2014

Instituto Politécnico de Coimbra

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Classificação de Literatura Biomédica

MESTRADO EM INFORMÁTICA E SISTEMAS

AUTOR | **João Pedro dos Santos Oliveira**

ORIENTADOR | **Prof. Doutor Carlos Pereira**

Coimbra, dezembro, 2014

Resumo

Atualmente existe uma enorme quantidade de informação online de literatura biomédica. A PubMed, o repositório de dados líder nesta área, destaca-se, à data atual, com mais de 23 milhões de citações a partir da Medline. Devido a esta quantidade de informação disponível torna-se difícil, para os utilizadores da área, a pesquisa, análise e organização da informação relevante.

Para apoiar estas tarefas, foi desenvolvida uma aplicação web, designada DoCluster 2.0, onde os utilizadores podem extrair informação relevante e classificar documentos em repositórios locais ou documentos obtidos através de uma pesquisa ao *Web service* da PubMed. Na extração de informação, para além da segmentação de documentos, normalmente realizada em qualquer processo de *text mining*, foram introduzidas duas ontologias que permitem uma extração de informação adequada a áreas específicas. As ontologias usadas foram a Gene Ontology que se foca na área de genes e produtos resultantes desses genes, adaptada neste caso ao domínio das peptidases e a Merops que se centra também na área das peptidases. Para a classificação dos documentos recorreu-se a técnicas de aprendizagem não supervisionada, através dos algoritmos *k-means*, *fuzzy c-mean*s e *subtractive clustering* e a uma técnica de aprendizagem supervisionada baseada em máquinas de vectores de suporte.

Para averiguar quais os melhores métodos de aprendizagem e de extração de caraterísticas do problema, foram realizados vários testes sobre *datasets* no domínio das peptidases. curados pela Merops. A medição dos resultados teve incidência em diferentes métricas, sendo elas a precisão e *recall* do classificador, o número de características extraídas no pré-processamento de documentos e o custo computacional de todo o processo de *text mining*.

Da análise dos resultados obtidos concluiu-se que as máquinas de vectores de suporte conseguem um melhor desempenho em relação aos algoritmos de aprendizagem não supervisionada, contudo exigindo um treino prévio dos classificadores. No pré-processamento de documentos, através do uso de ontologias, foi possível melhorar o desempenho de todo o processo e obter informação com um menor número de características sem que a qualidade do classificador diminua. Constatou-se também que o algoritmo *subtractive clustering*, por não necessitar da definição *a priori* do número de *clusters* é ideal para o tratamento de um conjunto de documentos em relação ao qual não existe um conhecimento prévio do seu conteúdo, como é o caso de documentos obtidos através da PubMed.

Palavras-chave: classificação de literatura biomédica, peptidases, *text mining*, ontologias, *k-means clustering*, *fuzzy c-means clustering*, *subtractive clustering*, máquinas de vectores de suporte.

Abstract

Currently there is a huge amount of online information on biomedical literature. PubMed, the leading data repository in this area stands out, at current date, with over 23 million citations from MEDLINE. Because of this amount of available information it is difficult for the users in this area, the search, analysis and organization of relevant information.

To support these tasks, in this project it was developed a web application, designated DoCluster 2.0, where users can extract relevant information and classify documents in local repositories or documents obtained by searching the Web service PubMed. In the information extraction phase, in addition to documents segmentation, usually performed at any text mining process, two ontologies have been introduced allowing the extraction of appropriate information to specific areas of research. The ontologies used were the Gene Ontology, that focuses on the area of genes and resulting products of these genes and adapted to the field of peptidases, and the Merops which also focuses on the area of peptidases. For documents classification phase, were used unsupervised learning techniques, through the k-means, fuzzy c-means and subtractive clustering algorithms, and a supervised learning technique based on support vector machines.

To determine the best methods of learning and features extraction, and in order to understand the best solutions in different situations, some tests on datasets cured by Merops were performed. The results focused on different metrics, being the accuracy and recall of the classifiers, the number of extracted features in the pre-processing phase and the computational cost of the whole process of text mining.

From the results analysis it was concluded that the support vector machines achieved a better classification performance compared to the unsupervised learning algorithms, requiring however, a prior training of the classifiers. In documents pre-processing, through the use of ontologies, it was possible to improve the performance of the whole process and obtain less information without decrease the quality of the classifiers. It was also found that the subtractive clustering algorithm, by not requiring the prior definition of the number of clusters its ideal for handling a set of documents for which there is no previous knowledge of its contents, as the case documents obtained through the PubMed.

Keywords: biomedical literature classification, peptidases, text-mining, ontologies, k-means clustering, fuzzy c-means clustering, subtractive clustering, support vector machines.

Agradecimentos

Aproveito para agradecer aos meus pais por todo o esforço que fizeram ao longo dos anos para que este momento fosse possível.

Ao meu irmão por todos os conselhos e amizade.

À minha namorada por sempre me apoiar e dar ânimo ao longo desta caminhada.

A toda a família pelo interesse demonstrado e pela vontade de me verem atingir esta meta.

Ao meu orientador pela ajuda e pela compreensão nos momentos em que estive mais afastado devido à distância.

Ao LIIT e ao projeto BIOINK que disponibilizaram todos os meios necessários para a realização deste trabalho.

Índice

Índice vii
Índice de Figuras ix
Índice de Tabelas xii
Termos e Acrónimos xiv
1. INTRODUÇÃO 1
1.1. Objetivos e motivação 2
1.2. Visão e Âmbito 2
1.3. Estrutura do relatório 3
2. ESTADO DE ARTE 5
2.1. Text Mining 5
2.2. Ontologias 6
2.3.1. Gene Ontology 7
2.3.2. MEROPS 8
2.3. Algoritmos de classificação 9
2.3.1. Algoritmo K-means 9
2.3.2. Algoritmo Fuzzy C-means 10
2.3.3. Algoritmo Subtractive Clustering 11
2.3.4. Máquinas de Vectores de Suporte 12
3. ANÁLISE DE REQUISITOS 17
3.1. Casos de uso 17
3.2.1. Atores do sistema 17
3.2.2. Lista de casos de uso 18
3.2.3. Diagrama de casos de uso 19
3.2.4. Descrição dos casos de uso 19
3.2. Diagramas de atividade 34
3.3. Diagramas de sequência 35
3.4. Protótipos de *interface* 35
4. DESENHO E ARQUITECTURA 37
4.1. Arquitetura 37
4.2. Diagrama de componentes 38
4.3.1. *Model* 38
4.3.2. *Controller* 38
4.3.3. *View* 38
4.3.4. *ViewModels* 39
4.3.5. *Entity Framework 4* 39
4.3.6. PubMed 39
4.3.7. *Matlab Application Type Library* 39
4.3.8. *WVTool* 39
4.3.9. *Nunrar* 39
4.3.10. *ICSharpCode.SharpZipLib* 39
4.3. Base de dados 40
5. IMPLEMENTAÇÃO 41

5.1. Obtenção de documentos 41
5.1.1. Obtenção de documentos do utilizador 42
5.1.2. Obtenção de documentos da PubMed 43
5.1.3. Obtenção de documentos do utilizador para LIBSVM 44
5.2. Extração de características 45
5.2.1. *NoOntology* 46
5.2.2. Gene Ontology 50
4.2.3. Merops 53
5.3. Classificação dos documentos 55
5.3.1. *K-means* 57
5.3.2. *Fuzzy C-means* 59
5.3.3. *Subtractive* 61
5.3.4. LIBSVM 62
5.3.4.1. LIBSVM com *cross-validation* 62
5.3.4.2. Treino de Máquinas de Vectores de Suporte 64
5.3.4.3.Classificação de documentos 65
6. ANÁLISE DE RESULTADOS 69
6.1. Resultados sem o uso de ontologias 71
6.1.1. *Pruning* 71
6.1.2. *Ngram* 73
6.1.3. *Vector Creation Type* 74
6.2. Resultados com ontologias 75
6.3. *Subtractive clustering* 77
7. CONCLUSÕES 81
8. REFERÊNCIAS BIBLIOGRÁFICAS 83
ANEXO A 85
ANEXO B 89
ANEXO C 109

Índice de Figuras

Figura 1 – Exemplo de grafo acíclico presente no Gene Ontology (Ontology Structure, 2014) 7
Figura 2 – Tabelas pertencentes à base de dados Merops 8
Figura 3 – Entradas na base de dados Merops relativas à peptidase *renin-2* 8
Figura 4 – Exemplo de escolha do melhor hiperplano (Computer Science Source, 2014) 13
Figura 5 – Exemplo de utilização de kernel para classificadores não lineares (Computer Science Source, 2014) 13
Figura 6 – Exemplo de SVM multiclasse usando a técnica um contra todos. 14
Figura 7 – Exemplo de SVM multiclasse usando a técnica um contra um 15
Figura 8 – Diagrama de casos de uso da aplicação DoCluster 19
Figura 9 – Diagrama de atividades: Treinar SMVs 34
Figura 10 – Diagrama de atividades: Agrupamento de documentos enviados por um utilizador 35
Figura 11 – Exemplo do padrão MVC dentro da aplicação DoCluster 37
Figura 12 – Diagrama de componentes 39
Figura 13 – Modelo de dados usado no DoCluster 40
Figura 14 – Padrão *template* usado na obtenção de documentos 41
Figura 15 – Excerto de código da classe ObtainDocuments 42
Figura 16 – Código com pedido ao *Web service* da PubMed 43
Figura 17 – Código com a obtenção de documentos do PubMed 44
Figura 18 – Diagrama de classes que representa a obtenção de documento para o LIBSVM 45
Figura 19 – Exemplo de uma matriz BOW 45
Figura 20 – Diagrama de classes relativo às diferentes formas de extração de características 46
Figura 21 – Processo de extração de características do *WVTool* 46
Figura 22 – Exemplo de corte de termos usando *prune frequence* com valor 3. 48
Figura 23 – Exemplo do esquema de indexação de palavras Gene Ontology 51
Figura 24 – *Singleton* que inicializa o esquema de indexação Gene Ontology 51
Figura 25 – Código da classe *GeneOntologySingleton* 52
Figura 26 – Código com extração de *features* usando GeneOntology 52
Figura 27 – Diagrama de classes do esquema de indexação Merops 53
Figura 28 – *Singleton* que inicializa o esquema de indexação Merops 54
Figura 29 – Diagrama de classes com padrão *template* para execução dos algoritmos. 55
Figura 30 – Código com a função execute da classe *Algorithm* 55
Figura 31 – Etapas de execução de um algoritmo 56
Figura 32 – Código da função *executeAlgorithm* 57
Figura 33 – Exemplos dos ficheiros de resultados criados pelo algoritmo K-means 58
Figura 34 – Diagrama de classes relativo ao algoritmo *K-means* 59
Figura 35 – Exemplo de ficheiros de resultados criados pelo algoritmo *Fuzzy C-means* 60
Figura 36 – Diagrama de classes relativo ao algoritmo *Fuzzy C-means* 60
Figura 37 – Diagrama de classes relativo ao algoritmo *Subtractive Clustering* 61

Figura 38 - Exemplo do ficheiro de resultados criados pelo LIBSVM com *cross-validation* 63
Figura 39 - Diagrama de classes relativo ao processo de LIBSVM com *cross-validation* 63
Figura 40 – Diagrama de classes relativo ao processo de treino de SVMs 64
Figura 41 – Exemplo dos ficheiros de resultados criados pelo algoritmo de classificação 66
Figura 42 – Diagrama de classes relativo ao processo de classificação de documentos 66
Figura 43 – Gráfico com resultados de todos os classificadores para diferentes valores de *pruning frequence* no *dataset 1* 72
Figura 44 – Gráfico com o resultado dos vários classificadores usando vários *ngram* no *dataset 3* 73
Figura 45 - Gráfico com o resultado dos vários classificadores usando vários *ngram* no *dataset 4* 73
Figura 46 – Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz BOW no *Dataset 1*. 74
Figura 47 - Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 2*. 74
Figura 48 – Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 3*. 75
Figura 49 - Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 5*. 75
Figura 50 - Gráfico com o resultado dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 1* 76
Figura 51 - Gráfico com o resultado dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 2* 76
Figura 52 – Tempo total de execução dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 1* 77
Figura 53 - Tempo total de execução dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 2* 77
Figura 54 – Visão global dos resultados obtidos com o *Subtractive clustering* 78
Figura 55 – Visão mais detalhada dos resultados obtidos com o algoritmo *Subtractive clustering* 78
Figura 56 – Diagrama de sequência: Treino de máquinas de vectores de suporte 85
Figura 57 – Diagrama de sequência: Classificação de documentos usando *SVMs* treinadas 86
Figura 58 – Diagrama de sequência: Classificação de documentos do utilizador usando o algoritmo *K-means* 87
Figura 59 – Protótipo de *interface*: Página inicial 89
Figura 60 – Protótipo de *interface*: Página de *login* 89
Figura 61 – Protótipo de interface: Registrar utilizador 90
Figura 62 – Protótipo de interface: Redefinir password 90
Figura 63 – Protótipo de *interface*: Alterar *password* 91
Figura 64 - Protótipo de *interface*: Informação detalhada de um utilizador 91

Figura 65 - Protótipo de *interface*: Lista de utilizadores registados 92
Figura 66 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *K-means* 93
Figura 67 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *Fuzzy C-means* 94
Figura 68 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *Subtractive Clustering* 95
Figura 69 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *K-means* 96
Figura 70 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *Fuzzy C-means* 97
Figura 71 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *Subtractive Clustering* 98
Figura 72 – Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos na lista de clusters 99
Figura 73 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos na lista de documentos 99
Figura 74 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos num documento especifico. 100
Figura 75 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco na visualização gráfica 100
Figura 76 - Protótipo de *interface*: Formulário para o treino de máquinas de vectores de suporte 101
Figura 77 – Protótipo de *interface*: Formulário para a utilização do LIBSVM com cross-validation 102
Figura 78 - Protótipo de *interface*: Resultados relativos à utilização do LIBSVM com *cross-validation* 103
Figura 79 - Protótipo de *interface*: Formulário para a classificação de documentos enviados pelo utilizador usando SVMs treinadas. 103
Figura 80 - Protótipo de *interface*: Formulário para a classificação de documentos pesquisados na PubMed usando SVMs treinadas. 104
Figura 81 - Protótipo de *interface*: Resultados com a classificação de documentos recorrendo a SVMs treinadas 105
Figura 82 - Protótipo de *interface*: Lista de agrupamentos de documentos guardados 106
Figura 83 - Protótipo de *interface*: Lista de máquinas de vectores de suporte treinadas 106
Figura 84 - Protótipo de *interface*: Lista de documentos guardados 107
Figura 85 - Protótipo de *interface*: Documento guardado 107

Índice de Tabelas

Tabela 1 – Atores do sistema, descrição dos atores e as suas responsabilidades. 17
Tabela 2 - Atores do sistema e respectivos casos de usos 18
Tabela 3 – Caso de uso: Agrupar documentos utilizador 20
Tabela 4 – Caso de uso: Agrupar documentos PubMed 21
Tabela 5 – Caso de uso: Usar LIBSVM com *cross-validation* 22
Tabela 6 – Caso de uso: Classificar documentos utilizador usando SVMs 22
Tabela 7 – Caso de uso: Classificar documentos PubMed usando SVMs 23
Tabela 8 – Caso de uso: Definir parâmetros de extração de *features* 23
Tabela 9 – Caso de uso: Enviar documentos 24
Tabela 10 – Caso de uso: Definir pesquisa PubMed 24
Tabela 11 – Caso de uso: Definir parâmetros *K-means* 25
Tabela 12 – Caso de uso: Definir parâmetros *Fuzzy C-means* 25
Tabela 13 – Caso de uso: Definir parâmetros *Subtractive* 25
Tabela 14 – Caso de uso: Definir Parâmetros LIBSVM 26
Tabela 15 – Caso de uso: Definir parâmetros LIBSVM com *cross-validation* 26
Tabela 16 – Caso de uso: Treinar SVMs 27
Tabela 17 – Caso de uso: Remover SVMs guardadas 27
Tabela 18 – Caso de uso: Guardar agrupamento de documentos 28
Tabela 19 – Caso de uso: Visualizar agrupamento de documentos 28
Tabela 20 – Caso de uso: Remover agrupamento de documentos 28
Tabela 21 – Caso de uso: Guardar documentos 29
Tabela 22 – Caso de uso: Visualizar documentos guardados 29
Tabela 23 – Caso de uso: Remover documentos guardados 30
Tabela 24 – Caso de uso: Validar novos utilizadores 30
Tabela 25 – Caso de uso: Bloquear utilizador 31
Tabela 26 – Caso de uso: Desbloquear utilizador 31
Tabela 27 – Caso de uso: Remover utilizador 32
Tabela 28 – Caso de uso: Editar limites utilizador 33
Tabela 29 – *Dataset 1* 70
Tabela 30 – *Dataset 2* 70
Tabela 31 – Dataset 3 71
Tabela 32 – *Dataset 4* 71
Tabela 33 – *Dataset 5* 71
Tabela 34 – Tabela com a legenda dos termos presentes nas tabelas de resultados 109
Tabela 35 – Resultados dos classificadores com extrações de características usando diferentes valores de *Prune Frequence* no *Dataset 1* 109
Tabela 36 – Resultados dos diferentes classificadores com extrações de *features* usando diferentes valores de *ngrams* no *Dataset 3* 110

Tabela 37 - Resultados dos diferentes classificadores com extrações de *features* usando diferentes valores de *ngrams* no *Dataset 4* 111
Tabela 38 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 1* 111
Tabela 39 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 2* 111
Tabela 40 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 3* 111
Tabela 41 – Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 5* 111
Tabela 42 - Resultados dos diferentes classificadores com extrações de características usando ontologias no *Dataset 1* 112
Tabela 43 - Resultados dos diferentes classificadores com extrações de características usando ontologias no *Dataset 2* 112
Tabela 44 – Resultados do algoritmo *Subtractive clustering* em extrações de *features* sem ontologias, com Gene Ontology e com Merops usando diferentes valores no parâmetro *radii* 112

Termos e Acrónimos

MEDLINE – do inglês *Medical Literature Analysis and Retrieval System Online*, é uma base de dados bibliográfica de informação biomédica. Inclui informações a partir de artigos obtidos em jornais de medicina, farmácia, cuidados de saúde, entre outros. Contém também informações de literatura na área de biologia e bioquímica.

NCBI – do inglês *Nacional Center for Biotechnology Information*.

PubMed – *Public MEDLINE* - Base de dados que contém mais de 23.000.000 de citações a partir da MEDLINE, revistas científicas e livros online.

MeSH – *Medical Subject Headings*, é uma enciclopédia de termos usada para indexar artigos para a PubMed.

SVM – *Support Vector Machines* - Máquinas de vectores de suporte.

FCM – *Fuzzy C-means*. É um algoritmo de aprendizagem não-supervisionada.

WVTool – *Word Vector Tool* - Biblioteca construída na linguagem de programação *java*, e que permite realizar a extração de *features* a partir de um conjunto de documentos.

LIBSVM – *Library for Support Vector Machines* - Biblioteca que implementa máquinas de vectores de suporte.

Matriz BOW – Matriz *Bag Of Words*.

GO – *Gene Ontology* - Projeto iniciado com o objetivo de criar uma representação standard de genes e produtos resultantes de genes espalhados por várias bases de dados.

IDF – *Inverse Document Frequency*.

TFIDF – *Term Frequency Inverse Document Frequency* - Opção para extração de features que cria uma matriz BOW tendo não só em atenção a frequência com que uma *feature* está presente num documento, mas também no conjunto de todos os documentos.

TP – *True Positives* - Documentos corretamente atribuídos a uma classe ou cluster.

FP – *False Positives* - Documentos erradamente atribuídos a uma classe ou cluster.

TN – *True Negatives* - Documentos corretamente não atribuídos a uma classe ou cluster.

FN – *False Negatives* - Documentos erradamente não atribuídos a uma classe ou cluster.

MVC – *Model-View-Controller* – padrão de desenho de software Modelo-Vista-Controlador

1. INTRODUÇÃO

Nos últimos anos tem-se assistido a um constante crescimento da área de biomedicina e a um aumento exponencial de literatura associada (livros, revistas, artigos científicos, entre outras). A maior base de dados de literatura biomédica, designada de PubMed, conta já com mais de 23.000.000 de citações a partir da MEDLINE, revistas cientificas e livros online (NCBI PubMed, 2014), tornando difícil para um utilizador a obtenção de informação relevante numa pesquisa. Na maior parte destas bases de dados online também não é dada a conhecer a similaridade entre os documentos retornados.

A PubMed possui, para uma parte considerável de documentos, termos MeSH (*Medical Subject Headings*) que ajudam à categorização dos documentos. No entanto esta informação é atribuída manualmente e considerando a enorme quantidade de textos, torna-se impraticável a categorização manual de todos os documentos. É por estas razões que se tem assistido a um investimento cada vez maior em ferramentas de *text mining* e categorização automática que possam resolver o problema.

No domínio da classificação de documentos, atualmente existem algumas ferramentas como a *toolbox* de *text mining* do RapidMiner (Rapid - I, 2012) ou o *add-on* de análise de texto da Orange (Orange, 2014) que conseguem obter bons resultados. No entanto essas ferramentas dependem bastante dos parâmetros definidos para o pré-processamento dos documentos e para os algoritmos de classificação. No caso dos parâmetros dos algoritmos de classificação essa dependência é ainda influenciada pelo processo de pré-processamento dos documentos.

É neste contexto, e principalmente para ajudar na definição dos parâmetros usados no pré-processamento dos documentos, que as ontologias podem ser importantes no processo de *text mining*. Atualmente já existem ferramentas que fazem uso de várias ontologias como será apresentado no capítulo 2.

Neste trabalho desenvolveu-se uma plataforma web de pesquisa inteligente de literatura biomédica, fazendo uso de técnicas de *text mining*, usando ontologias no auxílio ao pré-processamento de documentos, recorrendo a algoritmos de *clustering* como o *K-means*, *Fuzzy C-Means* e *Subtractive* e a classificadores supervisionados, tais como máquinas de vectores de suporte (SVM).

Abordagens semelhantes já foram seguidas em outras plataformas SVMSearch (Leite, 2009) e Biomedoc (Correia, 2010), contudo nesta plataforma foram adicionados novos parâmetros à extração de *features* com a utilização de *ngrams* e com a inclusão da ontologia Gene Ontology (Gene Ontology Consortium, 2012) e da base de dados Merops (Rawlings, Waller, Barrett, & Bateman, 2014).

1.1. Objetivos e motivação

O objetivo deste trabalho é construir e disponibilizar uma plataforma web para classificação de documentos através de algoritmos de aprendizagem não supervisionada (algoritmos de *clustering*) e supervisionada (máquinas de vectores de suporte).

A aprendizagem não supervisionada consiste em descobrir padrões interessantes a partir de conjuntos de dados que não estão etiquetados. Como estes dados não estão etiquetados não existe forma de avaliar a classificação que é realizada. Esta é, aliás, a grande diferença em relação à aprendizagem supervisionada, uma vez que esta conta com um conjunto de exemplos de treino, já etiquetados. Um algoritmo de aprendizagem supervisionada analisa o conjunto de exemplos de treino e cria uma função inferida, denominada como classificador. Através deste classificador é possível inferir a classe de um novo objeto de dados.

Para realizar a aprendizagem não supervisionada, a plataforma contém alguns algoritmos de *clustering* tais como o *K-Means*, *Fuzzy C-Means* e *Subtractive Clustering*, que são posteriormente avaliados.

Para a aprendizagem supervisionada são usadas máquinas de vectores de suporte. No final os resultados deste classificador também serão avaliados.

Por fim pretende-se aperfeiçoar as técnicas tradicionais de extração de *features*, introduzindo ontologias nesse processo. No final pretende-se também avaliar os resultados obtidos com esta optimização.

1.2. Visão e Âmbito

Atualmente várias pessoas ligadas à área da biomédica deparam-se com problemas na pesquisa e compreensão de documentos. Investigadores da área têm dificuldades na procura e organização de informação relevante. Curadores de documentos dispendem demasiado tempo na leitura completa e compreensão dos documentos para a posterior atribuição da classe a que pertencem. Até pessoas que não estão ligadas à área, mas pretendam encontrar informações que lhe são relevantes, sentem dificuldades em compreender o que a documentação da área oferece.

Para este conjunto de pessoas a ferramenta criada vai apresentar várias funcionalidades que as ajudarão a ultrapassar as suas dificuldades. As principais funcionalidades da ferramenta são:

- Agrupamento de documentos enviados pelo utilizador ou obtidos a partir da PubMed, recorrendo aos algoritmos *K-Means*, *Fuzzy C-Means* e *Subtractive Clustering*.
- Visualização gráfica dos resultados de agrupamento obtidos.
- Classificação de documentos fazendo uso de algoritmos de aprendizagem supervisionada, nomeadamente máquinas de vectores de suporte (SVM).
- Uso de ontologias na extração de informação dos documentos de forma a extrair a informação relevante no domínio das peptidases.
- Marcação, nos documentos, dos termos presentes nas ontologias utilizadas e ligações para esses termos nas páginas *online* das respectivas ontologias.
- Guardar modelos de classificadores.
- Guardar agrupamentos de documentos realizados com diferentes algoritmos.
- Guardar documentos relevantes.

1.3. Estrutura do relatório

Este relatório é constituído por um total de sete capítulos ordenados da seguinte forma: Introdução, Estado de arte, Análise de requisitos, Desenho e arquitetura, Implementação, Análise de resultados e Conclusões.

O capítulo corrente dá uma visão sobre os objetivos e visão para este projeto.

No segundo capítulo é apresentado o estado de arte relativamente ao processo de *text mining*, aos algoritmos de classificação e às ontologias que serão usados na aplicação.

Na análise de requisitos são especificados diagramas de casos de uso, diagramas de atividade mais relevantes, diagramas de sequência e protótipos de interface.

No capítulo de desenho e arquitetura é explicada a arquitetura da aplicação, é apresentado o seu diagrama de componentes e o diagrama da base de dados usada.

No quinto capítulo é apresentada uma visão mais aprofundada relativamente aos pontos fulcrais da implementação. O capítulo divide-se nas diferentes fases do *text-mining* e na forma como estão implementados pela aplicação. É especificado como são obtidos os documentos, como a informação é extraída desses documentos e como são classificados pelos diferentes algoritmos.

Na análise de resultados são apresentados alguns testes com diferentes parâmetros de extração de características, incluindo o uso de ontologias, e testados diferentes parâmetros em cada um dos algoritmos. Para a avaliação destes resultados, são usados indicadores como a precisão dos classificadores e a velocidade de execução dos mesmos.

Por fim, a partir dos resultados obtidos, são retiradas as conclusões mais pertinentes e projetado o trabalho futuro a desenvolver.

2. ESTADO DE ARTE

Este projeto envolve de forma bastante perceptível todas as fases de um processo de *text mining*. Estas fases são visíveis nas diferentes formas de obtenção de documentos, nas técnicas de extração de conhecimento a partir dos documentos e na utilização de diferentes algoritmos de aprendizagem não supervisionada e supervisionada para a classificação dos mesmos.

Para além do envio de documentos por parte de um utilizador foi disponibilizada a possibilidade de obtenção de documentos a partir do repositório líder na área de literatura biomédica, a PubMed, através de um *Web service* onde podem ser obtidos vários artigos de acordo com um conjunto de parâmetros de pesquisa (Entrez Programming Utilities Help, 2012).

Na fase de extração de conhecimento dos documentos, para além de uma biblioteca *open source* denominada *WVTool* (Word Vector Tool, 2010), foram usadas ontologias na obtenção de uma informação melhorada, mais "limpa" e orientada a uma área específica. Nesta ferramenta é focada a área das peptidases e os seus inibidores, áreas para as quais foram escolhidas as ontologias. Essas ontologias são a GeneOntology (The Gene Ontology Consortium, 2000) e Merops (Rawlings, Waller, Barrett, & Bateman, 2014).

Para a classificação de documentos foram usados vários algoritmos de aprendizagem não supervisionada como são os casos dos algoritmos de *clustering Kmeans*, *Fuzzy C-Means* e *Subtractive*. Para além destes, no caso da aprendizagem supervisionada, foram usadas máquinas de vectores de suporte através de uma biblioteca denominada LIBSVM (Chang & Lin, 2011).

De seguida é apresentado o estado de arte de algumas das técnicas, algoritmos e componentes que formam esta ferramenta.

2.1. Text Mining

O *text mining* consiste num processo onde se pretendem encontrar padrões e informações relevantes em textos escritos em linguagem natural e não estruturada. O processo divide-se em quatro fases, sendo elas a obtenção de documentos, o pré-processamento de texto, a classificação de documentos e a avaliação de resultados.

A obtenção de documentos é a fase mais simples deste processo consistindo apenas na escolha dos documentos a serem classificados.

A fase de pré-processamento de documentos é uma das fases mais importantes no processo de *text mining*. Nesta fase a informação não estruturada dos documentos é transformada num modelo de dados que é depois utilizado pelos diferentes algoritmos de classificação. O modelo de dados consiste numa matriz, conhecida como matriz *BOW*. Nesta matriz são representados os documentos, as características extraídas desses documentos e a importância

de cada característica para cada documento através de valores numéricos. Para a obtenção da matriz *BOW*, um documento é sujeito a um conjunto de técnicas de pré-processamento. Algumas dessas técnicas, também usadas neste projeto, serão dadas a conhecer mais à frente.

Após a fase de pré-processamento, já com um modelo de dados criado, é executada a fase de classificação de documentos onde são usados algoritmos de aprendizagem supervisionada e não supervisionada. Estes algoritmos têm como objetivo procurar padrões e relações entre os documentos. Também serão focados mais à frente os algoritmos usados nesta ferramenta.

Na última fase serão avaliados os resultados da classificação dos documentos. Esta fase é apenas executada quando existe uma etiquetação prévia dos mesmos. Em algoritmos de aprendizagem não supervisionada não é possível a sua realização uma vez que não existe um conhecimento prévio dos documentos a classificar.

O *text mining* pode ter várias aplicações práticas, sendo os motores de busca provavelmente as mais conhecidas em relação a esta área. No entanto a base do surgimento do *text mining* e uma das suas aplicações mais promissoras é a área das biociências onde se procuram encontrar padrões e informações relevantes dentro de subáreas das biociências como a medicina ou a genómica. É precisamente nesta área que este trabalho está focado.

Existe uma variedade enorme de ferramentas de *text mining* disponíveis. Aplicações como AeroText (Rocket Software, 2012) ou os já citados Orange e RapidMiner são algumas delas.

Em termos de ferramentas relacionadas com a área das biociências destacam-se, entre muitas outras, o GoPubMed (Doms & Schroeder, 2005) e o Coremine medical (Coremine medical, 2012).

2.2. Ontologias

A palavra ontologia tem origem na filosofia e neste contexto tem como base o estudo da existência. Em sistemas baseados em conhecimento o que existe é o que pode ser representado, sendo tudo o que existe acerca de um domínio, aquilo que as ontologias pretendem representar. Segundo *Gruber* (Gruber, 1993) as ontologias podem ser definidas como uma especificação explicita de uma conceptualização.

Uma ontologia é constituída por termos que pretendem representar um domínio especifico, sendo que esses termos definem um conjunto de entidades desse domínio. Essas entidades são classes, atributos dessas classes, suas instâncias e relações entre elas. Estas relações devem formar axiomas de forma a impedir a existência de mais do que uma interpretação.

As ontologias podem ser divididas em ontologias de domínio e ontologias superiores. Nas ontologias de domínio os termos são representados de acordo com um domínio especifico, ou seja, existindo um termo que pode conter vários significados é possível definir esse termo apenas dentro do domínio pretendido. No caso de uma ontologia superior são descritos conceitos genéricos normalmente aplicáveis em diferentes ontologias de domínio.

Várias áreas como a engenharia de software, inteligência artificial ou a web semântica criam ontologias de forma a organizar e limitar a complexidade da informação.

2.3.1. Gene Ontology

O Gene Ontology (GO) é um projeto iniciado em 2000 com o objetivo de criar uma representação padrão, entre várias bases de dados de genes e os seus produtos resultantes. Este projeto foi criado pelo GO Consortium que é constituído por um conjunto de bases de dados e comunidades de pesquisa da área de biomédica. Estas entidades são responsáveis pelo continuo desenvolvimento da ontologia e da sua aplicação.

A ontologia é constituída por um conjunto termos e as relações existentes entre eles. Estes termos descrevem os genes e os seus produtos resultantes de acordo com os processos biológicos, componentes celulares e funções moleculares. Por cobrir estes três domínios específicos o GeneOntology é considerado como um conjunto de três sub-ontologias.

A estrutura da GO pode ser vista como um grafo em que cada nó representa um termo e cada aresta entre dois nós representa a relação entre dois termos. Como pode ser visto pela Figura 1 esta é considerada uma estrutura hierárquica uma vez que a especialização dos termos vai aumentando à medida que se desce no grafo, sendo que as raízes do grafo são os três termos correspondentes às três ontologias já enunciadas atrás. Apesar da estrutura hierárquica existe a particularidade de um nó poder ter diferentes relações com mais do que um nó superior.

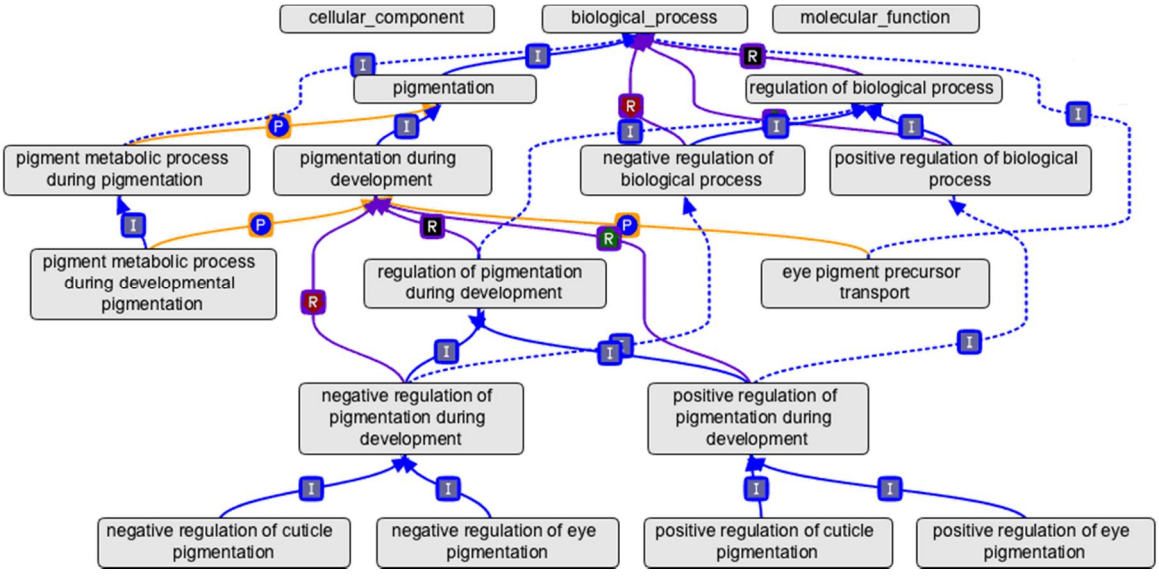


Figura 1 – Exemplo de grafo acíclico presente no Gene Ontology (Ontology Structure, 2014)

Cada termo GO é constituído por um conjunto de propriedades, sendo as mais importantes na sua definição um identificador único, a sub-ontologia a que pertence, a definição do termo com referências para a fonte de informação, sinónimos e referências cruzadas entre base de dados e relações para outros termos. Para relacionar os termos entre si a GO definiu também alguns tipos de relações sendo as mais comuns o "*is a*", "*part of*", "*has part*" e "*regulates*".

Atualmente existem algumas ferramentas que fazem uso desta ontologia. Uma dessas ferramentas já referenciada anteriormente é o GoPubMed. Esta ferramenta tem como principal funcionalidade a pesquisa de resumos de literatura biomédica a partir da base de dados PubMed e a categorização desses resumos de acordo com a GeneOntology. Todos os

termos GO relacionados com a pesquisa são apresentados ao utilizador bem como a sua marcação nos resumos obtidos. É possível também a posterior filtragem de resumos de acordo com os termos GO mais especializados, permitindo desta forma ao utilizador uma melhor análise da informação apresentada.

2.3.2. MEROPS

A Merops é uma base de dados de peptidases e proteínas inibidoras de peptidases. Não sendo considerada como uma ontologia possui ainda assim várias similaridades. Na Figura 2 são apresentadas algumas das tabelas que constituem esta base de dados e aquelas que são mais relevantes para este trabalho.

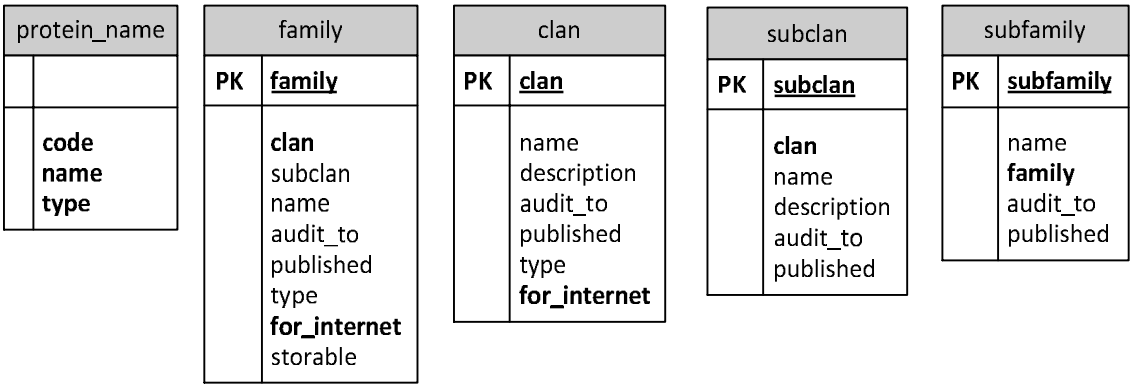


Figura 2 – Tabelas pertencentes à base de dados Merops

A base de dados Merops divide as peptidases em famílias e subfamílias que por sua vez pertencem a clãs e subclãs.

Todas as peptidases presentes na base de dados encontram-se na tabela *protein_name*. Cada peptidase é identificada através de um código, sendo que cada peptidase pode ter diferentes nomes. Para definir se o nome de uma peptidase é o nome principal ou apenas um sinónimo existe um campo *type* que pode ter três estados diferentes: "*real*" quando o nome da peptidase é o nome mais usado, "*alternative*" quando existem outros nomes que representam a mesma peptidase e "*index*" que consiste maioritariamente numa mistura de alguns nomes da peptidase separados por vírgulas. Na Figura 3 é apresentada a peptidase *renin-2* e todos os seus nomes alternativos.

Code	Name	Type
A01.008	renin-2	real
A01.008	Ren2 g.p. ({Mus musculus})	alternative
A01.008	submandibular renin	alternative
A01.008	renin, submandibular	index

Figura 3 – Entradas na base de dados Merops relativas à peptidase *renin-2*

A tabela *family* contém todas as famílias de peptidases presentes na ontologia. Os atributos com maior importância nesta tabela são: *family* com o código Merops que representa uma família de peptidases, *name* que contém o nome atribuído a cada família, *clan* e *subclan* que indicam respectivamente o código do clã e subclã ao qual a família está associada.

A interligação entre as famílias de peptidases e todas as peptidases presentes na tabela *protein_name* é obtida através do código Merops (campo *code* na tabela *protein_name* e campo *family* na tabela *family*). Cada código Merops de uma família é constituído por uma letra e um número. A letra é atribuída de acordo com o tipo catalítico das peptidases pertencentes à família em questão, ou seja, se as peptidases de uma família pertencem ao grupo catalítico das aspárticas o código começa com a letra A, caso pertençam ao grupo das cisteínas, a letra já será a C. Em relação aos números não existe uma razão especifica para a sua atribuição a uma família a não ser a ordem de descoberta das mesmas. A ligação entre famílias e suas peptidases é obtida neste mesmo código uma vez que uma peptidase terá como prefixo o código da família a que pertence seguido de um ponto e um número que identifica a peptidase em questão. Na Figura 3 é possível verificar que a peptidase *renin-2* com código *A01.008* pertence à família *pepsin* com código *A01*.

Em relação à tabela *clan* os seus principais atributos são o campo *clan*, com o código correspondente e o campo *name* com o nome desse clã.

As tabelas *subfamily* e *subclan* para além dos campos com o seu código e nome, contêm também o campo *family* e *clan* respectivamente.

A similaridade entre a base de dados Merops e uma ontologia deve-se à presença dos elementos básicos que constituem qualquer ontologia. Existem classes como é o caso das peptidases, suas famílias ou clãs. Cada classe possui vários atributos como acontece com o código, o nome ou o tipo de nome numa classe de peptidases. Existem instâncias dessas classes que são as próprias peptidases ou famílias de peptidases. Por fim existem também relações entre essas instâncias como é o caso de uma peptidase pertencer a uma família ou uma família pertencer a um clã.

2.3. Algoritmos de classificação

Como visto anteriormente no capitulo 1, são utilizados na plataforma os algoritmos de aprendizagem não supervisionada *K-means, Fuzzy C-means* e *Subtractive* e máquinas de vectores de suporte como aprendizagem supervisionada. De seguida é apresentado o estado de arte relativamente a cada um dos algoritmos usados pela plataforma.

2.3.1. Algoritmo K-means

O algoritmo *K-means*, também conhecido como *Hard C-means*, é um algoritmo que pretende organizar os elementos de um conjunto de dados num determinado número de *clusters* definidos por um utilizador. Estes dados devem ser organizados nos diferentes *clusters* de acordo com a similaridade entre si.

O funcionamento do algoritmo *K-means* pode ser dividido em quatro passos. No primeiro passo é realizada a inicialização dos centros de cada *cluster*. Normalmente estes centros são inicializados escolhendo *k* pontos aleatoriamente do conjunto de dados sendo *k* o número de *clusters* definidos no inicio pelo utilizador.

Depois de inicializados os centros dos *clusters*, é atribuído a cada um dos elementos do conjunto de dados um desses *clusters*. Esta atribuição é realizada com base na menor distância entre um elemento e os centros dos *clusters* definidos, ou seja, o *cluster* ao qual o elemento estiver mais próximo.

No terceiro passo é calculado o valor de uma função de custo dada pela equação (1). Esta função de custo consiste na soma das distâncias de cada elemento ao *cluster* que lhe foi atribuído. Neste caso é usada a distância euclidiana embora seja possível a utilização de outros tipos de distâncias entre vectores.

$$J = \sum_{i=1}^{c} Ji = \sum_{i=1}^{c} \left(\sum_{k, x_k \epsilon G_i} \|x_k - c_i\|^2 \right) \quad (1)$$

Após o calculo da função de custo é verificado se existiu uma melhoria do resultado em relação ao valor calculado na iteração anterior. Caso o valor tenha melhorado, ou seja, tenha diminuído, o algoritmo passa à próxima iteração. Caso este valor tenha aumentado o algoritmo termina a sua execução e retorna os resultados obtidos na iteração anterior.

Em caso do algoritmo não terminar a sua execução no terceiro passo existe um quarto e último passo que consiste na renovação dos centros de todos os clusters recorrendo à equação (2). Esta equação calcula os novos centros de cada cluster através da média entre os elementos desse cluster.

$$c_i = \frac{1}{|G_i|} \sum_{k, x_k \in G_i} x_k \quad (2)$$

Depois de calculados os novos centros o algoritmo volta ao segundo passo e continua a iterar até ser atingido o critério de paragem.

Muitas implementações deste algoritmo contêm também uma variante que consiste na sua repetição, escolhendo no final o melhor resultado entre todas as repetições, ou seja, aquele que tem um menor valor da função de custo. Esta variante foi introduzida devido a dois fatores. O primeiro devido à qualidade do algoritmo *K-means* estar dependente da inicialização dos centros dos clusters, pois caso a inicialização fosse realizada sempre com os mesmos centros o resultado obtido para um conjunto de dados seria sempre igual. Outro fator consiste na possibilidade do algoritmo atingir com bastante facilidade ótimos locais. Desta forma repetindo o algoritmo um determinado número de vezes e usando diferentes centros no inicio do algoritmo a probabilidade de conseguir melhores resultados aumenta.

2.3.2. Algoritmo Fuzzy C-means

O algoritmo *Fuzzy C-means* (*FCM*), tal como o algoritmo *K-means*, procura encontrar os melhores centros de cada *cluster* mediante a minimização de uma função de custo que mede a distância entre os diferentes elementos e os centros dos *clusters* escolhidos. Neste algoritmo, ao contrário do que acontece no algoritmo *K-means*, não é atribuído um *cluster* a cada elemento, mas sim um grau de pertença desse elemento a cada um dos *clusters*. Este grau de

pertença tem sempre um valor entre 0 e 1 sendo que a soma dos valores de um elemento a todos os *clusters* deve ser igual a 1.

O funcionamento do algoritmo *Fuzzy C-means* pode ser também definido em 4 passos, tal como acontece no algoritmo *K-means*. O algoritmo inicia a sua execução com a inicialização de uma matriz que contém o grau de pertença de cada elemento a um *cluster*. Esta inicialização normalmente é realizada de forma aleatória mantendo, no entanto, a condição de o valor da soma de todos os graus de pertença de um elemento aos diferentes *clusters* ser 1.

No segundo passo são calculados os centros dos *clusters* através da equação (3). Cada *cluster* é assim dado pela média de todos os elementos do conjunto de dados e o peso que o grau de pertença de cada elemento do *cluster* confere.

$$c_i = \frac{\sum_{j=1}^{n} u_{ij}^{m} x_j}{\sum_{j=1}^{n} u_{ij}^{m}} \quad (3)$$

Depois de calculados os centros dos *clusters* no passo seguinte é efetuado o cálculo da função de custo. Este cálculo é realizado através da equação (4) onde é realizada a soma das distâncias de cada elemento a cada cluster com o peso do grau de pertença de um elemento a esse cluster associado.

$$J(U, c_1, \ldots, c_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} d_{ij}^{2} \quad (4)$$

Ainda no terceiro passo e depois de calculada a função de custo, existe a verificação das condições de paragem do algoritmo. Estas condições de paragem normalmente são um valor mínimo de melhoria do valor da função de custo após cada iteração ou um número máximo de iterações do algoritmo.

Outra semelhança com o algoritmo *K-means* é o facto dos resultados finais dependerem do seu primeiro passo onde é inicializada a matriz *u* com valores aleatórios. Por esta razão é aconselhável a repetição do algoritmo várias vezes escolhendo de entre essas repetições o melhor resultado, ou seja, aquele que retornar o menor valor da função de custo.

2.3.3. Algoritmo Subtractive Clustering

O algoritmo *Subtractive* tem como característica distinta dos algoritmos *K-means* e *Fuzzy C-means* a capacidade de não necessitar da definição do número de *clusters* no qual um *dataset* se deve dividir.

O algoritmo *Subtractive* é considerado uma extensão do método *Mountain* proposto por Yager e Filev (Yager & Filev, 1994). No método *Mountain*, é criada uma grelha no espaço de dados e é calculado um potencial numérico para cada ponto dessa grelha de acordo com a distância aos dados existentes. Os pontos com mais dados perto de si terão um maior potencial, sendo escolhido como primeiro centro de um *cluster* o ponto com o maior potencial entre todos os pontos da grelha. A partir daqui a cada novo centro encontrado é diminuído o

potencial dos pontos da grelha de acordo com a sua distância a esse novo centro. Quanto maior for a proximidade de um ponto da grelha ao novo centro, maior será a redução do seu potencial. O algoritmo vai terminar quando o potencial de todos os pontos na grelha estiver já abaixo de um determinado limite.

Apesar de ser um método simples e concreto existe uma limitação do algoritmo relativamente ao elevado processamento inerente à avaliação de cada ponto da grelha.

Devido a esta limitação, Chiu (Chiu, 1994) acabou por propor o algoritmo *Subtractive* resolvendo o problema anterior, usando os próprios elementos do conjunto de dados como os pontos candidatos a centros de *clusters*. Com esta nova variante a computação passou a ser proporcional ao tamanho do problema e não à sua dimensão. Contudo os centros dos *clusters* podem não estar localizados num ponto do conjunto de dados, sendo, no entanto uma boa aproximação.

Para além da mudança de pontos candidatos a centros de *clusters* todo o restante funcionamento do algoritmo manteve-se inalterado. Assim para cada ponto x_i do conjunto de dados é calculada a medida de densidade, ou potencial, através da equação (5), onde r_a é uma constante positiva que representa o raio de vizinhança.

$$D_i = \sum_{j=1}^{n} \exp\left(\frac{\left\|x_i - x_j\right\|^2}{\left(\frac{r_a}{2}\right)^2}\right) \quad (5)$$

Após ser calculada a medida de densidade em todos os pontos, escolhe-se o ponto que obteve o maior potencial para ser usado como primeiro centro de um *cluster*. Após ser efetuada esta primeira escolha é recalculada a densidade em todos os pontos segundo a equação (6).

$$D_i = D_i - D_{c_1} \exp\left(\frac{\left\|x_i - x_{c_1}\right\|^2}{\left(\frac{r_a}{2}\right)^2}\right) \quad (6)$$

Com este novo cálculo as medidas de densidade de todos os pontos irão diminuir sendo os mais afetados aqueles que se encontram mais próximos do centro do *cluster*.

A partir daqui o algoritmo vai continuar a iterar até atingir a condição de paragem que neste caso consiste em todos os pontos passarem um valor mínimo limite para um ponto se poder tornar centro de um *cluster*.

2.3.4. Máquinas de Vectores de Suporte

Máquinas de vectores de suporte são uma das técnicas de aprendizagem mais conhecidas e mais usadas atualmente. Foi criada inicialmente por Vapnik em 1963 e tem vindo a ser aperfeiçoada ao longo dos tempos.

Ao contrário dos algoritmos *K-means*, *Fuzzy C-means* e *Subtractive*, as máquinas de vectores de suporte são um método de aprendizagem supervisionado. Como já foi explicado sucintamente no primeiro capitulo, estes métodos de aprendizagem fazem uso de um conjunto

de dados previamente etiquetados para criar um classificador que irá inferir a classe a que um novo elemento pertence.

Mais detalhadamente, as SVMs recebem um conjunto de dados que devem estar devidamente identificados como pertencentes a uma de duas classes. A partir destes dados vai ser criado um modelo, num espaço dimensional, que consiste num hiperplano que vai definir uma divisão entre os elementos de cada classe.

No entanto se repararmos na Figura 4, um conjunto de dados pode ser separado por vários hiperplanos. O que o algoritmo vai procurar como o melhor hiperplano é aquele com a maior distância aos elementos de cada uma das classes que lhe estão mais próximos.

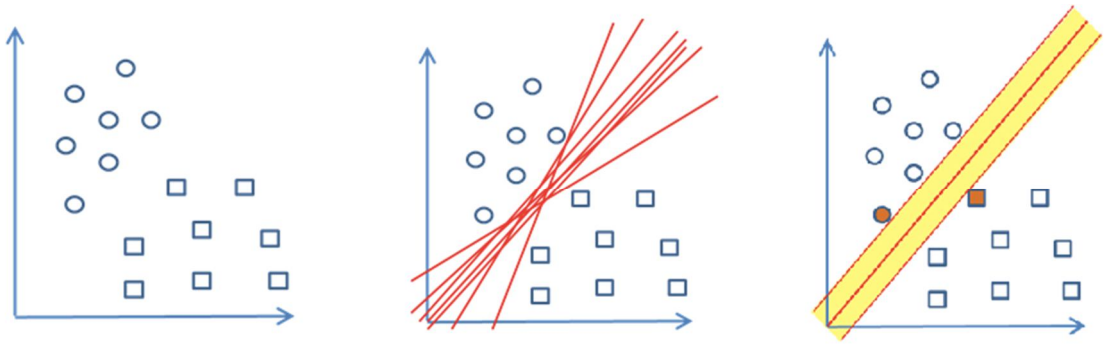


Figura 4 – Exemplo de escolha do melhor hiperplano (Computer Science Source, 2014)

Existem outros casos, como o que se pode verificar na Figura 5, em que um conjunto de treino não é separável linearmente. Para estes casos é possível criar um mapeamento dos elementos de treino para um espaço dimensional superior tornando desta forma o classificador não linear. O mapeamento para espaços dimensionais superiores é feito através de *kernels* sendo os mais conhecidos o polinomial, a função de base radial e o sigmoide.

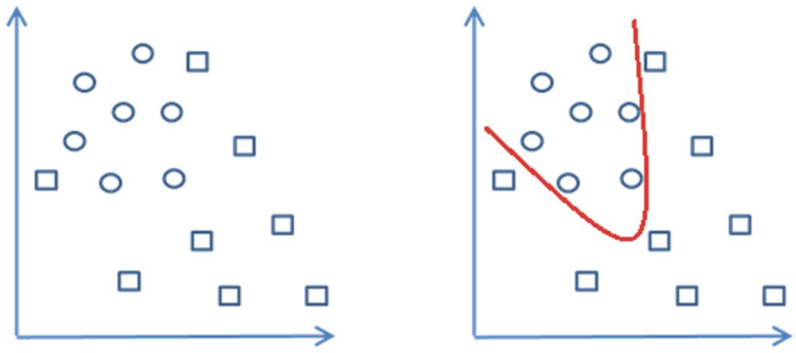


Figura 5 – Exemplo de utilização de kernel para classificadores não lineares (Computer Science Source, 2014)

Como foi visto até ao momento as SVMs foram originalmente desenhadas para classificação binária, tendo sido necessário encontrar soluções para ultrapassar esta limitação. Foram criadas ao longo do tempo técnicas para a utilização de SVMs com multiclasses, sendo as mais conhecidas o um contra todos (*one-against-all*) e o um contra um (*one-against-one*).

O primeiro método proposto para resolver este problema foi o método um contra todos. Neste método é construído um modelo para cada classe usando como exemplos positivos os elementos dessa classe e como elementos negativos todos os elementos das restantes classes. Ao contrário dos classificadores binários vistos anteriormente não será retornado um valor binário que indique se um novo documento pertence ou não à classe em questão. Neste caso é retornado um valor de confiança de o documento em questão pertencer à classe correspondente a esse classificador. Assim na fase de classificação os novos elementos são classificados usando todos os modelos criados sendo considerada no final como a classe pertencente, aquela para a qual a SVM correspondente teve um maior valor de confiança. Na Figura 6 é possível verificar este mesmo processo na classificação de um documento.

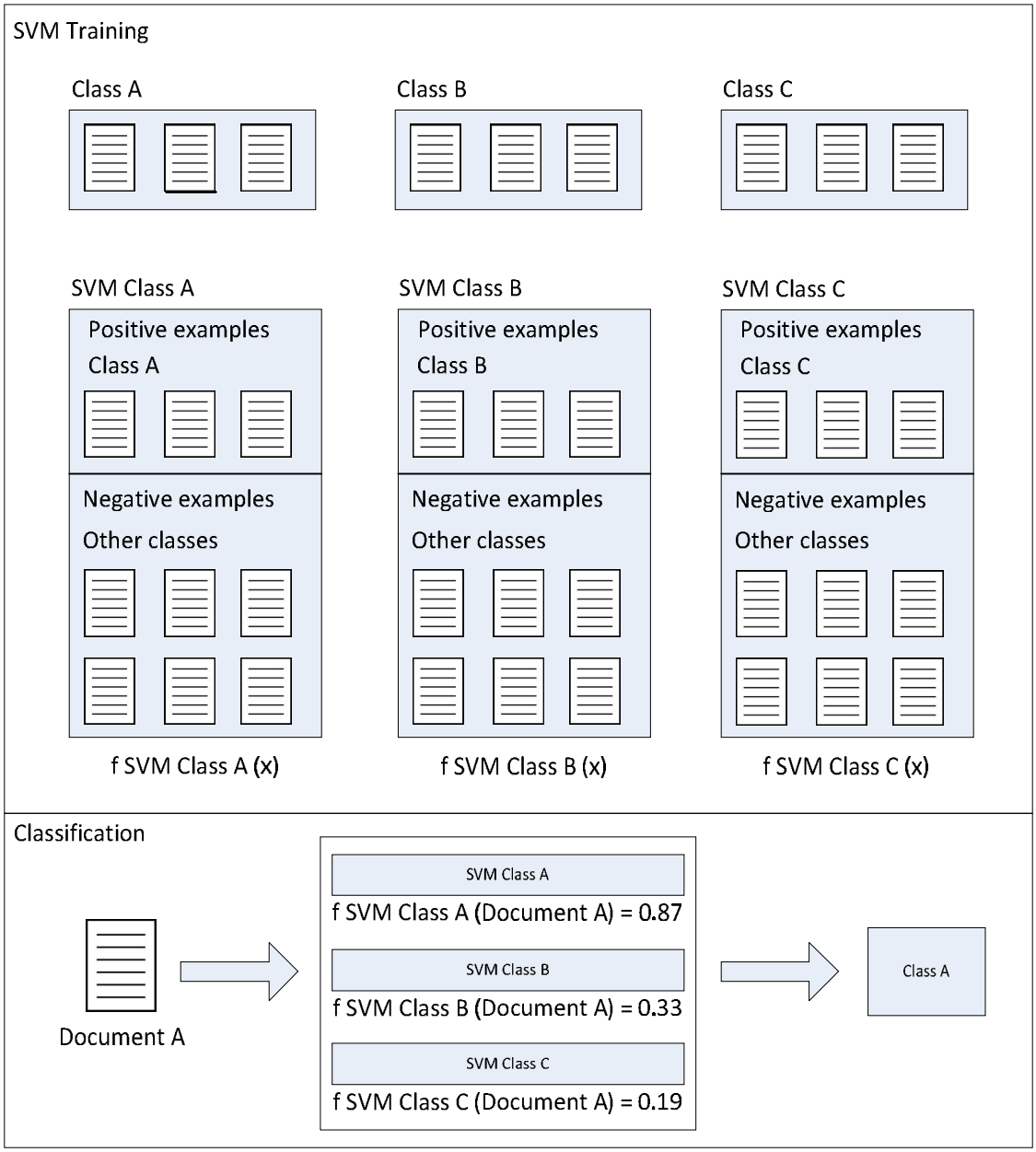


Figura 6 – Exemplo de SVM multiclasse usando a técnica um contra todos.

Uma das vantagens deste método prende-se com a redução do número de classificações binárias para o mesmo número de classes existentes. No entanto existem também desvantagens como é o caso do elevado processamento que a fase de treino requer, devido ao

aumento de amostras usadas nessa fase, e ao não balanceamento dos elementos de cada classe que deverão conter mais elementos negativos do que positivos na fase de treino.

Tendo em conta as desvantagens apontadas ao método um contra todos, foi criado um novo método denominado um contra um. Neste método são realizados vários treinos para cada classe, mais concretamente criando um classificador para cada par de classes possível. Na fase de classificação são usados todos os classificadores e é escolhida como classe pertencente aquela para a qual existiu um maior número de ocorrências. Pela Figura 7 é possível verificar a criação dos diferentes classificadores e a classificação de um documento recorrendo a esta técnica.

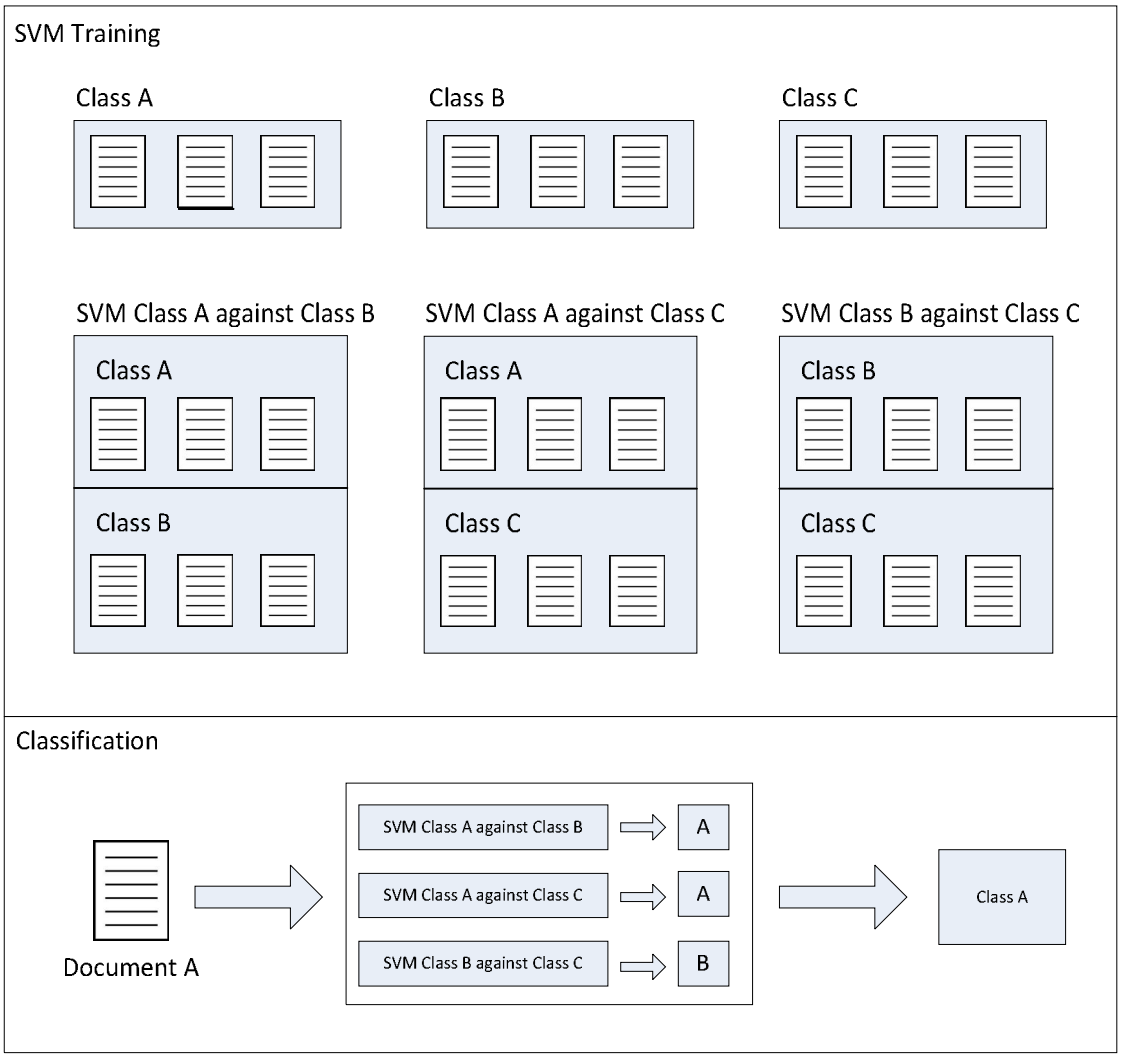


Figura 7 – Exemplo de SVM multiclasse usando a técnica um contra um

Através deste método é possível eliminar as desvantagens do método um contra todos, existindo ainda assim uma desvantagem relativa ao número de classificadores que em caso de um aumento considerável de classes irá também aumentar e criar desta forma um problema de processamento. Mesmo tendo esta desvantagem, segundo um estudo protagonizado por (Hsu & Lin, 2002) o método um contra um é o que proporciona melhores resultados, sendo mesmo o utilizado pela biblioteca LIBSVM que é usada na aplicação.

A biblioteca LIBSVM foi criada em 2001 por Chih-Chung Chang e Chih-Jen Lin e desde essa altura até à data atual tem vindo a ser melhorada. A biblioteca consiste na implementação de máquinas de vectores de suporte que podem ser integradas com facilidade em outras aplicações e conta com uma variedade enorme de *interfaces* e extensões em várias linguagens de programação sendo que nesta aplicação foi usada a extensão de Matlab.

3. ANÁLISE DE REQUISITOS

Nesta secção são apresentados os requisitos da aplicação. Numa primeira instância os requisitos identificados são apresentados através de casos de uso, recorrendo a um diagrama, à descrição detalhada de cada caso de uso e aos atores presentes no sistema. Para complementar os casos de uso e de forma a melhorar a compreensão de situações mais complexas do sistema são apresentados diagramas de atividade e diagramas de sequência. No final, para uma visualização gráfica do que se pretende na plataforma, são apresentados os protótipos de interface e a descrição de algumas das opções tomadas na criação desses protótipos.

3.1. Casos de uso

A presente secção contém os vários atores do sistema e as suas responsabilidades. Também inclui uma lista de casos de uso com os atores que os poderão executar. É ainda apresentado um diagrama de casos de uso e é detalhado cada um dos casos de uso identificados.

3.2.1. Atores do sistema

Na Tabela 1 são apresentados os diferentes atores do sistema. Para os três tipos de utilizadores da aplicação: utilizadores não registados, utilizadores registados e administrador, é descrito o seu papel dentro da aplicação e é dada a conhecer as responsabilidades que lhes são atribuídas.

Tabela 1 – Atores do sistema, descrição dos atores e as suas responsabilidades.

Nome	Descrição	Responsabilidades
Utilizador não registado	Este utilizador poderá usar funcionalidades básicas do sistema.	Este utilizador poderá realizar agrupamento de documentos utilizando vários algoritmos de *clustering*, definindo os vários parâmetros desses algoritmos bem como os parâmetros para extração de conhecimento a partir dos documentos. Poderá também usar o algoritmo LIBSVM com *cross-validation*, definindo os seus parâmetros, e classificar documentos usando máquinas de vectores de suporte já treinadas e fornecidas por defeito pelo sistema.
Utilizador registado	Este utilizador poderá usar funcionalidades mais avançadas do sistema.	Em adição às funcionalidades existentes para um utilizador não registado, este tipo de utilizador pode guardar documentos relevantes, guardar agrupamentos de documentos e guardar máquinas de vectores de suporte treinadas por si. Para além destas funcionalidades é também

		possível classificar documentos usando as SVMs treinadas. Devido à possibilidade de guardar informação o utilizador pode ainda gerir essa mesma informação.
Administrador	Este utilizador irá administrar o sistema	Este utilizador poderá visualizar todos os utilizadores do sistema, validar novos utilizadores, bloquear ou desbloquear utilizadores já validados, remover utilizadores e atualizar os limites de documentos, classificadores ou agrupamentos de documentos guardados por um utilizador. Este utilizador também poderá treinar classificadores que serão usados pelos utilizadores não registrados na classificação de documentos.

3.2.2. Lista de casos de uso

Na Tabela 2 são apresentados os vários casos de uso divididos pelos atores que os poderão executar.

Tabela 2 - Atores do sistema e respectivos casos de usos

Atores	Casos de uso
Utilizador não registado	UC 1 - Agrupar documentos utilizador UC 2 - Agrupar documentos PubMed UC 3 – Usar LIBSVM com *cross-validation* UC 4 - Classificar documentos utilizador usando SVMs UC 5 - Classificar documentos PubMed usando SVMs UC 6 - Definir parâmetros de extração *features* UC 7 - Enviar documentos UC 8 - Definir pesquisa PubMed UC 9 - Definir parâmetros *K-means* UC 10 - Definir parâmetros *Fuzzy C-means* UC 11 - Definir parâmetros *Subtractive* UC 12 - Definir parâmetros LIBSVM UC 13 - Definir parâmetros LIBSVM com *cross-validation*
Utilizador registado	UC 14 – Treinar SVMs UC 15 – Remover SVMs guardadas UC 16 - Guardar agrupamento de documentos UC 17 – Visualizar agrupamento de documentos UC 18 – Remover agrupamento de documentos UC 19 - Guardar documentos UC 20 – Visualizar documentos guardados UC 21 – Remover documentos guardados
Administrador	UC 22 – Validar novos utilizadores UC 23 – Bloquear utilizador UC 24 – Desbloquear utilizador UC 25 – Editar limites utilizador UC 26 – Remover utilizador

3.2.3. Diagrama de casos de uso

Nesta secção, através da Figura 8 é apresentado o diagrama de casos de uso da aplicação.

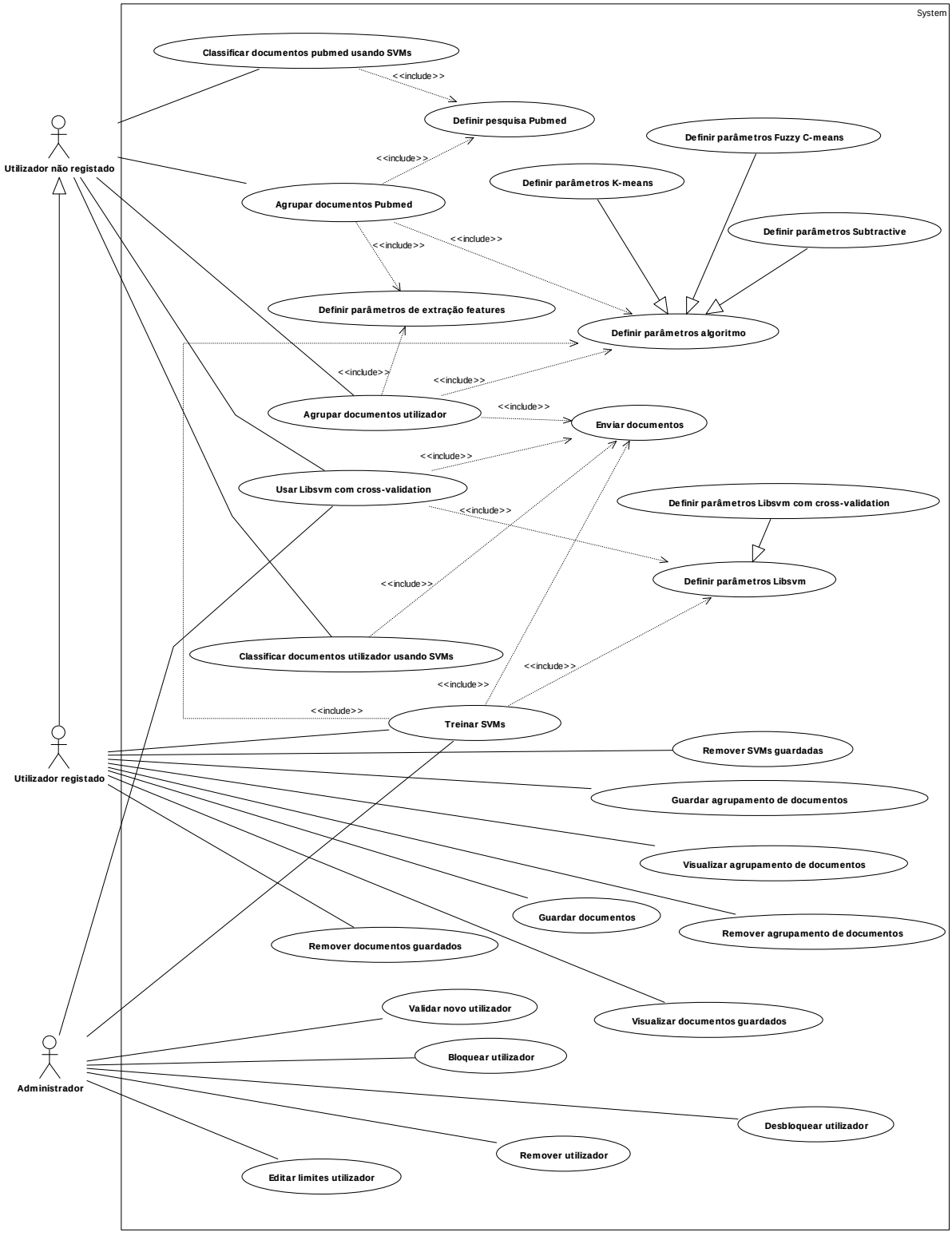


Figura 8 – Diagrama de casos de uso da aplicação DoCluster

3.2.4. Descrição dos casos de uso

De seguida é apresentada a descrição detalhada dos diferentes casos de uso. Na Tabela 3 e Tabela 4 são exibidos os casos de uso relativamente ao *clustering* de documentos enviados

pelo utilizador ou extraídos a partir de uma pesquisa realizada à PubMed. Na Tabela 5, Tabela 6, Tabela 7 e Tabela 16 são descritos os casos de uso relativos ao uso do LIBSVM, sendo que nestes casos de uso se insere o LIBSVM com *cross-validation*, o treino de máquinas de vectores de suporte e a classificação de documentos usando as máquinas de vectores de suporte treinadas. Nos casos de uso anteriores estão incluídos outros casos de uso relacionados com a definição dos parâmetros dos diferentes algoritmos e dos parâmetros para extração de conhecimento a partir dos documentos. A descrição destes casos de uso está presente na Tabela 8, Tabela 9, Tabela 10, Tabela 11, Tabela 12, Tabela 13, Tabela 14 e Tabela 15.

Em relação ao utilizador registado, todas as funcionalidades referentes à gestão da informação que este pode guardar como são o caso de documentos relevantes, *clustering* de documentos e máquinas de vectores de suporte treinadas, estão presentes na Tabela 17, Tabela 18, Tabela 19, Tabela 20, Tabela 21, Tabela 22 e Tabela 23. Por fim ainda são apresentados os casos de uso relativos ao administrador do sistema. Este ator tem as tarefas de validação de novos utilizadores, bloqueio, desbloqueio e remoção desses utilizadores e ainda a possibilidades de edição dos limites de informação que os utilizadores podem guardar. A descrição destes casos de uso é apresentada na Tabela 24, Tabela 25, Tabela 26, Tabela 27 e Tabela 28.

3.2.4.1. UC 1: Agrupar documentos utilizador

Tabela 3 – Caso de uso: Agrupar documentos utilizador

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite agrupar documentos enviados pelo utilizador.
Pré-Requisitos:	Escolher no menu superior a opção *Cluster your own documents*, escolher nas sub-opções o algoritmo a usar, definir os parâmetros para o pré-processamento dos documentos (extração de *features*) e os parâmetros do algoritmo escolhido.
Pós-Condições:	Apresentar os resultados do agrupamento de documentos.
Fluxo Principal:	1. O utilizador agrupa os seus documentos usando o algoritmo *K-means*. 1.1. O utilizador escolhe no menu superior a opção *Cluster your own documents* e a sub-opção *K-means clustering*. 1.2. O sistema devolve uma página com um campo para carregar um ficheiro, as opções de extração de *features* e os parâmetros do algoritmo *K-means*. 1.3. O utilizador envia os seus documentos (UC 7). 1.4. O utilizador define os parâmetros de extração de *features* (UC 6). 1.5. O utilizador define os parâmetros do algoritmo *K-means* (UC 9). 1.6. O utilizador carrega no botão *Run K-means Algorithm* para que o algoritmo seja executado. 1.7. O sistema executa o algoritmo e apresenta os resultados do agrupamento de documentos, bem como as opções utilizadas.
Fluxo Alternativo:	2. O utilizador agrupa os seus documentos usando o algoritmo *Fuzzy C-means*. 2.1. O utilizador escolhe no menu superior a opção *Cluster your own documents* e a sub-opção *Fuzzy C-means clustering*. 2.2. O sistema devolve uma página com um campo para carregar um ficheiro, as opções de extração de *features* e os parâmetros do algoritmo *Fuzzy C-means*. 2.3. Volta para o passo 1.3. 2.4. O utilizador define os parâmetros do algoritmo *Fuzzy C-means* (UC 10) (a seguir ao passo 1.4). 2.5. O utilizador carrega no botão *Run Fuzzy C-means Algorithm*. 2.6. Volta para o passo 1.7.

	3. O utilizador agrupa os seus documentos usando o algoritmo *Subtractive*. 3.1. O utilizador escolhe no menu superior a opção *Cluster your own documents* e a sub-opção *Subtractive clustering*. 3.2. O sistema devolve uma página com um campo para carregar um ficheiro, as opções de extração de *features* e os parâmetros do algoritmo *Subtractive*. 3.3. Volta para o passo 1.3. 3.4. O utilizador define os parâmetros do algoritmo *Subtractive* (UC 11) (a seguir ao passo 1.4). 3.5. O utilizador carrega no botão *Run Subtractive Algorithm*. 3.6. Volta para o passo 1.7.
Inclusões:	Enviar documentos (UC 7), Definir parâmetros de extração *features* (UC 6), Definir parâmetros *K-means* (UC 9), Definir parâmetros *Fuzzy C-means* (UC 10), Definir parâmetros *Subtractive* (UC 11).

3.2.4.2. UC 2: Agrupar documentos PubMed

Tabela 4 – Caso de uso: Agrupar documentos PubMed

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite agrupar documentos da PubMed.
Pré-Requisitos:	Escolher no menu a opção de *Cluster PubMed documents*, escolher o algoritmo a usar, definir os parâmetros de pesquisa de documentos na PubMed, definir os parâmetros para pré-processamento dos documentos (extração de *features*) e os parâmetros do algoritmo de agrupamento escolhido.
Pós-Condições:	Apresentar os resultados do agrupamento de documentos PubMed.
Fluxo Principal:	1. O utilizador irá agrupar os documentos PubMed usando o algoritmo *K-means*. 1.1. O utilizador escolhe a opção *Cluster PubMed documents* e a sub-opção *K-means clustering* no menu superior. 1.2. O sistema devolve uma página com um formulário de opções para pesquisa de documentos na PubMed, com as opções de extração de *features* e com os parâmetros do algoritmo *K-means*. 1.3. O uitlizador define a pesquisa para o PubMed (UC 8). 1.4. O utilizador define os parâmetros de extração de *features* (UC 6). 1.5. O utilizador define os parâmetros do algoritmo *K-means* (UC 9). 1.6. O utilizador carrega no botão *Run K-means Ãlgorithm*. 1.7. O sistema executa o algoritmo e apresenta os resultados do agrupamento de documentos, bem como as opções utilizadas.
Fluxo Alternativo:	2. O utilizador irá agrupar os documentos usando o algoritmo *Fuzzy C-Means.* 2.1. O utilizador escolhe a opção *Cluster PubMed documents* e a sub-opção *Fuzzy C-means clustering* no menu superior. 2.2. O sistema devolve uma página com um formulário de opções para pesquisa de documentos na PubMed com as opções de extração de *features* e com os parâmetros do algoritmo *Fuzzy C-means.* 2.3. Volta para o passo 1.3. 2.4. O utilizador define os parâmetros do algoritmo *Fuzzy C-means* (UC 10) (a seguir ao passo 1.4). 2.5. O utilizador carrega no botão *Run Fuzzy C-means Algorithm.* 2.6. Volta para o passo 1.7. 3. O utilizador irá agrupar os documentos usando o algoritmo *Subtractive*. 3.1. O utilizador escolhe a opção *Cluster PubMed documents* e a sub-opção *Subtractive clustering* no menu superior. 3.2. O sistema devolve uma página com um formulário de opções para pesquisa de documentos na PubMed, com as opções de extração de *features* e com os parâmetros do algoritmo *Subtractive.*

	3.3. Volta para o passo 1.3. 3.4. O utilizador define os parâmetros do algoritmo *Subtractive* (UC 11) (a seguir ao passo 1.4). 3.5. O utilizador carrega no botão *Run Subtractive Algorithm* 3.6. Volta para o passo 1.7.
Inclusões:	Definir pesquisa PubMed (UC 8), Definir parâmetros de extração *features* (UC 6), Definir parâmetros *K-means* (UC 9), Definir parâmetros *Fuzzy C-means* (UC 10), Definir parâmetros *Subtractive* (UC 11).

3.2.4.3. UC 3: Usar LIBSVM com *cross-validation*

Tabela 5 – Caso de uso: Usar LIBSVM com *cross-validation*

Atores:	Utilizador não registado, Utilizador registado e Administrador.
Descrição:	Permite usar o algoritmo LIBSVM e verificar a qualidade dos resultados obtidos.
Pré-Requisitos:	Escolher no menu a opção *LIBSVM with cross-validation*, definir os parâmetros de pré-processamento de documentos (extração de *features*) e os parâmetros do LIBSVM.
Pós-Condições:	Apresentar os resultados do LIBSVM.
Fluxo Principal:	1. O utilizador irá usar o algoritmo LIBSVM com *cross-validation*. 1.1. O utilizador escolhe a opção *Support Vector Machines* e sub-opção *LIBSVM with cross-validation* no menu superior. 1.2. O sistema devolve uma página com um campo para carregar um ficheiro, com as opções de extração de *features* e os parâmetros do algoritmo LIBSVM. 1.3. O utilizador envia os seus documentos (UC 7). 1.4. O utilizador define os parâmetros de extração de *features* (UC 6). 1.5. O utilizador define os parâmetros para o algoritmo LIBSVM com *cross-validation* (UC 13). 1.6. O utilizador carrega no botão *Run LIBSVM with cross-validation* 1.7. O sistema executa o algoritmo e apresenta os resultados ao utilizador, bem como as opções utilizadas.
Fluxo Alternativo:	Não existem fluxos alternativos
Inclusões:	Enviar documentos (UC 7), Definir parâmetros de extração *features* (UC 5), Definir parâmetros LIBSVM com *cross-validation* (UC 13).

3.2.4.4. UC 4: Classificar documentos utilizador usando SVMs

Tabela 6 – Caso de uso: Classificar documentos utilizador usando SVMs

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite classificar documentos enviados pelo utilizador.
Pré-Requisitos:	Escolher no menu a opção *Support Vector Machines* e sub-opção *Classify your documents*, enviar documentos e escolher o modelo a usar na classificação.
Pós-Condições:	Apresentar os documentos classificados
Fluxo Principal:	2. O utilizador irá classificar os seus documentos escolhendo um modelo anteriormente treinado. 2.1. O utilizador escolhe a opção *Support Vector Machines* e sub-opção *Classify your documents* no menu superior. 2.2. O sistema devolve uma página com um campo para carregar um ficheiro e uma lista com modelos já treinados para serem usados na classificação. 2.3. O utilizador envia os seus documentos (UC 7). 2.4. O utilizador escolhe um modelo para ser usado na classificação. 2.5. O utilizador carrega no botão *Classify documents*. 2.6. O sistema executa o algoritmo e apresenta ao utilizador os documentos

	classificados.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Enviar documentos (UC 7).

3.2.4.5. UC 5: Classificar documentos PubMed usando SVMs

Tabela 7 – Caso de uso: Classificar documentos PubMed usando SVMs

Atores:	Utilizador não registado e Utilizador registado
Descrição:	Permite classificar documentos da PubMed.
Pré-Requisitos:	Escolher no menu a opção *Support Vector Machines* e sub-opção *Classify PubMed documents*, definir os parâmetros de pesquisa de documentos na PubMed e escolher o modelo a usar na classificação.
Pós-Condições:	Apresentar os documentos classificados.
Fluxo Principal:	3. O utilizador irá classificar documentos vindos da PubMed escolhendo um modelo anteriormente treinado. 3.1. O utilizador escolhe a opção *Support Vector Machines* e sub-opção *Classify PubMed documents* no menu superior. 3.2. O sistema devolve uma página com um formulário de opções para pesquisa de documentos na PubMed e uma lista de modelos já treinados para serem usados na classificação. 3.3. O utilizador define a pesquisa para o PubMed (UC 8). 3.4. O utilizador escolhe um modelo para ser usado na classificação. 3.5. O utilizador carrega no botão *Classify documents*. 3.6. O sistema executa o algoritmo e apresenta ao utilizador os documentos classificados.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Definir pesquisa PubMed (UC 8).

3.2.4.6. UC 106: Definir parâmetros de extração de *features*

Tabela 8 – Caso de uso: Definir parâmetros de extração de *features*

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros para a extração de *features*.
Pré-Requisitos:	Escolher qualquer uma das sub-opções de *Cluster your own documents* ou *Cluster PubMed documents* presentes no menu principal ou as sub-opções *Train SVMs* e *LIBSVM with cross-validation* relativas à opção *Support vector machines* também presente no menu principal.
Pós-Condições:	Opções de extração de *features* definidas e validadas.
Fluxo Principal:	1. O utilizador define os parâmetros de extração de *features* sem recorrer a ontologias. 1.1. O utilizador escolhe o valor *NoOntology* na primeira opção do formulário. 1.2. O utilizador define os seguintes parâmetros: *Prune below*, *Prune above*, *Pruning frequence*, *TermNGramGenerator*, *Minimum lenght of words*, *Vector creation type*, *Tokenizer type* e *Stemmer type.*
Fluxo Alternativo:	2. O utilizador define os parâmetros de extração de *features* recorrendo à ontologia *GeneOntology*. 2.1. O utilizador escolhe a opção *GeneOntology* na primeira opção do formulário. 2.2. O sistema bloqueia os parâmetros *Tokenizer type* e *Stemmer type* colocando por defeito os valores *DummyTokenizer* e *DummyStemmer* respectivamente. 2.3. O utilizador define os seguintes parâmetros: *Prune below*, *Prune above*,

	Pruning frequence, *TermNGramGenerator*, *Minimum lenght of words* e *Vector creation type*. 3. O utilizador define os parâmetros de extração de *features* recorrendo à ontologia Merops: 3.1. O utilizador escolhe o valor *Merops* na primeira opção do formulário. 3.2. O sistema bloqueia os parâmetros *Tokenizer type* e *Stemmer type* colocando por defeito os valores *DummyTokenizer* e *DummyStemmer* respectivamente. 3.3. O sistema bloqueia os parâmetros *Prune below*, *Prune above*, *Pruning frequence*, *TermNGramGenerator* e *Minimum lenght of words*. 3.4. O utilizador define o parâmetro *Vector creation type*.
Inclusões:	Não existem inclusões.

3.2.4.7. UC 7: Enviar documentos

Tabela 9 – Caso de uso: Enviar documentos

Atores:	Utilizador não registado e Utilizador Registado.
Descrição:	Permite enviar um ficheiro, em formato zip ou rar, com documentos de texto.
Pré-Requisitos:	Escolher qualquer uma das sub-opções de *Cluster your own documents* no menu principal ou as sub-opções *LIBSVM with cross-validation*, *Train SVMs* e *Classify your documents* relativas à opção *Support vector machine*.
Pós-Condições:	Ficheiro enviado devidamente validado.
Fluxo Principal:	1. O utilizador envia documentos para serem classificados através de algoritmos de *clustering* e de SVMs treinadas. 1.1. O utilizador deve carregar no botão Escolher ficheiro 1.2. O sistema deve abrir uma janela para o utilizador procurar o ficheiro. 1.3. O utilizador deve escolher um ficheiro, com formato zip ou rar, contendo o conjunto de documentos no formato de texto.
Fluxo Alternativo:	2. O utilizador envia documentos para treinar SVMs com o LIBSVM e para usar o LIBSVM com *cross-validation* (a seguir ao passo 1.2.). 2.1. O utilizador deve escolher um ficheiro, com o formato zip ou rar, contendo ficheiros em formato txt divididos por pastas que vão corresponder a cada classe.
Inclusões:	Não existem inclusões.

3.2.4.8. UC 8: Definir pesquisa PubMed

Tabela 10 – Caso de uso: Definir pesquisa PubMed

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros de pesquisa para obter documentos da PubMed.
Pré-requisitos:	Escolher a opção de agrupar ou classificar documentos da PubMed.
Pós-Condições:	Obter um conjunto de documentos através do *Web service* da PubMed.
Fluxo Principal:	1. O utilizador define os parâmetros para a obtenção de documentos a partir da PubMed. 1.1. O utilizador insere o termo de pesquisa a enviar ao *Web service* da PubMed. 1.2. O utilizador escolhe os campos nos quais quer que a pesquisa incida: *All Fields*, *Title*, *Title and Abstract*, *Authors* ou *Journal Name*. 1.3. O utilizador define o número máximo de documentos pretendidos. 1.4. O utilizador define a ordem pela qual pretende que os documentos sejam retornados pelo *Web service* da PubMed: *Recently Added*, *Publication Date*, *Author* ou *Journal*.
Fluxo Alternativo:	Não existem fluxos alternativos.

Inclusões:	Não existem inclusões.

3.2.4.9. UC 9: Definir parâmetros *K-means*

Tabela 11 – Caso de uso: Definir parâmetros *K-means*

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros do algoritmo *K-means*.
Pré-Requisitos:	Escolher no menu principal as opções *Cluster your own documents* ou *Cluster PubMed documents* e a sub-opção *K-means clustering.*
Pós-Condições:	Opções definidas e validadas.
Fluxo Principal:	1. O utilizador define os parâmetros para o algoritmo *K-means.* 1.1. O utilizador define o número de *clusters.* 1.2. O utilizador define o número de vezes que o algoritmo vai ser executado. 1.3. O utilizador escolhe o tipo de distância que o algoritmo vai usar para medir a distância entre documentos e centros dos *clusters*: *Squared Eucledian*, *City Block*, *Cosine* ou *Correlation.* 1.4. O utilizador escolhe a forma de inicialização dos clusters: *Sample*, *Uniform* ou *Cluster*.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.10. UC 10: Definir parâmetros *Fuzzy C-means*

Tabela 12 – Caso de uso: Definir parâmetros *Fuzzy C-means*

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros do algoritmo *Fuzzy C-means*.
Pré-Requisitos:	Escolher a opção *Fuzzy C-means* no sub-menu das opções *Cluster your documents* e *Cluster PubMed documents.*
Pós-Condições:	Opções definidas e validadas.
Fluxo Principal:	2. O utilizador define os parâmetros para o algoritmo *Fuzzy C-means*. 2.1. O utilizador define o número de *clusters*. 2.2. O utilizador define o parâmetro *Exponent.* 2.3. O utilizador define o número máximo de iterações do algoritmo. 2.4. O utilizador define o mínimo de melhoria que tem de existir em cada iteração para o algoritmo não terminar.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.11. UC 11: Definir parâmetros *Subtractive*

Tabela 13 – Caso de uso: Definir parâmetros *Subtractive*

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros do algoritmo Subtractive.
Pré-Requisitos:	Escolher a opção *Subtractive* no sub-menu das opções *Cluster your documents* e *Cluster PubMed documents.*
Pós-Condições:	Opções definidas e validadas.
Fluxo Principal:	3. O utilizador define os parâmetros para o algoritmo *Subtractive.* 3.1. O utilizador define o parâmetro *Radii*. 3.2. O utilizador define o parâmetro *QuashFactor*. 3.3. O utilizador define o parâmetro *AcceptRatio*.

	3.4. O utilizador define o parâmetro *RejectRation*.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.12. UC 12: Definir Parâmetros LIBSVM

Tabela 14 – Caso de uso: Definir Parâmetros LIBSVM

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros do algoritmo LIBSVM.
Pré-Requisitos:	Escolher uma das opções *LIBSVM with cross-validation* ou *Train SMVs* no sub-menu das opções *Support Vector Machines*.
Pós-Condições:	Opções definidas e validadas.
Fluxo Principal:	1. O utilizador define os parâmetros LIBSVM usando o *kernel Linear*. 1.1. O utilizador escolhe o *kernel Linear*. 1.2. O sistema bloqueia os parâmetros *Gamma*, *Degree* e *Coef0*. 1.3. O utilizador define o parâmetro *Cost*. 1.4. O utilizador define o parâmetro *Cache Size*.
Fluxo Alternativo:	2. O utilizador define os parâmetros LIBSVM usando o *kernel Polynomial* 2.1. O utilizador escolhe o *kernel Polynomial*. 2.2. O utilizador define o parâmetro *Cost*. 2.3. O utilizador define o parâmetro *Gamma*. 2.4. O utilizador define o parâmetro *Degree*. 2.5. O utilizador define o parâmetro *Coef0*. 2.6. O utilizador define o parâmetro *Cache Size*. 3. O utilizador define os parâmetros LIBSVM usando o *kernel Radial Basis Function*. 3.1. O utilizador escolhe o *kernel Radial Basis Function*. 3.2. O sistema bloqueia os parâmetros *Degree* e *Coef0*. 3.3. O utilizador define o parâmetro *Cost*. 3.4. O utilizador define o parâmetro *Gamma*. 3.5. O utilizador define o parâmetro *Cache Size*. 4. O utilizador define os parâmetros LIBSVM usando o *kernel Sigmoid*. 4.1. O utilizador escolhe o *kernel Sigmoid*. 4.2. O sistema bloqueia os parâmetros *Degree* e *Coef0*. 4.3. O utilizador define o parâmetro *Cost*. 4.4. O utilizador define o parâmetro *Gamma*. 4.5. O utilizador define o parâmetro *Cache Size*.
Inclusões:	Não existem inclusões.

3.2.4.13. UC 13: Definir parâmetros LIBSVM com *cross-validation*

Tabela 15 – Caso de uso: Definir parâmetros LIBSVM com *cross-validation*

Atores:	Utilizador não registado e Utilizador registado.
Descrição:	Permite definir parâmetros para o algoritmo LIBSVM com *cross-validation*.
Pré-Requisitos:	Escolher a opção *LIBSVM with cross-validation* no sub-menu da opção *Support Vector Machines*.
Pós-Condições:	Opções definidas e validadas.
Fluxo Principal:	1. O utilizador define os parâmetros para usar o algoritmo LIBSVM com *cross-validation*. 1.1. O utilizador define os parâmetros LIBSVM (UC 12). 1.2. O utilizador define o parâmetro *Folds number for cross validation*.

Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Definir parâmetros LIBSVM (UC 12).

3.2.4.14. UC 14: Treinar SVMs

Tabela 16 – Caso de uso: Treinar SVMs

Atores:	Utilizador registado e Administrador.
Descrição:	Permite treinar um classificador através do algoritmo LIBSVM.
Pré-Requisitos:	Escolher a opção *Train SVMs* no sub-menu da opção *Support Vector Machines*.
Pós-Condições:	Classificador guardado na base de dados.
Fluxo Principal:	1. O utilizador treina um classificador usando o algoritmo LIBSVM. 1.1. O utilizador escolhe a opção *Support Vector Machines* e a sub-opção *Train SVMs* no menu superior. 1.2. O sistema devolve uma página com um campo para carregar um ficheiro, com as opções de extração de *features*, os parâmetros do algoritmo LIBSVM e as opções de identificação do treino. 1.3. O utilizador envia os seus documentos (UC 7). 1.4. O utilizador define os parâmetros de extração de *features* (UC 6). 1.5. O utilizador define os parâmetros LIBSVM (UC 12). 1.6. O utilizador define o nome para o treino. 1.7. O utilizador define a descrição para o treino. 1.8. O utilizador carrega no botão *Train SVMs*. 1.9. O sistema executa o algoritmo. 1.10. O sistema guarda o classificador na base de dados. 1.11. O sistema notifica o utilizador que guardou o classificador.
Fluxo Alternativo:	2. O utilizador treina SVMs aproveitando o LIBSVM com *cross-validation*. 2.1. O utilizador usa o LIBSVM com *cross-validation* (UC 3). 2.2. O utilizador carrega no botão Train SVMs presente no final dos resultados apresentados. 2.3. O sistema devolve uma página com um campo para carregar um ficheiro, com as opções de extração de *features* e os parâmetros do algoritmo LIBSVM já preenchidas com os valores usados no LIBSVM com *cross-validation* e as opções de identificação do treino. 2.4. O utilizador envia os seus documentos (UC 7). 2.5. Voltar para o passo 1.6.
Inclusões:	Enviar documentos (UC 7), Definir parâmetros de extração de *features* (UC 6), Definir parâmetros LIBSVM (UC 12), Usar LIBSVM com *cross-validation* (UC 3)

3.2.4.15. UC 15: Remover SVMs guardadas

Tabela 17 – Caso de uso: Remover SVMs guardadas

Atores:	Utilizador registado e Administrador.
Descrição:	Permite remover SVMs guardadas para classificação de documentos.
Pré-Requisitos:	Existirem SVMs guardadas e aceder no menu superior à sub-opção *Trained SVMs* da opção *Manage your account*, no caso do utilizador registado ou *Manage System*, no caso do Administrador.
Pós-Condições:	SVMs removidas da base de dados.
Fluxo Principal:	1. O utilizador remove uma SVM treinada anteriormente. 1.1. O utilizador escolhe no menu superior a opção *Manage your account* e a sub-opção *Trained SVMs*. 1.2. O sistema apresenta uma lista com as SVMs treinadas. 1.3. O utilizador procura a SVM treinada a remover.

	1.4. O utilizador carrega no botão *Delete* presente no bloco com a informação resumida dessa SVM. 1.5. O sistema apresenta uma caixa a questionar se o utilizador realmente quer remover a SVM. 1.6. O utilizador carrega no botão *Confirm.* 1.7. O sistema notifica o utilizador que removeu a SVM.
Fluxo Alternativo:	2. O administrador remove uma SVM treinada anteriormente. 2.1. O administrador escolhe no menu superior a opção *Manage system* e a sub-opção *Trained SVMs*. 2.2. Voltar ao passo 1.2.
Inclusões:	Não existem inclusões.

3.2.4.16. UC 16: Guardar agrupamento de documentos

Tabela 18 – Caso de uso: Guardar agrupamento de documentos

Atores:	Utilizador registado.
Descrição:	Permite guardar os resultados de um *clustering*
Pré-Requisitos:	Realizar o agrupamento de documentos e estar na página de resultados.
Pós-Condições:	Resultados do agrupamento de documentos guardados na base de dados.
Fluxo Principal:	1. O utilizador guarda os resultados obtidos após a execução de um dos algoritmos de *clustering.* 1.1. O utilizador realiza o agrupamento dos seus documentos (UC 1) ou o agrupamento de documentos da PubMed (UC 2) . 1.2. O utilizador carrega no botão *Save clustering* presente na barra lateral do lado direito. 1.3. O sistema notifica o utilizador que guardou os resultados do *clustering.*
Fluxo Alternativo:	Não existem fluxos alternativos
Inclusões:	Agrupar documentos utilizador (UC 1), Agrupar documentos PubMed (UC 2).

3.2.4.17. UC 17: Visualizar agrupamento de documentos

Tabela 19 – Caso de uso: Visualizar agrupamento de documentos

Atores:	Utilizador registado
Descrição:	Permite visualizar os resultados de um *clustering* guardado.
Pré-Requisitos:	Existir *clusterings* de documentos guardados.
Pós-Condições:	Resultados de um agrupamento de documentos.
Fluxo Principal:	1. O utilizador visualiza os resultados de um *clustering* de documentos guardado. 1.1. O utilizador escolhe no menu superior a opção *Manage your account* e a sub-opção *Clustered documents*. 1.2. O sistema apresenta uma lista com a informação resumida dos agrupamentos guardados. 1.3. O utilizador escolhe o agrupamento que pretende visualizar e carrega no bloco com essa informação. 1.4. O sistema redireciona para uma página de resultados detalhada.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.18. UC 18: Remover agrupamento de documentos

Tabela 20 – Caso de uso: Remover agrupamento de documentos

Atores:	Utilizador registado

Descrição:	Permite remover *clustering* de documentos guardados.
Pré-Requisitos:	Existir *clustering* de documentos guardados.
Pós-Condições:	Remoção da base de dados de toda a informação de um agrupamento de documentos.
Fluxo Principal:	1. O utilizador remove um *clustering* de documentos guardado. 1.1. O utilizador escolhe no menu superior a opção *Manage your account* e a sub-opção *Clustered documents*. 1.2. O sistema apresenta uma lista com a informação resumida dos agrupamentos guardados. 1.3. O utilizador escolhe o agrupamento que pretende remover e carrega no botão *Delete* presente no bloco com essa informação. 1.4. O sistema apresenta uma caixa a questionar se o utilizador realmente quer remover o agrupamento de documentos. 1.5. O utilizador carrega no botão *Confirm.* 1.6. O sistema notifica o utilizador que toda a informação relativa ao *clustering* foi removida.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.19. UC 19: Guardar documentos

Tabela 21 – Caso de uso: Guardar documentos

Atores:	Utilizador registado.
Descrição:	Permite guardar documentos.
Pré-Requisitos:	Estar numa página com toda a informação relativa a um documento.
Pós-Condições:	Documento guardado na base de dados.
Fluxo Principal:	1. O utilizador guarda um documento na base de dados. 1.1. O utilizador realiza o agrupamento dos seus documentos (UC 1), agrupamento de documentos da PubMed (UC 2), classificação dos seus documentos (UC 4) ou classificação de documentos da PubMed (UC5). 1.2. Na lista de documentos o utilizador carrega no bloco com a informação resumida do documento a guardar. 1.3. O sistema apresenta uma caixa com a informação detalhada do documento. 1.4. O utilizador carrega no botão Save. 1.5. O sistema guarda o documento na base de dados. 1.6. O sistema notifica o utilizador que guardou o documento.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Agrupamento documentos utilizador (UC 1), Agrupamento documentos PubMed (UC2), Classificar documentos utilizador usando SVMs (UC 4), Classificar documentos PubMed usando SVMs (UC 5).

3.2.4.20. UC 20: Visualizar documentos guardados

Tabela 22 – Caso de uso: Visualizar documentos guardados

Atores:	Utilizador registado.
Descrição:	Permite visualizar documentos guardados.
Pré-Requisitos:	Utilizador possuir documentos guardados.
Pós-Condições:	Informação detalhada de um documento apresentada ao utilizador.
Fluxo Principal:	1. O utilizador visualiza um documento guardado. 1.1. O utilizador escolhe no menu superior a opção *Manage your account* e sub-opção *Documents*.

	1.2. O sistema apresenta uma lista com todos os documentos guardados pelo utilizador. 1.3. Na lista de documentos o utilizador carrega no bloco com a informação resumida do documento a visualizar. 1.4. O sistema apresenta uma caixa com a informação detalhada do documento.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.21. UC 21: Remover documentos guardados

Tabela 23 – Caso de uso: Remover documentos guardados

Atores:	Utilizador registado.
Descrição:	Permite remover documentos guardados.
Pré-Requisitos:	Utilizador possuir documentos guardados.
Pós-Condições:	Documento removido da base de dados.
Fluxo Principal:	1. O utilizador remove um documento guardado. 1.1. O utilizador escolhe no menu superior a opção *Manage your account* e sub-opção *Documents.* 1.2. O sistema apresenta uma lista com todos os documentos guardados pelo utilizador. 1.3. Na lista de documentos o utilizador carrega no botão *Delete* presente no bloco com a informação resumida do documento a remover. 1.4. O sistema apresenta uma caixa a questionar se o utilizador realmente quer remover o documento. 1.5. O utilizador carrega no botão *Confirm.* 1.6. O sistema notifica o utilizador que o documento foi removido.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2.4.22. UC 22: Validar novos utilizadores

Tabela 24 – Caso de uso: Validar novos utilizadores

Atores:	Administrador.
Descrição:	Permite validar novos utilizadores registados.
Pré-Requisitos:	Existir utilizadores não validados.
Pós-Condições:	Utilizadores validados.
Fluxo Principal:	1. O administrador valida um novo utilizador a partir da lista de utilizadores. 1.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 1.2. O administrador carrega na opção *Waiting* do menu *Filter* presente na barra lateral do lado esquerdo. 1.3. O sistema devolve a lista de utilizadores à espera de validação. 1.4. O utilizador carrega no botão *Approve* presente no bloco com a informação resumida do utilizador que pretende validar. 1.5. O sistema notifica o administrador que validou o utilizador.
Fluxo Alternativo:	2. O administrador valida um novo utilizador a partir de uma caixa com a informação detalhada do utilizador. 2.1. Passo 1.1. 2.2. O administrador carrega no bloco com a informação resumida do utilizador que pretende validar (a seguir ao passo 1.3.). 2.3. O sistema apresenta uma caixa com a informação detalhada do utilizador. 2.4. O administrador carrega no botão *Approve*.

	2.5. O sistema notifica o administrador que validou o utilizador.
Inclusões:	Não existem inclusões.

3.2.4.23. UC 23: Bloquear utilizador

Tabela 25 – Caso de uso: Bloquear utilizador

Atores:	Administrador.
Descrição:	Permite bloquear utilizadores já validados.
Pré-Requisitos:	Existirem utilizadores validados.
Pós-Condições:	Utilizador bloqueado.
Fluxo Principal:	1. O administrador bloqueia um utilizador pesquisando pelo seu nome ou email. 1.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 1.2. O administrador realiza uma pesquisa a um utilizador pelo seu *username* ou email carregando nos botões correspondentes, que estão presentes no menu *Filter* que se encontra na barra lateral do lado direito. 1.3. O administrador insere na caixa de pesquisa o username ou email do utilizador pretendido. 1.4. O administrador carrega no botão de pesquisa. 1.5. O sistema devolve o utilizador pesquisado. 1.6. O administrador carrega no botão *Block* presente no bloco com a informação resumida do utilizador. 1.7. O sistema notifica o administrador que bloqueou o utilizador.
Fluxo Alternativo:	2. O administrador bloqueia um utilizador filtrando os utilizadores validados. 2.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 2.2. O administrador carrega na opção *Approved* do menu *Filter* presente na barra lateral do lado esquerdo. 2.3. O sistema devolve a lista de utilizadores aprovados. 2.4. O administrador procura o utilizador que pretende bloquear na lista de utilizadores e carrega no botão *Block* presente no bloco com a informação resumida desse utilizador. 2.5. O sistema notifica o administrador que bloqueou o utilizador. 3. O administrador bloqueia um utilizador a partir de uma caixa com a sua informação detalhada. 3.1. Ir para o passo 1.1. ou para o passo 2.1. 3.2. O administrador carrega no bloco com a informação resumida do utilizador que pretende bloquear (a seguir ao passo 1.5. ou passo 2.3.). 3.3. O sistema apresenta uma caixa com a informação detalhada do utilizador. 3.4. O administrador carrega no botão *Block*. 3.5. O sistema notifica o administrador que bloqueou o utilizador.
Inclusões:	Não existem inclusões.

3.2.4.24. UC 24: Desbloquear utilizador

Tabela 26 – Caso de uso: Desbloquear utilizador

Atores:	Administrador.
Descrição:	Permite desbloquear utilizadores bloqueados.
Pré-Requisitos:	Existirem utilizadores bloqueados.
Pós-Condições:	Utilizador validado
Fluxo Principal:	1. O administrador desbloqueia um utilizador pesquisando pelo seu nome ou

	email. 1.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 1.2. O administrador realiza uma pesquisa a um utilizador pelo seu *username* ou email carregando nos botões correspondentes, que estão presentes no menu *Filter* que se encontra na barra lateral do lado direito. 1.3. O administrador insere na caixa de pesquisa o username ou email do utilizador pretendido. 1.4. O administrador carrega no botão de pesquisa. 1.5. O sistema devolve o utilizador pesquisado. 1.6. O administrador carrega no botão *Unblock* presente no bloco com a informação resumida desse utilizador. 1.7. O sistema notifica o administrador que desbloqueou o utilizador.
Fluxo Alternativo:	2. O administrador desbloqueia um utilizador filtrando os utilizadores bloqueados. 2.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior 2.2. O administrador carrega na opção *Blocked* do menu *Filter* presente na barra lateral do lado esquerdo. 2.3. O sistema devolve a lista de utilizadores bloqueados. 2.4. O administrador procura o utilizador que pretende desbloquear na lista de utilizadores e carrega no botão *Unblock* presente no bloco com a informação resumida desse utilizador. 2.5. O sistema notifica o administrador que desbloqueou o utilizador. 3. O administrador desbloqueia um utilizador a partir de uma caixa com a sua informação detalhada. 3.1. Ir para o passo 1.1. ou para o passo 2.1. 3.2. O administrador carrega no bloco com a informação resumida do utilizador que pretende desbloquear (a seguir ao passo 1.5. ou passo 2.3.). 3.3. O sistema apresenta uma caixa com a informação detalhada do utilizador. 3.4. O administrador carrega no botão *Unblock*. 3.5. O sistema notifica o administrador que desbloqueou o utilizador.
Inclusões:	Não existem inclusões.

3.2.4.25. UC 25: Remover utilizador

Tabela 27 – Caso de uso: Remover utilizador

Atores:	Administrador.
Descrição:	Permite remover utilizadores.
Pré-Requisitos:	Existirem utilizadores registados.
Pós-Condições:	Utilizador removido do sistema.
Fluxo Principal:	1. O administrador remove um utilizador pesquisando pelo seu nome ou email. 1.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 1.2. O administrador realiza uma pesquisa a um utilizador pelo seu *username* ou email carregando nos botões correspondentes que estão presentes no menu *Filter* que se encontra na barra lateral do lado direito. 1.3. O administrador insere na caixa de pesquisa o username ou email do utilizador pretendido. 1.4. O utilizador carrega no botão de pesquisa. 1.5. O sistema devolve o utilizador pesquisado. 1.6. O administrador carrega no botão *Delete* 1.7. O sistema apresenta uma caixa a perguntar se o administrador realmente quer eliminar o utilizador. 1.8. O administrador carrega no botão *Confirm.*

	1.9. O sistema notifica o administrador que removeu o utilizador.
Fluxo Alternativo:	2. O administrador remove um utilizador filtrando os utilizadores existentes. 2.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 2.2. O administrador procura o utilizador que pretende remover na lista de utilizadores e carrega no botão *Remove.* 2.3. O sistema apresenta uma caixa a perguntar se o administrador realmente quer eliminar o utilizador. 2.4. O administrador carrega no botão *Confirm.* 2.5. O sistema notifica o administrador que removeu o utilizador. 3. O administrador desbloqueia um utilizador a partir de uma caixa com a sua informação detalhada. 3.1. Ir para o passo 1.1. ou para o passo 2.1. 3.2. O administrador carrega no bloco com a informação resumida do utilizador que pretende remover (a seguir ao passo 1.5. ou passo 2.1.). 3.3. O sistema apresenta uma caixa com a informação detalhada do utilizador. 3.4. O administrador carrega no botão *Delete.* 3.5. Voltar para o passo 1.6. ou o passo 2.3.
Inclusões:	Não existem inclusões.

3.2.4.26. UC 26: Editar limites utilizador

Tabela 28 – Caso de uso: Editar limites utilizador

Atores:	Administrador.
Descrição:	Permite editar o limite de documentos, SVMs e agrupamentos de documentos que um utilizador pode guardar.
Pré-Requisitos:	Existirem utilizadores registados.
Pós-Condições:	Utilizador registrado.
Fluxo Principal:	1. O administrador edita os limites de um utilizador. 1.1. O administrador escolhe a opção *Manage System* e a sub-opção *Manage Users* no menu superior. 1.2. O administrador realiza uma pesquisa por um utilizador pelo seu *username* ou email carregando nos botões correspondentes presentes no menu *Filter* que se encontra na barra lateral do lado direito. 1.3. O administrador insere na caixa de pesquisa o username ou email do utilizador pretendido. 1.4. O utilizador carrega no botão de pesquisa. 1.5. O sistema devolve o utilizador pesquisado. 1.6. O administrador carrega no bloco onde se encontra a informação resumida do utilizador. 1.7. O sistema apresenta uma caixa com toda a informação detalhada do utilizador e com os campos relativos aos limites para guardar os diferentes tipos de informação editáveis. 1.8. O administrador insere um novo valor para o limite de documentos guardados. 1.9. O administrador insere um novo valor para o limite de classificadores guardados. 1.10. O administrador insere um novo valor para o limite de *clusterings* guardados. 1.11. O administrador carrega no botão *Update.* 1.12. O sistema notifica o administrador que atualizou o utilizador.
Fluxo Alternativo:	Não existem fluxos alternativos.
Inclusões:	Não existem inclusões.

3.2. Diagramas de atividade

Devido à complexidade e importância de alguns dos casos de uso apresentados, foram criados diagramas de atividade. Os casos de uso para os quais foram criados os diagramas de atividade são o agrupamento de documentos enviados por um utilizador e o treino de máquinas de vectores de suporte.

O caso de uso relativo ao agrupamento de documentos da PubMed também apresenta um grau de complexidade e de importância semelhante ao agrupamento de documentos enviados por um utilizador, mas uma vez que existe uma grande similaridade entre ambos, apenas é apresentado um deles.

O primeiro diagrama de atividade apresentado é o diagrama relativo ao treino de máquinas de vectores de suporte. Este diagrama pode ser visto na Figura 9. Em relação ao agrupamento de documentos enviados por um utilizador, o seu diagrama está presente na Figura 10.

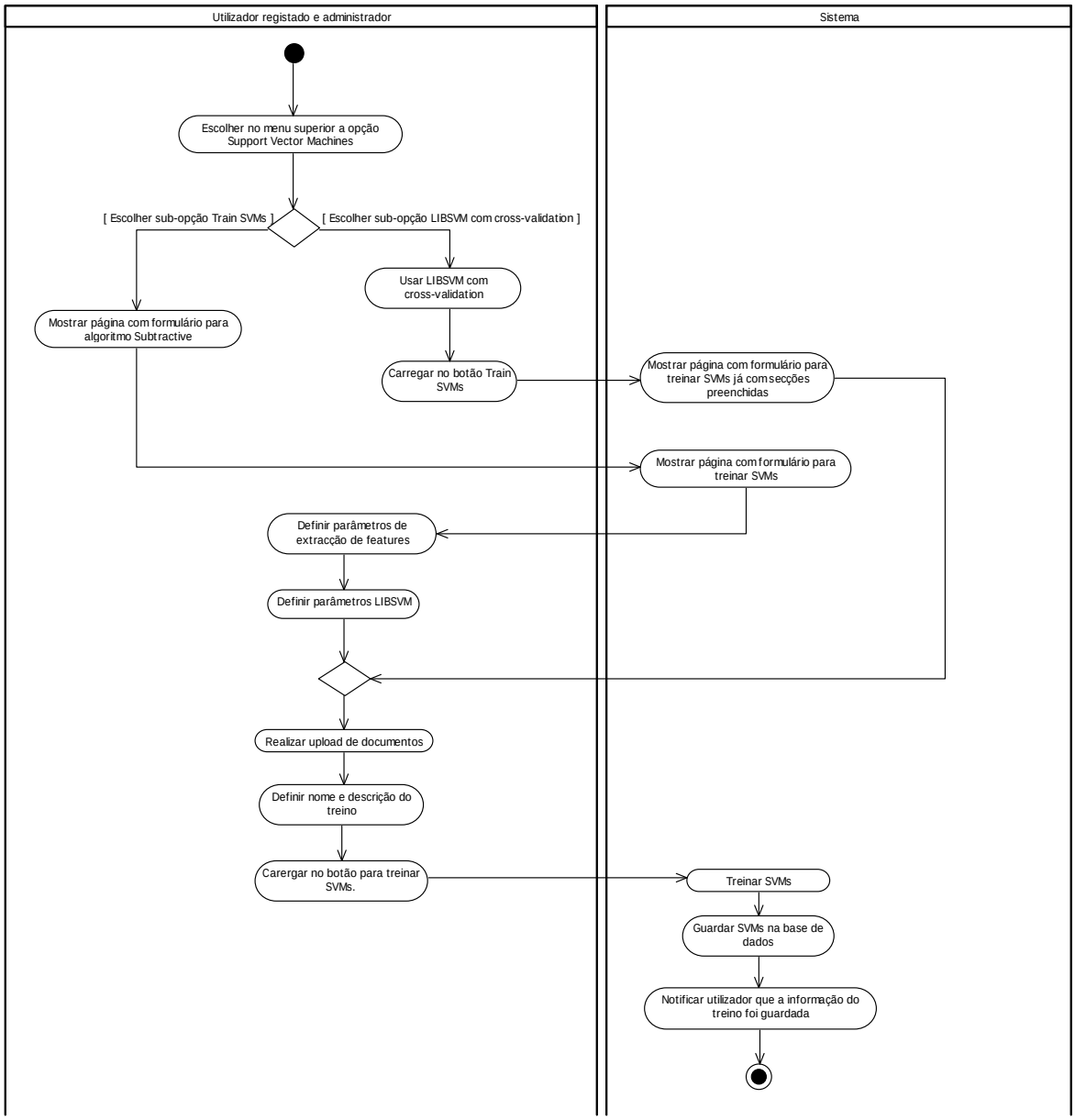


Figura 9 – Diagrama de atividades: Treinar SMVs

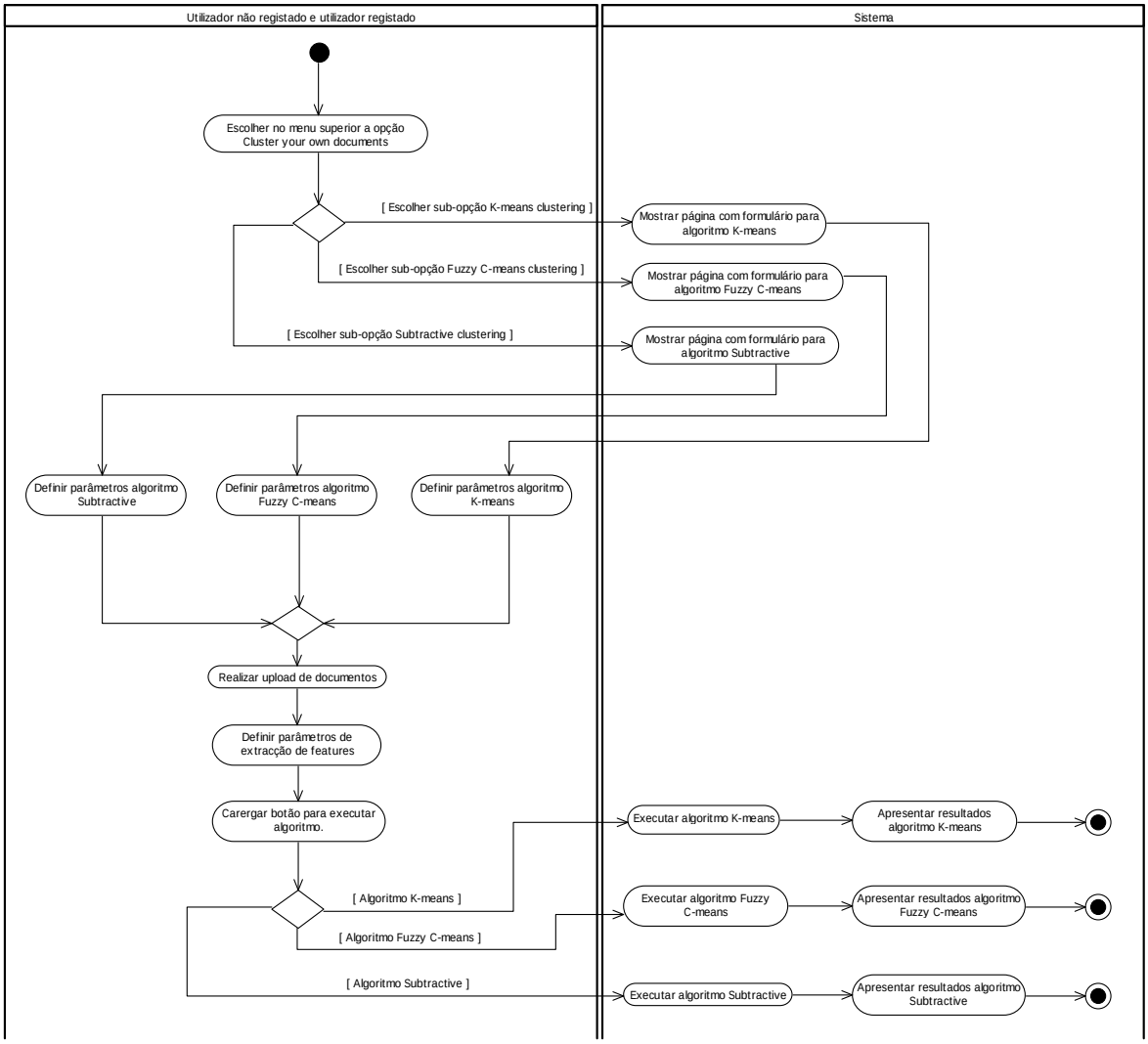


Figura 10 – Diagrama de atividades: Agrupamento de documentos enviados por um utilizador

3.3. Diagramas de sequência

Para além dos diagramas de atividade também foram criados alguns diagramas de sequência. Foram escolhidos três casos específicos para a criação destes diagramas: classificação de documentos enviados por um utilizador usando o algoritmo *K-means*, treino de máquinas de vectores de suporte e classificação de documentos recorrendo às *SVMs* treinadas. Os diagramas para estes casos podem ser consultados no anexo A.

3.4. Protótipos de *interface*

Ao longo do processo de criação dos protótipos de interface, apresentados no anexo B, foram tidas em conta algumas práticas que devem ser usadas no desenho do *interface* de uma aplicação.

Uma das práticas usadas é a consistência da aplicação em situações semelhantes. Ao longo dos vários formulários é possível verificar essa consistência ao ser mantida a estrutura, as cores e o *layout* dos mesmos. Também na apresentação dos diferentes tipos de resultados é mantida essa consistência. Na Figura 72, Figura 73, Figura 74, Figura 75, Figura 78 e Figura

81 é possível reparar que a informação com os resultados é sempre apresentada no centro das páginas, enquanto que do lado esquerdo existe um menu que permite a navegação entre os diferentes tipos de informação e do lado direito é fornecida a informação sobre as opções tomadas anteriormente e ações que podem ser executadas.

Ainda nos formulários é utilizada uma prática relativa à organização em grupos, das sequências de ações a serem tomadas. Na Figura 66, Figura 67, Figura 68, Figura 69, Figura 70, Figura 71, Figura 76 e Figura 77 é possível verificar uma divisão clara do tipo de opções a introduzir através de uma separação dessas opções em diferentes grupos. A separação é realizada de acordo com a obtenção de documentos, escolha dos parâmetros para extração de *features* e escolha dos parâmetros do algoritmo a utilizar.

Permitir ao utilizador reverter as suas ações facilmente é um principio que pode ser encontrado na apresentação de resultados. Como já foi explicado anteriormente, no ecrã de resultados é possível consultar os parâmetros usados na extração de *features* e nos algoritmos utilizados na classificação dos documentos e também voltar a realizar a ação anterior, seja essa ação o agrupamento ou classificação de documentos.

Finalmente, em alguns locais da aplicação é realizada uma prevenção de erros. Um desses exemplos ocorre na criação de uma nova conta, onde o utilizador ao colocar um nome de utilizador ou email já existente, é notificado sem ser necessário submeter o formulário. Outro exemplo acontece no ecrã de treino de máquinas de vectores de suporte onde o utilizador é notificado quando atinge o limite de *SVMs* guardadas e impossibilitado de realizar esse treino.

4. DESENHO E ARQUITECTURA

4.1. Arquitetura

Em termos de arquitetura a aplicação DoCluster usa o padrão Modelo-Vista-Controlador (*MVC*). Este padrão tem como principal objetivo separar o modelo de dados da aplicação (modelo) da camada de controlo (controlador) e do *interface* com o utilizador (vista) fazendo-as comunicar entre si. Esta separação faz com que exista um aumento da segurança dos dados uma vez que o *interface* não tem contato direto com o modelo de dados e necessita sempre de passar pela camada de controlo da aplicação para obter ou tomar ações sobre a informação. Outro dos pontos positivos deste padrão é o cumprimento de dois dos princípios mais importantes no ciclo de vida de um *software*. Esses princípios são a redução do acoplamento e o aumento da coesão tornando desta forma a aplicação mais escalável.

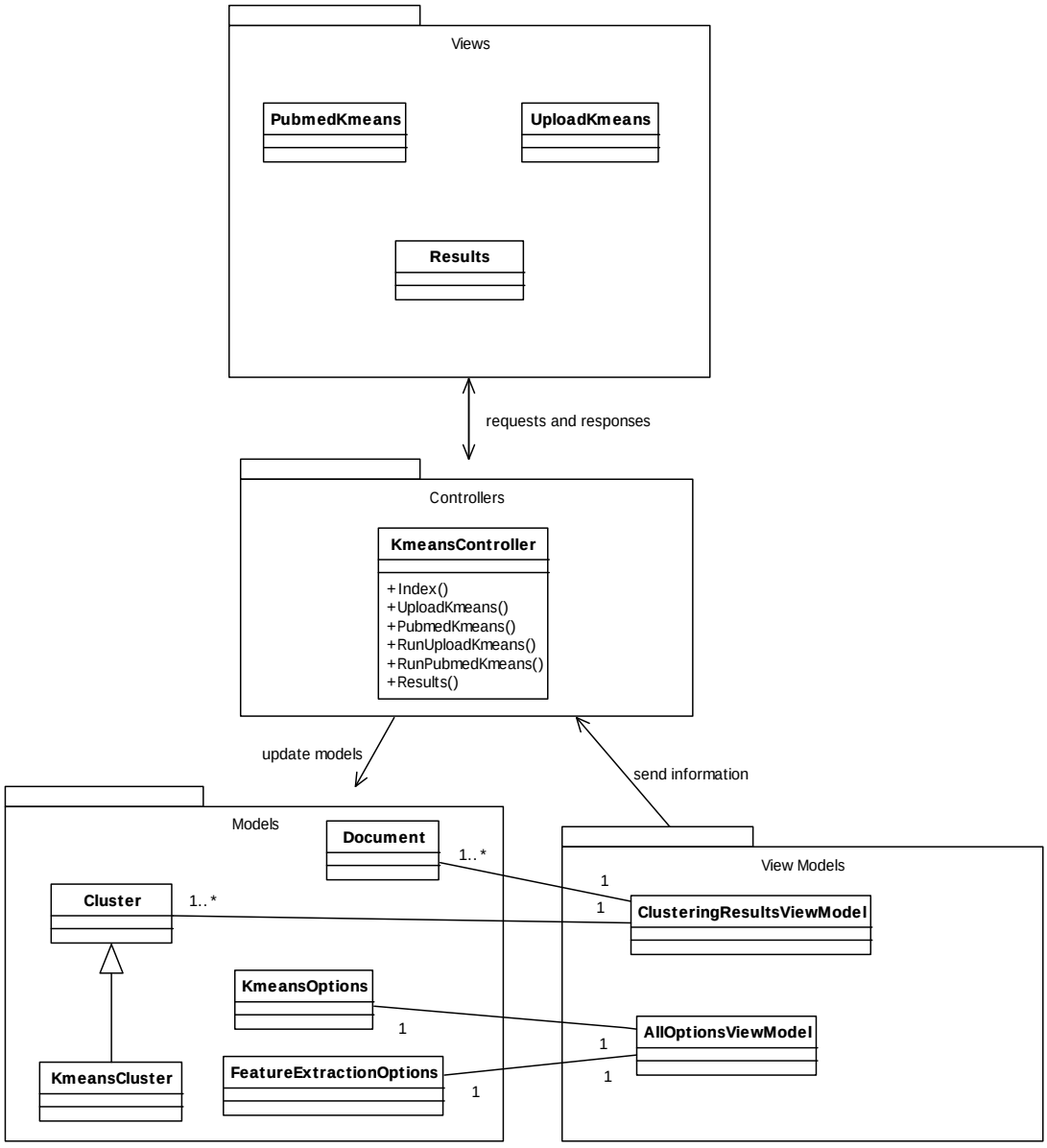


Figura 11 – Exemplo do padrão MVC dentro da aplicação DoCluster

Na Figura 11 é apresentado um diagrama com o funcionamento deste padrão recorrendo ao controlador *KmeansController*, às vistas que lhe estão associadas e às classes de dados que utiliza.

Uma forma de explicar mais pormenorizadamente o seu funcionamento é descrevendo os passos que são realizados numa situação especifica. Para esta explicação é usado o caso de classificação de documentos enviados por um utilizador recorrendo ao algoritmo *K-means*.

O primeiro passo neste cenário consiste em realizar o pedido ao controlador, através da função *UploadKmeans*, para obter a vista onde podem ser introduzidas as opções relativas à extração de *features* e aos parâmetros do algoritmo. Através de um *ViewModel*, *AllOptionsViewModel*, é possível associar à vista ambos os modelos *KmeansOptions* e *FeatureExtractionOptions* com os diferentes tipos de opções.

Depois de definidos todos os parâmetros necessários para a execução do algoritmo é realizado um pedido ao controlador através da função *RunUploadKmeans*, enviando para essa função o *AllOptionsViewModel* preenchido.

No controlador é executado o algoritmo, e após a execução desse algoritmo é guardada a informação com os resultados. De seguida é chamado o método *Results* que vai ler a informação guardada e vai criar um novo *ViewModel*, *ClusteringResultsViewModel* que terá associado os modelos *KmeansCluster* e *Document* com toda a informação dos resultados. Essa informação é por fim enviada para a vista e é apresentada ao utilizador.

Este foi apenas um exemplo prático de como o padrão MVC é implementado na aplicação, sendo que os restantes controladores usam a mesma arquitetura.

4.2. Diagrama de componentes

O diagrama de componentes da aplicação é apresentado na Figura 12, sendo os seus componentes descritos de seguida.

4.3.1. *Model*

O componente *model* é o componente que contém toda a parte lógica da aplicação. Neste componente são executadas ações como a obtenção de documentos, extração de *features* e classificação de documentos. Este componente interage com a maioria dos restantes componentes do sistema.

4.3.2. *Controller*

O *controller* é o componente que permite a interação entre o utilizador e a aplicação. Este componente recebe os dados e pedidos vindos do utilizador, trata e valida essa informação e ainda faz todos os pedidos e atualizações no modelo de dados.

4.3.3. *View*

O componente *view* contém um conjunto de vistas que permitem a visualização da informação presente nos modelos.

4.3.4. *ViewModels*

Este componente permite ultrapassar uma das limitações do padrão *MVC*. Com o *ViewModel* é possível encapsular várias classes do modelo para serem usadas em apenas uma vista.

4.3.5. *Entity Framework 4*

Entity Framework 4 é uma *framework* que permite o mapeamento entre objetos do modelo e a base de dados. A criação e todas as operações de leitura e escrita na base de dados são realizadas por esta *framework* de forma simples e eficaz.

4.3.6. PubMed

Este componente é um *Web service* baseado em SOAP que permite à aplicação aceder a documentos presentes na base de dados PubMed.

4.3.7. *Matlab Application Type Library*

Esta libraria permite, a partir de código *C#*, executar ficheiros Matlab.

4.3.8. *WVTool*

WVTool é uma biblioteca construída na linguagem de programação *java* que permite realizar a extração de *features* a partir de um conjunto de documentos.

4.3.9. *Nunrar*

NUnrar é uma biblioteca que permite extrair todos os ficheiros de um ficheiro *.rar*.

4.3.10. *ICSharpCode.SharpZipLib*

ICSharpCode.SharpZipLib consiste numa biblioteca que possibilita a extração de ficheiros a partir de um ficheiro *.zip*.

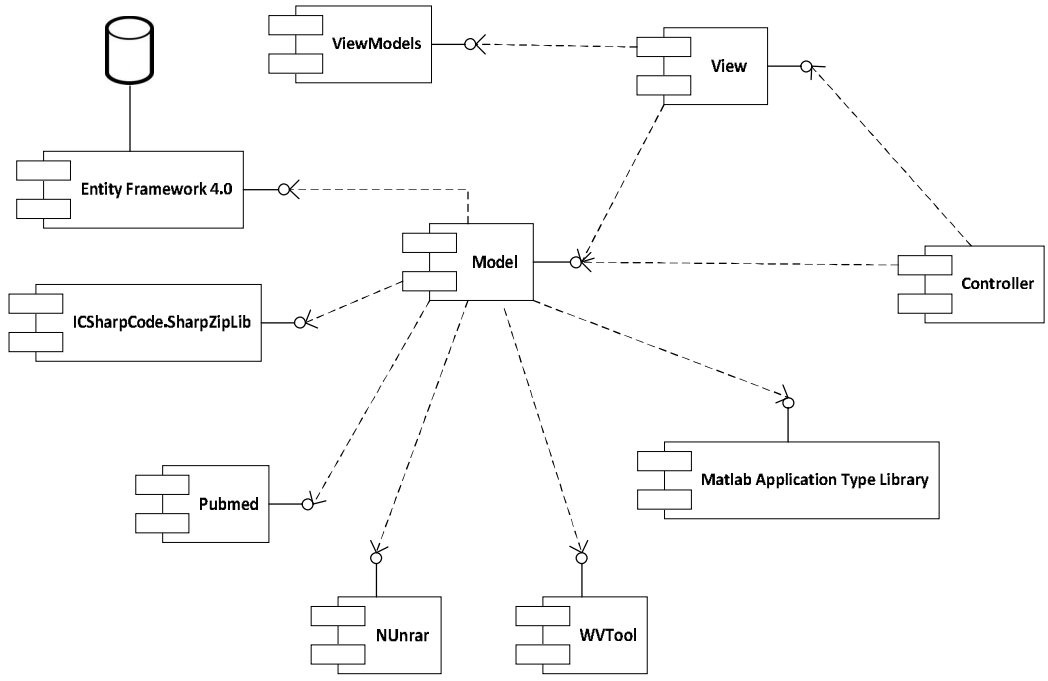


Figura 12 – Diagrama de componentes

4.3. Base de dados

O modelo de dados usado na aplicação é apresentado na Figura 13. Neste diagrama é possível visualizar todas as tabelas existentes, bem como as ligações entre elas.

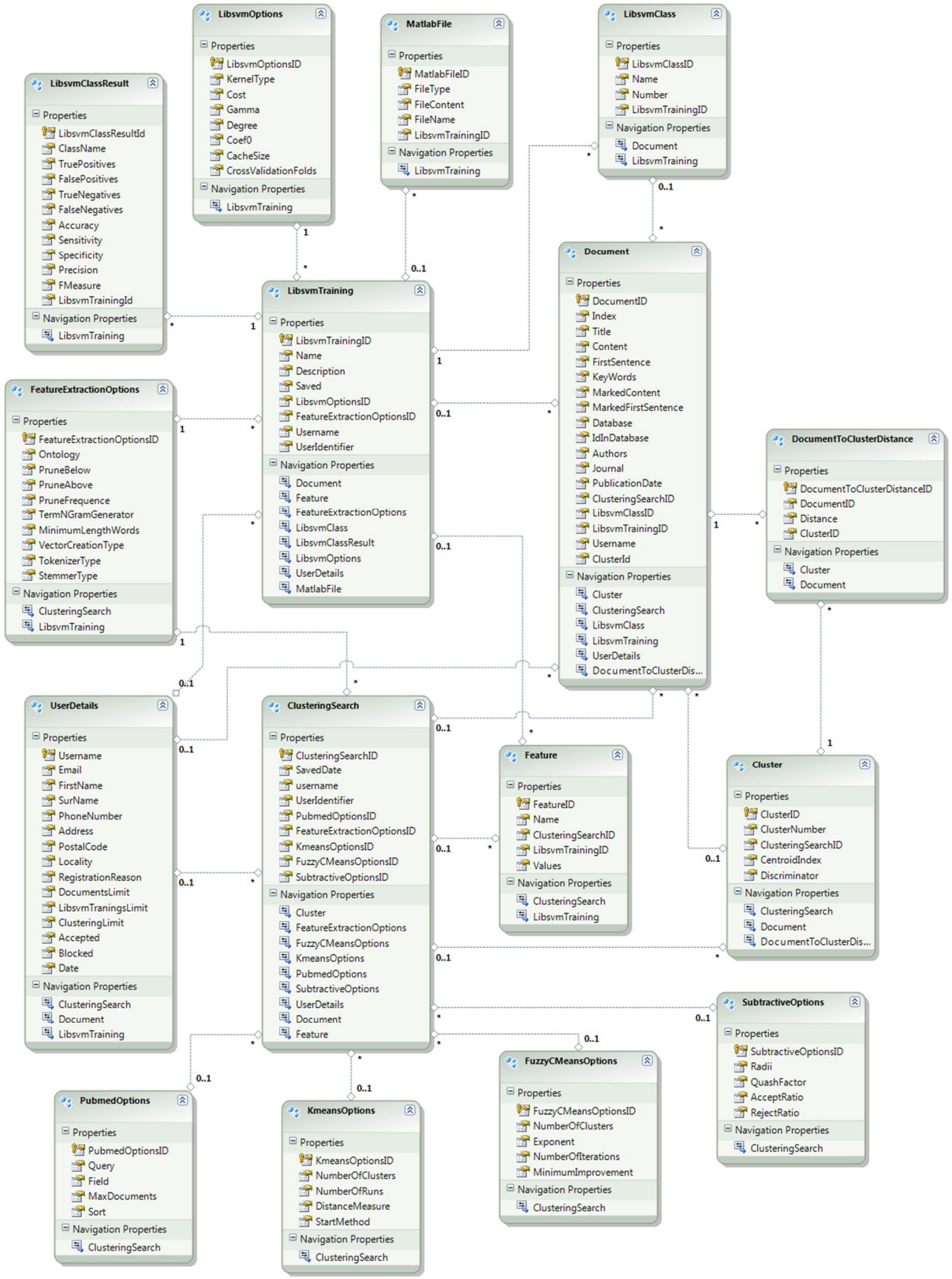


Figura 13 – Modelo de dados usado no DoCluster

5. IMPLEMENTAÇÃO

Com a implementação da ferramenta DoCluster não se pretendeu apenas criar uma plataforma para a classificação de documentos e visualização de resultados, mas também uma plataforma que seja extensível e de fácil manutenção.

Como na plataforma existem várias formas de obtenção de documentos, diferentes modos de extração de *features* e variados algoritmos de classificação, optou-se por realizar uma "separação" do código em três partes distintas. Estas partes coincidem exatamente com as fases do *text mining*: obtenção de documentos, extração de características e classificação de documentos.

De seguida são apresentadas algumas das opções de implementação tomadas, padrões de desenho utilizados e descrição da interligação entre todos os componentes do sistema.

5.1. Obtenção de documentos

Para se proceder à obtenção de documentos foi usado o padrão de desenho conhecido como o padrão *template*. Na Figura 14 é apresentado o diagrama de classes relativo à obtenção de documentos, que implementa esse padrão.

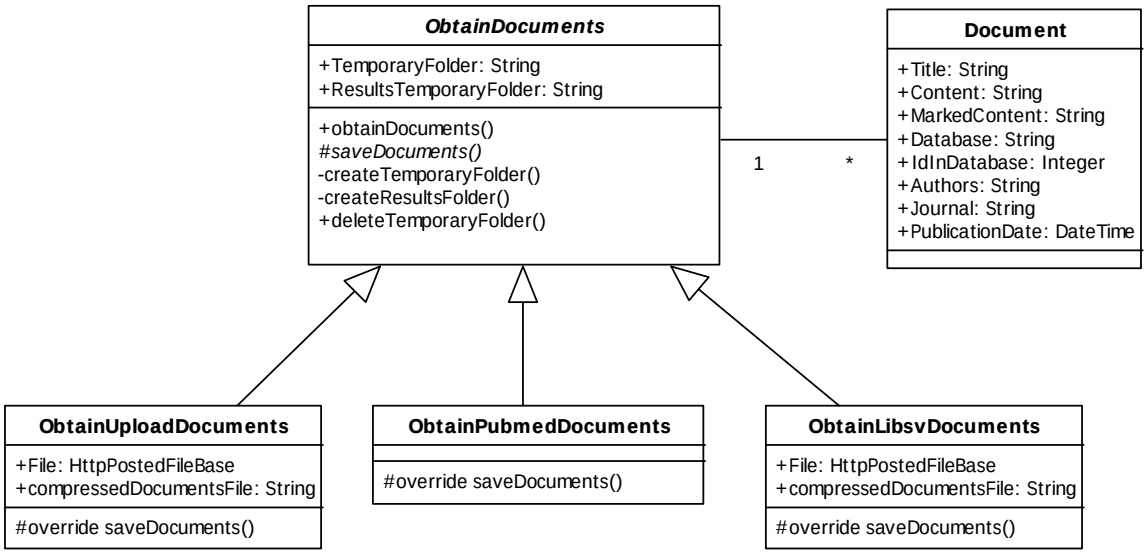


Figura 14 – Padrão *template* usado na obtenção de documentos

O padrão *template* é um padrão comportamental que define o esqueleto de um algoritmo dentro de um método. Alguns passos deste algoritmo podem variar nas diferentes classes derivadas, permitindo assim mudanças no algoritmo, mas sempre mantendo o seu esqueleto.

Para criar um padrão *template* é necessária uma classe abstrata que no seu interior possui um método, designado como o método *template*. Esse método vai conter várias chamadas a outros métodos, que vão definir o seu comportamento. Todo o comportamento que se prevê variável entre as classes derivadas deve estar em métodos abstratos, que terão obrigatoriamente de ser

implementados nessas classes. O comportamento que se prevê igual é implementado na classe base e não deve ser alterado nas classes derivadas.

A escolha para a utilização deste padrão de desenho deve-se a dois fatores. O primeiro consiste na repetição de algumas ações em todos os tipos de obtenção de documentos. Neste caso as ações repetidas são a criação de duas pastas temporárias onde são guardados os documentos a classificar (método *createTemporaryFolder*), e onde são colocados os resultados do algoritmo (método *createResultsFolder*). Outro método existente é o método *deleteTemporaryFolder* que permite a posterior eliminação das pastas e todos os seus ficheiros.

```
public void obtainDocuments() {

       //Criação de uma pasta temporária para guardar os documentos
       //O caminho da pasta é guardado
       this.temporaryFolder = createTemporaryFolder();

       //Obtém os documentos e guarda-os numa pasta temporára
       //Preenche um Dictionary<Int32, Document> com a informação dos
       //documentos
       this.documents = SaveDocuments();

       //Criação de uma pasta temporária de resultados
       //O caminho da pasta é guardado
       this.resultsTemporaryFolder = createResultsFolder();
}

private string createTemporaryFolder() {...}

private string createResultsFolder() {...}

public void deleteTemporaryFolder() {...}

protected abstract Dictionary<Int32, Document> SaveDocuments();
```

Figura 15 – Excerto de código da classe ObtainDocuments

O segundo fator para a escolha do padrão *template* é a forte possibilidade de no futuro virem a ser integradas novas formas de obtenção de documentos. Com a implementação deste padrão isso é possível criando apenas uma nova classe que derive da classe abstrata *ObtainDocuments*, e que implemente o método *SaveDocuments* que deverá obter os documentos a partir de uma fonte e guardá-los na pasta temporária.

O código referente ao método *template obtainDocuments*, bem como a declaração dos restantes métodos da classe *ObtainDocuments*, referidos anteriormente, é apresentado na Figura 15.

5.1.1. Obtenção de documentos do utilizador

Uma das formas de obtenção de documentos é através do *upload* de documentos por parte dos utilizadores. Os documentos devem ser enviados num ficheiro do tipo *rar* ou *zip* e devem ser ficheiros de texto com o formato *txt*.

Uma vez recebido esse ficheiro, o seu conteúdo vai ser extraído para a pasta temporária usando duas librarias: *ICSharpCode.SharpZipLib.Zip* e *NUnrar.Archive*, para ficheiros do tipo *zip* ou *rar* respectivamente.

Por fim estes documentos são lidos e a sua informação é colocada em objetos do tipo *Document*.

5.1.2. Obtenção de documentos da PubMed

Para a obtenção de documentos a partir da base de dados da PubMed são usados dois *Web services* disponibilizados pela NCBI (*Nacional Center for Biotechnology Information*). A obtenção destes documentos é dividida em duas partes sendo que em cada uma delas é usado um desses *Web services.*

Na primeira parte é usado o *Web service eUtilsService* que através das opções de pesquisa inseridas pelo utilizador irá retornar uma lista com todos os identificadores PubMed relativos aos artigos encontrados. O código relativo a esta fase é apresentado na Figura 16.

```
//Criação de um novo serviço
eUtilsService.eUtilsServiceSoapClient eUtilsServiceSoapClient =
       new eUtilsService.eUtilsServiceSoapClient();

//Criação de um novo pedido
eUtilsService.eSearchRequest request = new eUtilsService.eSearchRequest();

//Base de dados onde se pretende obter os documentos
request.db = "pubmed";

//Termo usado para pesquisa. Se o utilizador definiu campos especificos dos
artigos onde a sua query deve incidir é indicado através de parentesis rectos
esse campo.
if (!String.IsNullOrEmpty(this.options.Field))
       request.term = this.options.Query + "[" + this.options.Field + "]";
else
       request.term = this.options.Query;

//Número máximo de documentos a retornar
request.RetMax = this.options.MaxDocuments.ToString();

//Definir o tipo de ordenação dos resultados
if (!String.IsNullOrEmpty(this.options.Sort))
       request.sort = this.options.Sort;

//Execução do pedido
eUtilsService.eSearchResult result =
       eUtilsServiceSoapClient.run_eSearch(request);
```

Figura 16 – Código com pedido ao *Web service* da PubMed

Como se pode verificar no código apresentado é possível definir vários parâmetros de pesquisa. Esses parâmetros são o texto a pesquisar, o número máximo de artigos retornados, o(s) campo(s) onde se pretende que a pesquisa incida e a ordem pela qual se pretende que os resultados sejam retornados (data de adição na PubMed, data de publicação, titulo, autores ou nome da publicação).

Na segunda parte, os identificadores dos documentos retornados são utilizados no *Web service eFetchPubmedService* a partir do qual é obtida toda a informação relativa a esses artigos.

```
//Criação de um novo serviço
eFetchPubmedService.eUtilsServiceSoapClient
       eUtilsServiceSoapClientEFetchPubmed =
              new eFetchPubmedService.eUtilsServiceSoapClient();

//Criação de um novo pedido
eFetchPubmedService.eFetchRequest eFetchRequest =
       new eFetchPubmedService.eFetchRequest();

//Adicionar todos os identificadores separados por uma virgula
for (int i = 0; i < result.IdList.Length; i++)
       eFetchRequest.id += result.IdList[i] + ",";

//Remove a ultima virgula
eFetchRequest.id +=
       eFetchRequest.id.Substring(0, eFetchRequest.id.Length - 1);

//Pede os Abstracts dos artigos
eFetchRequest.rettype = "abstract";

                              //Executa o pedido
eFetchPubmedService.eFetchResult eFetchResult =
       eUtilsServiceSoapClientEFetchPubmed.run_eFetch(eFetchRequest);

//Percorre todos os objectos do tipo PumedArticleSet
//Guarda a informação em ficheiros de texto e coloca-os na pasta temporária
//Preenche a lista de objectos Document
for (int i = 0; i < eFetchResult.PubmedArticleSet.Length; i++) {

       ...

}
```

Figura 17 – Código com a obtenção de documentos do PubMed

Na Figura 17 é apresentado o código onde é realizado o pedido de todos os artigos a partir dos identificadores PubMed obtidos na primeira fase.

Da informação retornada é guardado o titulo dos artigos, o seu resumo, os autores, a publicação onde os artigos se encontram e a data dessa publicação.

5.1.3. Obtenção de documentos do utilizador para LIBSVM

A obtenção de documentos neste caso é bastante semelhante à verificada no *upload* de documentos. A diferença incide na forma como os documentos devem estar distribuídos no ficheiro *zip* ou *rar*. Como é necessário etiquetar documentos no treino de máquinas de vectores de suporte, essa etiquetação deve ser realizada através da distribuição dos ficheiros pelas pastas que representam a sua classe.

Assim, para além dos documentos adicionados como já era realizado anteriormente, é também adicionado um conjunto de classes às quais os documentos pertencem, sendo que a informação relativa a cada classe é guardada na classe *LibsvmClass*. Como se pode verificar pelo diagrama de classes presente na Figura 18, esta classe tem como atributos um identificador numérico e um nome que será o nome da pasta na qual um conjunto de documentos se encontram.

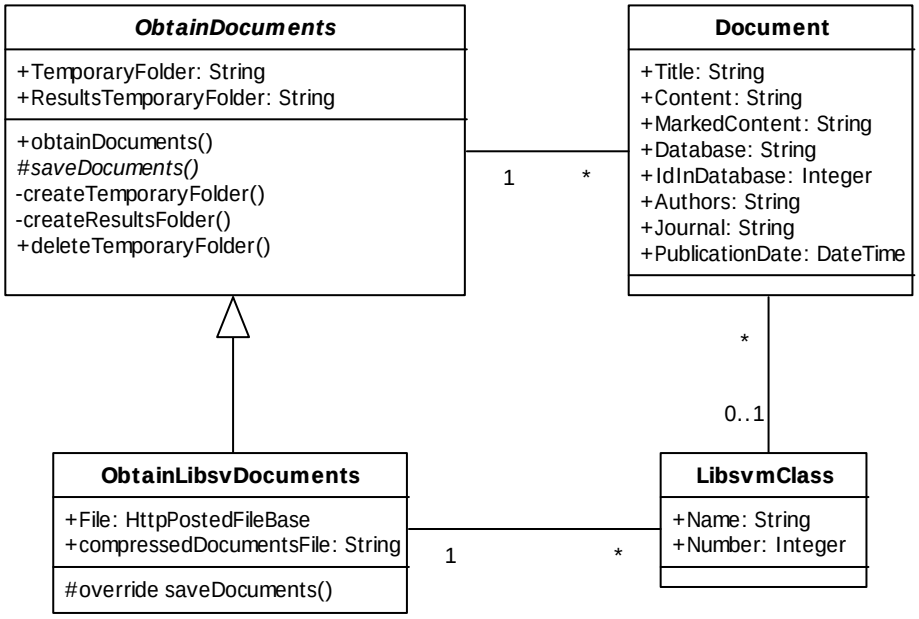


Figura 18 – Diagrama de classes que representa a obtenção de documento para o LIBSVM

5.2. Extração de características

Na extração de características são disponibilizadas três formas distintas para a realização dessa extração e respectiva criação do modelo de dados a ser usado pelos algoritmos de classificação. Usando o *WVTool* sem qualquer recurso a ontologias, e usando as ontologias GeneOntology e Merops.

Apesar das diferentes formas de extração de características, no final deste processo são sempre apresentados dois ficheiros de texto. Um desses ficheiros irá conter os termos extraídos enquanto que o outro irá apresentar o modelo de dados, cuja estrutura será sempre igual. O ficheiro com o modelo de dados consiste na matriz *BOW* e na Figura 19 é apresentado um exemplo com a informação que deverá estar presente nesta matriz. Este exemplo apresenta uma matriz na qual estão representadas três *features* extraídas a partir de sete documentos. Cada coluna da matriz representa as features F1, F2 e F3 e cada linha representa os documentos Doc1, Doc2, Doc3, Doc4, Doc5, Doc6 e Doc7. Nas diferentes células está representado o valor que traduz a importância de uma *feature* num documento, ou seja, no caso da *feature* F1 o valor da sua importância no documento Doc1 é 0,02.

Doc.\Feat.	F1	F2	F3
Doc1	0,02	0,10	0,54
Doc2	0,15	0,01	0
Doc3	0,02	0	0,14
Doc4	0,06	0,31	0,13
Doc5	0	0,42	0,20
Doc6	0,14	0,25	0,64
Doc7	0,32	0	0,26

Figura 19 – Exemplo de uma matriz BOW

Em termos de desenho da aplicação não será usado nenhum padrão de desenho como aconteceu na obtenção de documentos. No entanto a classe *Ontology* é também uma classe abstrata. Dentro da classe existe o método abstrato *extractFeatures*, que terá de ser implementado em todas as classes derivadas como se pode verificar pela Figura 20.

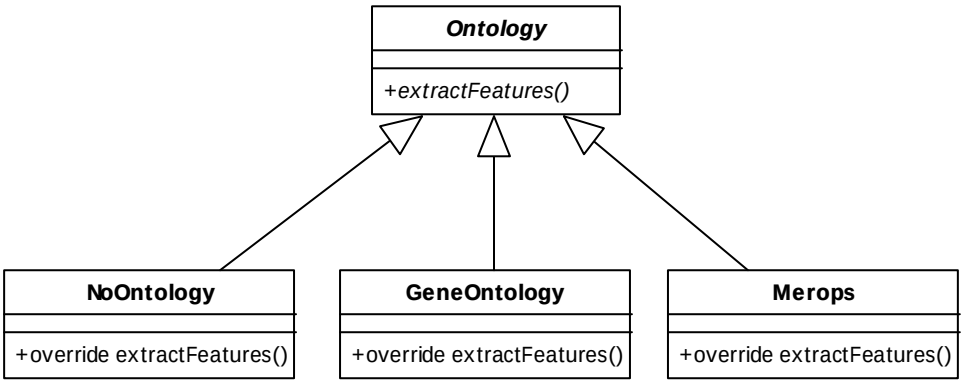


Figura 20 – Diagrama de classes relativo às diferentes formas de extração de características

5.2.1. *NoOntology*

Para a extração de *features* sem o recurso a ontologias, a aplicação faz uso da biblioteca *WVTool*. Na Figura 21 são demonstradas as fases a que todos os documentos de um *dataset* são submetidos até toda a sua informação estar disponível para classificação.

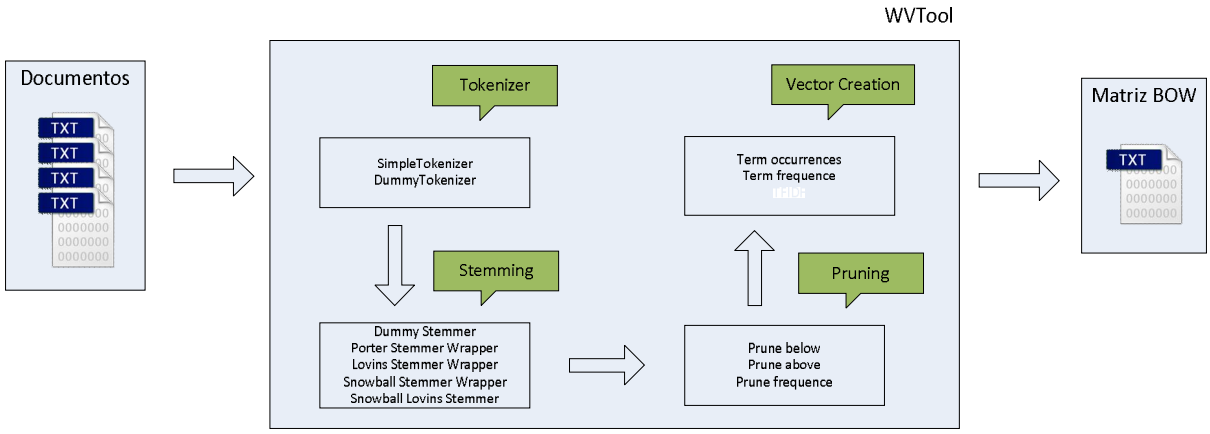


Figura 21 – Processo de extração de características do *WVTool*

Para a utilização desta biblioteca na aplicação, é necessário em primeira instância preencher um objeto, *WVTFileInputList*, com o conjunto de documentos obtidos anteriormente. De seguida a função *WVTClassify.wvcreate* irá realizar a extração de *features*.

Na função *WVTClassify.wvcreate* são passados como parâmetros os nomes dos ficheiros a criar, o objeto *WVTFileInputList* preenchido com todos os documentos e várias opções de extração de caraterísticas definidas pelo utilizador.

As opções de extração de features disponibilizados pelo *WVTool* dividem-se em quatro fases especificas: *tokenizer*, *stemming*, *pruning* (*prune below*, *prune above* e *prune frequence*) e criação de vectores. Para além destas opções são ainda disponibilizadas as opções de

TermNGramGenerator e tamanho mínimo das características. Estas duas opções foram implementadas em adição ao *WVTool*. De seguida são apresentadas e explicadas cada uma delas.

5.2.1.1. *Tokenizer*

O primeiro passo numa extração de termos a partir de documentos tem o nome de *tokenizer*. *Tokenizer* é o processo de segmentação dos documentos em vários *tokens* (termos). Este processo pode ser realizado de várias formas, sendo que o *WVTool* usa apenas duas: *SimpleTokenizer* e o *DummyTokenizer*.

No primeiro caso o documento é segmentado de acordo com todos os caracteres que não sejam letras. Assim, caracteres especiais, números ou espaços, serão os elementos de separação de *tokens* ao longo de um documento.

A segunda opção, *DummyTokenizer*, é semelhante ao *SimpleTokenizer*, não considerando no entanto números como um elemento de divisão. Esses números farão depois parte dos termos extraídos.

5.2.1.2. *Stemming*

Stemming é uma técnica de pré-processamento que consiste na redução dos termos à sua raiz. Para o efeito existem vários algoritmos, sendo os utilizados na plataforma o *DummyStemmer*, *PorterStemmer*, *LovinsStemmer*, *SnowballStemmer* e *SnowballLovinsStemmer*.

5.2.1.3. *Prune Below*

Esta opção permite realizar o corte de algumas *features* já extraídas dos documentos. Ao definir um valor para o *prune below*, o utilizador indica que pretende remover todas as features que não estejam presentes nesse número mínimo de documentos. Este parâmetro não pode ter um valor superior ao número de documentos, pois, se tal acontece todas as *features* são removidas.

5.2.1.4. *Prune Above*

Esta opção permite o corte de todas as *features* que apareçam em mais do que um determinado número, definido pelo utilizador, de documentos. O valor deste parâmetro terá de ser superior a zero uma vez que nesse caso todas as features seriam removidas. Caso o valor escolhido seja igual ou superior ao número de documentos esta opção não irá ter qualquer efeito prático uma vez que todas as *features* serão selecionadas.

5.2.1.5. *Prune Frequence*

O *Prune Frequence* é outra das opções existentes para o corte de termos. Neste caso é possível fazer o corte de todos os termos que após um determinado número de documentos aparecem em apenas um desses documentos. A Figura 22 mostra o funcionamento deste corte com um valor de *prune frequence* igual a 3.

Pela figura é possível perceber que de três em três documentos é verificado se os termos aparecem em apenas um desses documentos. Aqueles que não aparecem em mais do que um documento são retirados da lista de termos extraídos. Também é possível constatar através do

termo “manga”, que após a segmentação de todos os documentos o número de documentos em que as *features* extraídas aparecem é atualizado. Desta forma apesar do termo “manga” ter sido eliminado após os três primeiros documentos o valor é atualizado no final.

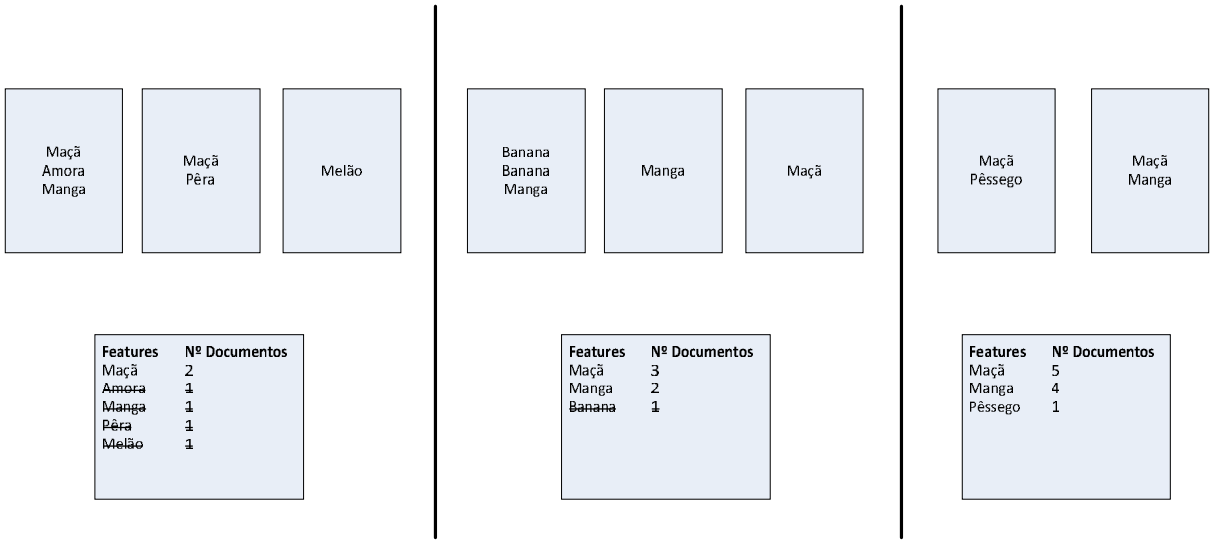


Figura 22 – Exemplo de corte de termos usando *prune frequence* com valor 3.

5.2.1.6. *TermNGramGenerator*

O *TermNGramGenerator* é uma técnica que não faz parte do *WVTool*. Esta opção foi implementada tendo em vista a possibilidade de extrair termos de um documento que possuam mais do que uma palavra. Tendo em conta que na área da biomédica existem vários termos que são bastante relevantes quando constituídos por mais do que uma palavra e não pelas palavras individualmente, é natural que esses termos compostos por várias palavras possam ser bastante úteis na distinção de documentos.

Dando um exemplo concreto temos a peptidase “*pepsin a*”. Um termo constituído por estas duas palavras é relevante uma vez que tal acontece apenas se o conteúdo do documento estiver relacionado, ou fizer referência a esta peptidase especifica. Se as palavras “*pepsin*” e “*a*” forem tratadas como termos distintos, verifica-se que o termo “*pepsin*” é um termo bastante vago relativamente a documentos da área biomédica, pois existem vários tipos de pepsinas, e por outro lado “*a*” é ainda mais vago uma vez que se pode encontrar em vários documentos e com bastante frequência em cada um deles. Neste ultimo caso o termo “*a*” pode mesmo ser removido na próxima etapa de pré-processamento.

A implementação desta técnica é bastante simples e resume-se à adição para cada *token*, de um número de *tokens* seguintes especificado pelo utilizador. Supondo que existem os *tokens* ”*angiotensin*”, “*converting*”, “*enzyme*” e “*peptidase*”, extraídos de um documento pela ordem em que são apresentados e que o utilizador definiu o tamanho máximo de três *tokens* para o novo termo, obtêm-se os seguintes termos finais: “*angiotensin*”, “*angiotensin converting*”, “*angiotensin converting enzyme*”, “*converting*”, “*converting enzyme*”, “*converting enzyme peptidase*”, “*enzyme*”, “*enzyme peptidase*” e “*peptidase*”.

Através desta técnica o número de *features* extraídas no final do pré-processamento dos documentos é praticamente multiplicada pelo número definido pelo utilizador. Na aplicação

está limitado a três devido aos problemas de desempenho computacional que este processo acarreta e que serão analisados na secção de resultados.

5.2.1.7. Tamanho mínimo de palavras

Este parâmetro tem como objetivo eliminar termos que possuam um tamanho inferior ao definido. Desta forma é possível eliminar termos constituídos por duas ou três letras, que sejam irrelevantes para a classificação de um documento e que possam adicionar "ruído" a essa mesma classificação.

Para além de todos os parâmetros e técnicas anteriores, o *WVTool* ainda possui um filtro que elimina as palavras comummente usadas numa determinada língua que neste caso é a língua inglesa. Desta forma palavras como "*the*", "*this*", "*is*" ou "*a*" são removidas.

5.2.1.8. Tipo de criação de vectores

Após a extração de características é necessário criar um modelo de dados (matriz *BOW*) que possa refletir em termos numéricos a importância dos termos extraídos para os diferentes documentos de um *dataset*.

Existem três formas de criar os vectores que vão formar a matriz *BOW*. As opções referentes a estes três métodos de cálculo de vectores são o *Term Occurrence*, *Term Frequency* e *TFIDF*.

No primeiro caso apenas é considerado o número de vezes que um termo aparece num documento. Este método é o mais simples dos três, mas também o mais falível uma vez que não consegue traduzir o peso real de um termo nos diferentes documentos do conjunto. Como exemplos, se pensarmos num termo que pode aparecer o mesmo número de vezes em dois documentos, mas que o número de palavras de ambos os documentos difere largamente, o peso da palavra para cada um desses documentos é objetivamente diferente.

No segundo caso, a opção *Term Frequency* já considera o peso de um termo no conjunto de todos os termos presentes nesse documento. Assim o calculo da frequência de um termo num documento é realizado através da equação (7).

$$\text{TF (Term Frequency)} = \frac{n_{j,i}}{\sum_k n_{k,i}} \quad (7)$$

$n_{j,i}$ é o numero de ocorrências do termo *j* no documento *i*.

$\sum_k n_{k,i}$ é o somatório da ocorrência de todos os termos no documento *i*.

Este processo de criação de vectores, apesar de resolver uma das desvantagens do *Term Occurrence*, continua a ter a desvantagem de não ter em atenção o peso de um termo dentro do conjunto de documentos. Desta forma um termo existente na maioria dos documentos vai ter a mesma importância de um termo que aparece apenas em alguns documentos. Para melhor compreender esta situação vamos imaginar que temos um conjunto de documentos sobre vários tipos de doenças. O termo doença irá aparecer regularmente em todos os documentos deste conjunto e embora não seja um termo diferenciador terá a mesma

importância que os termos coração, pulmão ou fígado que realmente ajudam a distinguir os vários tipos de doenças.

Usando o método *TFIDF* na criação de vectores este ultimo problema pode ser ultrapassado. O *TFIDF* consiste na multiplicação da frequência de um termo num documento pelo logaritmo da frequência desse termo na coleção de documentos (IDF). O valor do *IDF* é dado pela equação (8).

$$IDF\ (Inverse\ Document\ Frequency) = \log\left(\frac{|D|}{|\{d: t_j \in d\}|}\right) \quad (8)$$

|D| é o número de documentos analisados.

$|\{d: t_j \in d\}|$ é o numero de documentos onde o termo j aparece.

Por fim a equação do *TFIDF* é multiplicação das duas anteriores:

$$TFIDF\ (Term\ Frequency - Inverse\ Document\ Frequency) = TF * IDF \quad (9)$$

Com este valor é possível atribuir a um termo um valor numérico mais de acordo com a importância desse termo num documento tendo em conta também todo o conjunto de documentos.

5.2.2. Gene Ontology

Em termos das ontologias utilizadas na aplicação, a Gene Ontology foi a primeira a ser utilizada na ajuda ao pré-processamento de documentos. Esta ontologia é disponibilizada através de uma base de dados presente no *website* da Gene Ontology. A sua utilização na aplicação tem como base a filtragem de termos que apenas estejam presentes nessa ontologia, sendo que a forma mais fácil de realizar esta tarefa seria, para cada termo de um documento segmentado, pesquisar esse termo na base de dados. No entanto para fazer essa filtragem são necessários muitos acessos, sendo o tempo de cada acesso demasiado moroso para o que é pretendido. Por essa razão foi implementado um esquema que otimiza o desempenho da aplicação.

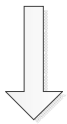
O esquema implementado contempla a colocação de uma parte da base de dados em memória. Esta pequena parte da base de dados é constituída por todos os termos que contenham as palavras peptidases e seus sinónimos. Esta decisão deveu-se a dois fatores. O primeiro devido ao tamanho demasiado elevado da ontologia, que levaria a um tempo de processamento inaceitável para a aplicação. Quanto ao segundo tem a ver com a necessidade de focar a extração de *features* numa área especifica de forma a conseguir grandes resultados com documentos dessas mesmas áreas.

Na implementação deste esquema foi usado o padrão de desenho *singleton*. Este padrão tem como objetivo restringir a instanciação de uma classe a somente um objeto, permitindo que toda a aplicação use esse objeto sem ser necessária nova instanciação. Para conseguir este propósito é colocada dentro da classe uma instância da própria classe definida como privada. Também o construtor será privado não podendo desta forma ser criado qualquer objeto fora da própria classe. Para finalizar existe um método publico e estático *getInstance* que vai

verificar se o objeto instanciado dentro da classe já foi criado. Caso ainda não tenha sido criado chama o construtor e retorna essa instância. Para uma melhor compreensão em termos práticos deste padrão é apresentado na Figura 24 o diagrama da classe *GeneOntologySingleton* e na Figura 25 o código que implementa o padrão *singleton* nesta classe.

Com este padrão é assim possível aceder uma única vez à base de dados Gene Ontology, tratar a informação e criar um esquema de indexação de palavras. Este esquema de indexação de palavras é formado por todas as palavras constituintes dos termos da base de dados que foram entretanto segmentados com o recurso ao *DummyTokenizer*. Após esta separação de palavras também é guardada informação de todos os termos onde cada palavra está presente, bem como o local no próprio termo. Na Figura 23 é demonstrada a forma como é construído este sistema de indexação de palavras.

ID	Term
1	aminopeptidase
2	Carboxypeptidase N catalytic chain
3	Carboxypeptidase B2



Dictionary<String, Dictionary<Int32, List<Int32>>>		
Palavra	Termo	Posições
Aminopeptidase	1	[1]
carboxypeptidase	2	[1]
	3	[1]
n	2	[2]
catalytic	2	[3]
chain	2	[4]
b2	3	[2]

Figura 23 – Exemplo do esquema de indexação de palavras Gene Ontology

Toda esta informação é obtida e indexada a partir da função *InitializerGeneOntology* e vai ser guardada em variáveis presentes no *GeneOntologySingleton* para posteriormente serem usadas na extração de características.

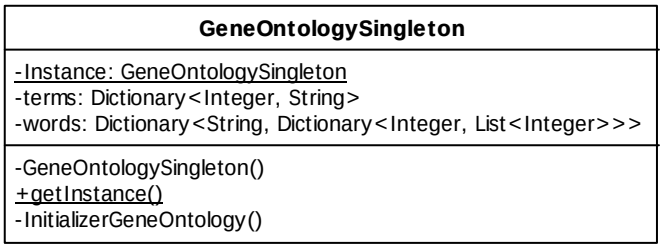


Figura 24 – *Singleton* que inicializa o esquema de indexação Gene Ontology

Com o esquema de indexação de palavras criado torna-se possível verificar rapidamente se num documento se encontram palavras presentes na ontologia. Como é guardada a informação da posição de uma palavra num termo é mesmo possível verificar a existência de termos compostos por mais do que uma palavra.

```
public class GeneOntologySingleton {

	private static GeneOntologySingleton Instance;
	private Dictionary<Int32, String> terms;
	private Dictionary<String, Dictionary<Int32, List<Int32>>> words;

	private GeneOntologySingleton() {
		InitializerGeneOntology();
	}

	public static GeneOntologySingleton getInstance()
	{
		if (Instance == null) {
			Instance = new GeneOntologySingleton();
		}

		return Instance;
	}

	private void InitializerGeneOntology() {...}
```

Figura 25 – Código da classe *GeneOntologySingleton*

```
public override void extractFeatures() {

	this.createWordVectorWithWVTool(this.getWVTFileInputList(),
		"ExtractedFeatures.txt", "null.txt", null, null);

	List<String> features =
		getExctractedFeatures("ExtractedFeatures.txt");

	if (!CreateGeneOntologyDictionary(features, "dictionary.txt"))
		throw new Exception();

	this.createWordVectorWithWVTool(this.getWVTFileInputList(),
			"wlist.txt", "wv.txt", null, "dictionary.txt");

	this.changeWordVectorFormat("wv.txt", "dados.txt");
	this.extractedFeatures =
		this.getFeatureExtractionModel("dados.txt", "wlist.txt");

	features = getExctractedFeatures("wlist.txt");

	Dictionary<String, Int32> ExtractedFeatures =
			new Dictionary<String, Int32>();

	foreach (String feature in features) {
		ExtractedFeatures.Add(feature, 0);
	}

}
```

Figura 26 – Código com extração de *features* usando GeneOntology

No entanto para criar o modelo de dados apenas com os termos GO encontrados nos documentos não basta usar este esquema de indexação de palavras. Assim é realizada uma primeira extração de *features* com o *WVTool* a partir da qual apenas é pretendido o ficheiro com todos os termos extraídos. A partir deste ficheiro são filtrados os termos pertencentes à ontologia recorrendo desta vez ao esquema de indexação. O resultado desta filtragem é outro ficheiro apenas com esses termos, e que é usado numa ultima fase. Esta fase consiste em usar novamente o *WVTool* sendo-lhe dado o ficheiro criado, para que apenas sejam extraídos os termos lá disponibilizados. Desta forma no final do processo de extração de *features* vamos ter um modelo de dados onde estão apenas presentes termos do Gene Ontology. Na Figura 26 é apresentado o método *extractFeatures* da classe *GeneOntology* onde é implementado todo este processo.

4.2.3. Merops

Tal como o Gene Ontology também a Merops disponibiliza uma base de dados com toda a informação que está presente na sua página *web*. Neste caso também é usado o padrão *singleton* de forma a adquirir a informação da base de dados e colocá-la disponível em memória. Para além de a informação ser colocada em memória é igualmente usado o sistema de indexação de palavras para, tal como no caso anterior, aumentar a *performance* da aplicação.

O sistema de indexação de palavras e o algoritmo de extração de *features* apresentam neste caso uma maior complexidade relativamente ao esquema de indexação criado para o Gene Ontology. Para este efeito foi criado um conjunto de classes de modo a guardar toda a informação necessária em memória e posteriormente usá-la no algoritmo de extração de *features* e na apresentação de resultados onde se pretende identificar nos documentos as *features* extraídas. Como é possível verificar na Figura 27, foram criadas quatro classes que serão usadas em todo este processo: *MeropsIdentifier*, *MeropsPeptidase*, *MeropsTerm* e *MeropsWord*.

A classe *MeropsIdentifier* é constituída pelos atributos *Code*, *Name* e *Url*. Cada instância desta classe irá referenciar uma peptidase, identificada através do código e do seu nome principal. Para além destes atributos ainda irá conter o *url* para a página Merops relativa à peptidase. O propósito desta classe será principalmente o de ajudar à marcação dos termos encontrados nos documentos e na ligação desses termos à página *web* da Merops.

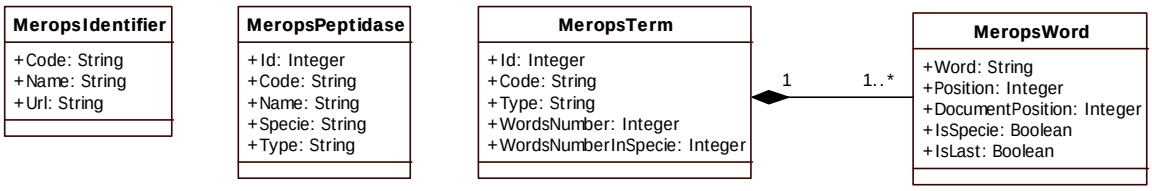


Figura 27 – Diagrama de classes do esquema de indexação Merops

A classe onde se pretende guardar a informação de toda a base de dados Merops é a classe *MeropsPeptidase*. Esta classe irá guardar a informação, devidamente curada, de todas as entradas presentes nas tabelas da base de dados da Merops (apresentadas na Figura 2) e necessárias para a extração de *features* (*protein_name, family, clan, subfamily* e *subclan*). Esta informação curada consiste em atribuir um identificador único a cada registo, o código da peptidase, família ou clã a que se refere e o tipo de nome dado (*real*, *alternative* ou *index*). O campo *name* presente em todas as tabelas da base de dados Merops vai por sua vez dar origem às propriedades *Name* e *Specie*. Este é dividido em duas propriedades diferentes uma vez que em muitos casos o nome possuía também a informação da espécie entre parêntesis ou chavetas.

As classes *MeropsTerm* e *MeropsWord* podem ser consideradas como especializações da tabela *MeropsPeptidase*. Estas classes contêm várias propriedades que vão ser úteis no processo de extração de caraterísticas como são os casos das propriedades *WordsNumber* e *WordsNumberInSpecie* na classe *MeropsTerm*, e *Position*, *DocumentPosition*, *IsSpecie* ou *IsLast* na classe *MeropsWord*.

Na Figura 28 é apresentada a classe *MeropsOntologySingleton* que implementa o padrão *singleton* e que contém toda a informação criada anteriormente.

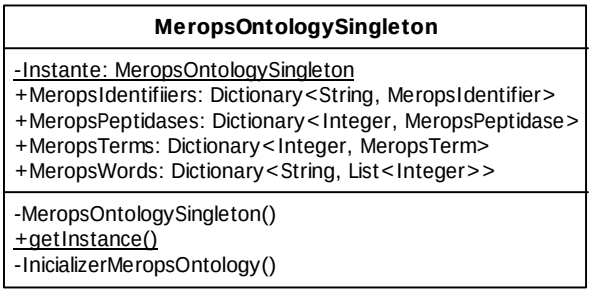


Figura 28 – *Singleton* que inicializa o esquema de indexação Merops

A Merops possui algumas vantagens na extração de features em relação ao Gene Ontology. Uma dessas vantagens é a base de dados especializada apenas em peptidases, que permite por sua vez especializar pesquisas a documentos desta área e obter assim melhores resultados. Outra melhoria em relação ao Gene Ontology é a possibilidade de termos diferentes serem considerados como um só. Isto pode ser feito quando a mesma peptidase aparece num documento com nomes diferentes ou quando se pretende filtrar pesquisas de acordo com clãs ou famílias de peptidases. Este ultimo caso não foi ainda implementado, mas fará parte do trabalho futuro a realizar.

A utilização desta ontologia no pré-processamento de documentos começa com a sua segmentação recorrendo às técnicas *SimpleTokenizer* e *DummyStemmer*. Após estarem segmentados são procurados os termos Merops presentes nesses documentos. Estes termos estão sempre ligados a uma peptidase especifica. Através desta aproximação os termos que sejam sinónimos de um termo principal de uma peptidase, são considerados como sendo o termo principal, não existindo desta forma a sua distinção na matriz *BOW*.

5.3. Classificação dos documentos

Após o pré-processamento de documentos é executado um dos algoritmos de classificação. Nesta fase, tal como na obtenção de documentos também é usado o padrão *template* como se pode verificar pela Figura 29.

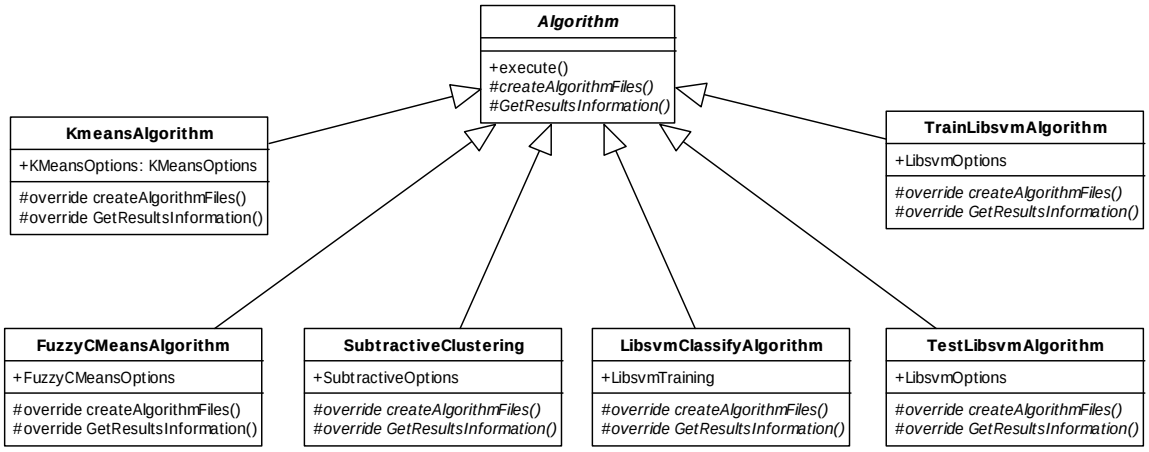


Figura 29 – Diagrama de classes com padrão *template* para execução dos algoritmos.

Neste caso a classe abstrata é a classe *Algorithm* e o método *template* é o método *execute* dessa classe. O código referente ao método *execute*, bem como a declaração de outras funções presentes na classe *Algorithm* são apresentadas na Figura 30.

```
public void execute() {

	//Cria ficheiros com opções para os algoritmos
	//Faz o download do ficheiro matlab
	createAlgorithmFiles();

	//Executa o algoritmo
	executeAlgorithm();

	//Lê os resultados
	GetResultsInformation();
}

protected abstract Boolean createAlgorithmFiles();

protected abstract Boolean GetResultsInformation();

protected Boolean createAlgorithmFile(string fileName, string content) {...}

protected Boolean downloadAlgorithmFile(int fileType) {...}

protected void executeAlgorithm() {...}
```

Figura 30 – Código com a função execute da classe *Algorithm*

O método *template* é constituído por três métodos. Os métodos *createAlgorithmFiles* e *GetResultsInformation* são métodos abstratos que devem ser implementados nas classes

derivadas enquanto que o método *executeAlgorithm* será comum na execução de todos os algoritmos.

Cada uma destas funções corresponde a uma etapa especifica na execução de um algoritmo. Na Figura 31 são demonstradas as três etapas e o que acontece em cada uma delas.

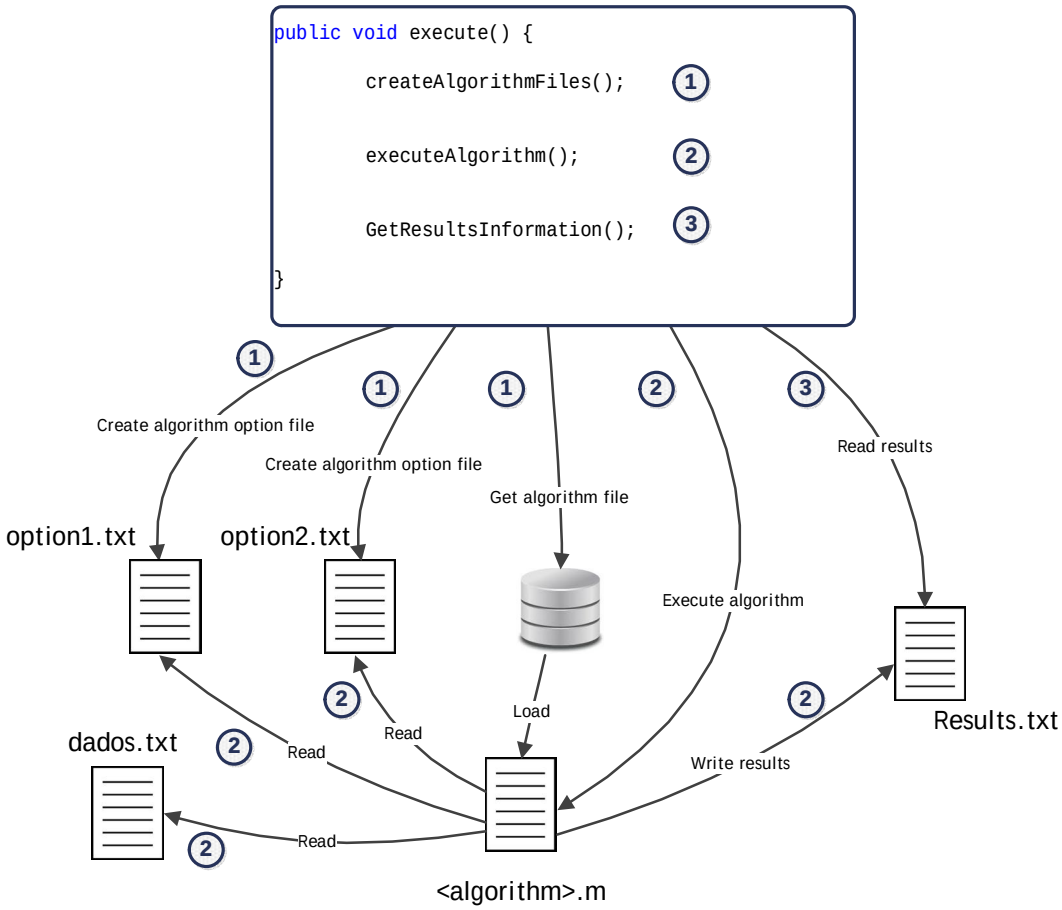


Figura 31 – Etapas de execução de um algoritmo

O método *createAlgorithmFiles*, que é implementado em cada uma das classe derivadas, tem como objetivo criar todos os ficheiros com as opções necessárias à execução dos algoritmos e realizar a leitura, a partir da base de dados da aplicação, do ficheiro Matlab com o algoritmo a executar. Para realizar estas tarefas cada classe derivada vai usar os métodos *createAlgorithmFile* e *downloadAlgorithmFile* da classe base. Todos estes ficheiros são colocados na pasta temporária de resultados juntamente com o ficheiro criado no pré-processamento de documentos contendo a matriz *BOW*.

No método *executeAlgorithm* é executado o ficheiro Matlab com o respectivo algoritmo. Para executar o ficheiro Matlab é usada a libraria *Matlab Application Type Library*, disponível na pasta de instalação do Matlab. O código para a execução de todos os ficheiros Matlab é apresentado na Figura 32.

Neste método é necessário em primeiro lugar aceder à pasta temporária de resultados e aí executar o algoritmo. O código Matlab, como será visto mais à frente em cada um dos algoritmos implementados, vai criar ficheiros de texto com os resultados dos algoritmos.

Para obter a informação dos ficheiros de texto com os resultados de cada algoritmo, existe o método *GetResultsInformation* que também é redefinido em cada uma das classes derivadas. Neste método são lidos os resultados dos algoritmos de classificação, sendo essa informação colocada em objetos criados de acordo com as necessidades de cada um desses algoritmos.

```
protected void executeAlgorithm() {

	MLApp.MLApp matlab = new MLApp.MLApp();

	DirectoryInfo directory = new
	DirectoryInfo(this.obtainedDocuments.resultsTemporaryFolder);

	List<String> directoriesNames = new List<String>();
	directoriesNames.Add(directory.Name);

	while (directory.Parent != null) {
		directory = directory.Parent;
		directoriesNames.Add(directory.Name);
	}

	for (int i = directoriesNames.Count - 1; i >= 0; i--)
		matlab.Execute("cd '" + directoriesNames[i] + "'");

	matlab.Execute(this.algorithmName + "();");

}
```

Figura 32 – Código da função *executeAlgorithm*

De seguida são indicadas as funções Matlab que implementam os algoritmos de classificação, as opções disponíveis para cada uma dessas funções e o esquema de classes que vai guardar no final da execução de cada algoritmo os resultados obtidos.

5.3.1. *K-means*

O algoritmo *K-means* usado na plataforma é executado pela função *kmeans* do Matlab.

Para a execução deste algoritmo é necessária a matriz *BOW* criada na fase de pré-processamento e são fornecidas algumas opções ao utilizador de forma a este poder escolher o número de *clusters* no qual os documentos devem ser divididos e a melhorar a qualidade desse agrupamento. As opções disponíveis para além do número de *clusters* são o tipo de distância entre vectores, o tipo de inicialização dos centros na primeira iteração do algoritmo e ainda o número de vezes que o algoritmo pode ser repetido.

Em termos de distâncias entre vectores é possível escolher as distâncias euclidiana, cosseno, correlação ou *manhattan* (também conhecida como *city block*), sendo que a distância euclidiana é a que será usada por defeito.

Em relação à inicialização dos centroides existem as seguintes opções:

- *sample* é a opção usada por defeito e consiste na escolha aleatória de *x* vectores presentes na matriz *BOW* como primeiros centroides.

- *uniform* forma os centroides a partir de pontos selecionados aleatoriamente de alguns vectores da matriz *BOW*.
- *cluster* executa o algoritmo em 10% de elementos escolhidos aleatoriamente a partir do conjunto de vectores. Esta execução preliminar recorre ao método *sample* na inicialização dos centroides sendo os centroides resultantes deste processamento preliminar, usados no inicio da execução do algoritmo com 100% dos elementos.

Por fim, a última opção permite que o algoritmo seja executado várias vezes usando diferentes centros no seu inicio. No final é escolhido o melhor resultado entre todas as repetições.

Em termos de resultados são criados os dois ficheiros de texto presentes na Figura 33. O ficheiro *índices.txt* indica os clusters a que cada documento pertence sendo que cada linha desse ficheiro corresponde ao documento presente no mesmo índice da matriz *BOW*. O valor presente nessa linha corresponde ao cluster que lhe foi atribuído. O segundo ficheiro, *distancias.txt*, contém as distâncias de cada documento ao centro de todos os *clusters* existentes. Tal como no ficheiro anterior cada linha corresponde a um documento e em cada uma delas existe um conjunto de valores correspondentes à distância desse documento ao centro de cada um dos *clusters*.

Document	Cluster
Doc1	1
Doc2	2
Doc3	1
Doc4	1
Doc5	2
Doc6	2
Doc7	1

indices.txt

Documents/Clusters	C1	C2
Doc1	3,42	6,34
Doc2	7,34	1,43
Doc3	2,32	4,23
Doc4	0,45	8,98
Doc5	5,93	2,43
Doc6	6,76	1,89
Doc7	2,32	5,77

distancias.txt

Figura 33 – Exemplos dos ficheiros de resultados criados pelo algoritmo K-means

Através do diagrama de classes presente na Figura 34 é possível verificar a arquitetura usada na aplicação para o algoritmo *K-means*. Como já foi visto anteriormente, após a execução do algoritmo, a função *GetResultsInformation* da classe *Algorithm* vai ler toda a informação presente nos documentos criados pelo algoritmo. Essa informação é colocada depois em vários objetos do tipo *KMeansCluster* que deriva da classe *Cluster*. Para além do campo *ClusterNumber* na classe *Cluster*, existe ainda o conjunto de documentos pertencentes a esse *cluster* e as distâncias de todos os documentos do *dataset* a esse *cluster* que são dados pelos objetos da classe *DocumentToClusterDistance*. Em relação à classe *KMeansCluster* existe o campo *CentroidIndex* que é preenchido pelo o índice na matriz *BOW* do documento que representa o centro desse *cluster*, ou seja, o documento que está a uma menor distância do centro do *cluster*. No exemplo de resultados da Figura 33 o documento que será considerado o centroide do primeiro *cluster* será o documento Doc4, uma vez que está apenas a uma distância de 0.45 desse *cluster*, enquanto que no segundo *cluster* será o documento Doc2, que se encontra a uma distância de 1.43.

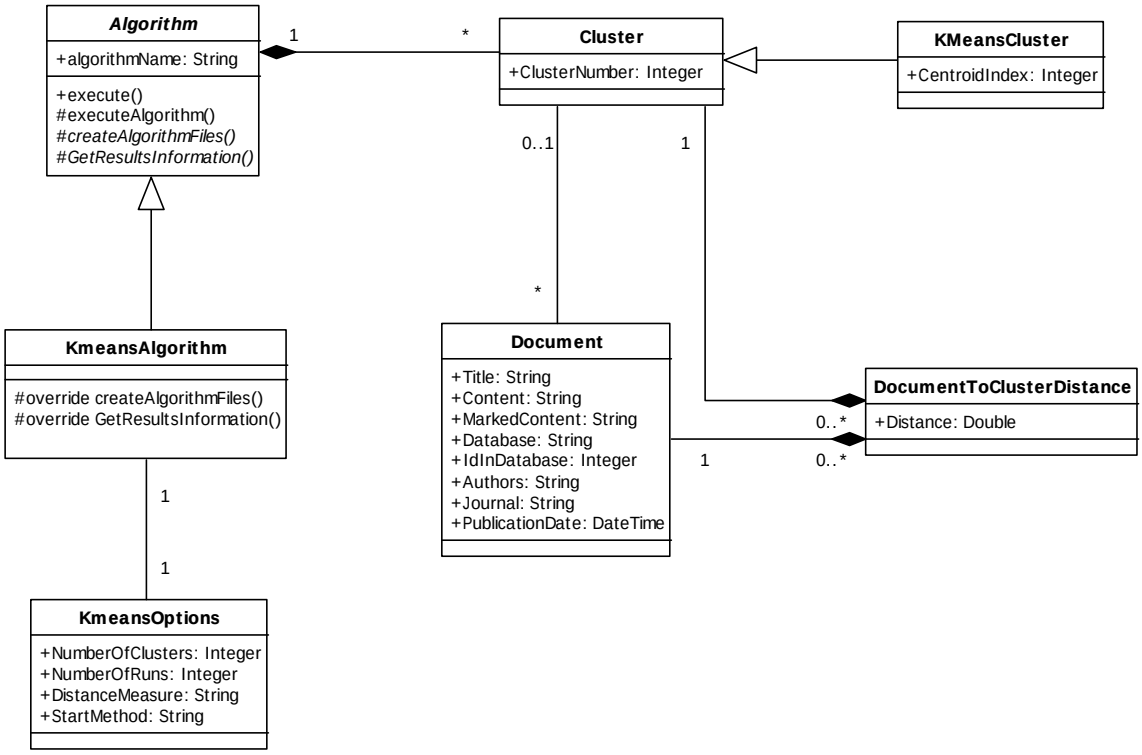


Figura 34 – Diagrama de classes relativo ao algoritmo *K-means*

5.3.2. *Fuzzy C-means*

O algoritmo *Fuzzy C-means* presente na aplicação faz uso da função *fcm* do Matlab.

Tal como o algoritmo *K-means*, para além do número de *clusters* no qual o conjunto de documentos deve ser dividido, existe um conjunto de opções que permitem aprimorar os resultados devolvidos pelo algoritmo. Essas opções são o *exponent*, o mínimo de melhoria em cada iteração do algoritmo e o número máximo de iterações do algoritmo. A opção *exponent* é a mais importante e tem como finalidade atribuir um peso à maior ou menor proximidade de um elemento a um *cluster*. As duas opções restantes são critérios de paragem do algoritmo. Um dos critérios de paragem é o valor atribuído como mínimo de melhoria entre iterações, sendo que essa melhoria é representada pela diminuição do valor de uma função de custo apresentada na equação (4). Este critério tem precedência sobre o segundo que consiste no número máximo de iterações do algoritmo.

Os resultados são colocados nos ficheiros representados na Figura 35. Tal como o algoritmo *K-means*, o algoritmo *Fuzzy C-means* cria o ficheiro *indices.txt* onde é apresentado o *cluster* a que cada documento pertence. O segundo ficheiro, *probabilidades.txt*, apresenta as probabilidades de um documento pertencer a um *cluster*. Nesse documento cada linha representa um *cluster* e cada coluna representa a probabilidade do documento pertencer a esse *cluster*. Os documentos mantêm a ordem pela qual se encontram na matriz *BOW*. Como se pode também verificar pela Figura 35, a soma das probabilidades de um documento pertencer aos vários clusters é sempre igual a 1.

Document	Cluster
Doc1	1
Doc2	2
Doc3	1
Doc4	1
Doc5	2
Doc6	2
Doc7	1

indices.txt

Clusters/Documents	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6	Doc7
Cluster 1	0,82	0,04	0,75	0,88	0,30	0,11	0,91
Cluster 2	0,18	0,96	0,25	0,12	0,70	0,89	0,09

Probabilidades.txt

Figura 35 – Exemplo de ficheiros de resultados criados pelo algoritmo *Fuzzy C-means*

Na Figura 36 é apresentado o diagrama de classes referente ao funcionamento do algoritmo *Fuzzy C-means* na aplicação. A informação presente nos ficheiros de resultados é colocada em objetos da classe *FuzzyCMeansCluster* e a probabilidade de cada documento pertencer a um *cluster* é colocada em objetos da classe *DocumentToClusterDistance* que no algoritmo *K-means* também são usados para colocar as distâncias dos documentos aos vários *clusters*.

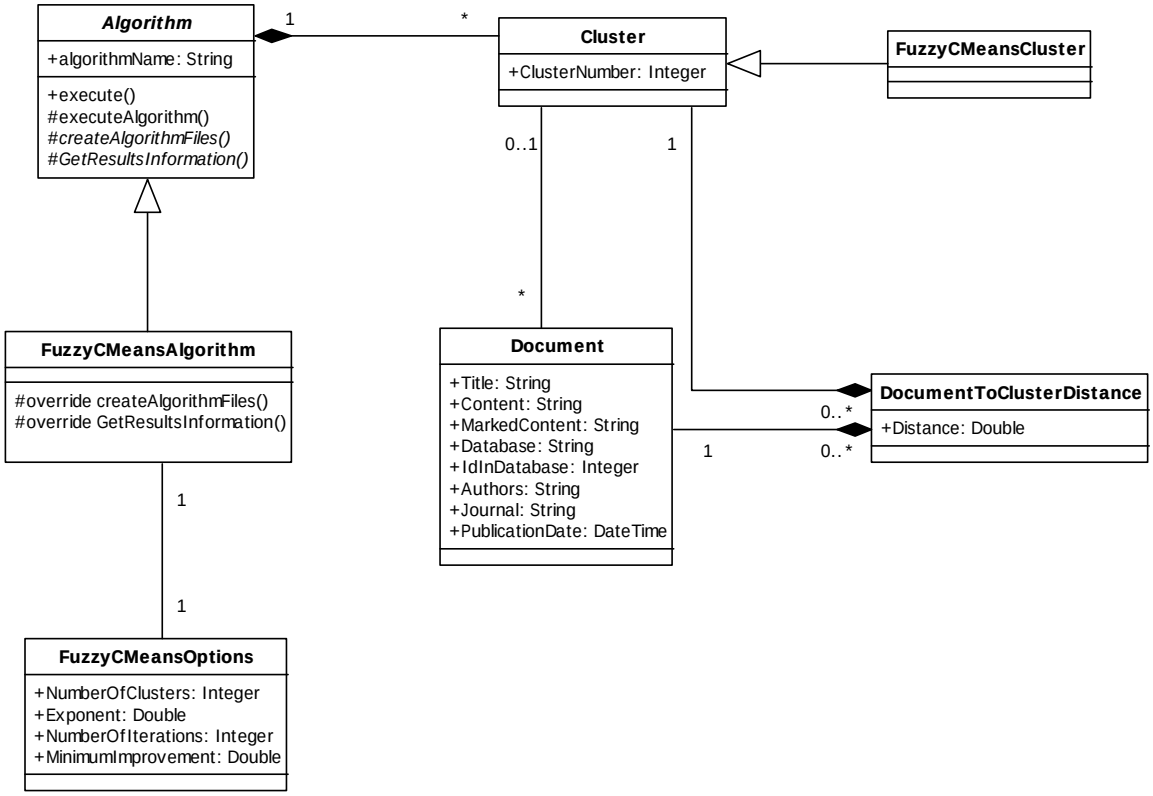


Figura 36 – Diagrama de classes relativo ao algoritmo *Fuzzy C-means*

5.3.3. *Subtractive*

Para o algoritmo *Subtractive* é usada a função *subclust* do Matlab. Esta função usa o algoritmo *Subtractive clustering* para determinar centros de clusters a partir de conjunto de dados. Após a definição dos centros é usada outra função Matlab, *pdist*, de forma a calcular para cada documento a distância a esses centros. A distância usada pela função *pdist* é a distância euclidiana. Desta forma é possível obter o *cluster* a que cada documento pertence e a distância de todos os documentos aos centros de todos os clusters.

As opções disponibilizadas na plataforma para este algoritmo são os parâmetros *influence range*, *quash factor*, *accept ratio* e *reject ratio*.

A opção *influence range* é a mais usada uma vez que permite definir um raio de ação para cada centro. Quanto maior for o valor dado a este parâmetro menor será o número de centros encontrados, sendo que no mínimo irá ser definido apenas um centro. Caso o valor do parâmetro *influence range* seja menor, o número de centros irá aumentar atingindo um limite máximo de centros igual ao número de documentos do *dataset*.

O parâmetro *quash factor* permite que documentos mais afastados de um centro de um *cluster* não façam parte desse *cluster*. Desta forma quanto maior for o valor dado ao parâmetro *quash factor* mais distantes estarão os centros uns dos outros.

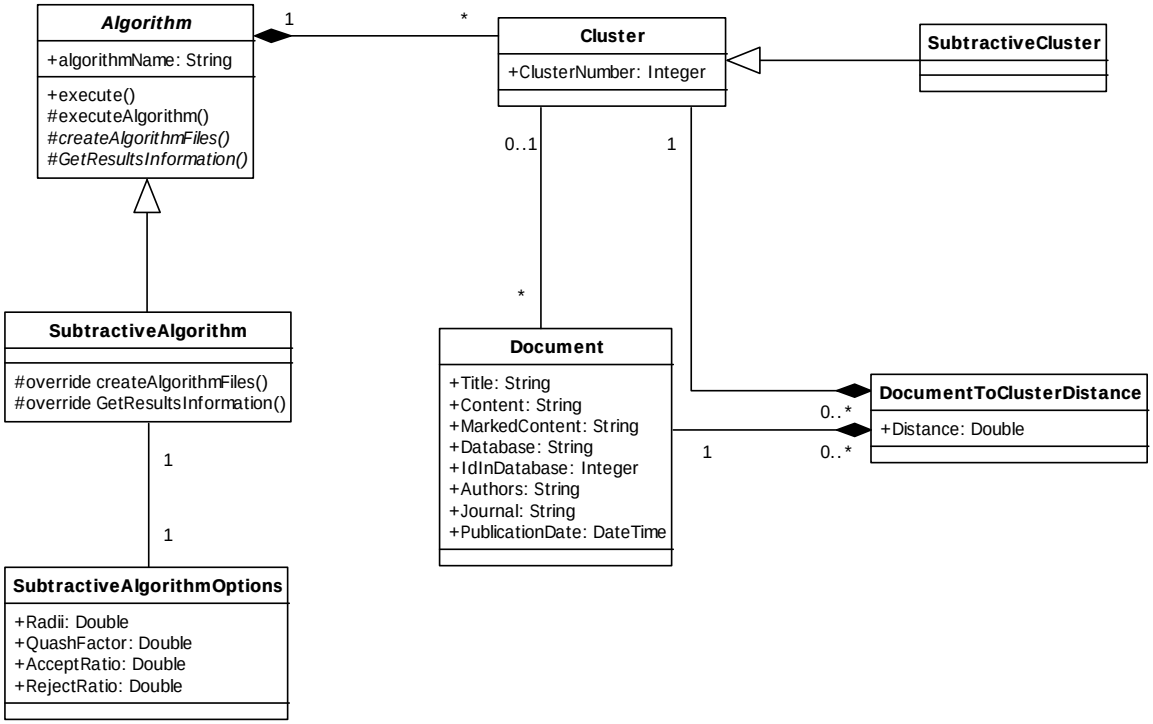


Figura 37 – Diagrama de classes relativo ao algoritmo *Subtractive Clustering*

Os parâmetros *accept ratio* e *reject ratio* permitem definir um valor que determina o potencial de aceitação ou rejeição para um ponto ser considerado um centro de um *cluster*. Desta forma o parâmetro *accept ratio* define o potencial mínimo para um ponto ser aceite como um centro

de um *cluster*, ao passo que o parâmetro *reject ratio* define o potencial máximo a partir do qual um ponto pode ser rejeitado como um centro de um *cluster*. Um valor elevado do *accept ratio* obriga a que apenas pontos com um grande potencial possam ser aceites como centros de um *cluster*. No caso do *reject ratio* um valor elevado indica que serão rejeitados pontos com pouco potencial. Diminuindo este valor pontos com pouco potencial já poderão ser aceites.

Os resultados obtidos serão iguais aos do algoritmo *K-means* existindo um ficheiro *indices.txt* e *distancias.txt* tal como são demonstrados na Figura 33. Também como no algoritmo *K-means* o diagrama de classes é semelhante como pode ser visto na Figura 37.

5.3.4. LIBSVM

A biblioteca LIBSVM permite realizar três tarefas distintas: usar o algoritmo com *cross-validation*, dando a possibilidade de testar vários parâmetros e verificar os resultados obtidos com esses parâmetros, treinar máquinas de vectores de suporte e classificar documentos recorrendo a essas SVMs previamente treinadas.

5.3.4.1. LIBSVM com *cross-validation*

O LIBSVM com *cross-validation* consiste na execução do algoritmo dividindo o conjunto de dados recebido em conjuntos de treino e conjuntos de teste. A divisão dos conjuntos é realizada de acordo com um valor *k* recebido no parâmetro *Cross Validation Fold*. Este valor vai dividir o conjunto de documentos em *k* partes, onde (*k*-1) partes são usadas para treino e a parte restante é usada para classificação, retornando a precisão da classificação efetuada. O processo é ainda realizado *k* vezes e a precisão final será a média das *k* precisões de todas as iterações realizadas.

Para além do parâmetro *Cross Validation Fold* existem ainda outros parâmetros relacionados com o treino de classificadores, e que podem ser definidos pelo utilizador. É possível definir assim o tipo de *kernel* utilizado, os parâmetros *cost*, *gamma*, *degree* e *coef0*, cuja utilização depende do tipo de *kernel* escolhido, e ainda o tamanho da cache utilizado na execução do algoritmo.

Segundo (Hsui, Chih-Chung, & Chih-Jen, 2003) as melhores opções a serem utilizadas são o *kernel* linear, no caso do número de dimensões (*features*) ser superior ao número de elementos do conjunto de dados (documentos), ou o *kernel* de função de base radial nos restantes casos. Em termos de valores para os parâmetros *cost* e *gamma* são aconselhados vários testes com pequenos incrementos em ambos os valores. Esses incrementos para o *cost* são de 2^{-5}, 2^{-3}, ..., 2^{15} e para o *gamma* de 2^{-15}, 2^{-13}, ..., 2^{3}.

Como em todos os algoritmos de classificação, a função *createAlgorithmFiles* vai criar todos os documentos necessários para a função LIBSVM do Matlab. No entanto para além dos ficheiros que contêm os parâmetros para o LIBSVM é criado um novo ficheiro que irá catalogar cada documento com a classe a que este pertence. Este ficheiro será constituído por várias linhas, que representam os documentos na mesma ordem em que estes se encontram na matriz BOW, e cada linha tem um identificador numérico que representa a classe do documento.

Depois de executado o algoritmo, a função *GetResultsInformation* vai ler os resultados a partir do ficheiro de texto *resultados.txt*, que é apresentado na Figura 38. Este caso apresenta os resultados relativos a um *dataset* com duas classes. Como se pode verificar para além dos valores presentes nas duas classes também é apresentada uma secção para os resultados globais. Em cada secção é apresentado um conjunto de valores que pertencem aos seguintes indicadores pela ordem em que aparecem no ficheiro: *True Positives*, *False Positives*, *True Negatives*, *False Negatives*, *Accuracy*, *Sensitivity*, *Specificity*, *Precision* e *F-Measure*.

```
****************** Class 1 ******************
94
5
195
2
97.635135
97.916667
97.500000
94.949495
96.410256
****************** Class 2 ******************
96
3
193
4
97.635135
96.000000
98.469388
96.969697
96.482412
****************** Overall ******************
...
```

resultados.txt

Figura 38 - Exemplo do ficheiro de resultados criados pelo LIBSVM com *cross-validation*

Esta informação é colocada numa lista de objetos *LibsvmClassResult* que é apresentada no diagrama de classes da Figura 39, juntamente com as restantes classes usadas neste processo.

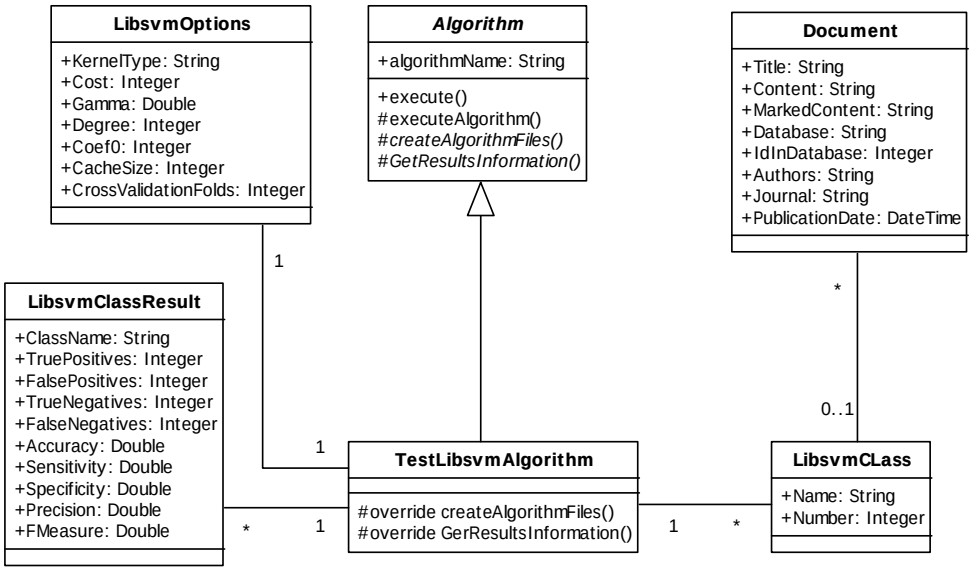


Figura 39 - Diagrama de classes relativo ao processo de LIBSVM com *cross-validation*

5.3.4.2. Treino de Máquinas de Vectores de Suporte

A primeira fase do treino de SVMs é idêntica à realizada no uso do LIBSVM com *cross-validation* uma vez que os parâmetros a definir por parte do utilizador são os mesmos, com exceção do parâmetro *Cross Validation Fold*. Devido à ausência deste parâmetro o algoritmo vai usar todos os documentos no treino das SVMs.

O resultado final do treino no LIBSVM será um classificador presente num ficheiro Matlab. Através da função *GetResultsInformation* este ficheiro será lido e guardado na base de dados da aplicação. Outro ficheiro a ser guardado é o ficheiro com todos os termos extraídos e usados no treino do classificador. Estes termos são guardados porque é necessária, na classificação de documentos, que a extração de características e respectiva matriz *BOW* correspondam àquela que foi usada no treino. Devido a este fator, quando na criação da matriz *BOW* é usado o método *TFIDF*, também os valores com a importância de cada termo dentro do conjunto de documentos (*IDF*) são guardados num ficheiro que por sua vez é guardado na base de dados. Neste ficheiro cada linha corresponde ao termo presente na mesma posição do ficheiro que contém todas as *features* extraídas para o treino. O valor em cada linha corresponde ao *IDF* relativo à extração efetuada no treino. Desta forma na classificação de documentos é possível recriar a matriz *BOW* recorrendo a estes valores.

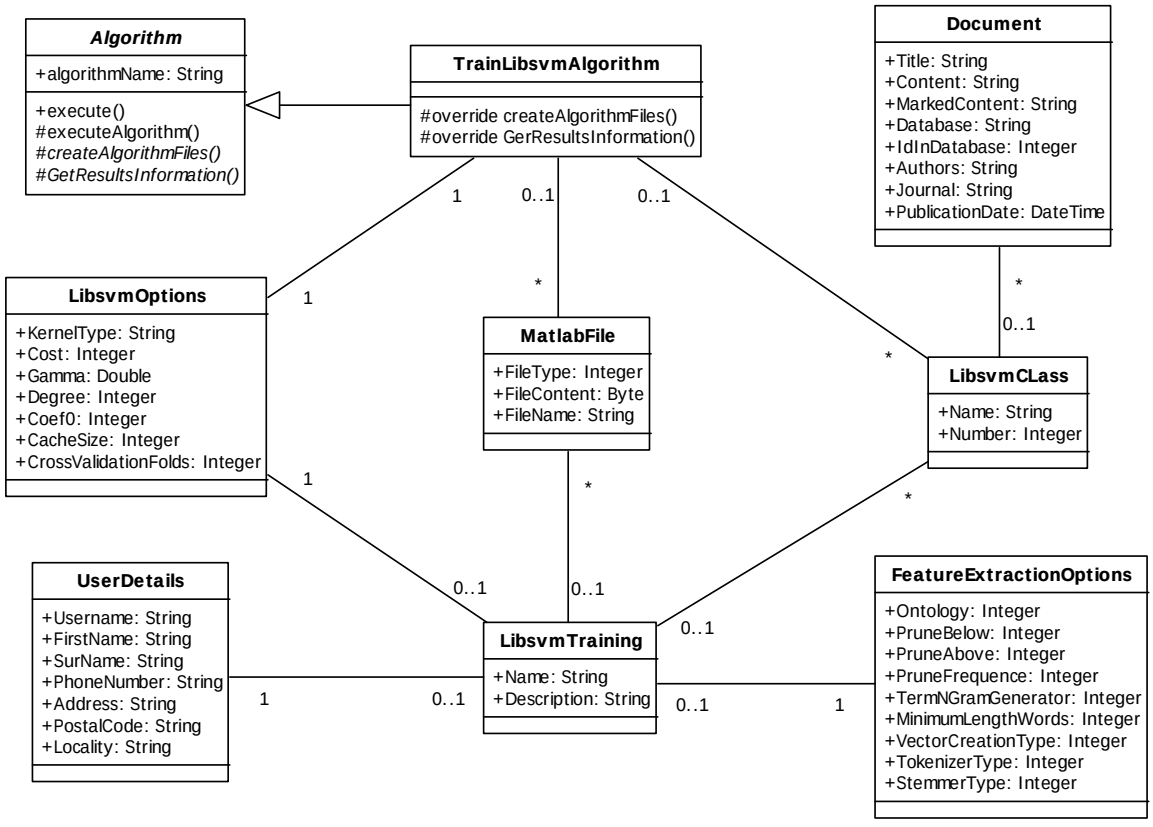


Figura 40 – Diagrama de classes relativo ao processo de treino de SVMs

Como se pode verificar pela Figura 40, para além dos ficheiros mencionados atrás, existe outra informação que é guardada na classe *LibsvmTraining*. Uma das informações mais importantes são os parâmetros usados na extração de *features* uma vez que a extração de conhecimento dos documentos a serem classificados deve usar estes parâmetros.

Em relação à restante informação que é guardada, existem as classes para as quais os documentos devem ser classificados, as opções do algoritmo LIBSVM usadas no treino e o utilizador que realizou esse mesmo treino.

5.3.4.3.Classificação de documentos

No processo de classificação de documentos existe uma diferença relativamente aos restantes algoritmos no que diz respeito à extração de *features*. Como já foi referido no treino das máquinas de vectores de suporte é necessário que na classificação de documentos a matriz *BOW* contenha os mesmos termos que foram usados no treino desses classificadores. É por essa razão que depois de realizado o treino de cada SVM é guardado na base de dados um ficheiro com todas as *features* extraídas e outro com os valores de *IDF* de todas as *features*, caso tenha sido usado o método *TFIDF* na criação dos vectores. Para além destes ficheiros, também foram guardadas as opções usadas na extração de características. Desta forma o utilizador apenas terá de escolher os documentos que quer classificar e o classificador que pretende usar para classificar esses documentos.

Após a obtenção dos documentos existem duas formas distintas de conseguir recriar a matriz *BOW* usada no treino. Estas formas dependem da ontologia que foi usada nessa extração de *features* existindo por isso, de um lado, a extração sem recurso a qualquer ontologia ou através da ontologia GO, e do outro a extração através da ontologia Merops.

No caso de no treino não ter sido usada qualquer ontologia, ou ter sido usado a ontologia GO a matriz é criada pelo *WVTool*. Esta funcionalidade teve de ser adicionada à biblioteca uma vez que não está disponível na versão original da mesma. Neste caso a principal diferença relativamente a uma extração de *features* normal consiste em criar logo no inicio a lista com os termos extraídos no treino do classificador em vez da adição dos termos a essa lista à medida que são segmentados os documentos. Ao longo dessa segmentação não são adicionados novos termos e apenas são considerados os que estão presentes nessa lista.

Existe ainda o caso especial da criação dos vectores ter sido realizado através do método *TFIDF*. Neste caso é necessário ter em atenção a importância que os termos extraídos tinham no conjunto de documentos usado no treino, sendo por essa razão que o valor *IDF* para cada termo foi guardado. Na altura de criação dos vectores que formam a matriz *BOW* é usada a frequência de uma *feature* no documento a classificar e multiplicado esse valor pela importância que a *feature* apresentava no conjunto de documentos de treino ao invés da importância que apresenta no conjunto de documentos a classificar.

No caso de no treino ter sido usada a ontologia Merops na extração de *features* é usado o mesmo processo que foi usado no caso anterior não recorrendo, no entanto, ao *WVTool*. Também aqui é criada uma lista inicial com os termos do treino e apenas serão contabilizados esses termos. Também no caso especial de criação de vectores da matriz *BOW* com o método *TFIDF*, o processo usado é mesmo.

Com a matriz criada, são preparadas de seguida todas as dependências para a classificação dos documentos. A função *createAlgorithmFiles* encarrega-se dessa tarefa e coloca na pasta de

resultados o ficheiro Matlab que irá classificar os documentos, bem como o modelo com o classificador treinado.

Tendo todos os ficheiros necessários preparados, é executado o algoritmo, sendo os resultados apresentados em dois ficheiros de texto: *resultados.txt* e *probabilidades.txt*. Estes ficheiros, apresentados na Figura 41 são bastante similares aos que são criados pelo algoritmo *K-means*. No caso do ficheiro *resultados.txt* cada linha corresponde a um documento e o valor nessa linha corresponde ao identificador da classe atribuída a esse documento.

Document	Class
Doc1	1
Doc2	2
Doc3	1
Doc4	1
Doc5	2
Doc6	2
Doc7	1

resultados.txt

Documents/Class	C1	C2
Doc1	0,99	0,01
Doc2	0,15	0,85
Doc3	0,77	0,23
Doc4	0,65	0,45
Doc5	0,06	0,94
Doc6	0,03	0,97
Doc7	0,88	0,12

probabilidades.txt

Figura 41 – Exemplo dos ficheiros de resultados criados pelo algoritmo de classificação

No caso do ficheiro *probabilidades.txt* cada linha corresponde também a um documento e cada coluna a uma classe. Os valores em cada célula correspondem à probabilidade de um documento pertencer a essa classe. A soma das probabilidades de um documento pertencer a cada uma das classes deve ser igual a 1.

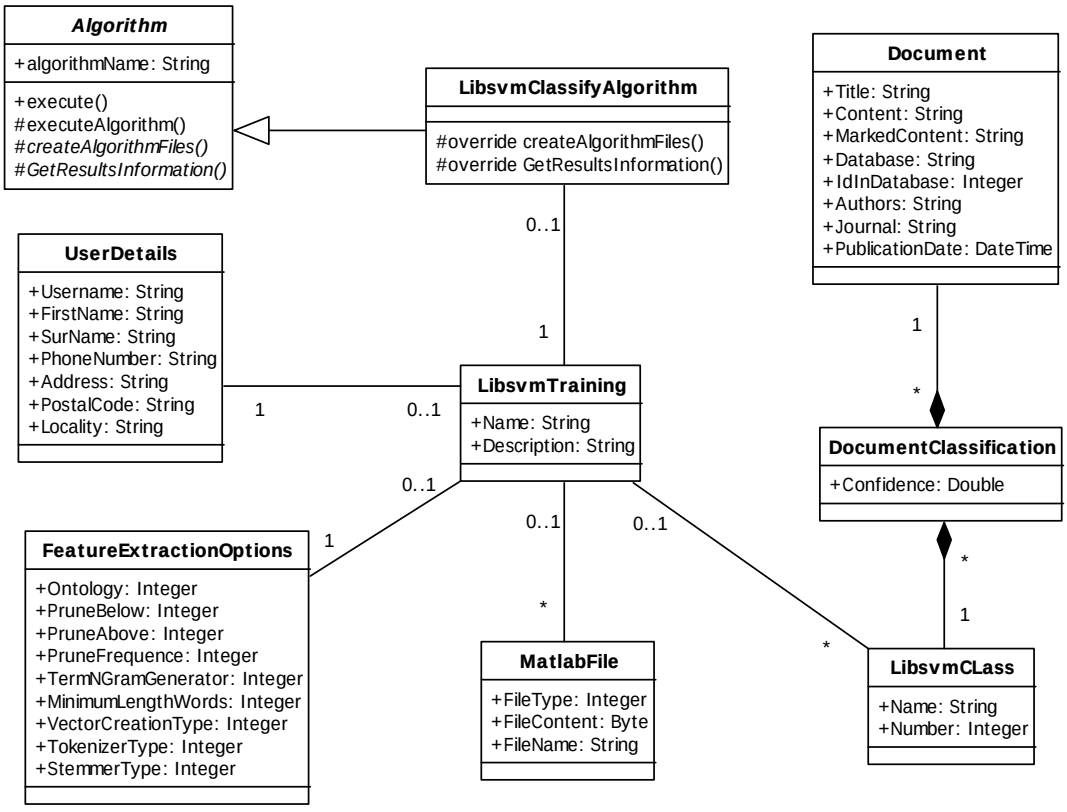


Figura 42 – Diagrama de classes relativo ao processo de classificação de documentos

A informação presente nestes documentos é por fim lida e colocada em objetos da classe *DocumentClassification*. Como se pode ver pela Figura 42, com o diagrama de classes referente ao processo de classificação, esta classe é composta por um objeto *LibsvmClass*, um objeto *Document* e o valor de confiança associado ao documento pertencer à classe em questão.

Apesar do LIBSVM usar a estratégia um contra um em problemas multiclasse, que conta com um sistema de votos na classificação dos documentos ao invés do calculo de probabilidades, esta biblioteca realiza ainda assim uma estimativa dessa probabilidade bastando para isso usar o parâmetro *b* do algoritmo (*probability estimates*).

6. ANÁLISE DE RESULTADOS

Neste capítulo são apresentados vários testes e respectivos resultados que pretendem aferir a qualidade dos algoritmos, definir os melhores parâmetros a serem utilizados na fase de pré-processamento de documentos e verificar o efeito das ontologias na classificação final dos documentos. Em relação aos parâmetros usados pelos diferentes algoritmos de classificação, não será realizado um estudo exaustivo para definir os melhores valores uma vez que a fase de execução dos algoritmos é bastante influenciada pelo pré-processamento dos documentos. No entanto, para cada extração de *features* efetuada foi executado várias vezes o mesmo algoritmo, utilizando diferentes valores em alguns dos seus parâmetros. Desta forma foi escolhido em cada algoritmo o melhor resultado entre todas as execuções e guardados os valores dos parâmetros utilizados.

O algoritmo *K-means* é o único para o qual não serão usados diferentes valores nos seus parâmetros. Para este algoritmo será usada a distância euclidiana e a inicialização dos centros será efetuada com o método *sample* (descrito na secção 5.3.1). Para além destes valores o algoritmo será repetido 10 vezes.

O algoritmo *Fuzzy C-means* irá apenas alterar os valores do parâmetro *exponent* através de incrementos de uma unidade desde o valor 2 até ao valor 25. Em relação aos critérios de paragem do algoritmo será sempre usado um número máximo de 1000 iterações ou uma melhoria mínima de 0,00001 no final de cada iteração.

O algoritmo *Subtractive clustering* é um caso especial uma vez que não necessita de uma definição prévia do número de *clusters* em que os documentos devem ser distribuídos. Por esta razão não será testado como os restantes algoritmos de classificação, uma vez que é impossível obter um valor de acerto se o número de clusters retornado não iguala o valor no qual os documentos estão realmente divididos. Desta forma será realizado um teste diferente que vai incidir sobre a sensibilidade do algoritmo à mudança de valores em um dos seus parâmetros. Esta sensibilidade vai ser atestada pelo número de *clusters* no qual os documentos serão divididos. Para este caso será criada uma secção à parte no qual serão analisados os seus resultados.

No LIBSVM serão usados os valores aconselhados pelos seus criadores e indicados na secção LIBSVM com *cross-validation* do Capitulo 5. Desta forma será sempre usado o *kernel* linear quando o número de *features* for superior ao número de documentos e o *kernel* de função de base radial quando tal não acontece. Em relação aos valores de *cost* e *gamma* serão incrementados em 2^{-5}, 2^{-3}, ..., 2^{15} e 2^{-15}, 2^{-13}, ..., 2^{3} respetivamente.

Para aferir a qualidade da classificação dos documentos vão ser tidos em conta vários fatores. Esses fatores são o grau de acerto do algoritmo, o seu tempo de processamento e a sensibilidade desses algoritmos de classificação relativamente às diferentes formas de extração de *features*.

Devido à natureza dos algoritmos de aprendizagem não supervisionada, onde não existe etiquetação prévia dos documentos, estes não retornam o grau de acerto do algoritmo através de indicadores como a precisão de cada *cluster* e a qualidade global da classificação obtida. Por esta razão é necessário usar uma forma de obter estes indicadores.

O primeiro passo para obter a precisão de cada *cluster* é encontrar o número de positivos verdadeiros (*TP*, do inglês *true positives*) e de positivos falsos (*FP*, do inglês *false positives*). Os *TP* são os documentos corretamente atribuídos pelo algoritmo a um *cluster* sendo os *FP* aqueles que são erradamente atribuídos. No entanto estes valores apenas podem ser conhecidos se cada *cluster* estiver associado a uma classe de documentos já conhecida à partida.

Para um cluster ficar associado a uma das classes de documentos que compõem o *dataset* é necessário verificar, entre os documentos atribuídos ao *cluster*, a classe que é mais representada, ou seja, a classe que tem um maior número dos seus documentos atribuídos a esse *cluster*.

Depois de obter os valores *TP* e *FP* é possível calcular a precisão de cada *cluster* e a qualidade global dos resultados obtidos recorrendo respectivamente às equações (10) e (11)

$$\textit{Precisão (Cluster i)} = \frac{\text{TP i}}{\text{Total de documentos i}} * 100 \quad (10)$$

$$\textit{Qualidade} = \frac{\text{TP}}{\text{Total de documentos}} * 100 \quad (11)$$

Para a realização dos testes foram usados vários *datasets* curados pela Merops de forma a que as ontologias possam ser testadas em documentos da área onde se inserem.

Tabela 29 – *Dataset 1*

Classes	Merops ID	Nº documentos
Renin	**A01.007**	**100**
Cathepsin B	**C01.060**	**100**
Calpain 1	**C02.001**	**100**
Aminopeptidase N	**M01.001**	**100**
ADAMTS13 peptidase	**M12.241**	**100**
Thrombin	**S01.217**	**100**
Total		**600**

Tabela 30 – *Dataset 2*

Classes	Merops ID	Nº documentos
Renin	**A01.007**	**125**
Cathepsin B	**C01.060**	**50**
Calpain 1	**C02.001**	**75**
Aminopeptidase N	**M01.001**	**75**
ADAMTS13 peptidase	**M12.241**	**100**
Thrombin	**S01.217**	**150**

Total	**600**

Tabela 31 – Dataset 3

Classes	Merops ID	Nº documentos
Pepsin A	**A01.001**	**50**
Presenilin 1	**A22.001**	**50**
Papain	**C01.001**	**50**
Neprilysin	**M13.001**	**50**
Prolyl oligopeptidase	**S09.001**	**50**
Total		**250**

Tabela 32 – *Dataset 4*

Classes	Merops ID	Nº documentos
HIV-1 retropepsin	**A02.001**	**50**
Cathepsin L	**C01.032**	**50**
Carboxypeptidase A1	**M14.001**	**50**
Lon-A peptidase	**S16.001**	**50**
Hepacivirin	**S29.001**	**50**
Total		**250**

Tabela 33 – *Dataset 5*

Classes	Merops ID	Nº documentos
Pepsin A	**A01.001**	**100**
Presenilin 1	**A22.001**	**25**
Papain	**C01.001**	**70**
Neprilysin	**M13.001**	**25**
Prolyl oligopeptidase	**S09.001**	**30**
Total		**250**

6.1. Resultados sem o uso de ontologias

Nesta primeira fase são apresentados testes realizados com diferentes valores nos parâmetros de pré-processamento dos documentos e sem recorrer a qualquer uma das ontologias presentes na aplicação.

6.1.1. *Pruning*

Como foi visto no Capitulo 5 existem três técnicas diferentes de *pruning*, sendo elas o *prune below,* o *prune above* e o *pruning frequence*.

Para os dois primeiros casos não serão realizados testes uma vez que a sua utilização depende em demasia de um prévio conhecimento do número de documentos pertencentes a cada grupo e de um *dataset* balanceado.

Analisando o *dataset 1*, e considerando que existe um conhecimento prévio do número documentos pertencentes a cada peptidase, é possível definir valores que possibilitem uma boa extração de *features*. Tendo em conta que este *dataset* é balanceado, contendo 100 documentos para cada peptidase, um exemplo de valores a utilizar seria o *prune below* com valor 90 e o *prune above* com valor 110, ou seja, seriam removidas todas as palavras que não estivessem presentes no mínimo em 90 documentos e todas as que estivessem presentes em mais 110 documentos. Com isto a probabilidade de serem extraídas caraterísticas que melhor definem cada classe é bastante elevada já que palavras presentes em grande parte dos documentos do *dataset* e que não ajudam na sua distinção, são removidas, bem como outras que apenas estão presentes num numero reduzido de documentos.

Se por outro lado tivermos em conta o *dataset 2*, não balanceado, torna-se impossível escolher um valor para cada um dos parâmetros que beneficie a recolha de informação relevante para a classificação de documentos. Se por exemplo forem definidos os mesmos valores de *prune below* e de *prune above* é possível recolher toda a informação importante de classes com 100 documentos, mas o mesmo não irá suceder em classes com diferente número de documentos uma vez que a informação pertinente dessas classes será removida.

Existe ainda o caso de não existir um conhecimento prévio do número de documentos pertencentes a cada uma das classes de um *dataset*. Neste caso torna-se impossível definir valores que possibilitem uma filtragem racional da informação.

Em relação ao *pruning frequence*, é possível averiguar se existe algum padrão que permita obter bons resultados. Para isso foram realizadas várias extrações de features no *dataset 1* incrementando o valor de *prune frequence* em 10 unidades até o seu valor ser igual ao número de documentos. Para cada extração de *features* foram executados os vários algoritmos de classificação e obtiveram-se os resultados presentes na Figura 43.

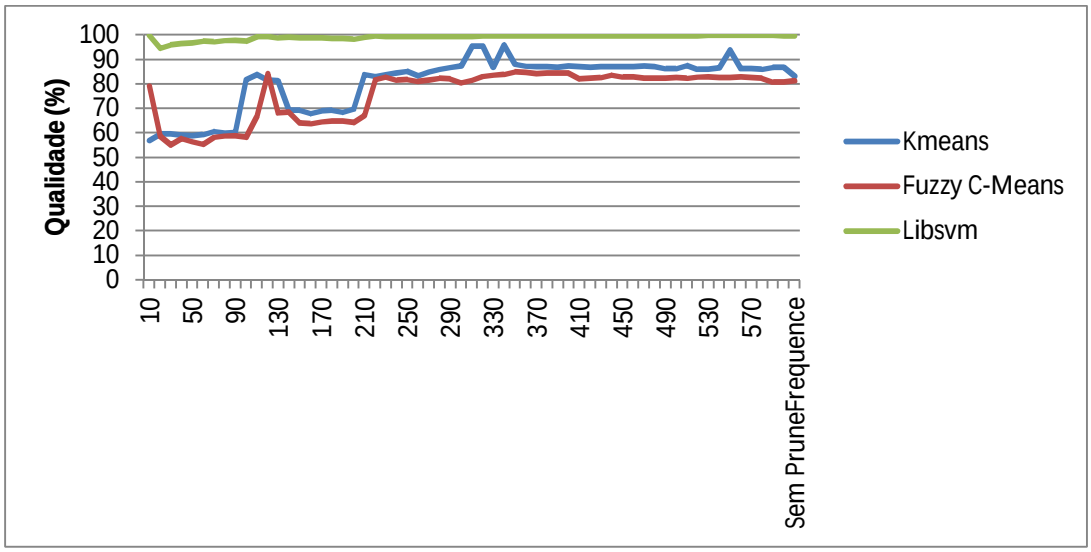


Figura 43 – Gráfico com resultados de todos os classificadores para diferentes valores de *pruning frequence* no *dataset 1*

Pelos resultados observados não se consegue verificar nenhum padrão interessante, existindo mesmo grandes variações de percentagens nos algoritmos *K-means* e *Fuzzy C-Means*. Como

existe uma grande imprevisibilidade na extração de caraterísticas usando este parâmetro, é normal que tal aconteça.

6.1.2. *Ngram*

O *ngram* é um parâmetro que permite adicionar valor a uma extração de *features* através da criação de características constituídas por mais do que uma palavra. Os termos compostos por mais do que uma palavra são bastante frequentes na área biomédica e numa grande parte dos casos são mesmo determinantes para a classificação de um documento. No entanto existe uma desvantagem relacionada com o elevado número de *features* extraídas no final do pré-processamento dos documentos que leva a um aumenta no tempo de computação dispendido na classificação do conjunto de documentos. Devido a este fator foram utilizados *datasets* de menor dimensão uma vez que o tempo de processamento é enorme.

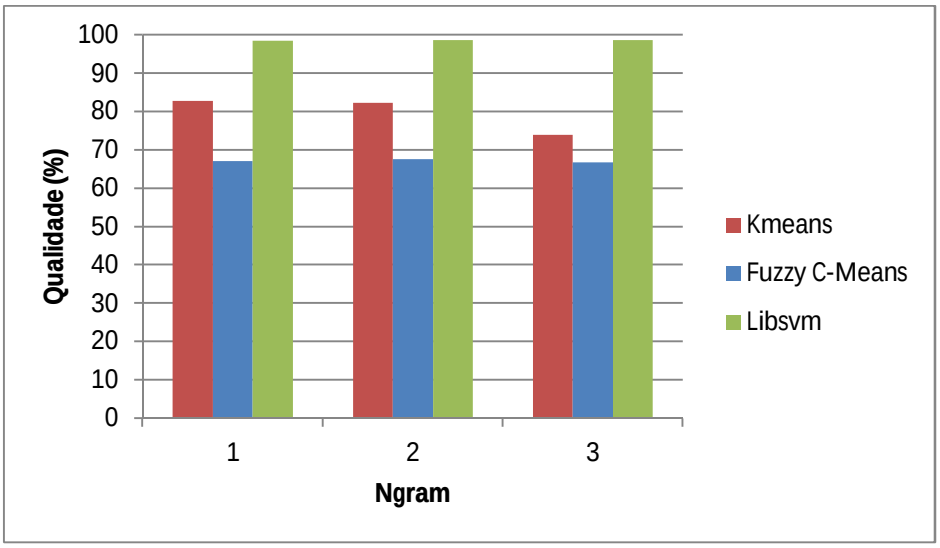


Figura 44 – Gráfico com o resultado dos vários classificadores usando vários *ngram* no *dataset 3*

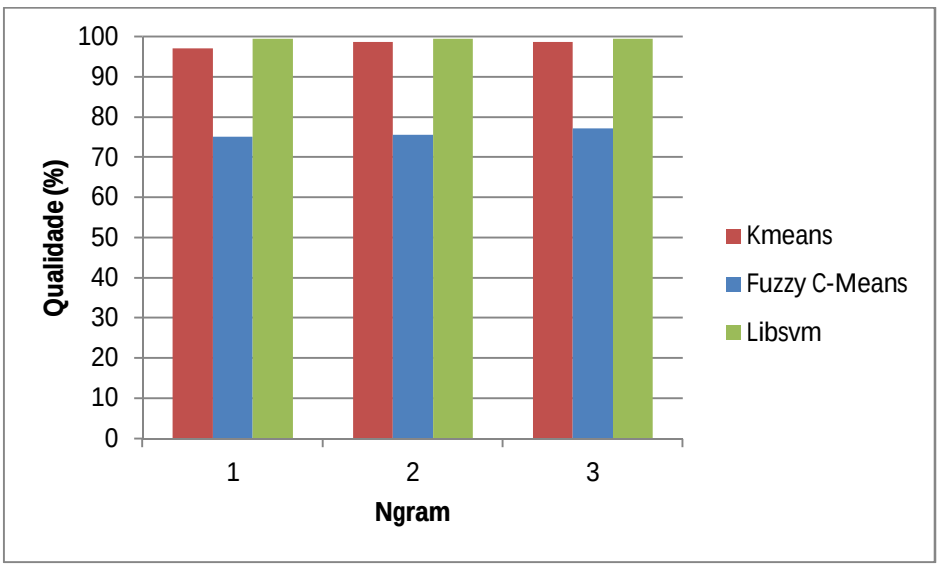


Figura 45 - Gráfico com o resultado dos vários classificadores usando vários *ngram* no *dataset 4*

Como se pode constatar pelos resultados da Figura 44 e da Figura 45 a adição de *ngrams* não se traduziu numa melhoria dos resultados. Mesmo selecionando características importantes para a classificação dos documentos essa melhoria não se verificou porque também são adicionadas demasiadas características que não contribuem positivamente para essa classificação. Este aspecto pode ser comprovado na Tabela 36 e na Tabela 37 através da diferença de *features* extraídas com 1, 2 ou 3 *ngrams*. Nessas tabelas também se pode comprovar o aumento do custo computacional, através do tempo de execução dos algoritmos, à medida que é incrementado o valor de *ngrams*.

6.1.3. Tipo de criação de vectores

Um dos parâmetros de extração de características é o tipo de criação dos vectores para a matriz *BOW*. Como referido anteriormente a opção *TermOccurence* era aquela que mostrava uma maior limitação por apenas ter em atenção o número de vezes que um termo aparecia num documento. Os resultados apresentados nos gráficos da Figura 46, Figura 47, Figura 48 e Figura 49 comprovam exatamente essa ideia por serem razoavelmente mais baixos que os restantes.

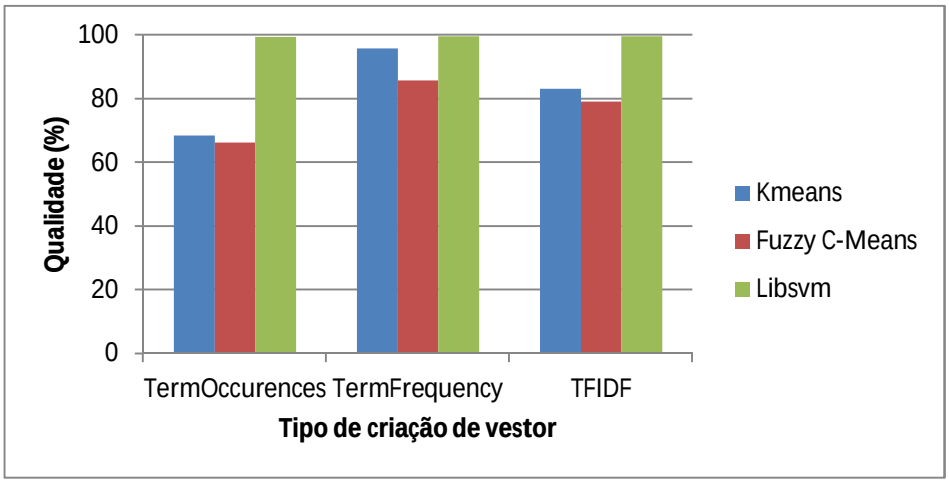


Figura 46 – Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz BOW no *Dataset 1*.

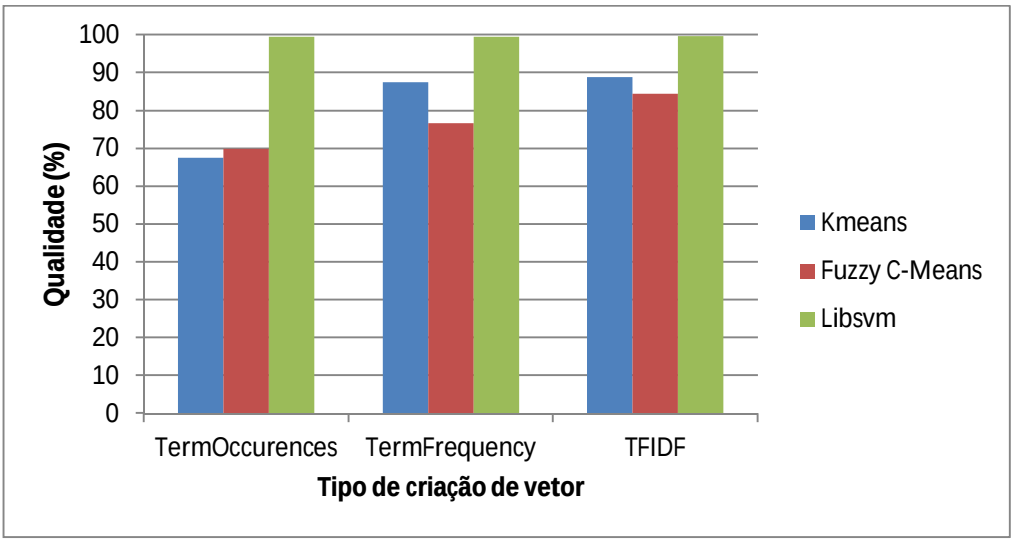


Figura 47 - Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 2*.

Em relação às opções *TermFrequency* e *TFIDF* os resultados diferiram conforme os *datasets* usados. Através dos *datasets* balanceados, *Dataset 1* e *Dataset 3*, é possível constatar pela Figura 46 e Figura 48 que a opção *TermFrequency* retorna melhores resultados do que as restantes opções. Por outro lado na Figura 47 e Figura 49 usando os *datasets* não balanceados, *Dataset 2* e *Dataset 5*, verifica-se que a opção *TFIDF* é aquela que apresenta uma melhor qualidade de resultados obtidos pelos algoritmos.

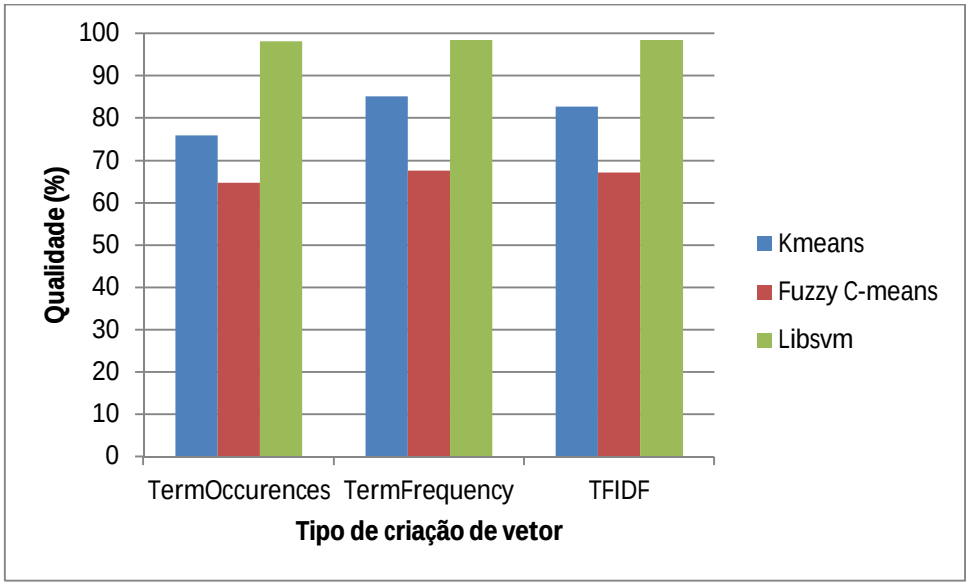


Figura 48 – Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 3*.

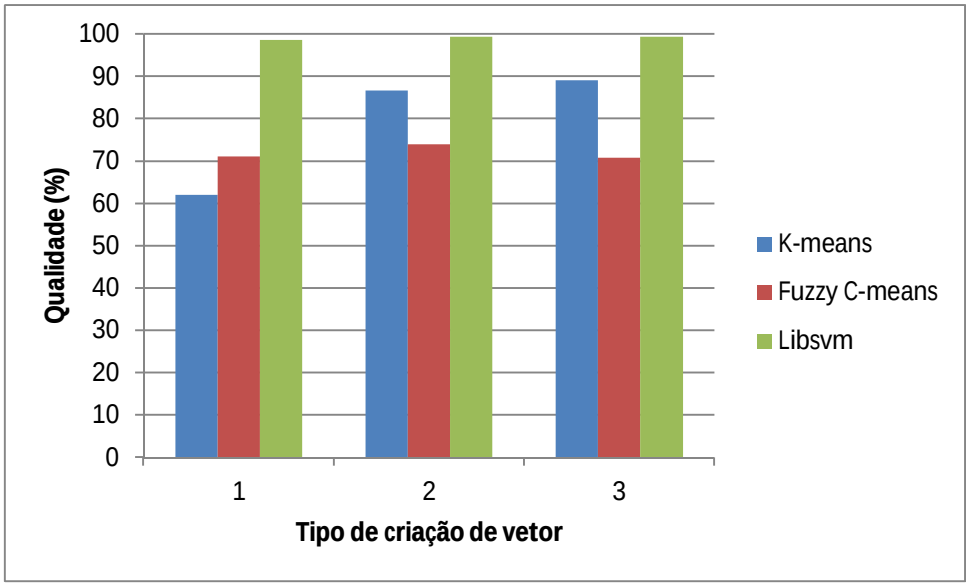


Figura 49 - Gráfico com o resultado de vários classificadores usando diferentes tipos de criação da matriz *BOW* no *Dataset 5*.

6.2. Resultados com ontologias

Nesta secção são apresentados alguns testes realizados sem o uso de qualquer ontologia e com as ontologias Gene Ontology e Merops. A extração de *features* sem recurso a ontologias é realizada com os seguintes parâmetros: sem qualquer tipo de cortes (*pruning*), com

SimpleTokenizer e *DummyStemmer* na segmentação dos documentos, com termos constituídos por 1 *ngram* e com o a criação dos vectores da matriz *BOW* usando o método *TFIDF*.

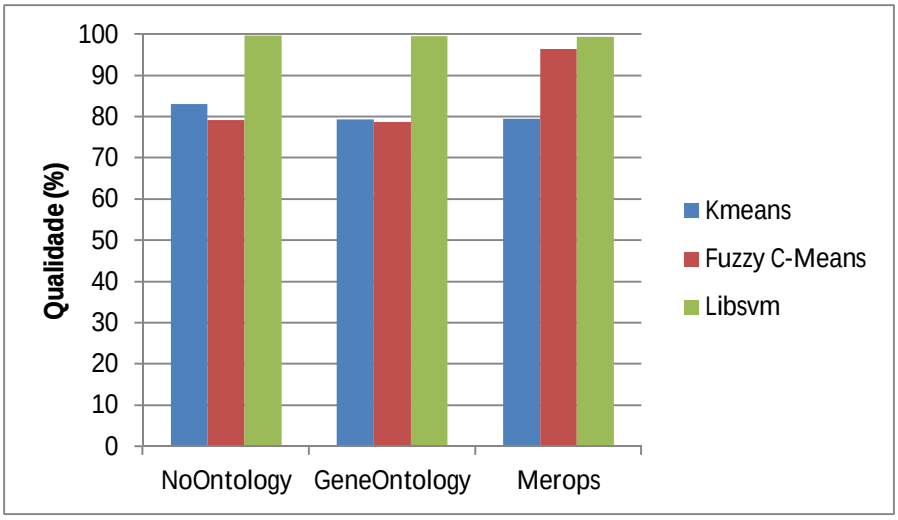


Figura 50 - Gráfico com o resultado dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 1*

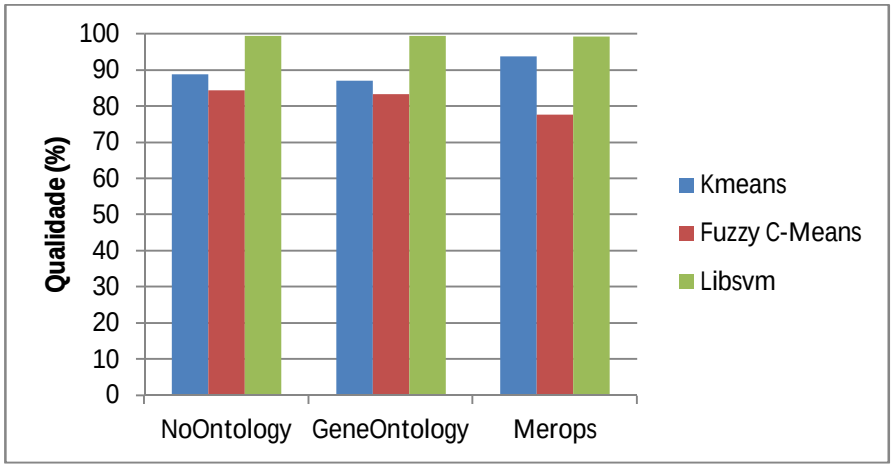


Figura 51 - Gráfico com o resultado dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 2*

Através dos resultados obtidos com dois *datasets* diferentes, visíveis nos gráficos da Figura 50 e Figura 51, é possível reparar que não existem grandes alterações relativamente à qualidade de classificação dos diferentes algoritmos. A única exceção acontece com a ontologia Merops em relação ao algoritmo *Fuzzy C-Means*, onde num caso melhora bastante a classificação e no outro piora.

No entanto existem diferenças substanciais em relação a outros indicadores, como é o caso do número de *features* extraídas e do tempo de processamento dos algoritmos. Através do uso de ontologias é possível verificar uma diminuição substancial em ambos esses indicadores. Já entre as duas ontologias a Merops é aquela que apresenta melhores resultados nesse sentido. Na Figura 52 e Figura 53 podem ser vistos estes resultados relativamente ao tempo de execução dos diferentes algoritmos. Em relação ao número de *features* extraídas é possível consultar essa informação na Tabela 42 e Tabela 43.

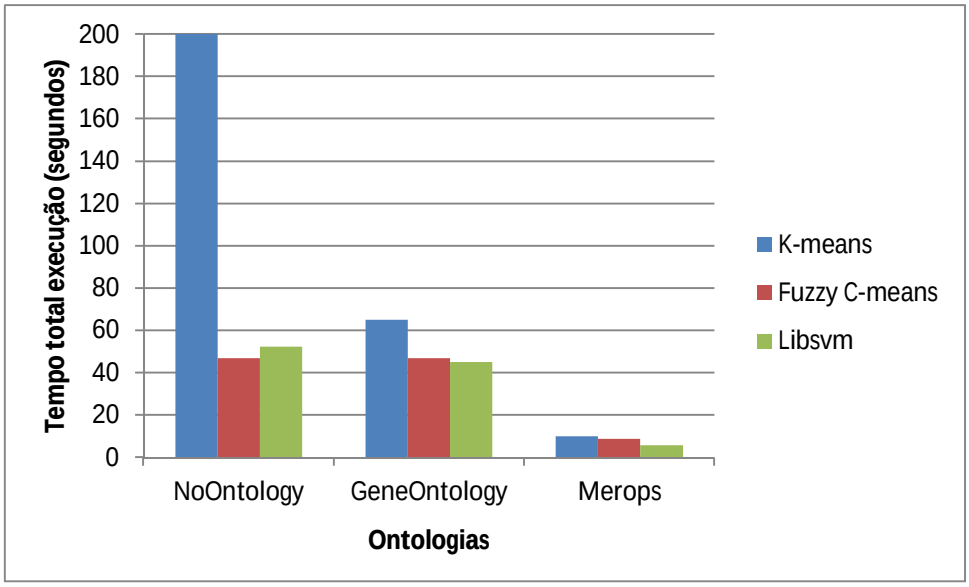


Figura 52 – Tempo total de execução dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 1*

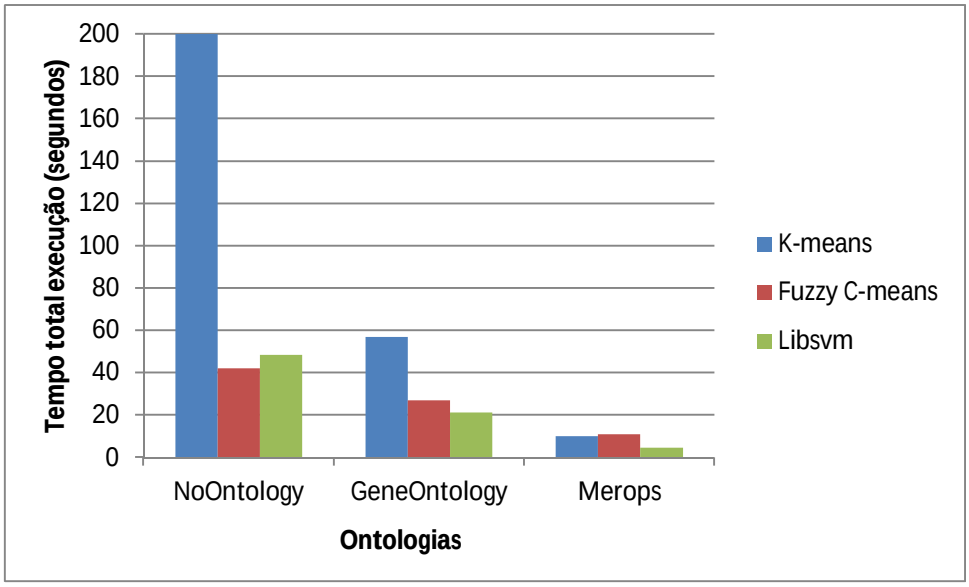


Figura 53 - Tempo total de execução dos vários classificadores em extrações de *features* sem recurso a ontologia, com Gene Ontology e com Merops Ontology no *Dataset 2*

6.3. *Subtractive clustering*

Como foi constatado anteriormente o algoritmo *Subtractive clustering* é um caso especial devido à sua habilidade de definir por si mesmo o número de *clusters* em que um conjunto de documentos deve ser dividido. Desta forma os testes realizados são centrados numa vertente diferente da que tem sido usada até ao momento.

Como já foi indicada anteriormente, a forma escolhida para testar uma possível melhoria na classificação de documentos através do algoritmo *Subtractive clustering*, é verificando a sensibilidade do algoritmo à mudança de valores de um dos seus parâmetros relativamente ao número de *clusters* criados no final. O parâmetro alterado é o parâmetro *radii*, devido à sua grande influência no resultado final do algoritmo uma vez que define o raio de cada *cluster* a

partir do seu centro. Desta forma à medida que esse valor é aumentado, o número de *clusters* diminui.

Nestes testes os valores do parâmetro *radii* foram incrementados 0.05 em cada iteração, começando no valor 0.05 e acabando no valor para o qual o resultado do algoritmo integra todos os documentos em apenas um *cluster*.

Como se pode comprovar na Figura 54 e Figura 55, os resultados obtidos demonstram uma maior sensibilidade do algoritmo quando não é usada qualquer ontologia ou quando é usada a ontologia Gene Ontology na extração de *features*. Em ambos estes casos existe uma variação máxima no número de clusters sendo que esse número altera em grande escala apenas com pequenas mudanças no valor *radii*. Com uma extração de *features* realizada pela ontologia Merops já existe uma menor variância do número de *clusters* nos quais os documentos são distribuídos, destacando-se desde logo o valor máximo de oito *clusters* que foram criados nesse caso.

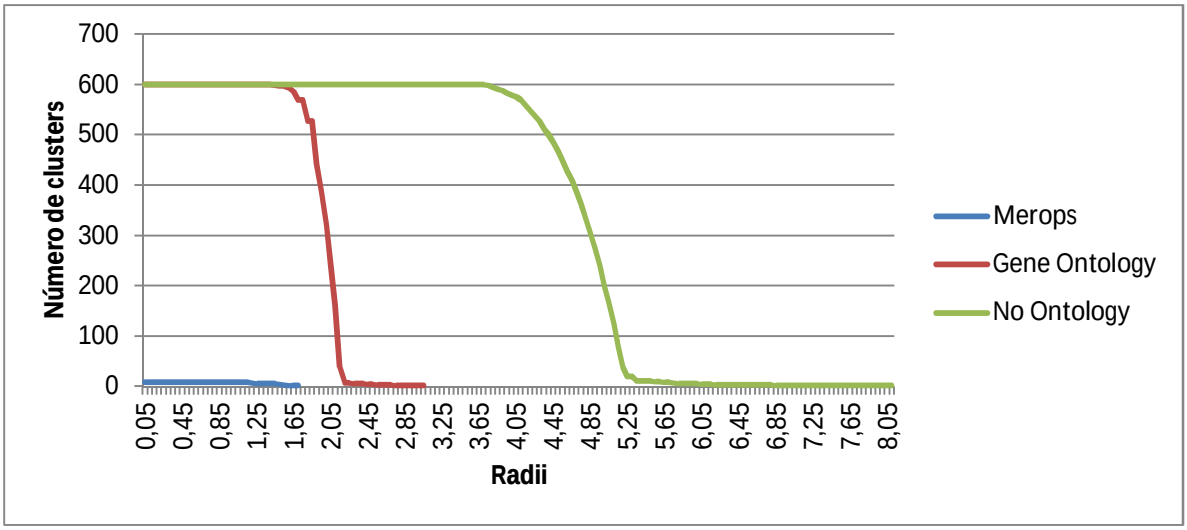


Figura 54 – Visão global dos resultados obtidos com o *Subtractive clustering*

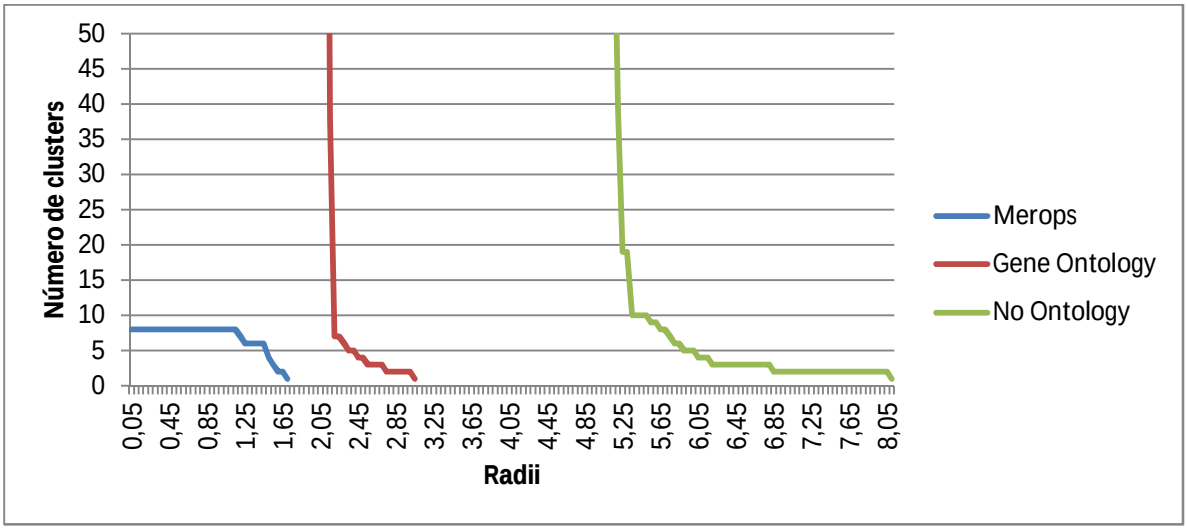


Figura 55 – Visão mais detalhada dos resultados obtidos com o algoritmo *Subtractive clustering*

Também em termos do tempo de execução do algoritmo se verificou uma enorme melhoria com a ontologia Merops, executando em alguns casos a classificação em menos de 1 segundo e nunca ultrapassando os 10 segundos, ao passo que com o Gene Ontology na maior parte das situações demorou mais de 1 minuto e sem ontologia mais de 10 minutos. Estes valores podem ser consultados na Tabela 44.

7. CONCLUSÕES

Começando pelos métodos de classificação usados, verificou-se uma hierarquia no que diz respeito aos classificadores que melhores resultados apresentam. Em todos os testes realizados o LIBSVM apresenta resultados próximos dos 100%. Num segundo plano encontra-se o algoritmo *K-means* que consegue ter quase sempre uma qualidade acima dos 80% chegando mesmo em alguns casos perto dos 100%. O algoritmo *Fuzzy C-Means* quando comparado com os restantes classificadores é aquele que demonstra uma maior dificuldade na obtenção de bons resultados, sendo de destacar, no entanto, que obtém os seus melhores resultados usando um valor entre 7 e 15 no parâmetro *exponent* do algoritmo.

Uns dos aspectos negativos do algoritmo *K-means* é o seu elevado tempo de processamento quando o número de *features* é elevado. Dos três algoritmos é aquele que apresenta maior tempo de execução, sendo essa diferença bastante considerável para os restantes.

Através dos testes realizados na extração de características ao parâmetro *ngram*, verificou-se que a sua adição não aumenta a qualidade de classificação nos diferentes algoritmos. Para além deste fator a sua utilização exige um processamento incomportável em *datasets* com um número médio ou elevado de documentos.

Outro dos testes realizados aos parâmetros usados na extração de *features* foi o tipo de criação de vectores para a matriz *BOW*. Neste teste observou-se que num *dataset* não balanceado o *TFIDF* consegue melhores resultados, ao passo que num *dataset* balanceado é o *TermFrequency* que se destaca. Estes indicadores fazem sentido uma vez que o *TFIDF* dá importância a um termo dentro do conjunto de documentos sendo assim melhor para *datasets* que não são balanceados.

Em relação às ontologias e usando os classificadores *K-means*, *Fuzzy C-means* e LIBSVM, em alguns casos observaram-se pequenas melhorias dos resultados e noutros uma pequena diminuição. Apesar de não existir uma melhoria substancial na classificação dos documentos, ouve uma melhoria clara noutros fatores. Esses fatores são a redução em grande escala do tempo de execução dos algoritmos e diminuição do número de *features*.

Para o caso do algoritmo *Subtractive clustering*, devido à sua característica única de não necessitar da definição à partida do número de clusters no qual deve dividir os documentos, foram realizados alguns testes diferentes. Através desses testes concluiu-se que o uso da ontologia Merops torna o algoritmo menos sensível à mudança dos seus parâmetros. Desta forma o número e clusters no qual o algoritmo divide os documentos é mais uniforme e não tão brusca como no caso em que não é usada qualquer ontologia ou quando é usada a Gene Ontology.

É possível por isso concluir que através das ontologias se consegue obter informação a partir dos documentos mais focada numa área específica e que o tempo de execução dos algoritmos

diminuiu em grande escala, sem que para tal a qualidade da classificação dos diferentes algoritmos seja comprometida.

Em termos de utilização da plataforma e olhando para os algoritmos de classificação presentes é possível verificar que o algoritmo *Subtractive* é mais indicado para a classificação de documentos a partir do *Web service* da PubMed. Tal acontece porque não existe um conhecimento prévio dos documentos e das classes a que eles pertencem, podendo dessa forma o utilizador organizar os documentos com um maior ou menor grau de semelhança conforme os valores usados nos parâmetros do algoritmo. Em termos de *datasets* enviados pelos utilizadores, onde existe um conhecimento prévio do número de classes a que eles podem pertencer, será melhor usar um algoritmo como o *K-Means* ou o *Fuzzy C-Means*.

Em relação à plataforma, através de alguns padrões de desenho, foi possível criar uma arquitetura que possibilita a sua rápida e fácil extensão. Foi dividido em termos de arquitetura todo o processo de classificação de documentos nas fases que compõem o *text mining*, ou seja, na obtenção de documentos, extração de conhecimento e classificação. Por existir um desacoplamento destas partes é possível adicionar novas formas de obtenção de documentos, novas ontologias ou formas de extração de *features* e novos algoritmos de classificação sem que para isso seja necessário alterar as outras fases do processo.

Considerando os resultados obtidos, consideramos que todos os objetivos inicialmente propostos foram atingidos, obtendo-se um trabalho útil e relevante para os investigadores no domínio biomédico. A aplicação irá ser facultada para domínio publico e o *feedback* obtido será considerado para melhorar futuras versões da aplicação.

Ainda como trabalho futuro e devido à extensibilidade da plataforma pretende-se usar novos algoritmos de classificação supervisionada e não supervisionada, bem como outras ontologias para a extração de conhecimento.

8. REFERÊNCIAS BIBLIOGRÁFICAS

Doms, A., & Schroeder, M. (2005). GoPubMed: exploring PubMed with the Gene Ontology. *Nucleic Acids Research*, W783–W786.

Yager, R., & Filev, D. (1994). Generation of Fuzzy Rules by Mountain Clustering. *Journal of Intelligent & Fuzzy Systems*, 209-219.

Word Vector Tool. (Maio de 2010). Obtido de SourceForge: http://sourceforge.net/projects/wvtool/

Coremine medical. (Novembro de 2012). Obtido de Coremine medical: http://www.coremine.com/medical/#search

Entrez Programming Utilities Help. (Agosto de 2012). Obtido de NCBI bookshelf: http://www.ncbi.nlm.nih.gov/books/NBK25501/

Gene Ontology Consortium. (Dezembro de 2012). Obtido de Gene Ontology Consortium: http://www.geneontology.org/

Rapid - I. (Novembro de 2012). Obtido de Rapid -I: http://rapid-i.com/content/view/202/206/lang,en/#text

Computer Science Source. (10 de Novembro de 2014). Obtido de Machine Learning – Support Vector Machines: https://computersciencesource.wordpress.com/2010/01/29/year-2-machine-learning-support-vector-machines/

NCBI PubMed. (Novembro de 2014). Obtido de NCBI PubMed: http://www.ncbi.nlm.nih.gov/pubmed/

Ontology Structure. (Dezembro de 2014). Obtido de Gene Ontology Consortium: http://geneontology.org/page/ontology-structure

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27.

Chiu, S. (1994). Fuzzy model identification based on cluster estimation. *J. Intell. Fuzzy Systems*, 267-278.

Correia, D. (2010). *Biomedoc*. Fonte: Biomedoc: http://193.137.78.18/biomedoc/

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 199-220.

Hsu, C.-W., & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions*, 415-425.

Hsui, C.-W., Chih-Chung, C., & Chih-Jen, L. (2003). A practical guide to support vector classification.

Leite, C. (2009). *SVM Search*. Fonte: SVM Search: http://193.137.78.18/svmsearch/

Orange. (Dezembro de 2014). *Orange*. Obtido de http://orange.biolab.si/download/

Rawlings, N. D., Waller, M., Barrett, A. J., & Bateman, A. (2014). MEROPS: the database of proteolytic enzymes, their. *Nucleic Acids Research Vol. 42*, D503–D509.

Rocket Software. (Novembro de 2012). *technology for extracting intelligence from text*. Obtido de Rocket Software: http://www.rocketsoftware.com/products/aerotext

The Gene Ontology Consortium. (2000). Gene ontology: tool for the unification of biology. *Nature genetics*, 25-29.

ANEXO A

Neste anexo são apresentados três diagramas de sequência. Os diagramas apresentados são relativos ao treino de máquinas de vectores de suporte (Figura 56), à classificação de documentos usando as máquinas de vectores de suporte treinadas (Figura 57) e classificação de documentos enviados por um utilizador utilizando o algoritmo *K-means* (Figura 58).

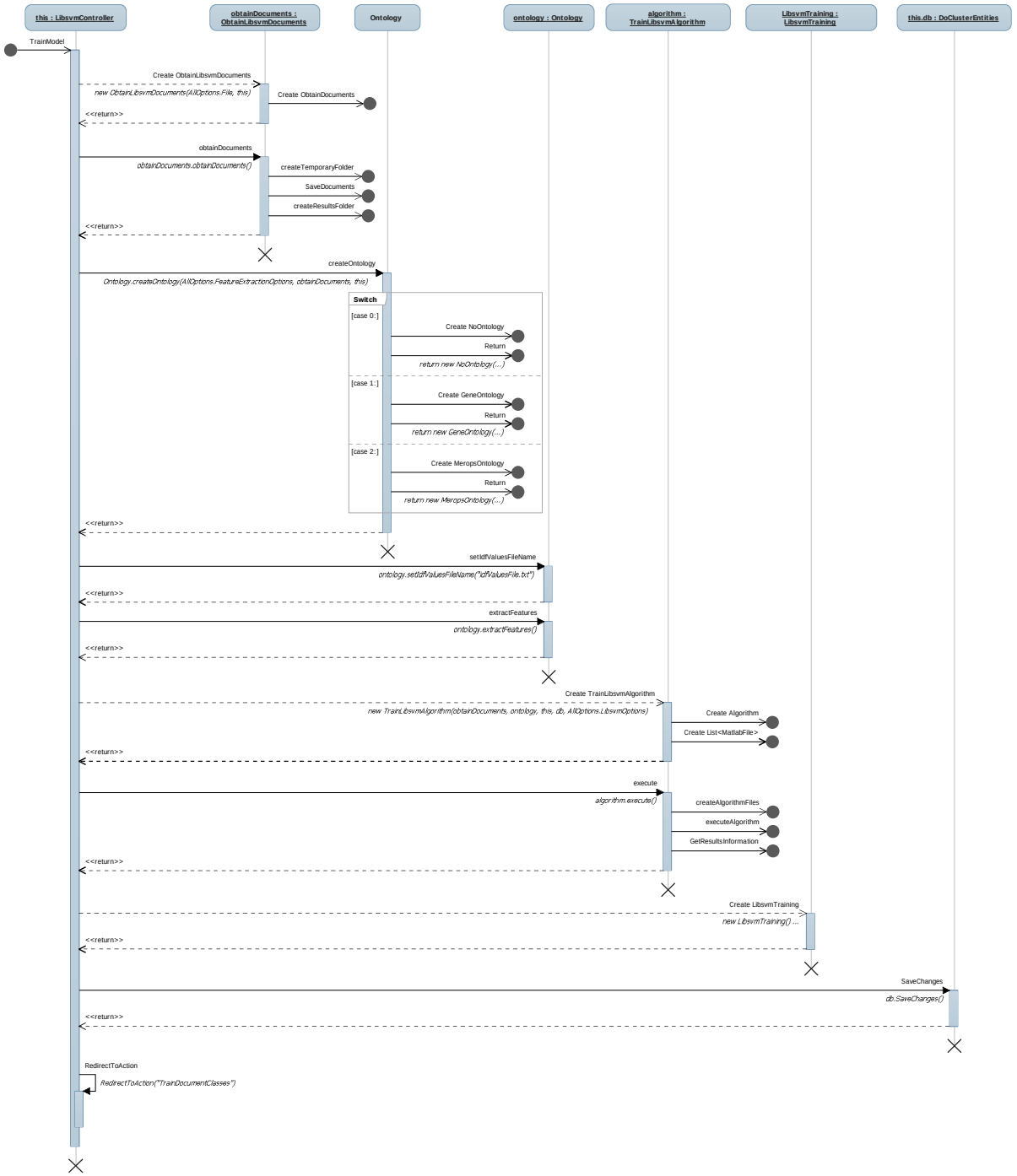


Figura 56 – Diagrama de sequência: Treino de máquinas de vectores de suporte

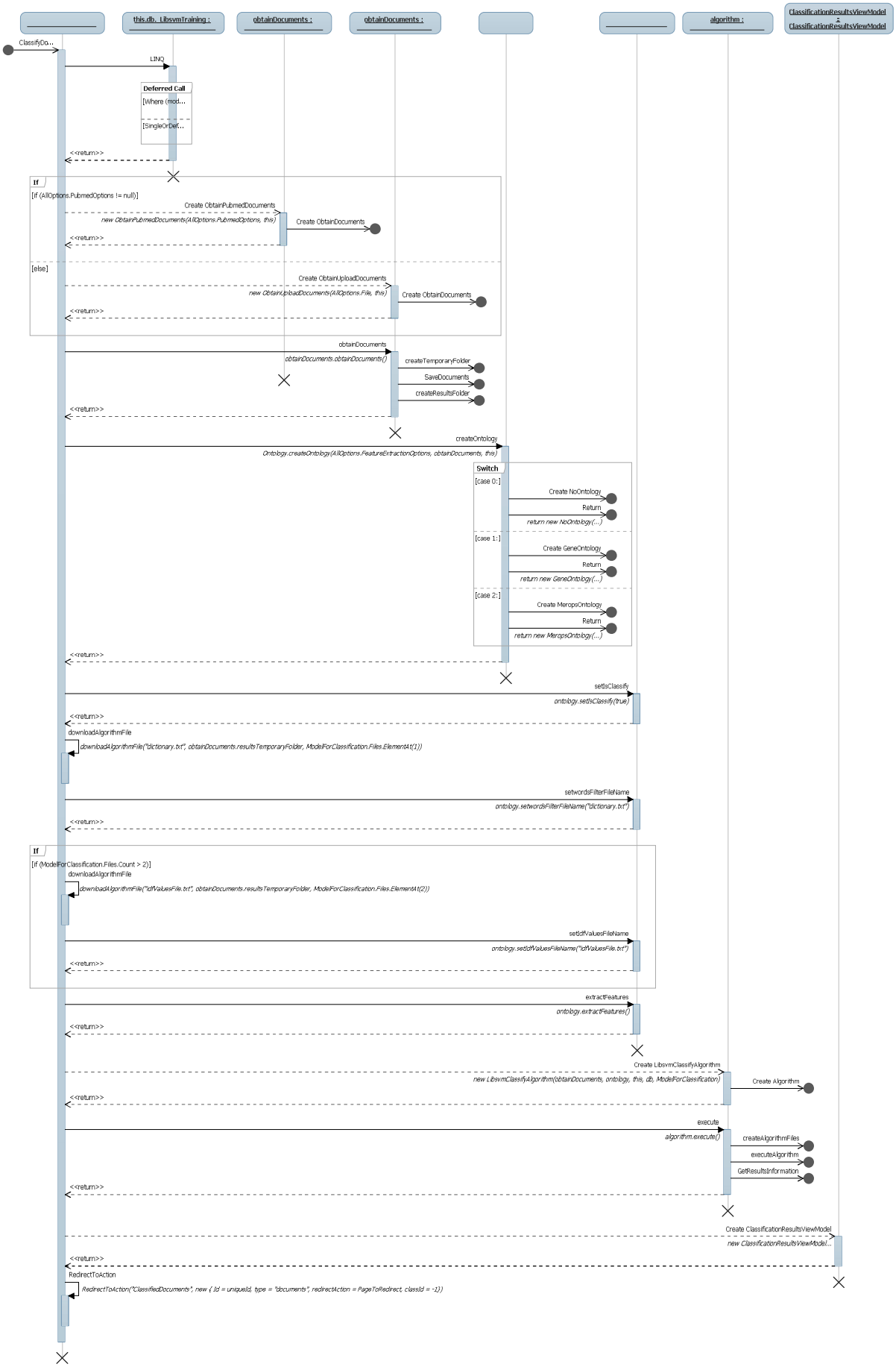


Figura 57 – Diagrama de sequência: Classificação de documentos usando *SVMs* treinadas

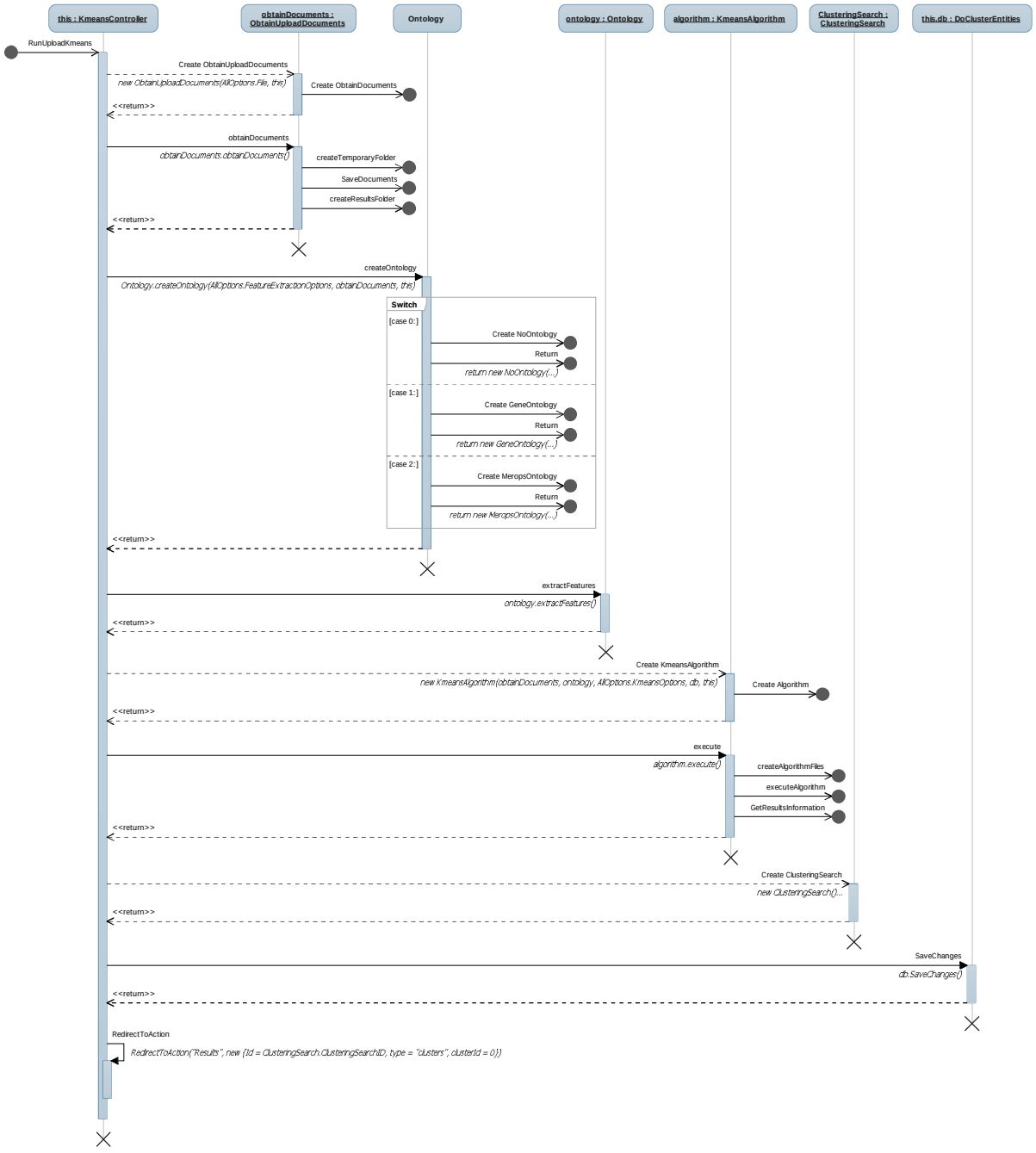


Figura 58 – Diagrama de sequência: Classificação de documentos do utilizador usando o algoritmo *K-means*

ANEXO B

Neste anexo são apresentados os vários protótipos de interface da aplicação.

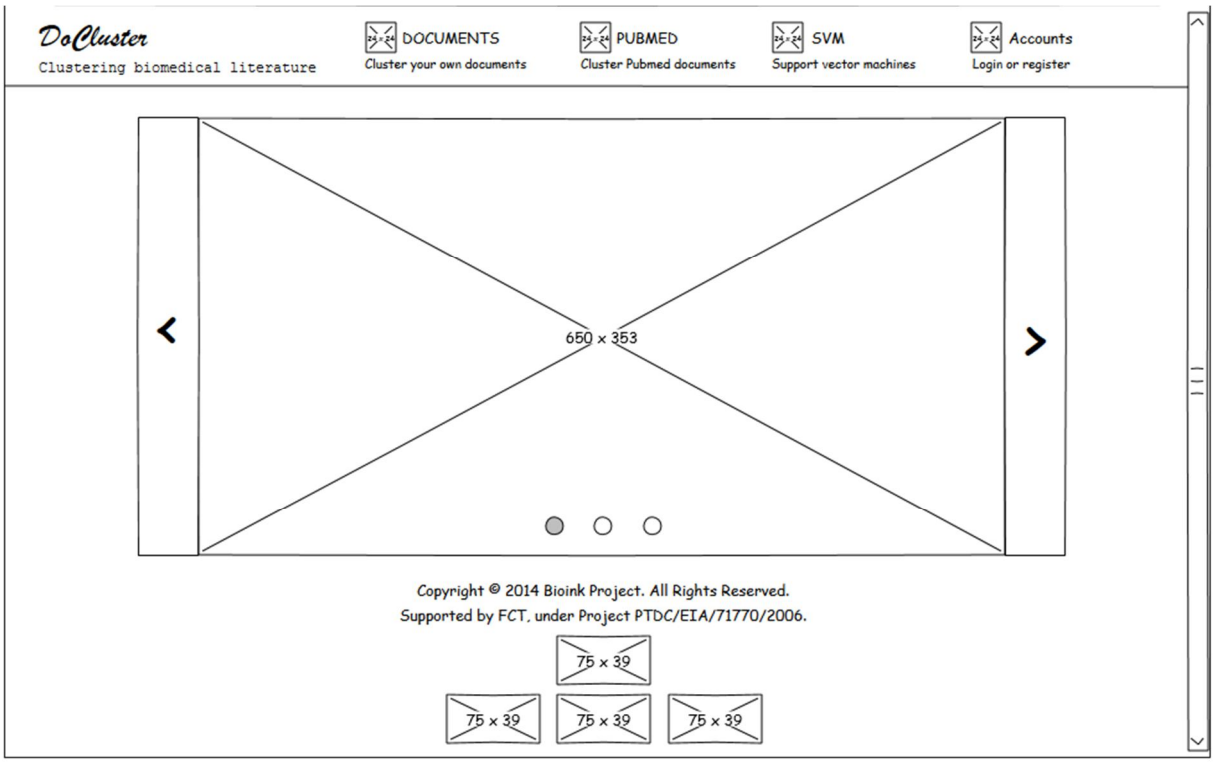


Figura 59 – Protótipo de *interface*: Página inicial

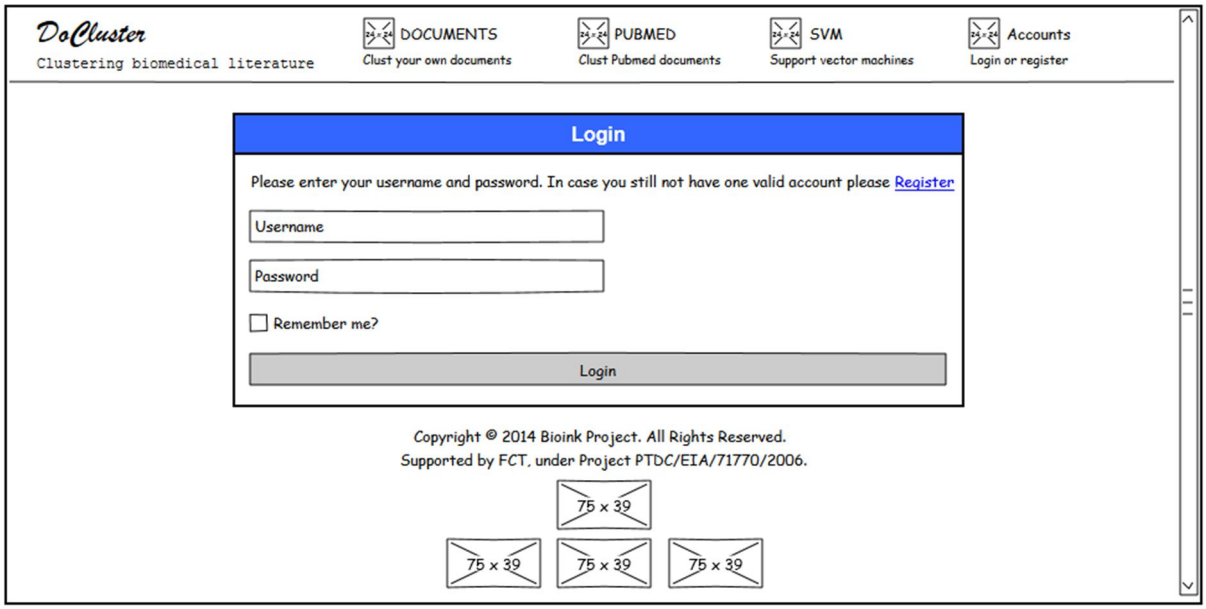


Figura 60 – Protótipo de *interface*: Página de *login*

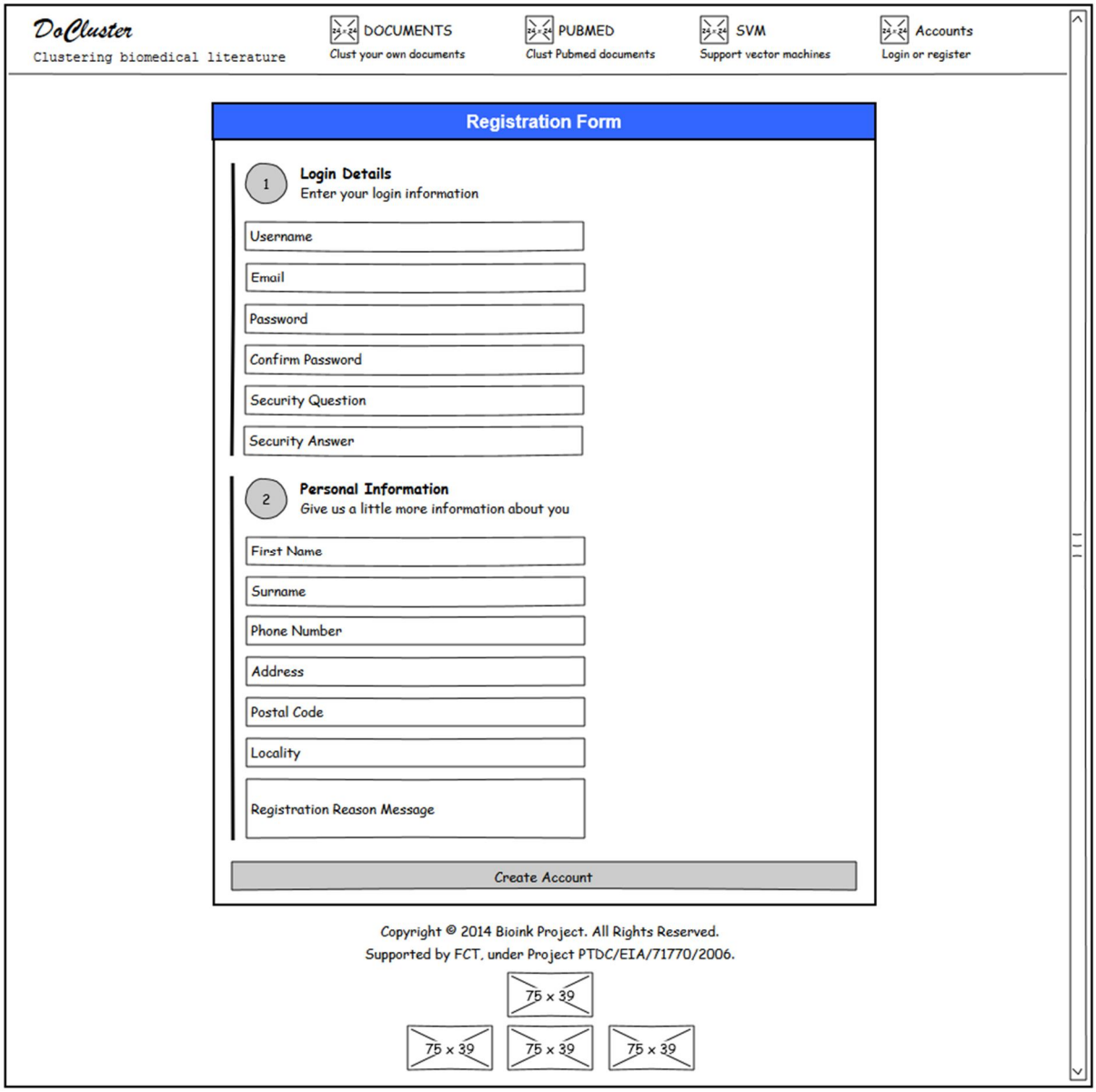


Figura 61 – Protótipo de interface: Registrar utilizador

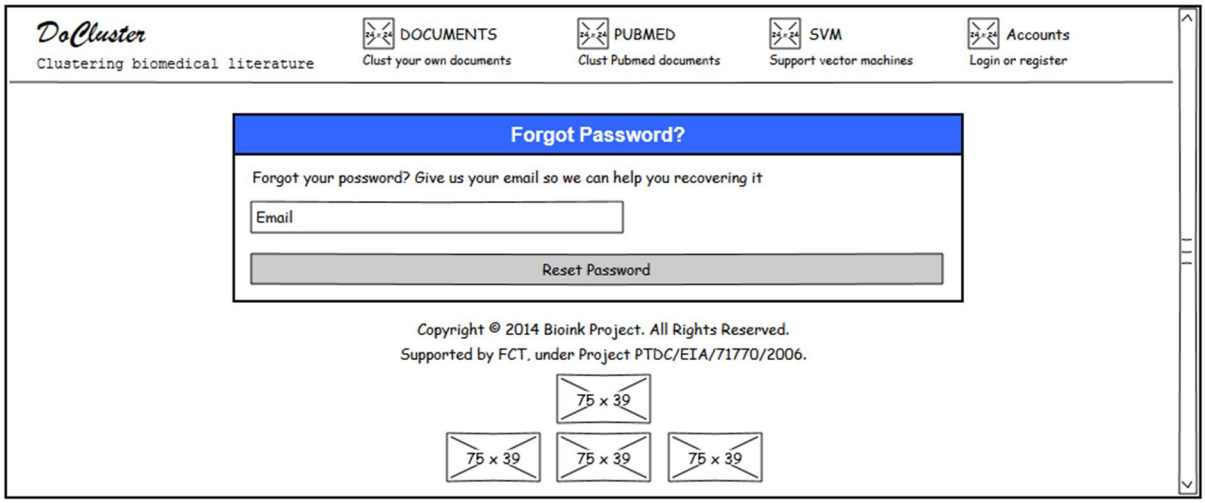


Figura 62 – Protótipo de interface: Redefinir password

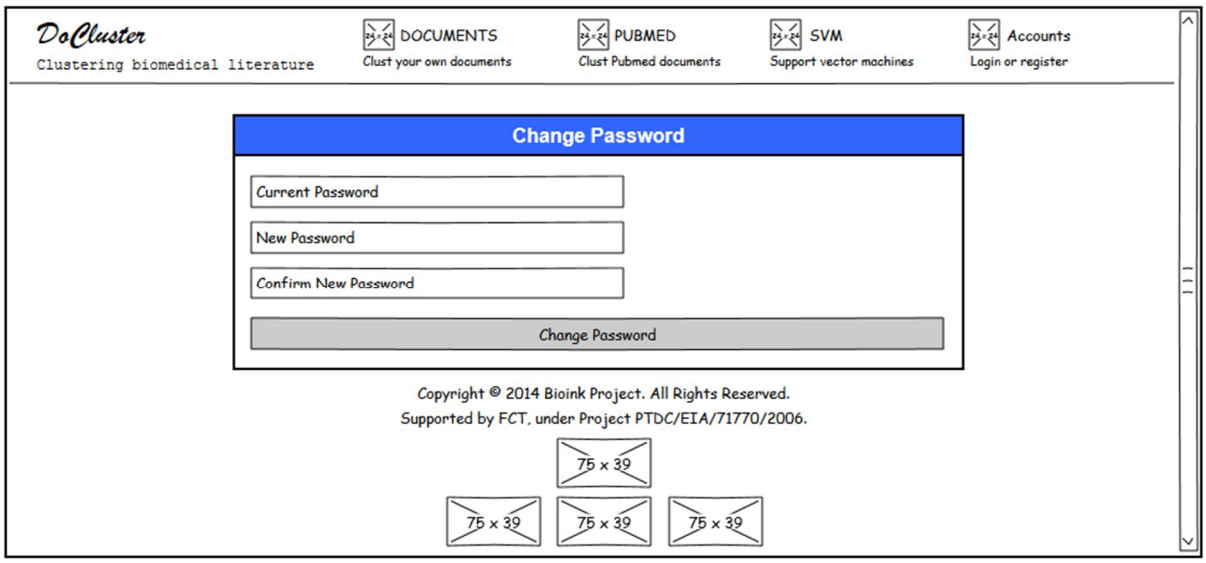


Figura 63 – Protótipo de *interface*: Alterar *password*

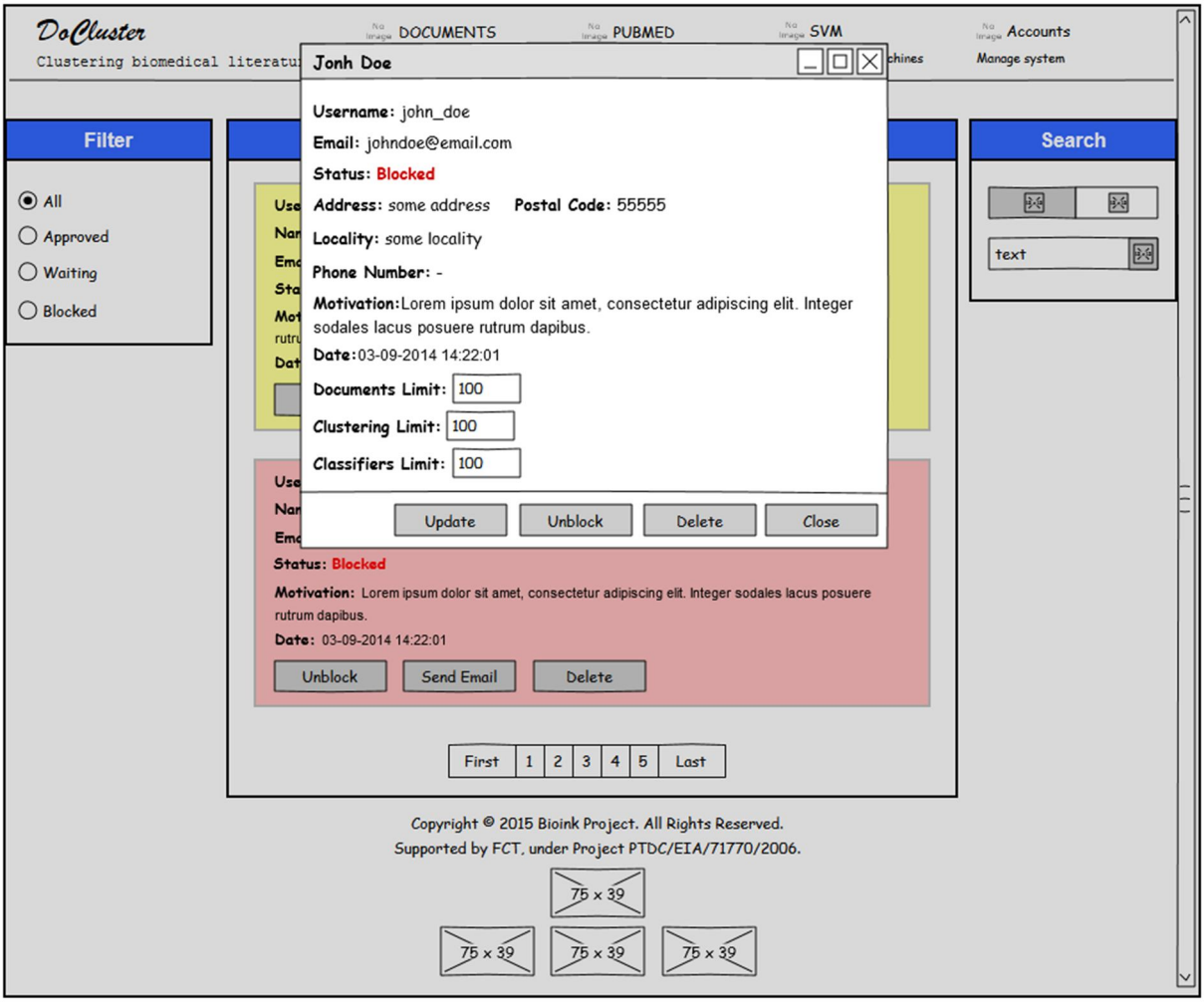


Figura 64 - Protótipo de *interface*: Informação detalhada de um utilizador

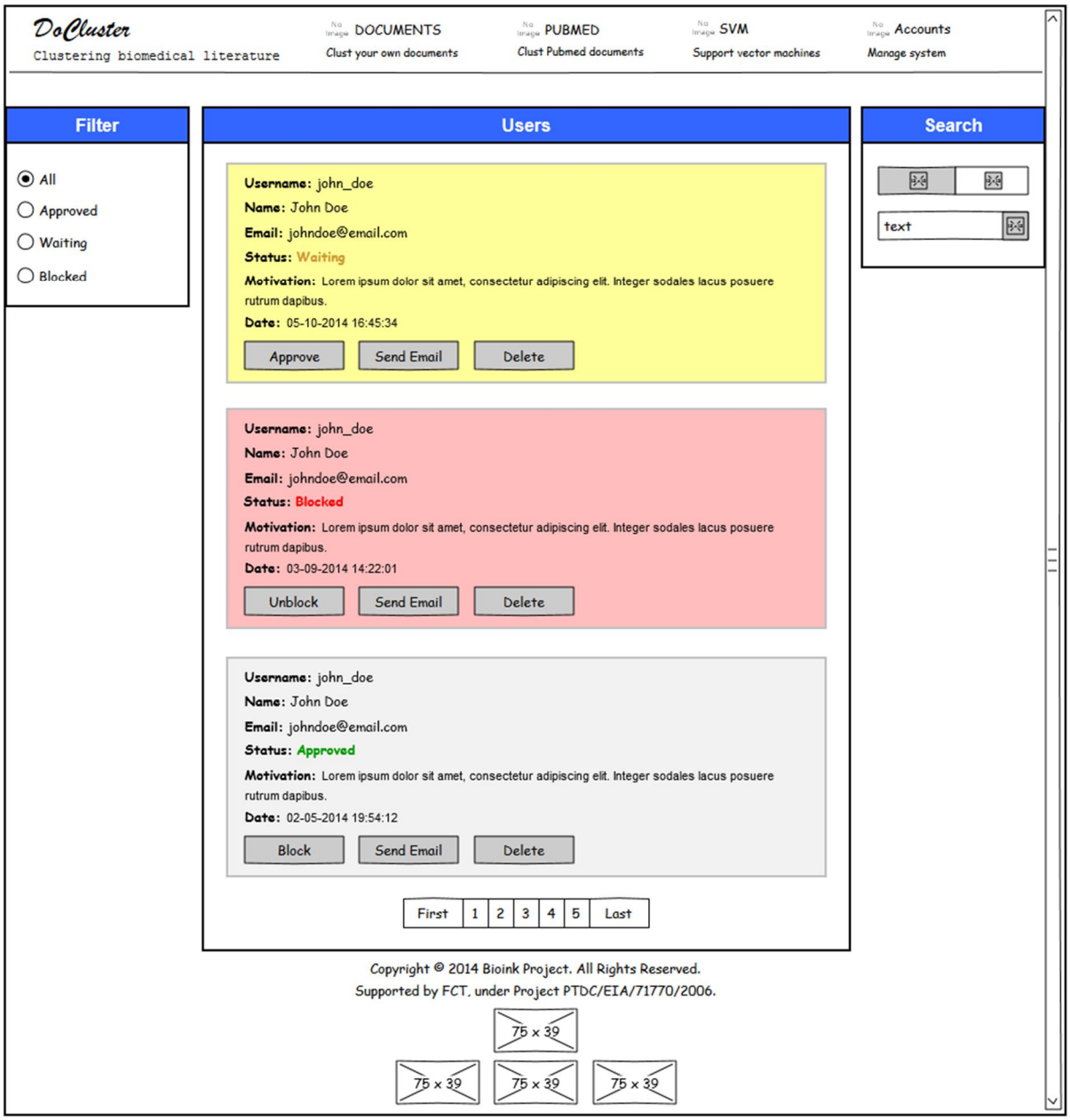


Figura 65 - Protótipo de *interface*: Lista de utilizadores registados

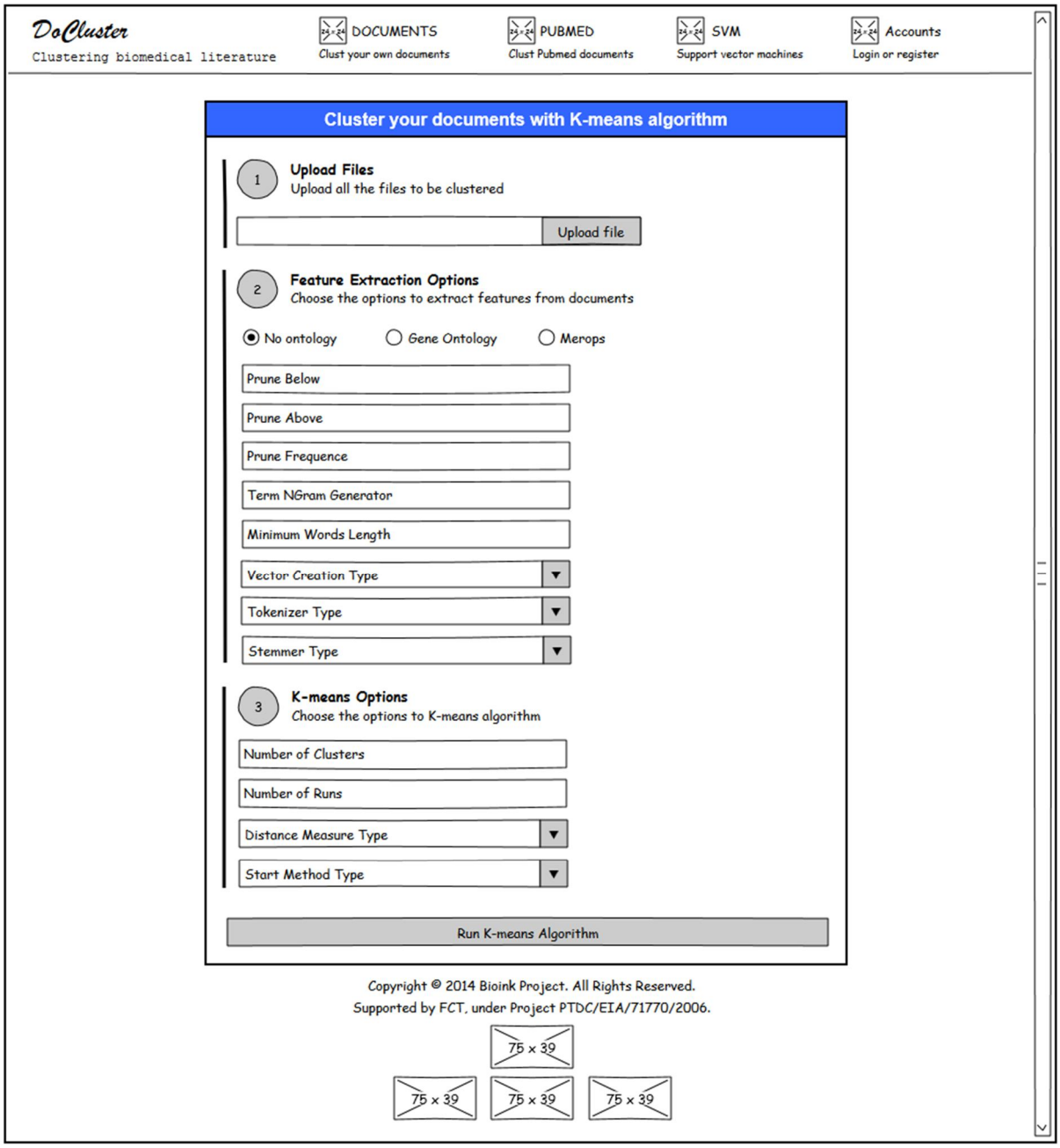


Figura 66 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *K-means*

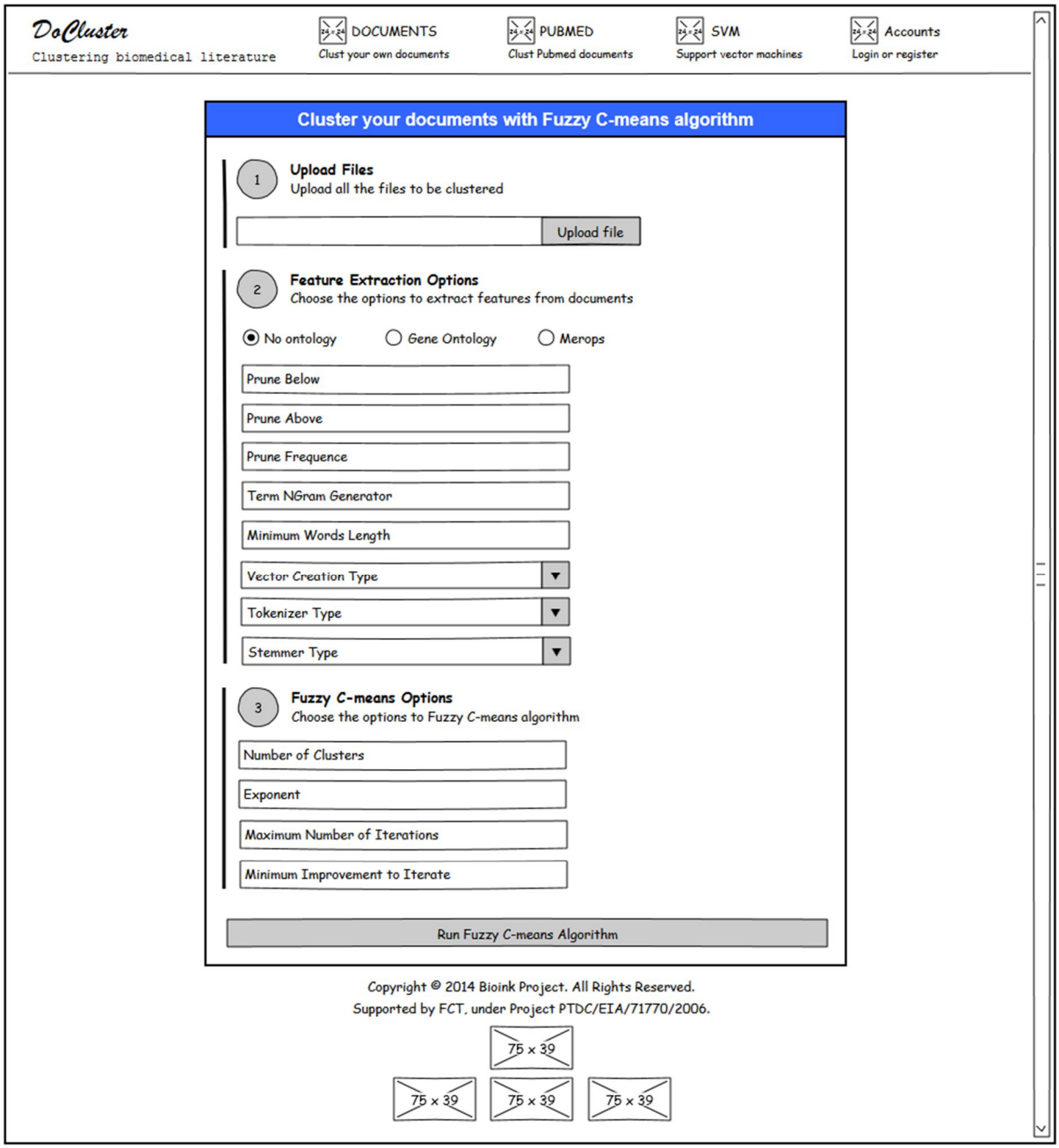


Figura 67 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *Fuzzy C-means*

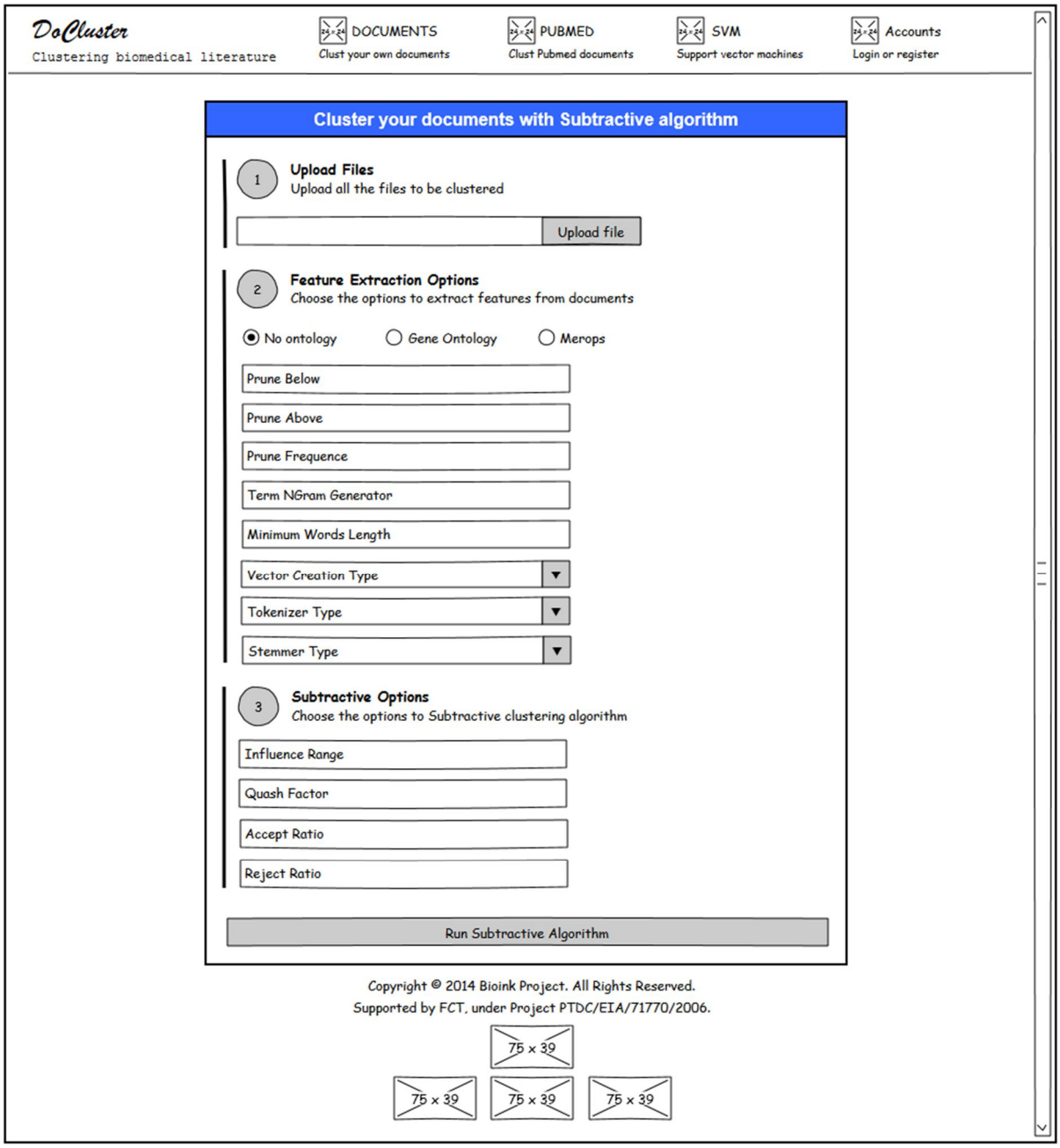


Figura 68 - Protótipo de *interface*: Formulário para o agrupamento de documentos enviados pelo utilizador usando o algoritmo *Subtractive Clustering*

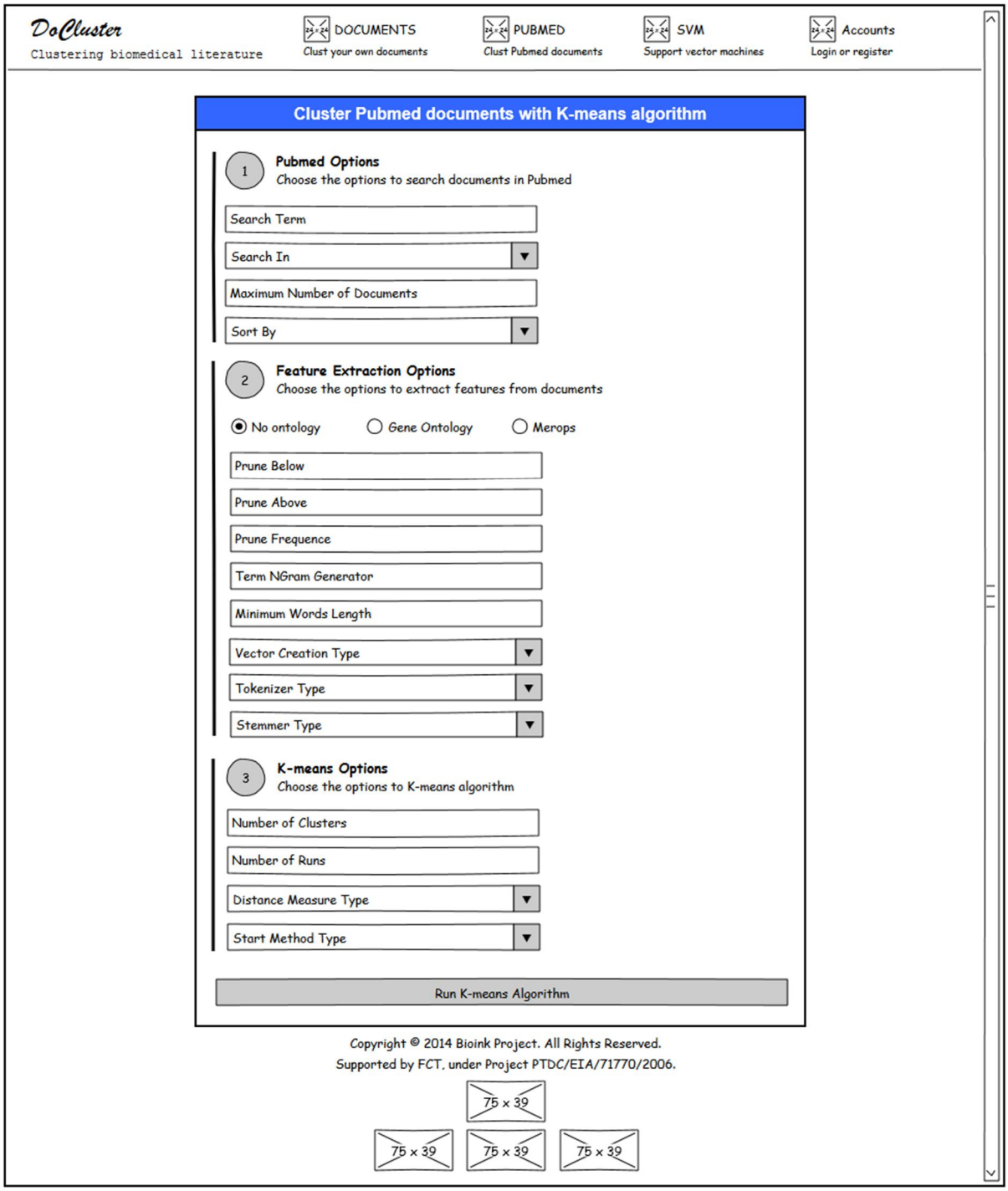


Figura 69 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *K-means*

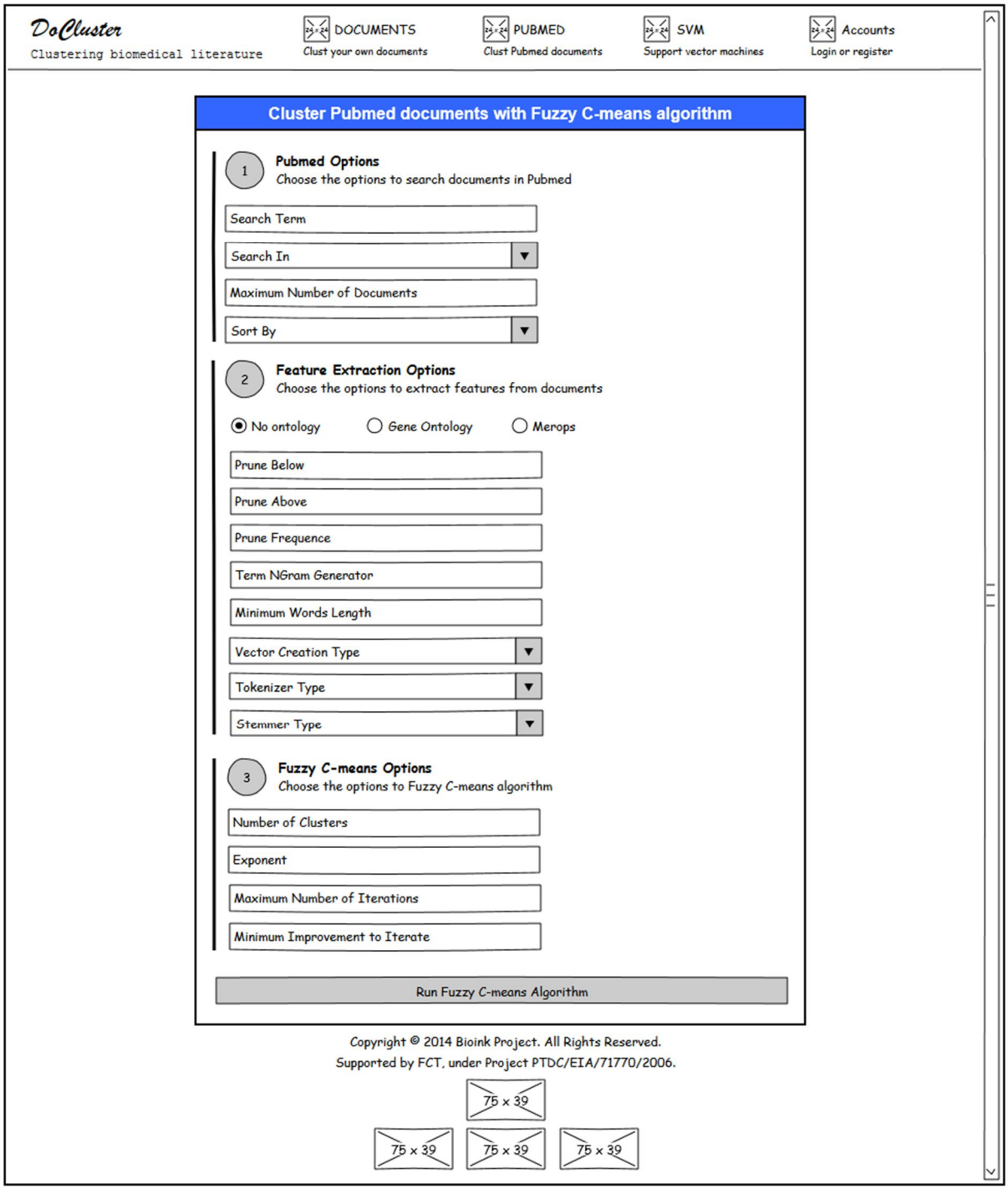


Figura 70 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *Fuzzy C-means*

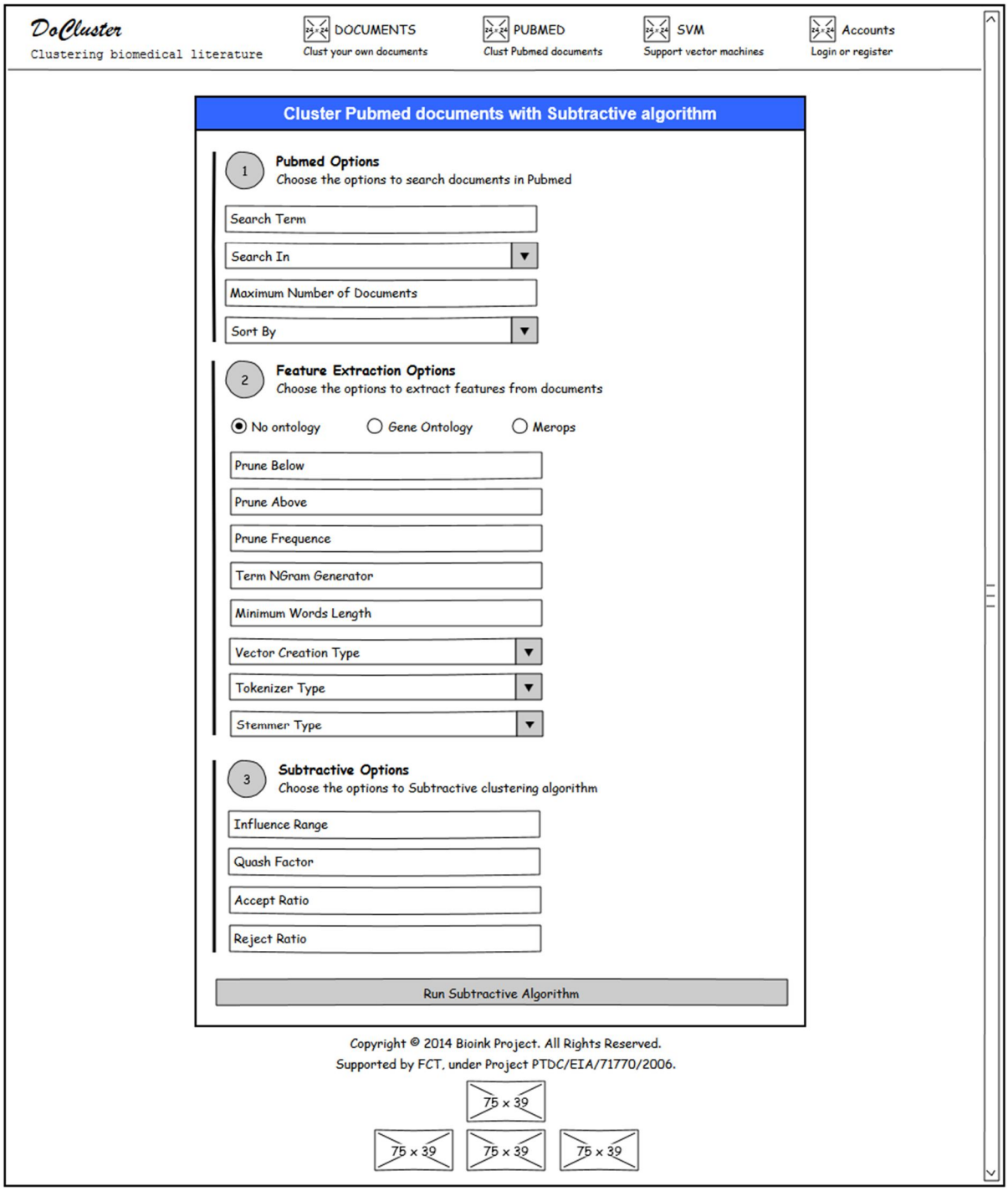


Figura 71 - Protótipo de *interface*: Formulário para o agrupamento de documentos pesquisados na PubMed usando o algoritmo *Subtractive Clustering*

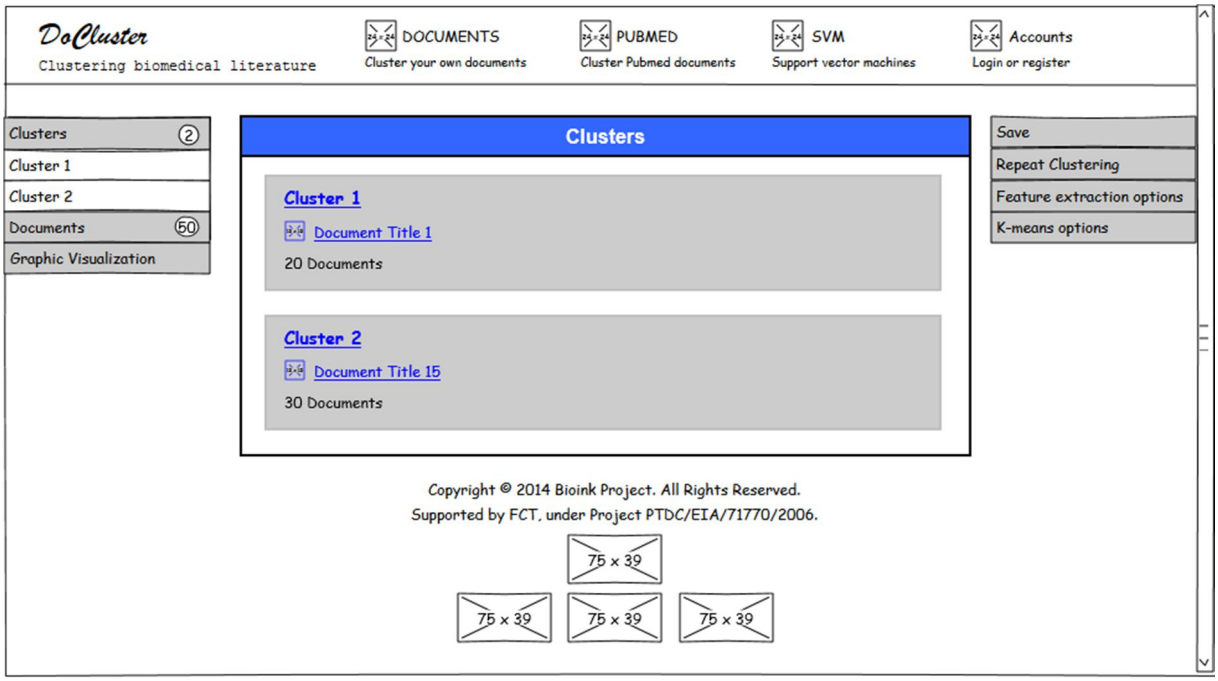


Figura 72 – Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos na lista de clusters

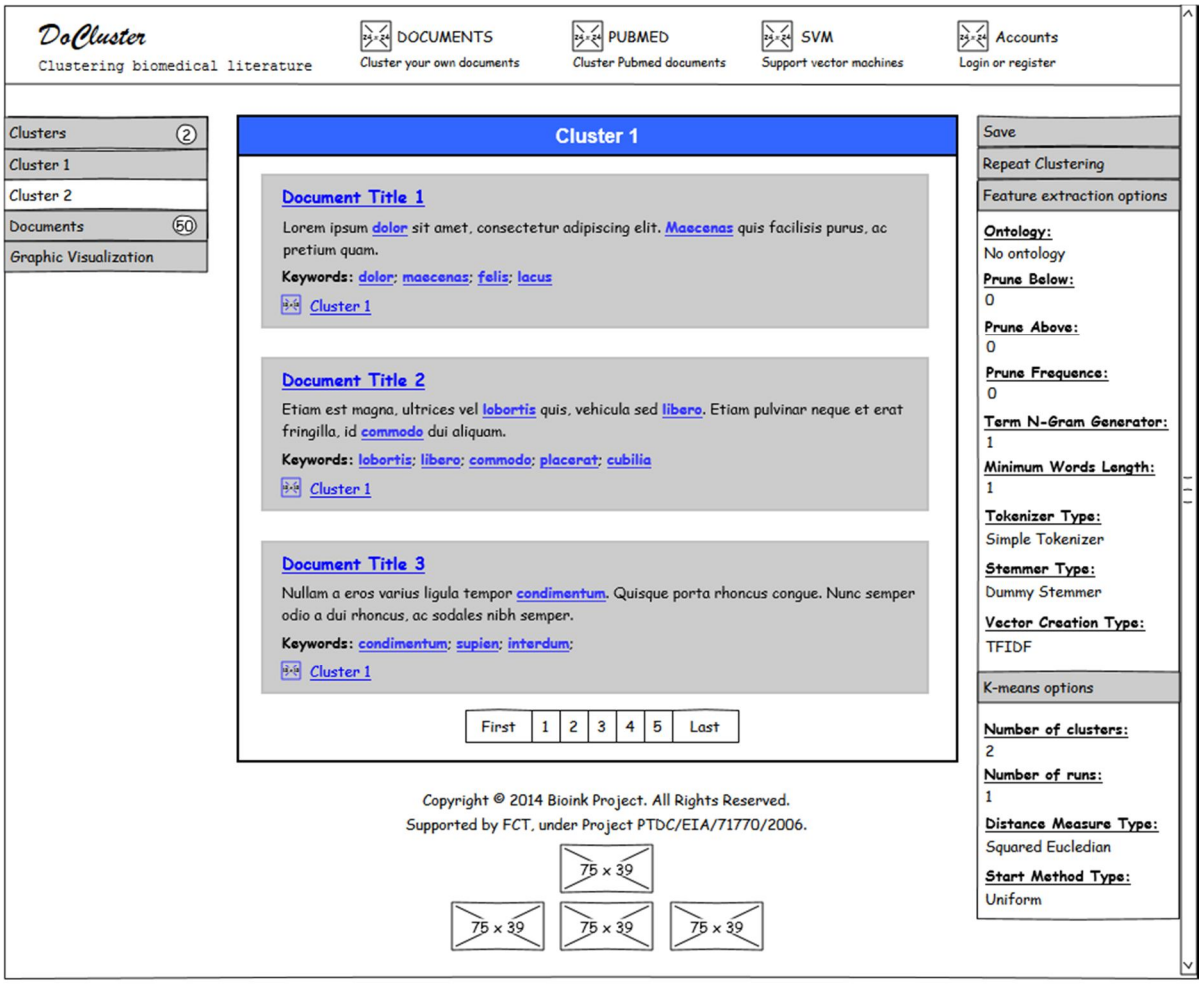


Figura 73 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos na lista de documentos

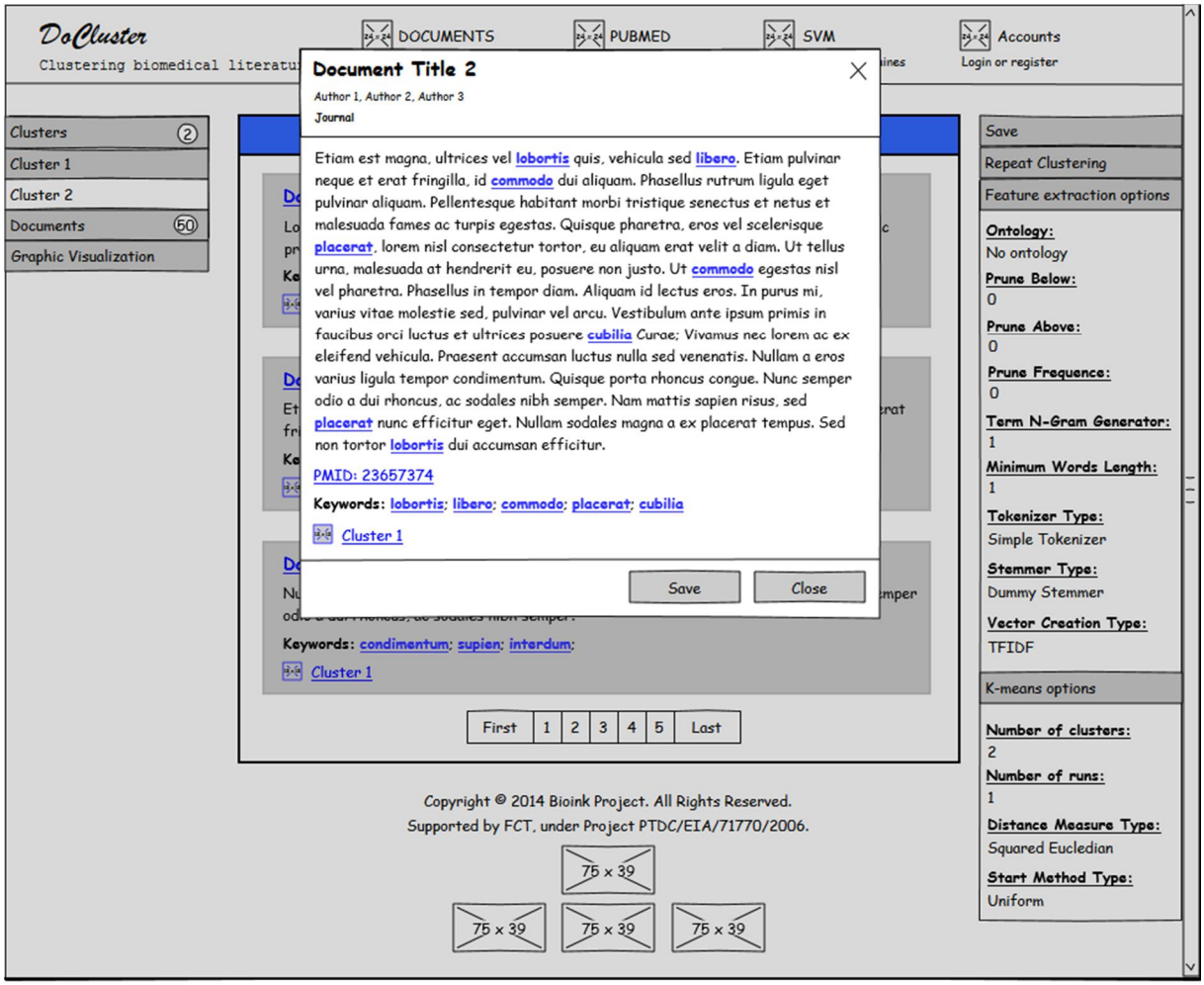


Figura 74 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco nos num documento especifico.

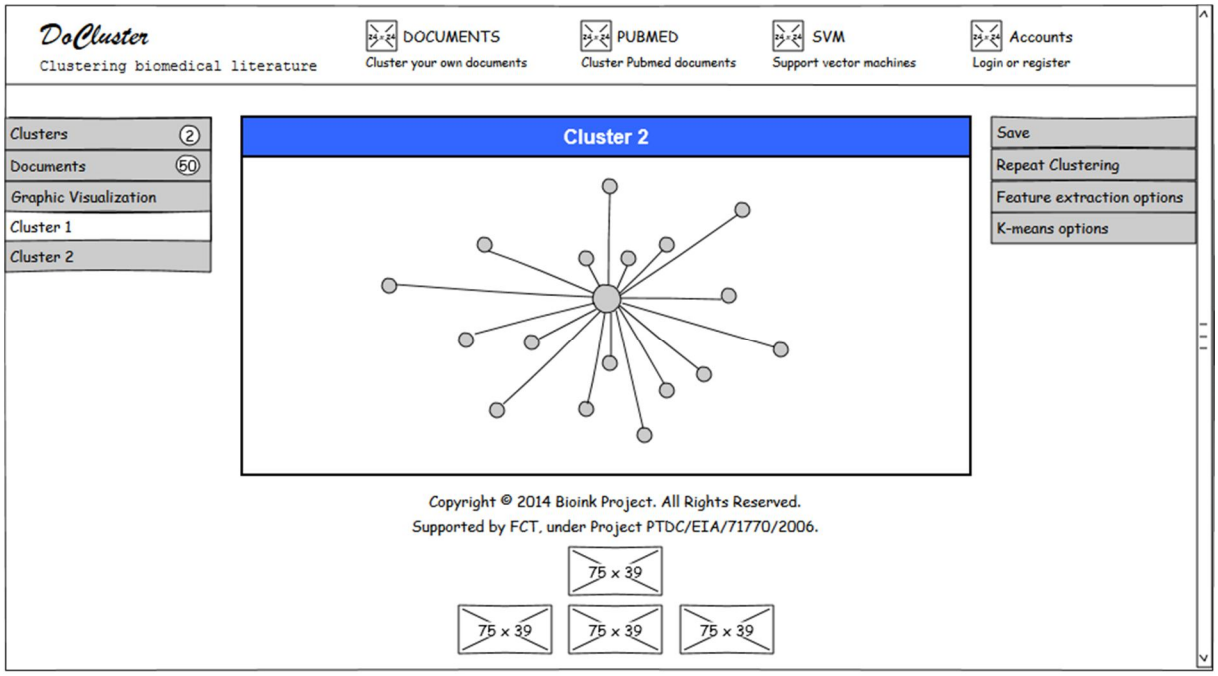


Figura 75 - Protótipo de *interface*: Resultados do agrupamento de documentos com foco na visualização gráfica

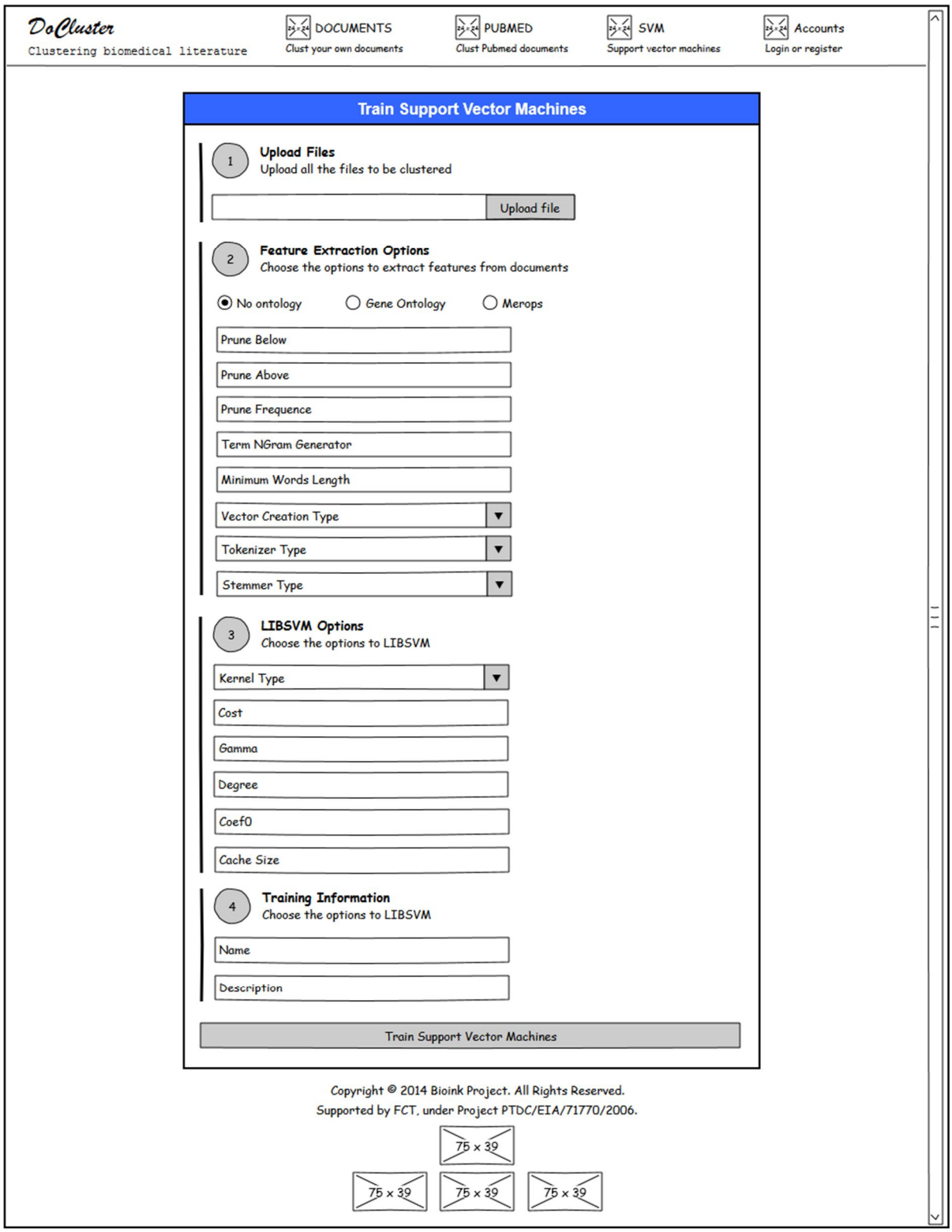


Figura 76 - Protótipo de *interface*: Formulário para o treino de máquinas de vectores de suporte

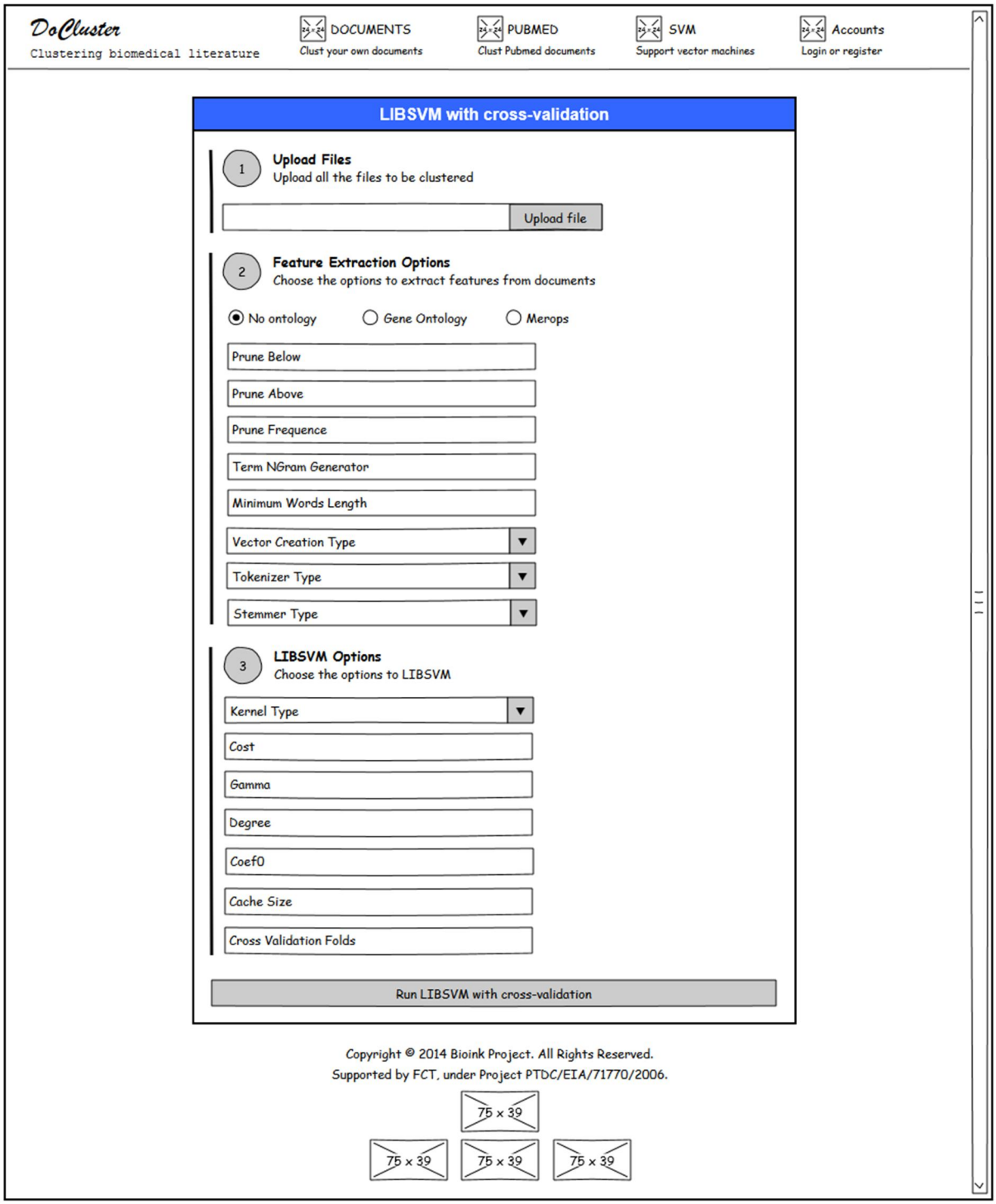


Figura 77 – Protótipo de *interface*: Formulário para a utilização do LIBSVM com cross-validation

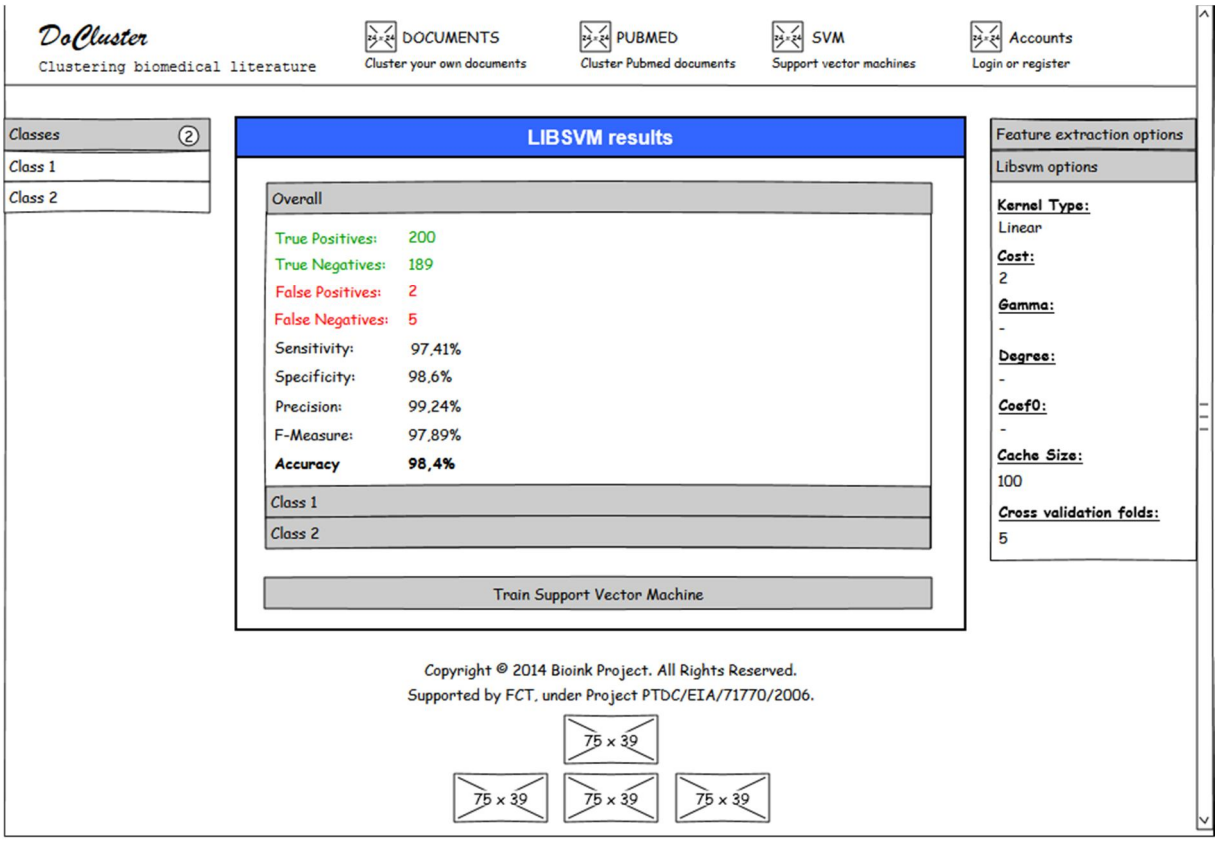


Figura 78 - Protótipo de *interface*: Resultados relativos à utilização do LIBSVM com *cross-validation*

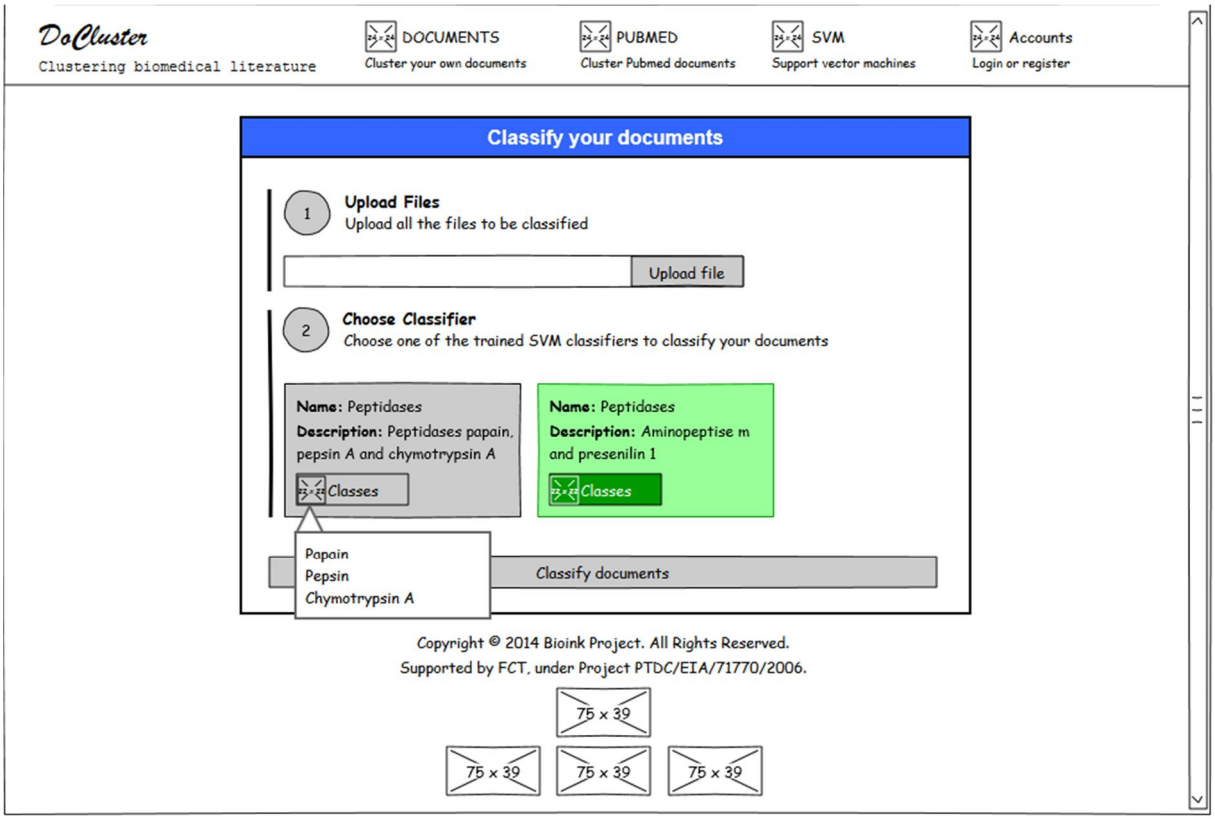


Figura 79 - Protótipo de *interface*: Formulário para a classificação de documentos enviados pelo utilizador usando SVMs treinadas.

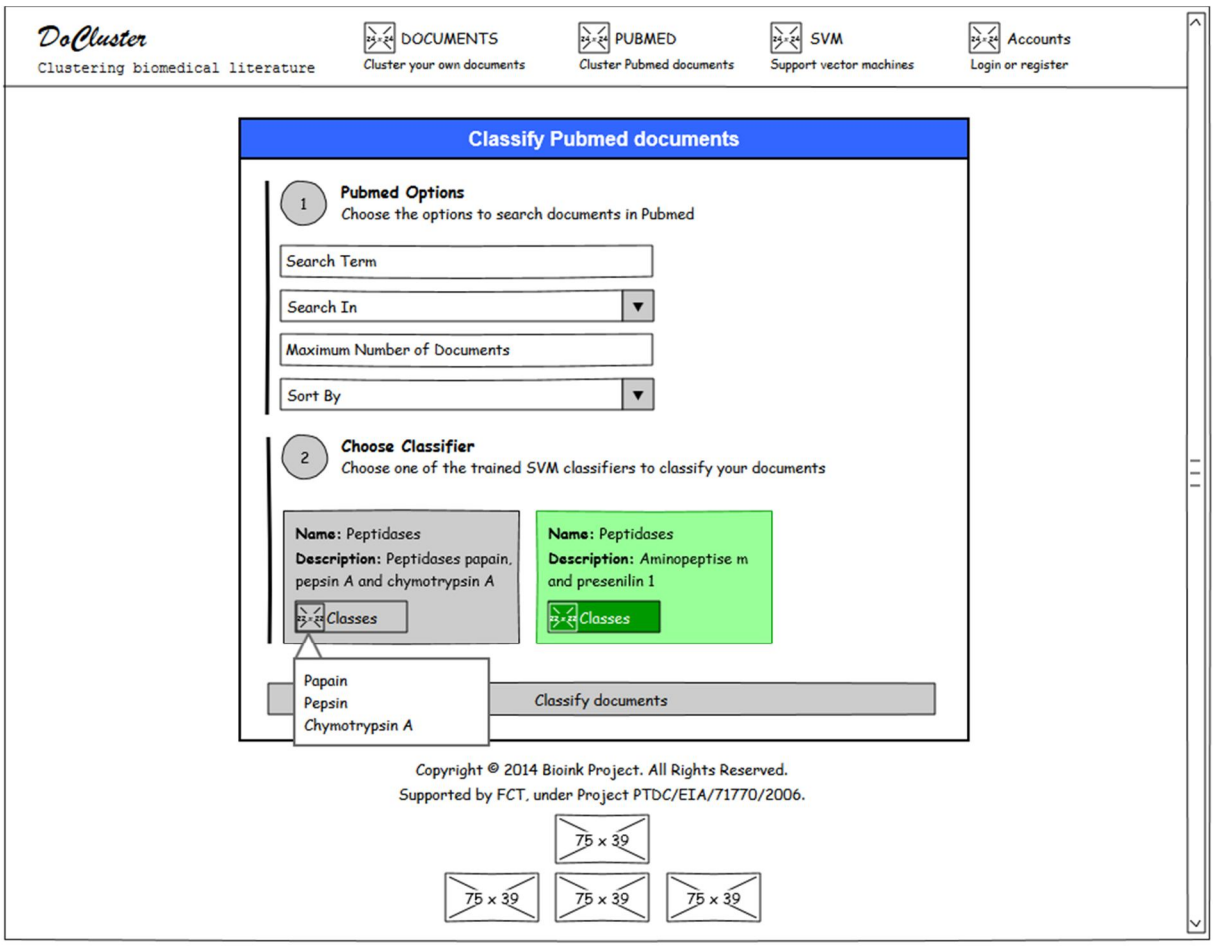


Figura 80 - Protótipo de *interface*: Formulário para a classificação de documentos pesquisados na PubMed usando SVMs treinadas.

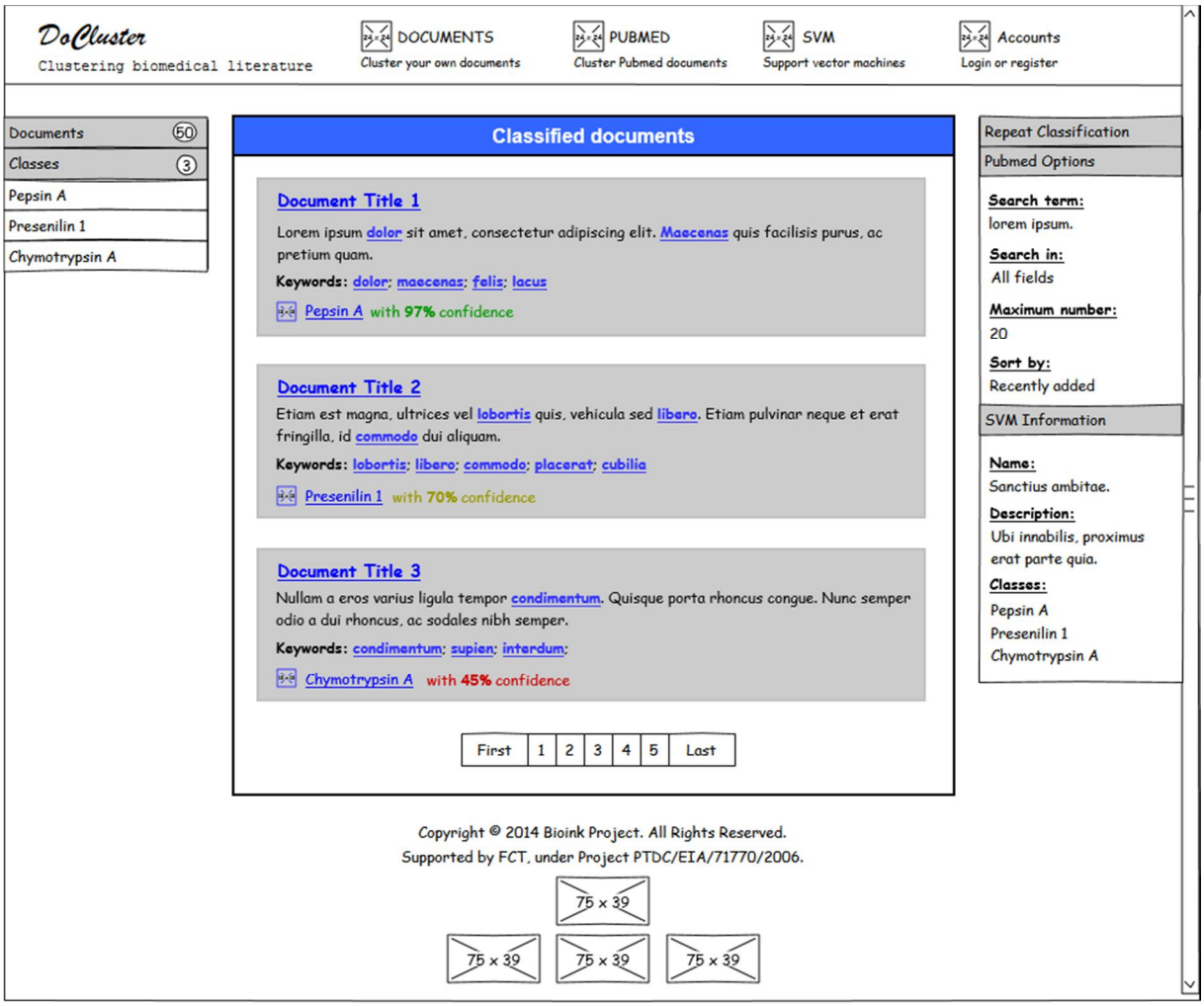


Figura 81 - Protótipo de *interface*: Resultados com a classificação de documentos recorrendo a SVMs treinadas

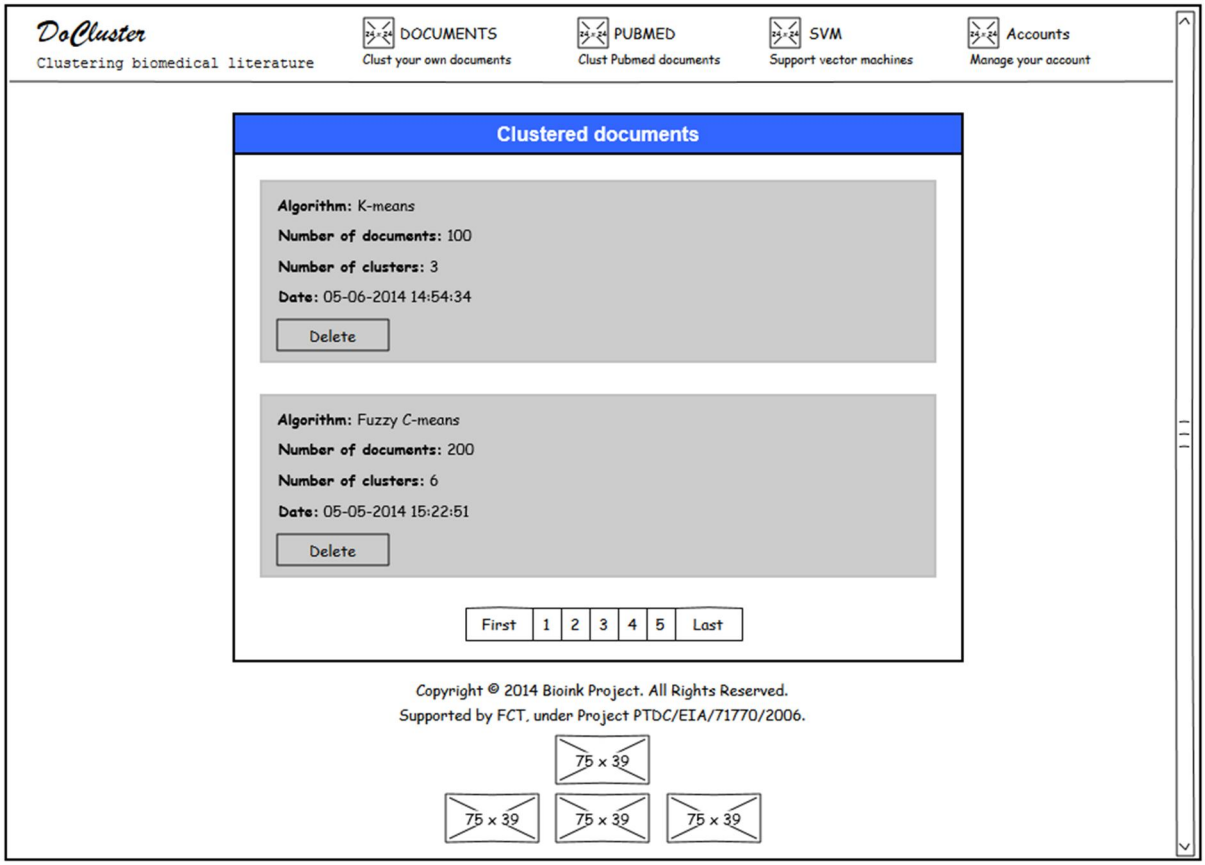


Figura 82 - Protótipo de *interface*: Lista de agrupamentos de documentos guardados

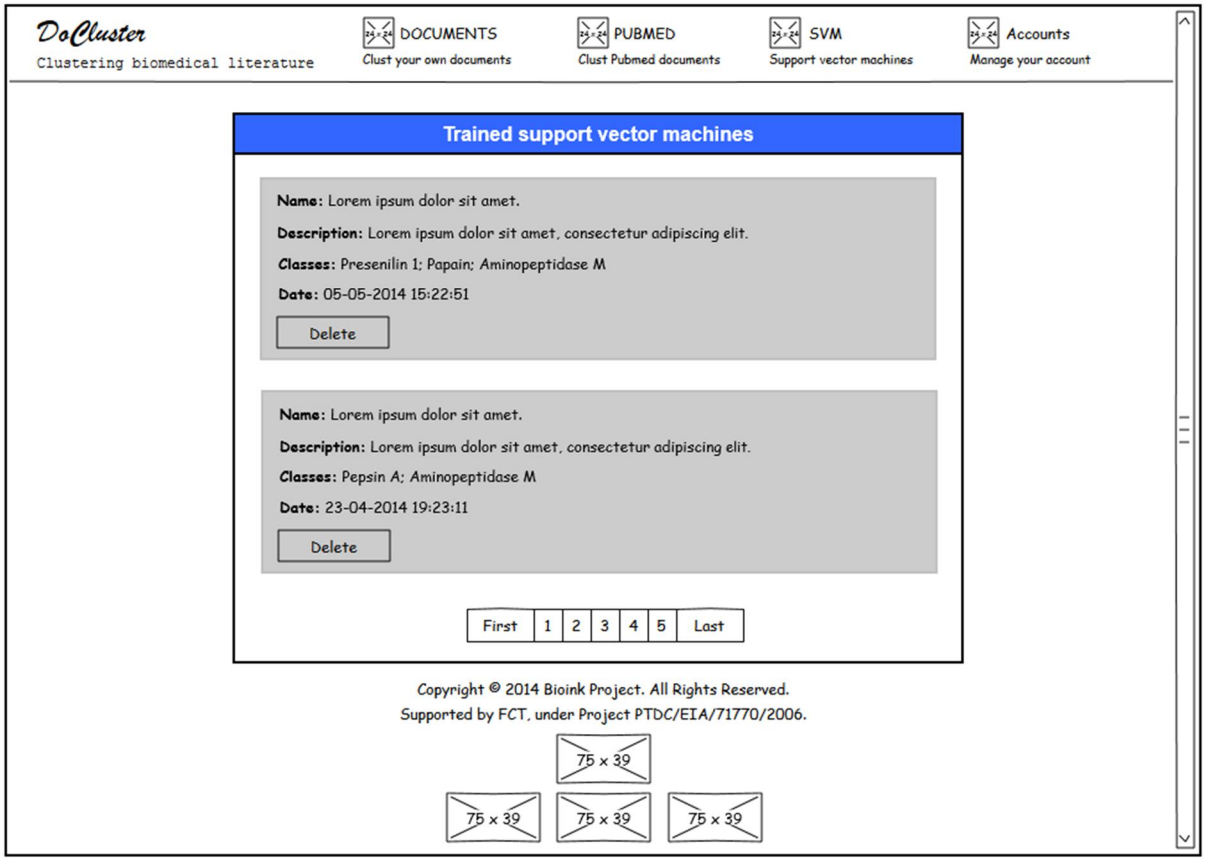


Figura 83 - Protótipo de *interface*: Lista de máquinas de vectores de suporte treinadas

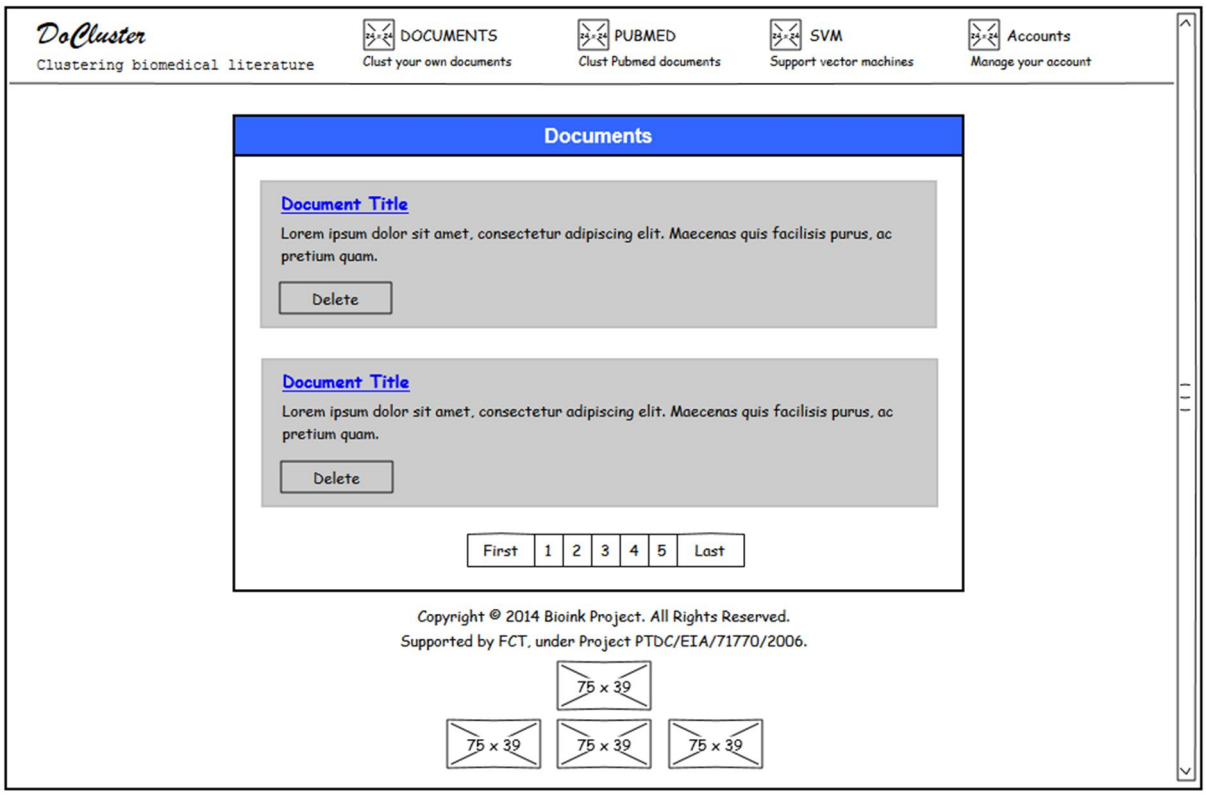


Figura 84 - Protótipo de *interface*: Lista de documentos guardados

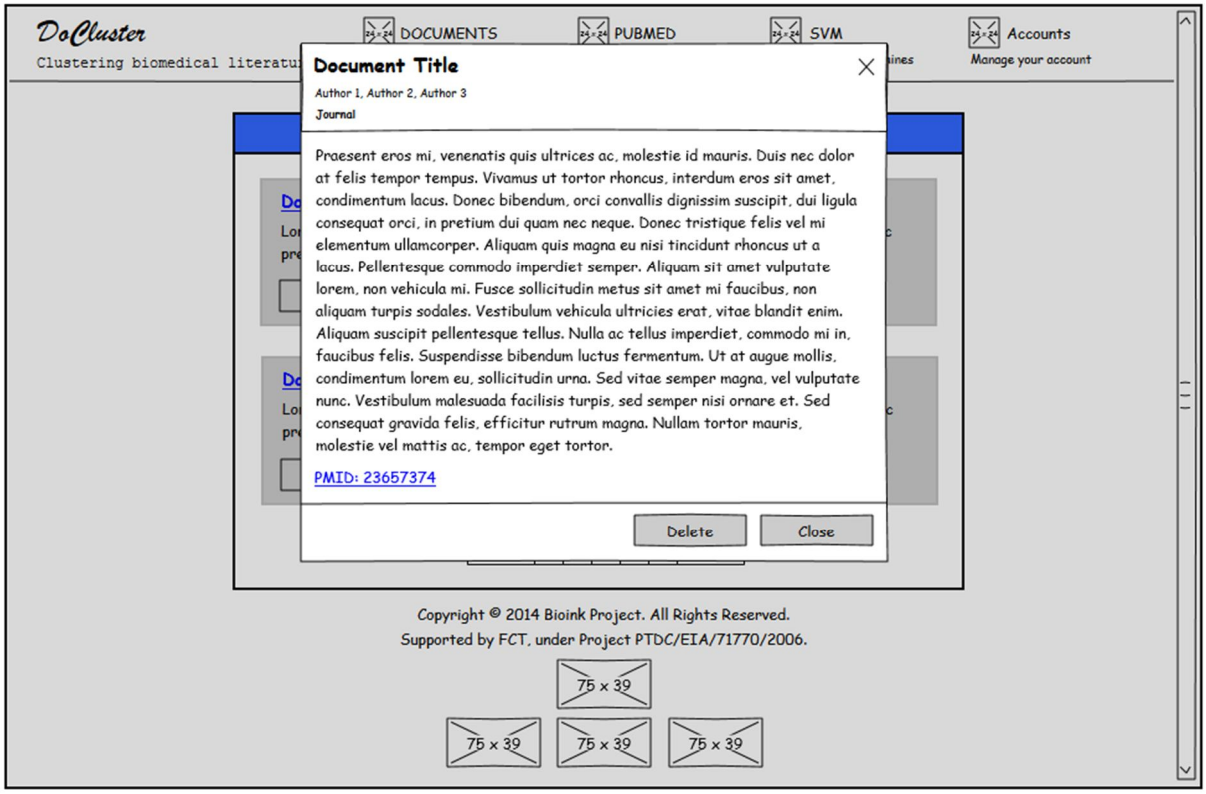


Figura 85 - Protótipo de *interface*: Documento guardado

ANEXO C

Neste anexo estão todas as tabelas com os resultados dos testes realizados aos vários algoritmos de classificação e vários métodos de extração de *features*. Como é possível constatar existe uma notação em todas as tabelas de resultados. Essa notação é apresentada na Tabela 34.

Tabela 34 – Tabela com a legenda dos termos presentes nas tabelas de resultados

Nfeatures	Número de *features* extraídas
tEF	Tempo de extração de *features* em segundos
tKM	Tempo de execução, em segundos, do algoritmo *K-Means*
KM	Algoritmo *K-Means*
tFCM	Tempo de execução, em segundos, do algoritmo *Fuzzy C-Means*
FCM	*Fuzzy C-Means*
tLIBSVM	Tempo de execução, em segundos, do Libsvm
NCSO	Número de *clusters* sem ontologia
tSSO	Tempo de execução, em segundos, do algoritmo *Subtractive* sem ontologia
NCGO	Número de *clusters* com Gene Ontology
tSGO	Tempo de execução, em segundos, do algoritmo *Subtractive* com Gene Ontology
NCM	Número de *clusters* com Merops
tSM	Tempo de execução, em segundos, do algoritmo *Subtractive* com Merops

Tabela 35 – Resultados dos classificadores com extrações de características usando diferentes valores de *Prune Frequence* no *Dataset 1*

Prune Frequence	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	Gamma	tLIBSVM	Accuracy Libsvm
10	370	7	10	57	14	10	79,17	Linear	1		28,15	99,72
20	785	9	41	59,67	3	14	58,5	RBF	1	2	3,4	94,5
30	1031	9	49	59,5	3	14	55	Linear	1		4,02	96
40	1267	10	74	59,17	4	10	57,67	Linear	4		7,81	96,28
50	1568	11	78	58,83	4	10	56,5	Linear	1		5,74	96,72
60	1604	11	91	59,33	4	10	55,5	Linear	1		9,29	97,5
70	2198	13	100	60,67	4	10	58,17	Linear	1		9,59	97,11
80	2289	13	105	60,17	5	10	58,67	Linear	1		7,82	97,67
90	2751	14	186	60,17	5	10	58,67	Linear	1		10,95	97,72
100	2426	13	90	81,83	6	10	58,33	Linear	4		11,47	97,56
110	2772	14	87	83,83	5	10	66,83	Linear	1		8,5	99,17
120	2388	13	77	81,5	6	11	84,17	Linear	1		8,63	99,39
130	3297	16	168	81,17	5	11	68,17	Linear	1		11,34	98,72
140	2906	15	193	69,17	6	11	68,33	Linear	2		9,65	98,94
150	2700	14	174	69,33	6	10	64	Linear	1		0,38	98,89
160	3783	18	228	67,83	6	10	63,83	Linear	1		12,36	98,78
170	3720	18	230	69	7	10	64,5	Linear	2		13,44	98,78
180	3486	16	220	69,17	7	12	64,83	Linear	1		13,58	98,5
190	3249	15	215	68,5	7	10	64,67	Linear	1		13,25	98,39

200	3072	15	176	69,67	7	11	64,33	Linear	1		10,2	98,33
210	4518	19	218	83,83	6	10	67,17	Linear	2		9,98	99
220	4432	19	209	83,17	8	11	81,67	Linear	2		11,31	99,44
230	4343	19	191	83,83	8	11	82,83	Linear	1		13,98	99,39
240	4218	19	208	84,5	8	12	81,5	Linear	1		14,25	99,28
250	4184	18	213	85,17	8	12	81,67	Linear	1		0,36	99,28
260	4059	18	205	83,33	8	12	81	Linear	2		11,15	99,39
270	3957	18	194	85	7	11	81,5	Linear	1		10,98	99,39
280	3764	17	211	86	7	11	82,17	Linear	1		0,38	99,33
290	3695	17	182	86,83	7	11	82	Linear	2		13,44	99,39
300	3587	16	139	87,33	7	10	80,5	Linear	1		10,73	99,33
310	5769	23	263	95,33	7	10	81,5	Linear	1		13,95	99,33
320	5737	23	267	95,33	10	10	83	Linear	1		16,09	99,61
330	5691	23	262	86,83	10	10	83,5	Linear	1		16,18	99,56
340	5660	23	265	95,83	10	10	83,83	Linear	1		16,17	99,56
350	5619	23	245	88	10	10	84,83	Linear	1		13,13	99,56
360	5544	23	256	87,17	9	10	84,67	Linear	1		13,09	99,56
370	5518	23	257	87	9	11	84	Linear	1		0,39	99,56
380	5492	23	253	87	9	11	84,33	Linear	1		0,4	99,56
390	5439	22	253	86,83	9	11	84,5	Linear	1		15,39	99,56
400	5412	22	262	87,17	9	11	84,33	Linear	1		15,63	99,67
410	5333	22	272	87	9	11	82	Linear	2		12,79	99,67
420	5305	22	255	86,83	9	12	82,17	Linear	1		0,39	99,67
430	5286	22	236	87	9	12	82,67	Linear	2		15,51	99,67
440	5268	22	233	87	9	10	83,5	Linear	1		15,25	99,67
450	5234	22	239	87	9	12	82,83	Linear	1		15,34	99,67
460	5218	21	250	87	9	12	82,83	Linear	1		15,35	99,61
470	5151	21	243	87,17	10	12	82,17	Linear	1		15,87	99,61
480	5106	21	228	87	9	12	82,33	Linear	1		12,52	99,67
490	5051	21	215	86,33	9	12	82,33	Linear	1		12,53	99,67
500	5042	21	218	86,33	9	12	82,67	Linear	1		12,46	99,67
510	5004	21	244	87,5	9	12	82,33	Linear	1		12,47	99,67
520	4907	21	238	86	9	12	82,83	Linear	1		12,45	99,56
530	4874	21	232	86	9	12	82,83	Linear	2		15,06	99,72
540	4810	21	236	86,5	9	12	82,67	Linear	1		12,33	99,72
550	4720	20	202	93,67	9	12	82,67	Linear	1		15,52	99,72
560	4635	20	200	86,17	8	12	82,83	Linear	1		14,83	99,72
570	4593	20	203	86,17	8	12	82,5	Linear	1		12,15	99,72
580	4536	19	176	86	8	12	82,33	Linear	1		12,1	99,72
590	4499	19	161	86,83	8	11	80,83	Linear	2		12,32	99,72
600	4441	19	172	86,83	8	11	80,67	Linear	1		12,09	99,67
Sem PruneFrequence	8883	33	764	83,17	8	12	81,33	Linear	1		15,09	99,67

Tabela 36 – Resultados dos diferentes classificadores com extrações de *features* usando diferentes valores de *ngrams* no *Dataset 3*

Ngram	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	Gamma	tLIBSVM	Accuracy Libsvm

1	5706	10	105	82,8	9	12	67,2	Linear	1	1	7,25	98,56
2	30346	47	1303	82,4	28	12	67,6	Linear	1	1	17,58	98,72
3	59607	91	2787	74	35	12	66,8	Linear	1	1	25,34	98,72

Tabela 37 - Resultados dos diferentes classificadores com extrações de *features* usando diferentes valores de *ngrams* no *Dataset 4*

Ngram	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	tLIBSVM	Accuracy Libsvm
1	5264	10	79	97,2	16	5	75,2	Linear	1	7,09	99,52
2	28457	45	1154	98,8	18	15	75,6	Linear	1	17,76	99,52
3	56293	96	2308	98,8	33	17	77,2	Linear	1	28,52	99,52

Tabela 38 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 1*

Tipo Vector	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	tLIBSVM	Accuracy Libsvm
TermOccurences	8883	33	860	68,5	14	17	66,33	Linear	1	46	99,5
TermFrequency	8883	32	555	95,83	16	12	85,83	Linear	1	16,38	99,56
TFIDF	8883	33	1647	83,17	14	10	79,17	Linear	2	20,83	99,67

Tabela 39 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 2*

Tipo Vector	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	tLIBSVM	Accuracy Libsvm
TermOccurences	8742	32	635	67,5	15	16	70	Linear	1	15,22	99,5
TermFrequency	8742	32	512	87,5	15	8	76,67	Linear	1	13,01	99,56
TFIDF	8742	33	824	88,83	16	8	84,5	Linear	2	20,8	99,67

Tabela 40 - Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 3*

Tipo Vector	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	tLIBSVM	Accuracy Libsvm
TermOccurences	5706	10	104	76	7	13	64,8	Linear	1	10,77	98,24
TermFrequency	5706	8	79	85,2	7	12	67,6	Linear	1	4,4	98,56
TFIDF	5706	8	103	82,8	7	12	67,2	Linear	1	5,76	98,56

Tabela 41 – Resultados dos diferentes classificadores com extrações de características usando diferentes tipos de criação de vectores da matriz *BOW* no *Dataset 5*

Tipo Vector	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	tLIBSVM	Accuracy Libsvm
TermOccurences	5523	10	127	62	6	11	71,2	Linear	1	8,05	98,72
TermFrequency	5523	8	71	86,8	9	15	74	Linear	1	4,1	99,36
TFIDF	5523	8	137	89,2	11	20	70,8	Linear	1	4,01	99,36

Tabela 42 - Resultados dos diferentes classificadores com extrações de características usando ontologias no *Dataset 1*

Ontology	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	Gamma	tLIBSVM	Accuracy Libsvm
NoOntology	8883	31	812	83,17	16	10	79,17	Linear	1	1	21,46	99,72
GeneOntology	1016	42	23	79,33	5	14	78,83	Linear	1	1	3,19	99,5
Merops	116	5	5	79,5	4	10	96,5	RBF	1	0,00012207	0,84	99,39

Tabela 43 - Resultados dos diferentes classificadores com extrações de características usando ontologias no *Dataset 2*

Ontology	Nfeatures	tEF	tKM	Qualidade KM	tFCM	FCM Exponent	Qualidade FCM	Kernel	Cost	Gamma	tLIBSVM	Accuracy Libsvm
NoOntology	8742	27	822	88,83	15	8	84,5	Linear	1	1	21,61	99,44
GeneOntology	996	21	36	87,17	6	15	83,33	Linear	1	1	0,37	99,44
Merops	107	4	6	93,83	7	8	77,67	RBF	1	0,00012207	0,7	99,33

Tabela 44 – Resultados do algoritmo *Subtractive clustering* em extrações de *features* sem ontologias, com Gene Ontology e com Merops usando diferentes valores no parâmetro *radii*

Radii	NCSO	tSSO	NCGO	tSGO	NCMO	tSMO
0,05	600	647	600	88	8	10
0,1	600	616	600	82	8	1
0,15	600	655	600	84	8	1
0,2	600	616	600	81	8	1
0,25	600	585	600	80	8	1
0,3	600	597	600	84	8	1
0,35	600	593	600	87	8	1
0,4	600	586	600	84	8	0
0,45	600	634	600	85	8	8
0,5	600	608	600	84	8	1
0,55	600	615	600	82	8	1
0,6	600	642	600	87	8	0
0,65	600	593	600	82	8	7
0,7	600	634	600	84	8	1
0,75	600	602	600	80	8	1
0,8	600	613	600	83	8	1
0,85	600	624	600	85	8	7
0,9	600	610	600	90	8	1
0,95	600	617	600	88	8	1
1	600	634	600	86	8	1
1,05	600	620	600	85	8	1
1,1	600	606	600	84	8	1
1,15	600	617	600	84	8	0
1,2	600	579	600	86	7	7
1,25	600	586	600	83	6	1
1,3	600	590	600	86	6	1

1,35	600	585	600	87	6	8
1,4	600	599	600	80	6	1
1,45	600	601	599	86	6	1
1,5	600	575	597	86	4	1
1,55	600	615	597	87	3	1
1,6	600	634	594	86	2	1
1,65	600	592	586	85	2	1
1,7	600	621	569	78	1	1
1,75	600	643	569	79		
1,8	600	623	527	77		
1,85	600	604	527	75		
1,9	600	589	442	70		
1,95	600	621	389	65		
2	600	598	321	57		
2,05	600	632	242	51		
2,1	600	620	159	41		
2,15	600	615	38	30		
2,2	600	657	7	27		
2,25	600	645	7	21		
2,3	600	632	6	27		
2,35	600	614	5	20		
2,4	600	700	5	27		
2,45	600	645	4	21		
2,5	600	601	4	20		
2,55	600	634	3	15		
2,6	600	630	3	16		
2,65	600	632	3	15		
2,7	600	623	3	15		
2,75	600	625	2	15		
2,8	600	638	2	15		
2,85	600	605	2	15		
2,9	600	602	2	15		
2,95	600	595	2	21		
3	600	591	2	15		
3,05	600	596	1	14		
3,1	600	607				
3,15	600	620				
3,2	600	587				
3,25	600	592				
3,3	600	602				
3,35	600	588				
3,4	600	611				
3,45	600	606				
3,5	600	600				
3,55	600	576				

3,6	600	596				
3,65	600	588				
3,7	600	586				
3,75	599	587				
3,8	595	589				
3,85	591	583				
3,9	588	588				
3,95	583	581				
4	580	580				
4,05	575	580				
4,1	569	573				
4,15	559	562				
4,2	549	559				
4,25	537	545				
4,3	527	549				
4,35	512	527				
4,4	501	518				
4,45	484	503				
4,5	468	493				
4,55	448	483				
4,6	428	459				
4,65	409	442				
4,7	386	424				
4,75	362	414				
4,8	334	391				
4,85	306	367				
4,9	273	340				
4,95	241	313				
5	200	287				
5,05	167	260				
5,1	125	230				
5,15	78	197				
5,2	39	171				
5,25	19	156				
5,3	19	155				
5,35	10	151				
5,4	10	152				
5,45	10	151				
5,5	10	150				
5,55	9	152				
5,6	9	150				
5,65	8	177				
5,7	8	148				
5,75	7	149				
5,8	6	148				
5,85	6	147				
5,9	5	151				

5,95	5	150				
6	5	148				
6,05	4	146				
6,1	4	146				
6,15	4	146				
6,2	3	146				
6,25	3	145				
6,3	3	146				
6,35	3	145				
6,4	3	145				
6,45	3	146				
6,5	3	146				
6,55	3	147				
6,6	3	145				
6,65	3	146				
6,7	3	147				
6,75	3	147				
6,8	3	146				
6,85	2	145				
6,9	2	139				
6,95	2	148				
7	2	139				
7,05	2	140				
7,1	2	146				
7,15	2	146				
7,2	2	137				
7,25	2	147				
7,3	2	138				
7,35	2	140				
7,4	2	146				
7,45	2	145				
7,5	2	147				
7,55	2	146				
7,6	2	147				
7,65	2	146				
7,7	2	145				
7,75	2	146				
7,8	2	138				
7,85	2	143				
7,9	2	145				
7,95	2	146				
8	2	144				
8,05	2	145				
8,1	1	138				