoErros.add((double)erros[i]);
		}
		paraGrafico.add((double) bagging.getSomasB()/numExemplos);
		paraGraficoErros.add((double)erros[modelos.size()]);
```

```
                if(numExemplos>1000) { valoresY.remove(0); valoresErro.remove(0); }
                valoresY.add(paraGrafico);
                valoresErro.add(paraGraficoErros);
        }
}
```

classe Friedman

```
package mb.utils;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class Friedman {
        // Variáveis
        // Ranks atribuídos a cada amostra, média por grupo
        ArrayList<ArrayList<Double>> ranks = new ArrayList<ArrayList<Double>>();
        double somas[];         // Soma dos ranks dos sujeitos sem grupo
        int k = 0;                      // Número de amostras
        int n = 0;                      // Número de sujeitos sem grupo
        int nPorGrupo=0;        // Número de sujeitos por grupo
        int indices[];          // Posição dos campos que interessam
        Scanner leitor;         // Ficheiro com os dados
        double rankAtual[];     // Ranks do sujeito atual
        // Construtores
        public Friedman(FileReader ficheiro, String campos[], int nPorGrupo) throws IOException {
                leitor = new Scanner(ficheiro);
                indices = new int[campos.length];
                rankAtual = new double[campos.length];
                somas = new double[campos.length];
                // Ver os campos
                String titulos;
                int num=0; int j=0;
                while(num<campos.length) {
                        titulos = leitor.next();
                        if(titulos.equals(campos[num])) {
                                ArrayList<Double> amostra = new ArrayList<Double>();
                                ranks.add(amostra); k++;
                                indices[num]=j;
                                somas[num] = 0.0;
                                num++;
                        }
                        j++;
                }
```

```
                leitor.nextLine();
                this.nPorGrupo = nPorGrupo;
        }
        // Métodos
        public boolean leSujeito() {
                double erros[] = new double[k];
                int posicoes[] = new int[k];
                boolean haSujeitos = true;
                Double valor = new Double(0.0);
                int num=0; int j=0;
                while(num<k) {
                        try { valor = leitor.nextDouble(); }
                        catch(Exception e) { haSujeitos=false; break; };
                        if(j==indices[num]) { erros[num]=valor; posicoes[num]=num; num++; }
                        j++; }
                if(haSujeitos) {
                        leitor.nextLine();
                        // ordena
                        int fim=k;
                        while(fim>0) {
                                for(j=0; j<fim-1; j++) {
                                        if(erros[j]>erros[j+1]) {
                                                double auxD = erros[j+1]; erros[j+1] = erros[j];
erros[j] = auxD;
                                                int auxI = posicoes[j+1]; posicoes[j+1] = posicoes[j];
posicoes[j] = auxI;
                                        }
                                }
                                fim--;
                        }
                        for(j=0; j<k; j++) rankAtual[j] = 0.0;
                        atualiza(erros,posicoes);
                        this.n++;
                        for(j=0; j<k; j++) somas[j] = somas[j]+rankAtual[j];
                        if(this.n == this.nPorGrupo) {
                                for(j=0; j<k; j++) {
                                        Double novoRank = new Double(somas[j]/nPorGrupo);
                                        this.ranks.get(j).add(novoRank);
                                }
                                this.n=0;
                                for(j=0; j<k; j++) somas[j] = 0.0;
                        }
                }
                return haSujeitos;
        }
        private void atualiza(double erros[], int posicoes[]) {
                int rEmp, vR, nEmp;
                vR=1; rankAtual[0]=vR; nEmp=1; rEmp=vR;
                for(int j=1; j<k; j++) {
                        vR++;
```

```
                        if(erros[j-1]==erros[j]) {
                                nEmp++;
                                rEmp+=vR;
                        }
                        else {
                                for(int s=j-1; s>j-1-nEmp; s--)
rankAtual[posicoes[s]]=(double)rEmp/nEmp;
                                nEmp=1; rEmp=vR;
                        }
                }
                for(int s=k-1; s>k-1-nEmp; s--) rankAtual[posicoes[s]] = (double) rEmp/nEmp;
        }
        public void fecha() { leitor.close(); }
        public void imprimir() {
                System.out.println("Número de amostras: "+this.k);
                for(int i=0; i<this.n; i++) {
                        System.out.print((i+1)+"\t");
                        for(int j=0; j<this.k; j++) System.out.print(ranks.get(j).get(i)+"\t");
                        System.out.println();
                }
                System.out.println("---");
        }
        public double estatistica(FileWriter stats, FileWriter multiplas) throws IOException {
                double et=0;
                double R[] = new double[k];
                int N = ranks.get(0).size();
                for(int j=0; j<k; j++) R[j]=0.0;
                // Grava ranks
                for(int i=0; i<N; i++) {
                        String linha="";
                        for(int j=0; j<k; j++) { linha += ranks.get(j).get(i)+"\t"; }
                        stats.write(linha.replace('.', ',')+"\n");
                }
                // Calcula anova não paramétrica
                for(int j=0; j<k; j++) for(int i=0; i<N; i++) R[j] += ranks.get(j).get(i);
                for(int j=0; j<k; j++) R[j] = (double) R[j]/N;
                double sq=0.0;
                for(int j=0; j<k; j++) {
                        sq += R[j]*R[j];
                }
                double a = (double) 12*N/(k*(k+1));
                double b = sq;
                double c = (double) k*(k+1)*(k+1)/4;
                et = a*(b-c);
                et = (N-1)*et/(N*(k-1)-et);
                // Grava múltiplas
                for(int i=0; i<k; i++) {
                        for(int j=0; j<k; j++) {
                                double T = (double) (R[i]-R[j])/Math.sqrt((double)k*(k+1)/(6*N));
                                String str = "i: "+i+"; j: "+j+"; T = "+T+"\n";
```

```
                                        multiplas.write(str);
                                }
                        }
                        return et;
                }
}
```

classe Modelo

```
package mb.utils;

import mb.classifiers.EnsembleHeterogenio;
import mb.classifiers.SubkNN;
import mb.classifiers.SubFIMTDD;

public class Modelo {
        // Variáveis
        private String tipo;
        private SubkNN knn;
        private SubFIMTDD fimtdd;
        private EnsembleHeterogenio bagging;
        // Construtores
        public Modelo(String str, SubkNN knn) {
                if(str.equals("SubkNN")) { this.tipo="SubkNN"; this.knn=knn; }
                else this.tipo=null;
        }
        public Modelo(String str, SubFIMTDD fimtdd) {
                if(str.equals("SubFIMTDD")) {
                        this.tipo="SubFIMTDD"; this.fimtdd=fimtdd; this.fimtdd.prepareForUse();
                } else this.tipo=null;
        }
        public Modelo(String str, EnsembleHeterogenio bagging) {
                if(str.equals("EnsembleHeterogenio")) { this.tipo="EnsembleHeterogenio";
this.bagging=bagging; }
                else this.tipo=null;
        }
        // Métodos
        public String getTipo() { return tipo; }
        public SubkNN getKnn() { return knn; }
        public SubFIMTDD getFimtdd() { return fimtdd; }
        public EnsembleHeterogenio getEnsembleHeterogenio() { return bagging; }
}
```

classe Minkowski

```
package mb.utils;

import com.yahoo.labs.samoa.instances.Instance;

public class Minkowski {
	// Variáveis
	private int p;
	// Construtor
	public Minkowski(int p) { this.p=p; }
	// Métodos
	public double distancia(Instance X, Instance Y) {
		double valor = 0;
		for(int i=0; i<X.numAttributes()-1; i++) {
			double a = Math.pow(X.value(i)-Y.value(i),this.p);
			valor = valor + a;
		}
		valor = Math.pow(valor,1.0/this.p);
		return valor;
	}
}
```

classe Vizinho

```
package mb.utils;

import com.yahoo.labs.samoa.instances.Instance;

public class Vizinho {
	private int indice;
	private Instance instância;
	private double distancia;
	public Vizinho(int i, Instance inst, double d) { this.indice=i; this.instância=inst;
this.distancia=d; }
	public int getIndice() { return this.indice; }
	public Instance getInstance() { return this.instância; }
	public double getDistance() { return this.distancia; }
	public void imprimir() {
		System.out.println("i: "+indice+"\tatt1: "+instância.value(1)+"\ty:
"+instância.value(instância.numAttributes()-1));
	}
}
```

classe Poisson

```
package mb.utils;

public class Poisson {
	// Variáveis
	private int lambda;
	// Construtores
	public Poisson(int lambda) { this.lambda=lambda; }
	// Métodos
	public int gera() {
		double xis = Math.random();
		double f=0; double F=0; int k=0;
		while(xis>F && k<12) {
			f=Math.exp(-
this.lambda)*Math.pow(this.lambda,(double)k)/(double)fatorial(k);
			F=F+f;
			k++; }
		return k-1; }
	private int fatorial(int n) { int m=1; for(int i=n; i>1; i--) m=m*i; return m; }
}
```

classe PrevisaoDependente

```
package mb.utils;

import java.util.ArrayList;

import mb.utils.Poisson;

public class PrevisaoDependente {
	// Variáveis
	ArrayList<Integer> kapas = new ArrayList<Integer>();
	ArrayList<Double> dependentes = new ArrayList<Double>();
	Double dependenteV;
	Double dependenteB;
	ArrayList<Double> pesos = new ArrayList<Double>();
	Poisson poisson = new Poisson(1);
	double eta=0.0001;
	PrevisaoDependente anterior;
	// Construtores
	public PrevisaoDependente() { }
	public PrevisaoDependente(double yPrevisto[], PrevisaoDependente anterior, double eta) {
		this.eta = eta;
		for(int i=0; i<yPrevisto.length; i++) {
			kapas.add(this.poisson.gera());
			Double Previsao = new Double(yPrevisto[i]);
```

```
                        dependentes.add(Previsao);
                }
                this.anterior = anterior;
        }
        // Métodos
        public double getSomaPesos() {
                double somaPesos=0;
                for(int i=0; i<pesos.size(); i++) somaPesos=somaPesos+pesos.get(i);
                return somaPesos;
        }
        public double[] getPesos() {
                int numPesos = this.pesos.size();
                double w[] = new double[numPesos];
                for(int i=0; i<numPesos; i++) w[i] = this.pesos.get(i).doubleValue();
                return w;
        }
        public double mediaPonderada(double yVerdadeiro) {
                double valor = 0;
                int num = dependentes.size();
                double peso[] = new double[num];
                int somaKapas=0;
                for(int i=0; i<num; i++) somaKapas = somaKapas+kapas.get(i).intValue();
                if(somaKapas!=0) {
                        for(int i=0; i<num; i++) {
                                peso[i] = (double) kapas.get(i).intValue()/somaKapas;
                                valor = valor + peso[i]*dependentes.get(i).doubleValue();
                        }
                } else valor=yVerdadeiro;
                return valor;
        }
        public double mediaAgregada(double yVerdadeiro, double yB) {
                double valor = 0, pValor, yValor;
                int numModelos = this.dependentes.size()+1;
                Double inicial = new Double(1.0/numModelos);
                this.dependenteV = new Double(yVerdadeiro);
                this.dependenteB = new Double(yB);
                if(anterior==null) for(int j=0; j<numModelos; j++) pesos.add(inicial);
                else {
                        // Atualizar pesos
                        double denominador = 0.0;
                        double d;
                        for(int k=0; k<numModelos-1; k++) {
                                d = anterior.dependentes.get(k).doubleValue()-
anterior.dependenteV.doubleValue();
                                denominador = denominador +
anterior.pesos.get(k).doubleValue()*Math.exp(-eta*d*d);
                        }
                        d = anterior.dependenteB.doubleValue()-anterior.dependenteV.doubleValue();
                        denominador = denominador + anterior.pesos.get(numModelos-
1).doubleValue()*Math.exp(-eta*d*d);
```

```
                        double somaPesos=anterior.getSomaPesos();
                        for(int j=0; j<numModelos-1; j++) {
                                d = anterior.dependentes.get(j).doubleValue()-
anterior.dependenteV.doubleValue();
                                Double novoPeso = anterior.pesos.get(j).doubleValue()*Math.exp(-
eta*d*d)*somaPesos/denominador;
                                this.pesos.add(novoPeso);
                        }
                        d = anterior.dependenteB.doubleValue()-anterior.dependenteV.doubleValue();
                        Double novoPeso = anterior.pesos.get(numModelos-
1).doubleValue()*Math.exp(-eta*d*d)*somaPesos/denominador;
                        this.pesos.add(novoPeso);

                }
                // Calcular a média
                double somaPesos = getSomaPesos();
                for(int i=0; i<pesos.size()-1; i++) {
                        pValor = pesos.get(i).doubleValue();
                        yValor = dependentes.get(i).doubleValue();
                        valor = valor + pValor*yValor;
                }
                pValor = pesos.get(pesos.size()-1).doubleValue();
                valor = valor + pValor*yB;
                valor = valor/somaPesos;
                return valor;
        }
}
```

classe FundoGrafico

```
package mb.utils;

import java.util.ArrayList;
import java.awt.Color;
import java.awt.Graphics;

import javax.swing.JPanel;

public class FundoGrafico extends JPanel {
	private static final long serialVersionUID = 1L;
	private static long numPainel=0L;
	ArrayList<ArrayList<Double>> valoresY = new ArrayList<ArrayList<Double>>();
	ArrayList<ArrayList<Double>> valoresErro = new ArrayList<ArrayList<Double>>();
	ArrayList<Modelo> modelos = new ArrayList<Modelo>();
	int tipoErro;		// 1:MSE; 2:MAD; 3:Pesos
	public FundoGrafico(ArrayList<ArrayList<Double>> valoresY,
ArrayList<ArrayList<Double>> valoresErro,
		ArrayList<Modelo> modelos, int tipoErro) {
		this.valoresY = valoresY; this.valoresErro = valoresErro; this.modelos = modelos;
numPainel++; this.tipoErro=tipoErro; }
	void escolheCor(int j, Graphics g) {
		switch(j) {
			case 0: g.setColor(Color.blue); break;
			case 1: g.setColor(Color.green); break;
			case 2: g.setColor(Color.red); break;
			case 3: g.setColor(Color.pink); break;
			case 4: g.setColor(Color.cyan); break;
			case 5: g.setColor(Color.yellow); break;
			case 6: g.setColor(Color.gray); break;
		}
	}
	@Override public void paintComponent(Graphics g) {
		int xa=0, ya=0, xb=0, yb=0; int ux=1; int uy=-3;
		int x0Um=50; int y0Um=275;
		g.setColor(Color.black);
		g.drawString("Duração média da chamada", 450, 50);
		g.drawLine(x0Um+ux*(-1),y0Um,x0Um+ux*1000,y0Um);
		g.drawLine(x0Um,y0Um+uy*(-1),x0Um,y0Um+uy*120);
		long v, pos;
		if(numPainel<=1000) { v=500; pos=v; }
		else { v = (numPainel+500)/1000*1000-500; pos = (v-(numPainel%1000))%1000; }
		String xis = ""+v;
		g.drawString(xis, (int)(x0Um+pos*ux), y0Um+20);
		g.drawString("Número de instâncias testadas: "+(numPainel-1), 800,50);
		for(int k=20; k<=100; k=k+20) {
			g.drawLine(x0Um+ux*(-5), y0Um+k*uy, x0Um+ux*5, y0Um+k*uy);
			String marca = ""+k;
```

```
                        g.drawString(marca, x0Um+ux*(-30), y0Um+k*uy);
                }
                int esq=-50;
                g.drawString("Real: ",esq+100,600);
                g.setColor(Color.blue);
                g.drawLine(esq+100, 580, esq+150, 580);
                g.setColor(Color.black);
                for(int j=0; j<modelos.size(); j++) {
                        g.drawString(modelos.get(j).getTipo()+": ",esq+250+j*150,600);
                        escolheCor(j+1,g);
                        g.drawLine(esq+250+j*150, 580, esq+300+j*150, 580);
                        g.setColor(Color.black);
                }
                g.drawString("Bagging: ",esq+250+150*modelos.size(),600);
                escolheCor(1+modelos.size(),g);
                g.drawLine(esq+250+modelos.size()*150, 580, esq+250+50+modelos.size()*150,
580);
                g.setColor(Color.black);
                g.drawString("Agregada: ",esq+250+150+150*modelos.size(),600);
                escolheCor(2+modelos.size(),g);
                g.drawLine(esq+250+150+modelos.size()*150, 580,
esq+250+150+50+modelos.size()*150, 580);
                g.setColor(Color.black);
                switch(tipoErro) {
                case 1: g.drawString("Erro quadrático médio (MSE)", 450, 350); break;
                case 2: g.drawString("Erro absoluto médio (MAD)", 450, 350); break;
                case 3: g.drawString("Pesos (Agregada)", 450, 350); break;
                }
                int x0Dois=50; int y0Dois=550;
                double uy2=0.0;
                int kIni=0, kFim=0, kPasso=0;
                switch(tipoErro) {
                case 1: kIni=5000; kFim=10000; kPasso=5000; uy2=-0.02; break;
                case 2: kIni=20; kFim=100; kPasso=20; uy2=uy; break;
                case 3: kIni=1; kFim=1; kPasso=1; uy2=-200.0; break;
                }
                g.drawString(xis, (int)(x0Dois+pos*ux), y0Dois+20);
                g.drawLine(x0Dois+ux*(-1),y0Dois,x0Dois+ux*1000,y0Dois);
                if(tipoErro!=3) g.drawLine(x0Dois,y0Dois+(int)(uy2*(-2)),x0Dois,
y0Dois+(int)(uy2*(kFim)));
                else g.drawLine(x0Dois,y0Dois+(int)(uy2*(-0.1)),x0Dois,
y0Dois+(int)(uy2*(kFim*1.1)));
                for(int k=kIni; k<=kFim; k=k+kPasso) {
                        g.drawLine(x0Dois+ux*(-5), y0Dois+(int)(k*uy2), x0Dois+ux*5,
y0Dois+(int)(k*uy2));
                        String marca = ""+k;
                        g.drawString(marca, x0Dois+ux*(-40), (int)(y0Dois+k*uy2));
                }
                for(int i=0; i<valoresY.size()-1; i++) {
                        for(int j=0; j<valoresY.get(i).size(); j++) {
```

```
                                        //System.out.println("Erro: "+valoresErro.get(i).size());
                                        escolheCor(j,g);
                                        xa=i*ux+x0Um;
ya=((int)valoresY.get(i).get(j).doubleValue())*uy+y0Um;
                                        xb=(i+1)*ux+x0Um;
yb=((int)valoresY.get(i+1).get(j).doubleValue())*uy+y0Um;
                                        g.drawLine(xa,ya,xb,yb);
                                        if(j!=0 && j<valoresErro.get(i).size()+1) {
                                                xa=i*ux+x0Dois; ya=(int)(valoresErro.get(i).get(j-
1).doubleValue()*uy2)+y0Dois;
                                                xb=(i+1)*ux+x0Dois; yb=(int)(valoresErro.get(i+1).get(j-
1).doubleValue()*uy2)+y0Dois;
                                                escolheCor(j,g);
                                                g.drawLine(xa, ya, xb, yb);

                                        }
                                }

                        }


                }
}
```

classe GraficoDuplo

```
package mb.utils;
import javax.swing.JFrame;

public class GraficoDuplo extends JFrame {
        private static final long serialVersionUID = 1L;
        public GraficoDuplo() {
        }
        public void inicializacao() { setTitle("Grafico"); setSize(1100,700);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                setVisible(true);

        }
}
```

Anexo B

Tabela Resultados

Neste anexo apresentam-se algumas tabelas de resultados.
Designadamente, tabelas com os resultados para o primeiro dia e tabelas com os resultados para o primeiro dia usando o ensemble homogéneo.

B.1 Tabela *Ensemble Heterogéneo* primeiro dia

yTrue	ySubFIMTDD	ySubkNN	ySubkNN	ySubkNN	yHeterog	yAgregada	mediaTrue	mediaSubFIMTDD	mediaSubkNN	mediaSubkNN
55	64,3986	83,8	54,95	100,3	86,51973	64,39863	67,81657	67,83396	67,74986	67,78625
69	66,6019	42,5	87,4	100,3	70,97549	66,60195	67,81657	67,83396	67,74986	67,78625
41	66,7837	43,6	89,6	100,3	49,39593	66,78374	67,81656	67,83396	67,74985	67,78625
86	66,2753	74,4	79,5	100,3	80,11883	66,27532	67,81657	67,83396	67,74986	67,78625
53	66,1357	38,1	39,85	100,3	68,47025	66,13574	67,81656	67,83396	67,74985	67,78625
96	65,7651	34,7	42,15	100,3	63,64924	65,76514	67,81657	67,83396	67,74985	67,78625
51	50,2679	94,1	77,25	100,3	89,33125	50,26790	67,81657	67,83396	67,74985	67,78625
51	50,5092	98,2	78,9	100,3	89,60000	50,50918	67,81656	67,83395	67,74985	67,78625
4	65,9330	51,7	47,15	98	57,67899	65,93298	67,81656	67,83395	67,74985	67,78625
130	65,9594	54,5	49,9	98	73,46562	65,95937	67,81656	67,83395	67,74985	67,78625
81	65,9216	56,5	62,9	98	70,33333	65,92159	67,81657	67,83395	67,74985	67,78624
4	65,9365	48	45,4	98	53,97884	65,93652	67,81656	67,83395	67,74985	67,78624
19	95,8027	53	43,3	98	74,40137	95,80275	67,81656	67,83396	67,74985	67,78624
221	91,6823	61,6	58,4	98	83,76078	91,68233	67,81657	67,83396	67,74985	67,78624
42	62,9123	82,3	67,25	98	82,30000	62,91231	67,81657	67,83396	67,74985	67,78624

70	63,0241	84,5	64,45	98	70,18272	63,02408	67,81657	67,83396	67,74985	67,78624
314	71,3328	161,9	99,2	98	149,12000	71,33279	67,81659	67,83396	67,74986	67,78624
33	65,7402	86,5	71,6	98	73,86005	65,74020	67,81658	67,83396	67,74986	67,78624
286	64,2572	112,1	88,85	98	78,07860225	64,2572045	67,81660222	67,83395669	67,74986345	67,78624433
56	96,6313	74,5	72,6	98	88,32626726	96,63133631	67,81660125	67,83395907	67,749864	67,78624473
35	64,8632	61,6	77,55	98	74,6875	64,863226	67,81659853	67,83395882	67,7498635	67,78624554
13	64,8562	60,5	70,95	98	74,33826981	64,85615846	67,816594	67,83395858	67,7498629	67,7862458
9	64,8700	58	57,45	98	68,01000429	64,87000858	67,81658913	67,83395833	67,74986209	67,78624494
20	62,9408	46,4	48,9	98	73,03333333	62,94075025	67,81658518	67,83395793	67,74986032	67,78624338
36	66,0046	40,5	78,95	98	70,86364964	66,00459856	67,81658255	67,83395778	67,74985807	67,78624431
46	66,0249	183,9	119,6	98	108,8874319	66,0248638	67,81658074	67,83395763	67,74986768	67,78624859
35	66,1011	93,1	71,35	98	75,47528375	66,101135	67,81657803	67,83395748	67,74986977	67,78624889

mediaSubkNN	mediaHeterog	mediaAgregada	MSESubFIMTDD	MSESubkNN	MSESubkNN	MSESubkNN	MSEHeterog	MSEAgregado
68,47796	67,95994	67,84746	15463,59887	16955,84269	16427,30583	18920,54158	16139,62244	15460,55372
68,47796	67,95994	67,84746	15463,59759	16955,84135	16427,3045	18920,5401	16139,62111	15460,55245
68,47797	67,95994	67,84746	15463,59637	16955,83995	16427,30334	18920,53882	16139,61978	15460,55122
68,47797	67,95994	67,84746	15463,59512	16955,83856	16427,30198	18920,53728	16139,61845	15460,54998
68,47797	67,95994	67,84746	15463,59385	16955,83717	16427,30064	18920,5359	16139,61713	15460,54871
68,47797	67,95994	67,84746	15463,59265	16955,83608	16427,29952	18920,53433	16139,61588	15460,54751
68,47798	67,95994	67,84745	15463,59137	16955,83483	16427,29822	18920,53297	16139,61467	15460,54623
68,47798	67,95995	67,84745	15463,59009	16955,83361	16427,29692	18920,5316	16139,61346	15460,54495
68,47798	67,95994	67,84745	15463,58913	16955,8324	16427,29572	18920,53077	16139,61236	15460,54399
68,47799	67,95995	67,84745	15463,58819	16955,83147	16427,29489	18920,52929	16139,61129	15460,54305
68,47799	67,95995	67,84745	15463,58693	16955,83011	16427,29356	18920,52775	16139,60997	15460,54179
68,47799	67,95994	67,84745	15463,58597	16955,82887	16427,29234	18920,52691	16139,60884	15460,54083
68,47799	67,95994	67,84745	15463,58518	16955,82756	16427,29103	18920,52587	16139,60776	15460,54004

68,47799	67,95995	67,84746	15463,58528	16955,82826	16427,29186	18920,52555	16139,60798	15460,54014
68,47800	67,95995	67,84746	15463,58404	16955,827	16427,29055	18920,52425	16139,60678	15460,5389
68,47800	67,95995	67,84746	15463,58276	16955,82561	16427,2892	18920,52275	16139,60544	15460,53762
68,47800	67,95995	67,84746	15463,58636	16955,82612	16427,29166	18920,52504	16139,60636	15460,54122
68,47800	67,95995	67,84746	15463,58517	16955,82496	16427,29042	18920,52383	16139,60516	15460,54003
68,47801	67,95996	67,84746	15463,58795	16955,82605	16427,29228	18920,52518	16139,6074	15460,54281
68,47801	67,95996	67,84746	15463,58681	16955,82468	16427,29094	18920,52376	16139,60615	15460,54167
68,47801	67,95996	67,84746	15463,58561	16955,82334	16427,28973	18920,52253	16139,60495	15460,54047
68,47801	67,95996	67,84746	15463,58455	16955,82212	16427,28865	18920,52156	16139,60392	15460,53941
68,47802	67,95996	67,84746	15463,58353	16955,82092	16427,28749	18920,52065	16139,60288	15460,53839
68,47802	67,95996	67,84746	15463,5824	16955,81957	16427,2862	18920,51959	16139,60177	15460,53726
68,47802	67,95996	67,84746	15463,5812	16955,81817	16427,28499	18920,51834	16139,60054	15460,53606
68,47802	67,95996	67,84746	15463,57995	16955,81834	16427,28408	18920,517	16139,59953	15460,53481
68,47803	67,95996	67,84746	15463,57875	16955,81722	16427,28283	18920,51576	16139,59833	15460,53361

MADSubFIMTDD	MADSubkNN	MADSubkNN	MADSubkNN	MADHeterog	MADAgregado
50,36273374	54,29774268	53,28188596	57,50493028	52,27052651	50,41658183
50,36272978	54,29774038	53,28188307	57,50492811	52,27052235	50,41657786
50,36272774	54,2977361	53,28188268	57,50492826	52,27051872	50,41657582
50,36272521	54,29773257	53,28187881	57,50492469	52,27051488	50,41657329
50,36272213	54,29772931	53,2818755	57,50492384	52,27051184	50,4165702
50,36272047	54,29772989	53,28187554	57,50491944	52,27051019	50,41656853
50,36271636	54,29772897	53,28187331	57,50491876	52,27050904	50,41656442
50,36271223	54,29772838	53,28187121	57,50491809	52,2705079	50,41656029
50,36271319	54,29772783	53,28187037	57,5049211	52,27050802	50,41656125
50,36271432	54,29772959	53,28187259	57,504919	52,27050837	50,41656237
50,3627114	54,29772712	53,28186968	57,50491564	52,27050493	50,41655945
50,36271236	54,29772627	53,28186869	57,50491866	52,27050474	50,4165604
50,36271455	54,29772459	53,2818663	57,50492044	52,270505	50,41656259
50,36272108	54,29773329	53,28187534	57,50492586	52,27051203	50,41656911
50,36271864	54,29773213	53,28187302	57,50492573	52,27051104	50,41656667
50,36271505	54,29772884	53,28186907	57,50492329	52,27050673	50,41656308
50,36273096	54,29773693	53,28188243	57,5049364	52,27051605	50,41657898
50,3627295	54,29773686	53,28188122	57,50493702	52,2705151	50,41657752
50,36274368	54,29774675	53,28189312	57,50494782	52,27052798	50,41659169
50,36274287	54,29774379	53,28189008	57,50494654	52,27052633	50,41659088
50,36274118	54,2977415	53,2818892	57,50494699	52,27052529	50,41658918
50,3627413	54,29774094	53,28188958	57,50494926	52,27052604	50,4165893
50,36274176	54,2977405	53,28188918	57,50495187	52,27052659	50,41658975
50,36274114	54,29773819	53,28188717	57,50495356	52,27052666	50,41658913
50,36273946	54,29773407	53,28188631	57,50495394	52,27052522	50,41658744
50,36273695	54,29774099	53,28188799	57,50495348	52,2705261	50,41658493
50,36273536	54,2977413	53,28188659	57,50495394	52,27052512	50,41658333

PesoSubFIMTDD	PesoSubkNN	PesoSubkNN	PesoSubkNN	PesoHeterog
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0

B.2 Tabela *Ensemble Heterogéneo* segundo dia

yTrue	ySubFIMTDD	ySubkNN	ySubkNN	ySubkNN	yHeterog	yAgregada	mediaTrue	mediaSubFIMTDD	mediaSubkNN	mediaSubkNN	mediaSubkNN
36	73,5769	87,2	74,75	39,8	78,1179	73,5769	73,6309	73,6484	73,4455	73,6078	74,8390
33	77,8129	30,5	48,6	38	60,1898	77,8129	73,6304	73,6484	73,4450	73,6073	74,8369
161	76,9278	110,6	84,45	99	81,9426	76,9278	73,6455	73,6766	73,4596	73,6194	74,7724
66	79,8746	101,8	87,75	100,9	87,2641	79,8746	73,6413	73,6797	73,4547	73,6141	74,8221
18	78,6819	67,4	60,1	35,4	72,4988	78,6819	73,6402	73,6804	73,4505	73,6106	74,8388
5	83,1748	124,4	99,6	37,4	67,2400	83,1748	73,6420	73,6820	73,4505	73,6111	74,8802
31	71,1123	55	78,35	57,5	58,9354	71,1123	73,6342	73,6854	73,4417	73,6010	74,8437
24	64,5370	83,7	88,15	135,5	110,9474	64,5370	73,6463	73,6900	73,4568	73,6155	74,8882
43	74,7245	37,3	72,65	26,1	74,3096	74,7245	73,6450	73,6911	73,4589	73,6167	74,8727
19	76,0128	45,6	53,55	38,8	68,5252	76,0128	73,6454	73,6938	73,4580	73,6164	74,8602
84	79,9888	162,8	113,3	147,4	142,9654	79,9888	73,6514	73,6946	73,4634	73,6226	74,8476
18	75,7188	201,4	125,25	95,6	107,4600	75,7188	73,6567	73,6959	73,4673	73,6283	74,8592
18	69,9784	60,2	63,95	53,7	61,9571	69,9784	73,6722	73,7016	73,4844	73,6431	74,8509
143	77,6863	56,3	45,95	47	71,5490	77,6863	73,6925	73,7083	73,5040	73,6650	74,8578
142	62,6466	202,2	127,3	64,7	122,2933	62,6466	73,6958	73,7226	73,5010	73,6627	75,0321
46	65,4268	55,7	48,1	41,6	47,2000	65,4268	73,6988	73,7231	73,5031	73,6647	75,0386
321	70,9139	43	43,65	85,2	61,6093	70,9139	73,7146	73,7278	73,5194	73,6830	75,0632
15	72,8081	50,3	43,55	81,8	81,8000	72,8081	73,7186	73,7347	73,5285	73,6897	75,0493
19	67,1434	29,3	33,85	53,6	33,8500	67,1434	73,7264	73,7395	73,5378	73,6977	75,0831
71	66,9715	59,9	66,8	55	62,2108	66,9715	73,7290	73,7450	73,5467	73,7057	75,1095
35	69,4878	69,6	61,25	26,4	63,3805	69,4878	73,7311	73,7460	73,5459	73,7058	75,1236
42	70,9399	79,7	83,15	59,2	71,0339	70,9399	73,7358	73,7465	73,5495	73,7094	75,1123
69	78,3515	47,8	43,85	43,3	43,3000	78,3515	73,7343	73,7477	73,5511	73,7122	75,1035
136	79,4710	69,4	67,7	74,2	71,3750	79,4710	73,7317	73,7502	73,5485	73,7095	75,1239
74	69,7294	76,3	94,95	30,1	71,9196	69,7294	73,7416	73,7589	73,5583	73,7193	75,1698

2

4

mediaHeterog	mediaAgregada	MSESubFIMTDD	MSESubkNN	MSESubkNN	MSESubkNN	MSEEHeterog	MSEAgregado
73,9106	73,7878	22430,7277	24159,8795	23410,0202	27096,4559	22980,9310	22404,0320
73,9103	73,7886	22430,3108	24159,6369	23409,9010	27095,9637	22980,6433	22403,6179
73,9109	73,7968	22427,0502	24156,4095	23406,5246	27085,6130	22976,4495	22400,4168
73,9102	73,7968	22426,5195	24155,8597	23406,0744	27084,9905	22975,9831	22399,8876
73,9074	73,8236	22456,6670	24198,9549	23444,4611	27090,6585	23010,2278	22430,2883
73,9182	73,8265	22448,3646	24190,2551	23436,0980	27086,1761	23002,7243	22422,0273
73,9212	73,8271	22456,7405	24195,7927	23443,5045	27094,8889	23010,9517	22430,4197
73,9326	73,8285	22451,0857	24191,1352	23438,3773	27091,7874	23006,0735	22424,7964
73,9194	73,8315	22442,5899	24182,4623	23429,9370	27073,6089	22997,0016	22416,3715
73,9384	73,8358	22449,7332	24193,4644	23439,9393	27090,6947	23006,7609	22423,5810
73,9355	73,8367	22450,1621	24194,2756	23440,2400	27087,8732	23006,8378	22424,0324
73,9332	73,8392	22446,1666	24190,8157	23437,6293	27081,3920	23002,6955	22420,0683
73,9332	73,8399	22443,6123	24188,7764	23435,4762	27075,4269	23000,2561	22417,5390
73,9388	73,8410	22447,8114	24193,2319	23440,2522	27076,4547	23004,2486	22421,7660
73,9456	73,8465	22461,7577	24210,2356	23456,2511	27085,0345	23018,1470	22435,7666
73,9597	73,8526	22472,8287	24224,8280	23470,9423	27092,6216	23031,1708	22446,9282
74,0055	73,8660	22443,2778	24196,4247	23443,2187	27100,9490	23007,9018	22417,5612
74,0079	73,8664	22445,6538	24198,4388	23445,3738	27100,9722	23009,9659	22419,9537
74,0241	73,8707	22458,4841	24215,8472	23462,0377	27109,3680	23024,5846	22432,8454
74,0259	73,8772	22456,1214	24218,2571	23464,0596	27104,5597	23025,0191	22430,5643
74,0392	73,8816	22474,2032	24243,0529	23486,3367	27139,7835	23047,8076	22448,7062
74,0508	73,8868	22482,4239	24255,1274	23496,9568	27155,6967	23058,7291	22456,9923
74,0541	73,8876	22488,5393	24262,2308	23504,0515	27163,5511	23065,6634	22463,1188
74,0529	73,8881	22502,1695	24276,4292	23517,9541	27176,5909	23079,3176	22476,7562
74,0521	73,8893	22497,5714	24272,8384	23513,9397	27169,5487	23074,6411	22472,1717
74,0560	73,8916	22496,7882	24272,4298	23513,0034	27165,8189	23073,8559	22471,4126
74,0752	73,9000	22495,6141	24273,8090	23514,1846	27163,2214	23073,5552	22470,2985

2

MADSubFIMTDD	MADSubkNN	MADSubkNN	MADSubkNN	MADHeterog	MADAgregado
59,1487	63,8362	62,7981	67,88942045	61,52746322	60,62414502
59,1487	63,8359	62,7980	67,88810459	61,52699874	60,62398838
59,1558	63,8427	62,8032	67,87835783	61,52844813	60,62780249
59,1551	63,8418	62,8024	67,87683772	61,52747094	60,62709616
59,1836	63,8681	62,8253	67,83692539	61,53740345	60,64151092
59,1810	63,8616	62,8187	67,86508033	61,53961765	60,63663168
59,1823	63,8590	62,8170	67,87632821	61,54178793	60,63703179
59,1835	63,8591	62,8168	67,9068263	61,54927418	60,63652133
59,1779	63,8476	62,8038	67,87150995	61,53346826	60,62704096
59,1893	63,8668	62,8211	67,91160119	61,55357429	60,63478855
59,1887	63,8659	62,8198	67,89917674	61,55019724	60,63290091
59,1905	63,8652	62,8203	67,89186764	61,54863773	60,63295934
59,1928	63,8699	62,8247	67,88379139	61,54969268	60,63383249
59,1968	63,8767	62,8330	67,88939207	61,55583801	60,63638322
59,2111	63,8989	62,8538	67,89050139	61,57045231	60,64764121
59,2251	63,9233	62,8784	67,90116987	61,58882659	60,65660384
59,2269	63,9145	62,8706	68,03078958	61,61439586	60,64822259
59,2285	63,9164	62,8729	68,03278915	61,61628551	60,64898608
59,2387	63,9352	62,8940	68,05282539	61,6345882	60,65574461
59,2436	63,9456	62,9021	68,05274193	61,63913181	60,65615436
59,2520	63,9613	62,9156	68,09523601	61,65656408	60,66121835
59,2590	63,9729	62,9255	68,12931985	61,67107665	60,66459948
59,2622	63,9756	62,9290	68,14449876	61,67639964	60,66715272
59,2668	63,9823	62,9356	68,14470833	61,68037408	60,67144427
59,2666	63,9828	62,9364	68,13581019	61,6784935	60,67041866
59,2657	63,9791	62,9321	68,14075847	61,67742942	60,66820341
59,2773	63,9931	62,9458	68,17256366	61,69487053	60,67652453

2

PesoSubFIMTDD	PesoSubkNN	PesoSubkNN	PesoSubkNN	PesoHeterog
1	0	2,45E-195	0	4,58E-110
1	0	2,04E-195	0	4,20E-110
1	0	7,90E-196	0	3,67E-110
1	0	7,43E-196	0	3,51E-110
1	0	2,19E-199	0	4,69E-112
1	0	1,10E-199	0	2,16E-112
1	0	1,29E-199	0	1,97E-112
1	0	5,84E-200	0	1,01E-112
1	0	1,65E-200	0	6,59E-113
1	0	1,35E-201	0	1,01E-113
1	0	9,64E-202	0	9,50E-114
1	0	2,89E-202	0	7,44E-114
1	0	1,53E-202	0	5,49E-114
1	0	7,11E-203	0	4,58E-114
1	0	1,02E-203	0	2,71E-114
1	0	3,63E-205	0	4,33E-115
1	0	5,25E-207	0	3,33E-117
1	0	4,30E-207	0	3,25E-117
1	0	2,22E-208	0	7,30E-118
1	0	5,93E-210	0	8,10E-119
1	0	2,55E-211	0	4,47E-120
1	0	2,31E-212	0	6,03E-121
1	0	1,17E-212	0	3,61E-121
1	0	8,93E-213	0	3,30E-121
1	0	5,19E-213	0	2,95E-121
1	0	3,51E-213	0	2,27E-121
1	0	3,54E-214	0	7,71E-122

B.3 Tabela *Ensemble homogéneo* primeiro dia

2

yTrue	ySubkNN	ySubkNN	ySubkNN	ySubkNN	ySubkNN	yEHomog	yAgregada	mediaTrue	mediaSubkNN	mediaSubkNN	mediaSubkNN
47	44	62,65	100,3	60,15	51,49	51,49	51,49	67,81655492	67,74985899	67,78624964	68,47795382
270	58,9	45,35	100,3	47,5	59,87	72,825	59,87	67,81657165	67,74985826	67,78624779	68,47795645
20	55,7	65,95	100,3	45,1	58,04	85,43333333	58,04	67,81656769	67,74985726	67,78624763	68,47795909
55	83,8	54,95	100,3	41,8	61,54	71,778	61,54	67,81656663	67,74985859	67,78624657	68,47796172
69	42,5	87,4	100,3	50,75	57	62,6375	57	67,81656673	67,7498565	67,7862482	68,47796435
41	43,6	89,6	100,3	52,15	57,16	64,1875	57,16	67,81656451	67,7498545	67,78625	68,47796698
86	74,4	79,5	100,3	49,35	79,7	86,88	79,7	67,81656602	67,74985505	67,78625097	68,47796962
53	38,1	39,85	100,3	66,8	57,64	55,4925	57,64	67,81656479	67,7498526	67,78624866	68,47797225
96	34,7	42,15	100,3	66,8	57,56	57,27666667	57,56	67,81656712	67,74984987	67,78624654	68,47797488
51	94,1	77,25	100,3	30,95	52,37	66,71	52,37	67,81656573	67,74985205	67,78624732	68,47797751
51	98,2	78,9	100,3	30,65	52,5	82,81666667	52,5	67,81656434	67,74985457	67,78624824	68,47798014
4	51,7	47,15	98	37,95	63	98	63	67,81655906	67,74985324	67,78624653	68,47798259
130	54,5	49,9	98	39,25	44,13	98	44,13	67,8165642	67,74985214	67,78624505	68,47798503
81	56,5	62,9	98	41,95	63,63	70,075	63,63	67,81656529	67,74985121	67,78624465	68,47798747
4	48	45,4	98	40,35	62,32	62,32	62,32	67,81656002	67,74984958	67,7862428	68,47798991
19	53	43,3	98	54,15	60,65	52,7	60,65	67,81655598	67,74984836	67,78624077	68,47799235
221	61,6	58,4	98	47,45	72,13	55,53571429	72,13	67,81656865	67,74984785	67,78624	68,4779948
42	82,3	67,25	98	57,8	74,15	68,0125	74,15	67,81656651	67,74984905	67,78623995	68,47799724
70	84,5	64,45	98	58,25	73,96	74,475	73,96	67,81656669	67,74985044	67,78623968	68,47799968
314	161,9	99,2	98	54,7	72	122,3142857	72	67,81658706	67,74985823	67,78624227	68,47800212
33	86,5	71,6	98	55,8	61	74,5375	61	67,81658418	67,74985978	67,78624259	68,47800456
286	112,1	88,85	98	57,65	64,12	76,87833333	64,12	67,81660222	67,74986345	67,78624433	68,478007
56,00	74,50	72,60	98,00	147,75	81,28	108,98	81,28	67,81660	67,74986	67,78624	68,47801
35,00	61,60	77,55	98,00	64,60	73,48	70,40	73,48	67,81660	67,74986	67,78625	68,47801
13,00	60,50	70,95	98,00	66,30	73,32	60,50	73,32	67,81659	67,74986	67,78625	68,47801

9,00	58,00	57,45	98,00	65,30	73,40	65,30	73,40	67,81659	67,74986	67,78624	68,47802
20,00	46,40	48,90	98,00	80,40	46,60	72,20	46,60	67,81659	67,74986	67,78624	68,47802
36,00	40,50	78,95	98,00	78,35	62,11	59,73	62,11	67,81658	67,74986	67,78624	68,47802
46,00	183,90	119,60	98,00	122,60	97,56	107,30	97,56	67,81658	67,74987	67,78625	68,47802
35,00	93,10	71,35	98,00	45,75	71,65	78,50	71,65	67,81658	67,74987	67,78625	68,47803

mediaSubkNN	mediaSubkNN	mediaEHomog	mediaAgregada	MSESubkNN	MSESubkNN	MSESubkNN	MSESubkNN	MSESubkNN	MSEEHomog
67,84410733	67,83569189	67,94602396	67,83710984	16955,84304	16427,30556	18920,54319	16781,16548	15930,60832	16160,39982
67,84410564	67,83569123	67,94602437	67,83710918	16955,84532	16427,30838	18920,54401	16781,16818	15930,61066	16160,4017
67,84410376	67,83569042	67,94602581	67,83710837	16955,84403	16427,30719	18920,54298	16781,16685	15930,60946	16160,40072
67,84410161	67,8356899	67,94602613	67,83710785	16955,84269	16427,30583	18920,54158	16781,16547	15930,60815	16160,39941
67,84410019	67,835689	67,94602569	67,83710695	16955,84135	16427,3045	18920,5401	16781,16411	15930,60684	16160,39807
67,8440989	67,83568812	67,94602538	67,83710607	16955,83995	16427,30334	18920,53882	16781,16274	15930,60554	16160,39678
67,84409737	67,8356891	67,94602694	67,83710705	16955,83856	16427,30198	18920,53728	16781,16146	15930,60423	16160,39544
67,84409728	67,83568825	67,94602591	67,83710621	16955,83717	16427,30064	18920,5359	16781,16009	15930,60291	16160,39411
67,84409719	67,8356874	67,94602503	67,83710536	16955,83608	16427,29952	18920,53433	16781,15877	15930,60172	16160,39289
67,84409414	67,83568613	67,94602493	67,83710408	16955,83483	16427,29822	18920,53297	16781,15741	15930,6004	16160,39158
67,84409107	67,83568486	67,94602616	67,83710281	16955,83361	16427,29692	18920,5316	16781,15606	15930,59908	16160,39032
67,84408859	67,83568446	67,94602865	67,83710241	16955,8324	16427,29572	18920,53077	16781,15477	15930,59805	16160,38972
67,84408623	67,8356825	67,94603113	67,83710045	16955,83147	16427,29489	18920,52929	16781,15406	15930,59735	16160,38847
67,84408409	67,83568215	67,94603131	67,8371001	16955,83011	16427,29356	18920,52775	16781,1528	15930,59605	16160,38714
67,84408181	67,83568169	67,94603084	67,83709964	16955,82887	16427,29234	18920,52691	16781,15152	15930,59502	16160,38608
67,84408068	67,8356811	67,94602958	67,83709905	16955,82756	16427,29103	18920,52587	16781,15023	15930,59384	16160,38484
67,84407899	67,83568145	67,94602856	67,8370994	16955,82826	16427,29186	18920,52555	16781,15134	15930,59436	16160,38577
67,84407816	67,83568198	67,94602856	67,83709993	16955,827	16427,29055	18920,52425	16781,14997	15930,59313	16160,38449
67,84407737	67,83568248	67,9460291	67,83710043	16955,82561	16427,2892	18920,52275	16781,14859	15930,59181	16160,38315
67,84407628	67,83568283	67,9460336	67,83710078	16955,82612	16427,29166	18920,52504	16781,15277	15930,59534	16160,38486
67,84407528	67,83568226	67,94603414	67,83710021	16955,82496	16427,29042	18920,52383	16781,15142	15930,59408	16160,38366
67,84407444	67,83568195	67,94603488	67,8370999	16955,82605	16427,29228	18920,52518	16781,15435	15930,59684	16160,38594
67,84408105	67,83568307	67,94603828	67,83710	16955,82468	16427,29094	18920,52376	16781,15366	15930,59557	16160,38484
67,84408078	67,83568353	67,94603848	67,83710	16955,82334	16427,28973	18920,52253	16781,15234	15930,59438	16160,38361
67,84408065	67,83568399	67,94603786	67,83710	16955,82212	16427,28865	18920,52156	16781,15119	15930,59336	16160,38246
67,84408044	67,83568445	67,94603764	67,83710	16955,82092	16427,28749	18920,52065	16781,15006	15930,59239	16160,38138
67,84408148	67,83568269	67,94603800	67,83710	16955,81957	16427,28620	18920,51959	16781,14897	15930,59113	16160,38027
67,84408235	67,83568222	67,94603732	67,83710	16955,81817	16427,28499	18920,51834	16781,14773	15930,58987	16160,37898
67,84408688	67,83568467	67,94604057	67,83710	16955,81834	16427,28408	18920,51700	16781,14683	15930,58877	16160,37795
67,84408505	67,83568499	67,94604144	67,83710	16955,81722	16427,28283	18920,51576	16781,14545	15930,58756	16160,37677

MSEAgregado	MADSubkNN	MADSubkNN	MADSubkNN	MADSubkNN	MADSubkNN	MADEHomog	MADAgregado
15928,86621	54,2977376	53,2818799	57,50492047	54,03824252	51,53712137	52,3785847	51,53415698
15928,86489	54,29773336	53,28187679	57,50492013	54,03823914	51,53711748	52,37858074	51,53415309
15928,86723	54,29774633	53,28189097	57,50492941	54,03825307	51,5371306	52,37859272	51,5341662
15928,86603	54,29774479	53,28189036	57,50493129	54,03825068	51,53712948	52,3785938	51,53416509
15928,86472	54,29774268	53,28188596	57,50493028	54,0382473	51,53712576	52,37859086	51,53416137
15928,86341	54,29774038	53,28188307	57,50492811	54,03824434	51,53712249	52,37858705	51,5341581
15928,86211	54,2977361	53,28188268	57,50492826	54,0382408	51,53711956	52,37858463	51,53415517
15928,8608	54,29773257	53,28187881	57,50492469	54,03823936	51,53711582	52,37858038	51,53415143
15928,85948	54,29772931	53,2818755	57,50492384	54,03823603	51,53711194	52,37857625	51,53414755
15928,85829	54,29772989	53,28187554	57,50491944	54,03823397	51,53711086	52,37857512	51,53414647
15928,85697	54,29772897	53,28187331	57,50491876	54,03823116	51,53710671	52,37857209	51,53414232
15928,85565	54,29772838	53,28187121	57,50491809	54,03822838	51,53710257	52,37857039	51,53413818
15928,85462	54,29772783	53,28187037	57,5049211	54,03822672	51,53710319	52,37857383	51,5341388
15928,85392	54,29772959	53,28187259	57,504919	54,03822975	51,53710603	52,37857214	51,53414164
15928,85262	54,29772712	53,28186968	57,50491564	54,03822851	51,5371032	52,37856871	51,53413881
15928,85159	54,29772627	53,28186869	57,50491866	54,03822705	51,53710376	52,37856921	51,53413937
15928,85041	54,29772459	53,2818663	57,50492044	54,03822549	51,53710295	52,37856766	51,53413855
15928,85093	54,29773329	53,28187534	57,50492586	54,03823537	51,537111	52,37857701	51,53414661
15928,8497	54,29773213	53,28187302	57,50492573	54,03823221	51,53710939	52,37857483	51,534145
15928,84838	54,29772884	53,28186907	57,50492329	54,03822871	51,53710546	52,37857087	51,53414107
15928,85191	54,29773693	53,28188243	57,5049364	54,03824569	51,53712121	52,37858239	51,53415682
15928,85065	54,29773686	53,28188122	57,50493702	54,03824311	51,53711926	52,3785815	51,53415488
15928,85341	54,29774675	53,28189312	57,50494782	54,03825752	51,53713335	52,37859446	51,53416897
15928,8521	54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
15928,8509	54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
15928,8499	54,2977	53,2819	57,5049	54,0383	51,5371	52,3786	51,5342
15928,8490	54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
15928,8477	54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342
15928,8464	54,2977	53,2819	57,5050	54,0383	51,5371	52,3786	51,5342

2,39E-46	0			0	0,8

Anexo C

Outras Tabelas

C.1 Normal

	0,00	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0,0	0,0000	0,0080	0,0160	0,0239	0,0319	0,0399	0,0478	0,0558	0,0638	0,0717
0,1	0,0797	0,0876	0,0955	0,1034	0,1113	0,1192	0,1271	0,1350	0,1428	0,1507
0,2	0,1585	0,1663	0,1741	0,1819	0,1897	0,1974	0,2051	0,2128	0,2205	0,2282
0,3	0,2358	0,2434	0,2510	0,2586	0,2661	0,2737	0,2812	0,2886	0,2961	0,3035
0,4	0,3108	0,3182	0,3255	0,3328	0,3401	0,3473	0,3545	0,3616	0,3688	0,3759
0,5	0,3829	0,3899	0,3969	0,4039	0,4108	0,4177	0,4245	0,4313	0,4381	0,4448
0,6	0,4515	0,4581	0,4647	0,4713	0,4778	0,4843	0,4907	0,4971	0,5035	0,5098
0,7	0,5161	0,5223	0,5285	0,5346	0,5407	0,5467	0,5527	0,5587	0,5646	0,5705
0,8	0,5763	0,5821	0,5878	0,5935	0,5991	0,6047	0,6102	0,6157	0,6211	0,6265
0,9	0,6319	0,6372	0,6424	0,6476	0,6528	0,6579	0,6629	0,6680	0,6729	0,6778
1,0	0,6827	0,6875	0,6923	0,6970	0,7017	0,7063	0,7109	0,7154	0,7199	0,7243
1,1	0,7287	0,7330	0,7373	0,7415	0,7457	0,7499	0,7540	0,7580	0,7620	0,7660
1,2	0,7699	0,7737	0,7775	0,7813	0,7850	0,7887	0,7923	0,7959	0,7995	0,8029
1,3	0,8064	0,8098	0,8132	0,8165	0,8198	0,8230	0,8262	0,8293	0,8324	0,8355
1,4	0,8385	0,8415	0,8444	0,8473	0,8501	0,8529	0,8557	0,8584	0,8611	0,8638

1,5	0,8664	0,8690	0,8715	0,8740	0,8764	0,8789	0,8812	0,8836	0,8859	0,8882
1,6	0,8904	0,8926	0,8948	0,8969	0,8990	0,9011	0,9031	0,9051	0,9070	0,9090
1,7	0,9109	0,9127	0,9146	0,9164	0,9181	0,9199	0,9216	0,9233	0,9249	0,9265
1,8	0,9281	0,9297	0,9312	0,9328	0,9342	0,9357	0,9371	0,9385	0,9399	0,9412
1,9	0,9426	0,9439	0,9451	0,9464	0,9476	0,9488	0,9500	0,9512	0,9523	0,9534
2,0	0,9545	0,9556	0,9566	0,9576	0,9586	0,9596	0,9606	0,9615	0,9625	0,9634
2,1	0,9643	0,9651	0,9660	0,9668	0,9676	0,9684	0,9692	0,9700	0,9707	0,9715
2,2	0,9722	0,9729	0,9736	0,9743	0,9749	0,9756	0,9762	0,9768	0,9774	0,9780
2,3	0,9786	0,9791	0,9797	0,9802	0,9807	0,9812	0,9817	0,9822	0,9827	0,9832
2,4	0,9836	0,9840	0,9845	0,9849	0,9853	0,9857	0,9861	0,9865	0,9869	0,9872
2,5	0,9876	0,9879	0,9883	0,9886	0,9889	0,9892	0,9895	0,9898	0,9901	0,9904
2,6	0,9907	0,9909	0,9912	0,9915	0,9917	0,9920	0,9922	0,9924	0,9926	0,9929
2,7	0,9931	0,9933	0,9935	0,9937	0,9939	0,9940	0,9942	0,9944	0,9946	0,9947
2,8	0,9949	0,9950	0,9952	0,9953	0,9955	0,9956	0,9958	0,9959	0,9960	0,9961
2,9	0,9963	0,9964	0,9965	0,9966	0,9967	0,9968	0,9969	0,9970	0,9971	0,9972
3,0	0,9973	0,9974	0,9975	0,9976	0,9976	0,9977	0,9978	0,9979	0,9979	0,9980
3,1	0,9981	0,9981	0,9982	0,9983	0,9983	0,9984	0,9984	0,9985	0,9985	0,9986
3,2	0,9986	0,9987	0,9987	0,9988	0,9988	0,9988	0,9989	0,9989	0,9990	0,9990
3,3	0,9990	0,9991	0,9991	0,9991	0,9992	0,9992	0,9992	0,9992	0,9993	0,9993
3,4	0,9993	0,9994	0,9994	0,9994	0,9994	0,9994	0,9995	0,9995	0,9995	0,9995
3,5	0,9995	0,9996	0,9996	0,9996	0,9996	0,9996	0,9996	0,9996	0,9997	0,9997
3,6	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9997	0,9998	0,9998	0,9998
3,7	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998	0,9998
3,8	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999
3,9	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999
4,0	0,9999	0,9999	0,9999	0,9999	0,9999	0,9999	1,0000	1,0000	1,0000	1,0000
4,1	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,2	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

4,3	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,4	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,5	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,6	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,7	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,8	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
4,9	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
5,0	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

2

C.2 Distribuição F-Snedcor 3%

4

	Num																
Den	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	449,65	555,06	599,98	624,58	640,05	650,65	658,37	664,23	668,83	672,55	675,6	678,16	680,33	682,2	683,83	685,25	686,51
2	31,841	32,333	32,499	32,581	32,631	32,664	32,688	32,706	32,72	32,731	32,74	32,748	32,754	32,759	32,764	32,768	32,772
3	15,179	14,036	13,533	13,251	13,069	12,943	12,85	12,779	12,723	12,677	12,639	12,608	12,58	12,557	12,537	12,519	12,503
4	10,874	9,547	8,9718	8,6483	8,4404	8,2954	8,1884	8,1061	8,041	7,988	7,9442	7,9072	7,8757	7,8485	7,8247	7,8038	7,7853
5	9,0173	7,6646	7,0803	6,7508	6,5383	6,3896	6,2796	6,1948	6,1275	6,0727	6,0273	5,989	5,9562	5,9279	5,9031	5,8813	5,862
6	8,0028	6,6549	6,0729	5,7439	5,531	5,3816	5,2708	5,1853	5,1172	5,0617	5,0156	4,9766	4,9433	4,9145	4,8893	4,867	4,8473
7	7,3689	6,0318	5,4545	5,1272	4,915	4,7656	4,6546	4,5687	4,5003	4,4444	4,3978	4,3585	4,3248	4,2956	4,2701	4,2476	4,2275
8	6,937	5,6112	5,0386	4,7133	4,5018	4,3527	4,2416	4,1556	4,0868	4,0306	3,9838	3,9442	3,9102	3,8808	3,855	3,8322	3,8119
9	6,6245	5,3091	4,7407	4,4172	4,2065	4,0576	3,9465	3,8603	3,7914	3,735	3,6879	3,648	3,6138	3,5841	3,5581	3,5351	3,5146
10	6,3882	5,082	4,5172	4,1953	3,9853	3,8367	3,7256	3,6393	3,5702	3,5135	3,4663	3,4262	3,3917	3,3618	3,3355	3,3123	3,2916
11	6,2034	4,9052	4,3436	4,0231	3,8136	3,6652	3,5542	3,4678	3,3985	3,3417	3,2942	3,2539	3,2192	3,1891	3,1626	3,1392	3,1184
12	6,055	4,7638	4,2049	3,8856	3,6767	3,5285	3,4174	3,3309	3,2615	3,2045	3,1568	3,1163	3,0815	3,0511	3,0245	3,001	2,9799
13	5,9333	4,6481	4,0917	3,7735	3,565	3,4169	3,3058	3,2192	3,1497	3,0925	3,0447	3,004	2,969	2,9385	2,9117	2,8879	2,8668
14	5,8317	4,5519	3,9976	3,6802	3,4721	3,3242	3,2131	3,1264	3,0567	2,9994	2,9514	2,9105	2,8753	2,8447	2,8178	2,7939	2,7726
15	5,7456	4,4705	3,9181	3,6015	3,3938	3,2459	3,1348	3,048	2,9782	2,9208	2,8726	2,8316	2,7963	2,7655	2,7384	2,7144	2,6929

2

4

16	5,6718	4,4008	3,85	3,5342	3,3268	3,179	3,0679	2,981	2,9111	2,8535	2,8052	2,7641	2,7286	2,6976	2,6704	2,6463	2,6247
17	5,6077	4,3405	3,7912	3,476	3,2688	3,1211	3,01	2,9231	2,853	2,7953	2,7469	2,7056	2,67	2,6389	2,6115	2,5873	2,5656
18	5,5516	4,2877	3,7398	3,4252	3,2182	3,0706	2,9595	2,8724	2,8023	2,7445	2,6959	2,6545	2,6188	2,5876	2,5601	2,5357	2,5139
19	5,5021	4,2413	3,6945	3,3804	3,1737	3,0261	2,9149	2,8278	2,7576	2,6997	2,651	2,6095	2,5736	2,5423	2,5147	2,4902	2,4683
20	5,458	4,2	3,6543	3,3407	3,1341	2,9866	2,8754	2,7883	2,718	2,6599	2,6111	2,5695	2,5335	2,5021	2,4744	2,4498	2,4278
21	5,4186	4,1631	3,6184	3,3052	3,0988	2,9514	2,8401	2,7529	2,6825	2,6244	2,5755	2,5337	2,4977	2,4661	2,4383	2,4136	2,3915
22	5,3831	4,1299	3,5861	3,2733	3,0671	2,9197	2,8084	2,7211	2,6507	2,5924	2,5434	2,5016	2,4654	2,4338	2,4059	2,3811	2,3589
23	5,351	4,0999	3,5569	3,2445	3,0384	2,891	2,7797	2,6924	2,6219	2,5635	2,5145	2,4725	2,4363	2,4045	2,3766	2,3517	2,3294
24	5,3218	4,0727	3,5304	3,2183	3,0124	2,865	2,7537	2,6663	2,5957	2,5373	2,4881	2,4461	2,4098	2,378	2,3499	2,3249	2,3026
25	5,2952	4,0478	3,5063	3,1945	2,9886	2,8413	2,73	2,6426	2,5719	2,5134	2,4641	2,422	2,3856	2,3537	2,3256	2,3005	2,2781
26	5,2708	4,025	3,4842	3,1726	2,9669	2,8196	2,7083	2,6208	2,55	2,4915	2,4421	2,4	2,3634	2,3315	2,3033	2,2781	2,2556
27	5,2483	4,0041	3,4638	3,1525	2,9469	2,7997	2,6883	2,6008	2,5299	2,4713	2,4219	2,3797	2,3431	2,311	2,2827	2,2576	2,235
28	5,2275	3,9848	3,4451	3,134	2,9285	2,7813	2,6699	2,5823	2,5114	2,4527	2,4033	2,3609	2,3243	2,2922	2,2638	2,2386	2,2159
29	5,2083	3,9669	3,4277	3,1169	2,9115	2,7642	2,6528	2,5652	2,4943	2,4355	2,386	2,3436	2,3069	2,2747	2,2463	2,221	2,1983
30	5,1905	3,9503	3,4116	3,101	2,8957	2,7485	2,637	2,5494	2,4784	2,4196	2,37	2,3275	2,2907	2,2585	2,23	2,2046	2,1819
40	5,0637	3,8327	3,2976	2,9885	2,7838	2,6367	2,5251	2,4371	2,3657	2,3065	2,2564	2,2134	2,1761	2,1434	2,1144	2,0885	2,0653
50	4,9898	3,7644	3,2315	2,9233	2,7189	2,5718	2,4601	2,3719	2,3003	2,2407	2,1903	2,147	2,1093	2,0762	2,0469	2,0207	1,9972
60	4,9414	3,7197	3,1882	2,8807	2,6765	2,5295	2,4177	2,3293	2,2575	2,1977	2,147	2,1035	2,0656	2,0323	2,0027	1,9763	1,9525
70	4,9072	3,6882	3,1578	2,8507	2,6467	2,4997	2,3878	2,2993	2,2273	2,1674	2,1165	2,0728	2,0347	2,0012	1,9715	1,9449	1,9209
80	4,8818	3,6648	3,1352	2,8284	2,6245	2,4775	2,3656	2,2771	2,2049	2,1448	2,0939	2,05	2,0118	1,9781	1,9482	1,9215	1,8974
90	4,8622	3,6468	3,1178	2,8112	2,6074	2,4604	2,3485	2,2599	2,1876	2,1275	2,0764	2,0324	1,9941	1,9603	1,9303	1,9034	1,8792
100	4,8466	3,6324	3,1039	2,7976	2,5939	2,4469	2,3349	2,2462	2,1739	2,1136	2,0625	2,0184	1,98	1,9461	1,916	1,889	1,8647
105	4,8399	3,6263	3,098	2,7917	2,5881	2,4411	2,3291	2,2403	2,168	2,1077	2,0565	2,0124	1,9739	1,94	1,9099	1,8829	1,8585
110	4,8339	3,6208	3,0926	2,7865	2,5828	2,4358	2,3238	2,235	2,1627	2,1024	2,0511	2,007	1,9685	1,9345	1,9043	1,8773	1,8529
115	4,8284	3,6157	3,0877	2,7816	2,578	2,431	2,319	2,2302	2,1578	2,0975	2,0462	2,002	1,9635	1,9295	1,8993	1,8722	1,8477
∞	4,7106	3,5078	2,9836	2,6791	2,4761	2,3291	2,2167	2,1275	2,0545	1,9934	1,9413	1,8964	1,857	1,8222	1,7911	1,7632	1,7379