Classificação Automática de Emails

por

Maria João Alves Lima

2013

Tese de Mestrado - Análise de Dados e Sistemas de Apoio à Decisão

Orientadores:
Pavel Brazdil
João Gama

Palavras-chave: *text-mining*, email, classificação, *machine learning*

Resumo

As mensagens de email são utilizadas, cada vez mais, como meio de comunicação entre as pessoas. As mensagens podem ser de foro pessoal ou profissional e, dada a grande quantidade de mensagens que cada pessoa recebe por dia, é cada vez mais essencial organizar essas mensagens para não se perder informações importantes. O problema consiste na falta de disponibilidade de tempo em organizar manualmente essas mensagens. Para tal, surge a necessidade de que a classificação seja feita de forma automática. O objectivo deste trabalho é verificar se é possível utilizar a classificação automática para resolver este problema. Para alcançar o objectivo foi necessário preparar os dados de treino. Numa primeira fase, os dados fornecidos em formato de excel foram processados de forma a criar pastas com as mensagens em formato.dat no computador. De seguida, foram aplicados métodos de pré-processamento e processamento tradicionais de *text-mining*, como remoção de *stopwords*, transformação dos dados numa matriz documento-termo, etc. Por fim foi feito um estudo comparativo de quatro diferentes classificadores, Naive Bayes, Redes Neuronais, Máquina de Suporte Vectorial e k-Vizinhos mais Próximos, de forma a seleccionar o classificador que melhor performance apresenta na resolução deste problema. Os classificadores foram testados utilizando classificação binária e classificação multi-classe em linguagem de programação R. A classificação binária serviu para verificar se o sistema permitia distinguir se uma determinada mensagem pertence a uma dada pasta ou não. Nesta fase os resultados revelaram-se muito satisfatórios, obtendo valores de F1 superiores a 0.9 para algumas pastas. A classificação multi-classe tem como objectivo determinar qual a pasta correcta a que a mensagem pertence. O objectivo foi atingido pelo classificador SVM com uma taxa de erro inferior a 0.4.

Abstract

Email messages are used, increasingly, as a mean of communication between people. The messages can be personal or professional, and given the large number of messages that each person receives everyday, it is increasingly essential to organize these messages not to miss important information. The problem is the lack of time available to organize these messages manually. To this end, is necessary that the classification be done automatically. The aim of this work is to verify whether it is possible to use the automatic classification to solve this problem. To achieve the goal it was necessary to prepare the training data. Initially, the data provided in excel format were processed to create folders with messages in.dat format. Next, pre-processing and processing methods of traditional text-mining were applied, such as the removal of stopwords, processing data in a document-term matrix, etc. Finally, was made a comparative study of four different classifiers, Naive Bayes, Neural Networks, Support Vector Machine and k-nearest neighbors in order to select the best classifier performance in solving this problem. The classifiers were tested using binary classification and multi-class classification in the programming language R. A binary classification was used to verify that the system allowed to distinguish whether a particular message belongs to a particular folder or not. At this stage the results were very satisfactory, resulting in F1 values greater than 0.9 for some folders. A multi-class classification aims to determine the proper folder to which the message belongs. The goal was achieved by the SVM classifier with a error rate lower than 0.4.

Índice

1. Introdução

1.1. Motivação

As mensagens de email fazem parte do dia a dia actual, podendo ser pessoais ou profissionais. Os emails servem também como uma ferramenta de arquivo, existindo mensagens cuja informação poderá vir a ser útil mais tarde. Desta forma, surge a necessidade de organizar as mensagens recebidas, separando-as por pastas.

"Folders and messages are constantly being created, destroyed, and reorganized" (Segal and Kephart 1999), estes são os obstáculos com que vários utilizadores se cruzam todos os dias na sua caixa de email e que procuram ultrapassar de forma a manter uma boa organização dos seus emails.

A classificação das mensagens de emails pertence à área de *Text Mining* e pode ser feita de forma manual ou automática. Manualmente o utilizador atribui cada mensagem a uma determinada pasta; na forma automática, é utilizado um algoritmo que aprenderá que tipo de mensagens existe em cada pasta e, a cada nova mensagem que chegue à caixa de entrada, ele decide qual a pasta a que pertence essa nova mensagem. O utilizador pode ainda optar por definir determinadas regras que serão aplicadas à classificação das novas mensagens, no entanto essas regras são limitadas e, por isso, a sua definição pode tornar-se rapidamente um trabalho cansativo para o utilizador (Xiao-Lin and Cloete 2005).

1.2. Estrutura

A resolução do problema da classificação de email resume-se em três tarefas: a representação dos dados e o seu processamento, a escolha intuitiva dos classificadores e a selecção do modelo.

O trabalho começa com uma breve exploração do tema, métodos e técnicas utilizadas para a classificação de emails resumidas no capítulo 2. No capítulo 3 é feita uma descrição dos dados utilizados neste trabalho e no capítulo 4 e 5 encontra-se o processamento desses dados, desde a remoção de palavras com baixo nível de

informação à transformação dos dados numa matriz documento-termo. O processo de treino dos classificadores, a classificação e os resultados estão descritos no capítulo 6.

Todas as etapas deste trabalho foram realizadas em programação R, os códigos utilizados encontram-se em anexo.

2. Estado da arte

2.1. Text Mining

A *Text Mining* (ou Mineração de Textos) é uma técnica que assenta na extracção de informação e conhecimento de dados não estruturados. É também conhecida como Descoberta de Conhecimento de Textos (KDT). Usa técnicas de *Information Retrieval* (Recuperação de Informação), de extracção de informação assim como de processamento de linguagem natural (NLP) e relaciona-os com algoritmos e métodos de KDT (Hotho, Nürnberger et al. 2005). O problema da KDT consiste em extrair conceitos explícitos e implícitos e as relações semânticas entre os conceitos e tem como objectivo extrair conhecimento de grandes quantidades de dados de texto.

O processo de *Text Mining* divide-se em diversas fases. A primeira fase é o processamento do texto que consiste na remoção de pontuação, transformando o texto em palavras singulares, remoção de *stop-words*, que envolve a remoção de caracteres HTML e XML assim como de palavras com elevada frequência mas com baixo teor de informação, e a transformação de todas as palavras na sua origem (*stemming*). A segunda fase é a transformação do texto em *bag of words* (saco de palavras) ou num vector modelo. A terceira fase inclui a selecção de atributos. Os atributos com baixo teor de informação são eliminados, mantendo apenas os que apresentam informação relevante. A quarta fase é a fase mais complexa, nesta fase são utilizados os métodos de *text-mining* como o *clustering*, a classificação, a *Information Retrieval*, *etc.* A última fase inclui a avaliação e interpretação dos resultados obtidos com na fase quatro através da precisão, sensibilidade, medida F1, etc.

Um dos objectivos da *Text Mining* consiste em que um computador consiga fazer uma análise de textos assim como o ser humano, para tal, várias técnicas têm sido utilizadas. Essas técnicas consistem na extracção de informação, onde são identificadas frases-chave e a sua relação com o texto, na sumarização, que consiste em condensar o texto mantendo a informação, a classificação, ou seja, atribuir classes aos documentos, e o processo de linguagem natural (NLP), que tem como objectivo construir um sistema que analise, compreenda e gerar NLP (Patel and Soni 2012).

2.2. Representação e Processamento dos Dados

A mensagem de email é composta por diferentes atributos como "De", "Para", "Assunto" e "Corpo da mensagem". O conteúdo de cada atributo são todas as palavras que ocorrem nesse campo. A representação das mensagens é geralmente em forma de *bag of* words. O processamento mais comum deste tipo de textos passa por converter todas as palavras em palavras singulares, remover a pontuação e discriminar os endereços de email pelas palavras que o constituem. De seguida são eliminadas palavras de elevada frequência mas com baixo conteúdo de informação, como por exemplo "a", "os", a maioria das preposições e conjunções (Provost 1999).

Mais recentemente, Katakis, Tsoumakas et al. (2006) consideram que o modelo de representação mais predominante é o modelo de espaço vectorial. Nesta abordagem, cada mensagem é representada por um vector. Cada elemento da mensagem é associado a um atributo, que pode ser uma frase ou uma palavra. Geralmente os elementos dos vectores utilizam valores boleanos (0 ou 1) de forma a detectar a presença ou ausência de alguma particular palavra ou frase na mensagem, ou utilizam pesos (valores entre 0 e 1) para revelar a importância de cada palavra ou frase na mensagem (frequência). Relativamente ao pré-processamento das mensagens, as palavras que pertencem à mesma família são tratadas como uma só, pela sua raiz, por exemplo, "programa", "programador" e "programação". De seguida são eliminadas as *stop-words* por não terem poder de discriminação, no entanto, o uso dessas palavras pode ser determinante na identificação do autor do e-mail. Destacam ainda que uma palavra que aparece no campo "assunto" pode ser mas importante que a mesma palavra aparecendo no "corpo da mensagem", ou seja, o peso da palavra é superior quando aparece no "assunto". Os autores sugerem o uso de TF-IDF para o cálculo dos pesos de forma a que seja intuitivo que uma palavra é importante para a mensagem se ela aparecer muitas vezes nessa mensagem e, ao mesmo tempo, não ser uma palavra comum, isto é, que aparece em muitas mensagens.

A abordagem mais recente do processamento dos dados é apresentado por Chakravarthy, Venkatachalam et al. (2010) que tomam em consideração toda a estrutura de uma mensagem. As experiências realizadas estão explicadas na secção de classificação de email por pastas (secção 2.4.9).

2.3. Selecção de atributos

A selecção dos atributos consiste em encontrar as palavras que melhor distingam as categorias das mensagens de forma mais eficiente. Por norma, são calculadas medidas de qualidade especiais para cada palavras e de seguida são seleccionados os N melhores atributos. Uma das medidas mais tipicamente usadas é o TF-IDF, ainda aplicável, mas agora é possível utiliza-la para o total do *corpus* e não para um documento só. Existem ainda outras medidas no campo da teoria da informação que são amplamente mais aplicáveis e que se tornaram populares na área de classificação de emails, devido à sua eficácia em *Text Mining*, como o Ganho de Informação (Information Gain) e a medida do Chi-quadrado (Katakis, Tsoumakas et al. 2006). Os autores referem ainda a medida da Informação Mútua (Mutual Information) que indica quanto poder um termo tem de forma a distinguir a sua categoria das restantes.

Gomez and Moens (2010) testaram a validade de um método de extracção estatística de atributos que se baseia na redução da dimensão de forma a manter os atributos mais informativos das mensagens. A sua abordagem chama-se Análise Discriminante com Tendência (BDA) e consiste em encontrar uma transformação do espaço dos atributos que agrupe rigorosamente exemplos positivos enquanto afasta os exemplos negativos. Este novo espaço deverá expressar, em termos estatísticos, as diferenças e similaridades entre as mensagens. O método BDA foi aplicado a quatro diferentes conjuntos de dados, mas antes do método ser implementado os emails foram transformados em vectores. Primeiro foi removida a estrutura da informação, de seguida removeram as palavras com baixo nível de informação usando estatísticas de informação mútua, obtendo 5000 atributos iniciais. Por fim, as palavras escolhidas foram pesadas através de um esquema TF-IDF enquanto representavam cada mensagem como um vector. Os resultados revelaram uma boa performance quando o BDA é utilizado na filtração de emails usando validação. Demonstou também que os resultados não são dependentes do número de atributos pois estes mantiveram o seu valor discriminativo ao longo do tempo.

Os estudos realizados na área de classificação de emails baseiam-se em dois objectivos, classificação de emails por pastas e detecção de spam.

2.4. Detecção de spam

2.4.1. Custos de diferenciação de erros de classificação

Na área de detecção de spam, Sahami, Dumais et al. (1998) utilizaram uma abordagem bayesiana, redes bayesianas, para a filtração de spam. Consideraram que, aplicando o problema a um quadro teórico de decisão, é possível fazer uso de métodos probabilísticos de aprendizagem, em conjunto com os custos de diferenciação de erros de classificação, de forma a produzir filtros apropriados para esta problemática. Para a experiência foram usadas 1789 mensagens das quais 1578 estavam classificadas como spam e 211 como não-spam. No pré-processamento dos dados utilizaram uma análise Zipf que eliminou as palavras com uma frequência inferior a três vezes, ou seja, palavras que tinham menos de três ocorrências. Relativamente ao conjunto de atributos escolhidos foram realizados testes com diferentes regimes de atributos sendo que o regime com maior precisão utilizava como atributos as palavras, as frases e o domínio específico. Dado que, na classificação de emails como spam e não-spam os custos de má classificação diferem, isto é, o custo de classificar um email não-spam como spam é superior ao contrário, os autores aplicaram uma regra que determina que um email só é considerado como spam caso a probabilidade de ser spam seja superior a 99,9%. Os resultados obtidos levaram à conclusão de que atributos de domínio específico ao problema, a linha de texto das mensagens e a consideração dos custos de má classificação é possível produzir filtros com boa precisão.

2.4.2. Estudo comparativo: Árvore de Decisão, Naive Bayes, Redes Neuronais e Máquina de Suporte Vectorial

Youn and McLeod (2007) realizaram um estudo comparativo da performance de quatro algoritmos, Árvore de Decisão (J48), Naive Bayes, Redes Neuronais e Máquina de Suporte Vectorial. A experiência consistiu em classificar emails como sendo spam ou não-spam utilizando diferentes tamanhos de conjuntos de dados e diferentes números de atributos. A classificação através de Redes Neuronais consiste em três passos: processamento dos dados, treino e teste. No processamento dos dados foram seleccionados os atributos que revelaram mais informação, sendo posteriormente utilizados para o treino dos dados. O classificador foi aplicado aos dados de teste e a

eficiência foi calculada através de um algoritmo de erro *Back Propagation*. Relativamente ao classificador Máquina de Suporte Vectorial (SVM), o algoritmo aprende considerando diferentes exemplos de mensagens em que cada mensagem corresponde a um ponto do conjunto de dados que pode estar classificado como -1 ou +1. As duas classes são separadas por um hiperplano que minimiza a distância entre os pontos dentro de cada classe, sendo essa distância conhecida como vectores de suporte, e maximiza a distância entre as duas classes. O classificador Naive Bayes considera a probabilidade de determinado documento *d* com um vector *x* pertencer à categoria *c* tendo em conta que cada ponto do vector é condicionalmente independente dada a categoria. Quanto ao classificador J48 foi utilizada uma simples árvore binária. Os resultados obtidos com as experiências realizadas revelaram que os classificadores que demonstram melhor performance são o Naive Bayes e o J48, sendo este um resultado previsto dado tratar-se de uma decisão binária.

2.5. Classificação de email por pastas

2.5.1. Estudo comparativo: RIPPER e Método tradicional de IR

Relativamente à classificação de emails por pastas existem já algumas abordagens apresentadas. Cohen (1996) comparou dois métodos para classificação de emails, um tradicional método de IR (*information retrievial*) com base em pesos TF-IDF e um classificador baseado em regras chamado RIPPER, tendo focado particularmente a aprendizagem de categorias de mensagens através de um pequeno conjunto de mensagens já classificadas. No método IR os documentos ou mensagens são representados em vectores cujos elementos correspondem às palavras do corpo da mensagem. Para uma mensagem, o valor da componente da palavra depende da frequência com que a palavra ocorre na mensagem e do tamanho da mensagem. A aprendizagem é feita adicionando os vectores que correspondem aos exemplos positivos da classe *C* e subtraindo os vectores correspondentes aos exemplos negativos, desta forma produz-se um "vector protótipo" para a classe *C*. Os vectores das mensagens podem ser classificados de acordo com a sua distância ao vector protótipo. Um documento é classificado como positivo se a distância é inferior a um determinado limiar que pode ser escolhido de forma a equilibrar a precisão. Na experiência realizada

o limiar foi escolhido de forma minimizar o erro no conjunto de treino. O segundo algoritmo utilizado na experiência foi o algoritmo RIPPER que constrói um conjunto de regras adicionando repetitivamente regras a um conjunto de regras vazio até que todos os exemplos positivos estejam cobertos. De seguida é feita uma optimização ao conjunto de regras de modo a reduzir o seu tamanho e assim melhorar o seu ajuste aos dados de treino. Em todas as experiências realizadas neste artigo foram tidas em consideração o corpo e o cabeçalho das mensagens que revelaram que ambos os métodos obtêm generalizações significativas a partir de um pequeno número de exemplos e, assim, é possível comparar as suas performances para este tipo de problemas. Ambos os métodos são razoavelmente eficientes, no entanto, uma maior compreensão das regras poderá ser mais vantajoso para o utilizador, pois este poderá modificar as regras quando necessário.

2.5.2. Dinâmica e heterogeneidade

Vários são os desafios na área de classificação de textos ligados à distribuição de emails por pastas. Brutlag and Meek (2000) apresentaram um artigo focando-se em três desses desafios. Os desafios passam pela natureza dinâmica do domínio do email, pela heterogeneidade dos conteúdos e pelos tamanhos das pastas, e a memória prática e limitações da CPU na sua implementação. As tecnologias de classificação usadas na experiência são a Máquina de Suporte Vectorial (SVM), um classificador TF-IDF e um modelo de linguagem simples chamado "unigram model". Os dados utilizados na experiência corresponderam a vários conjuntos de dados estáticos de diferentes emails. As mensagens foram extraídas e de seguida foi feita a selecção de atributos. Relativamente ao pré-processamento dos dados foram eliminadas as *stop-words* assim como as palavras com muito baixa frequência. O conjunto de teste foi seleccionado usando 20% das mensagens mais recentes. A Máquina de Suporte Vectorial foi treinada para cada pasta em que os exemplos positivos constituíam nas mensagens que pertenciam a essa pasta. Os restantes classificadores foram treinados a partir dos vectores de palavras. Na experiência foram utilizadas cinco caixas de email com o objectivo de comparar os três classificadores de forma a estabelecer métodos de aprendizagem que ofereçam uma precisão competitiva apesar da heterogeneidade do conteúdo e do tamanho das pastas. Os resultados mostraram que o métodos de

aprendizagem testados oferecem uma boa competitividade de precisão comparado com o método SVM discriminativo. Quanto ao tamanho das pastas, o estudo revelou que a performance dos classificadores varia consoante se as pastas são densas ou dispersas, ou seja, se o mecanismo de um classificador aumentar os pesos da contagem das pastas dispersas então essas pastas irão subir para o topo da lista na classificação. Relativamente aos custos computacionais, foi demonstrado que é possível recorrer a classificadores incrementais computacionalmente tratáveis sem sacrificar a precisão.

2.5.3. Co-training

Um dos problemas da classificação de textos é a inexistência de dados rotulados assim como o facto de ser dispendioso, ao nível do tempo, rotular esses dados. Desta forma, Kiritchenko and Matwin (2001) aplicaram o algoritmo Co-Training em dados de email. Este algoritmo foi inicialmente desenvolvido por Blum and Mitchell (1998). A ideia base assenta na utilização de dados não rotulados juntamente com alguns dados já rotulados para aumentar o desempenho de um classificador. Assumindo que, por vezes, os atributos que descrevem os dados são redundantes e que podem ser divididas em dois conjuntos, cabeçalho e corpo da mensagem, em que cada um por si só é suficiente para uma classificação correcta, é possível construir dois classificadores, um para cada conjunto. Se um classificador consegue encontrar um exemplo que seja muito semelhante a um dos exemplos já classificados, então pode prever com confiança a classe desse exemplo transformando-o em exemplo de treino para o outro classificador, melhorando assim a performance do segundo classificador. Na experiência os autores utilizaram o email de um utilizador que contém 1500 mensagens divididas por três pastas com 250, 500 e 700 mensagens respectivamente. Com estes três grupos foi criados três problemas de duas classes, a classe positiva e a classe negativa. Portanto, cada problema tem uma diferente distribuição de exemplos negativos e positivos como 1:5 (problema altamente desequilibrado), 1:2 (problema moderadamente desequilibrado) e 1:1 (problema equilibrado). Para cada problema 25% dos exemplos foram seleccionados como teste, os restantes foram escolhidos consoante a proporção descrita acima mantendo a maioria dos exemplos não classificados. Os algoritmos implementados foram o Naive Bayes e a Máquina de Suporte Vectorial. Os resultados da experiência revelaram que o desempenho do co-training depende do algoritmo de

aprendizagem utilizado. Com os dados desta experiencia, o Naive Bayes não obteve uma boa performance, ao contrário do que aconteceu com o SVM (Máquina de Suporte Vectorial) que revelou sucesso na aplicação. A explicação apontada para este facto cai sobre a escassez de vectores característicos, caso que o SVM consegue lidar com eficiência, mas que é crucial para o Naive Bayes.

2.5.4. Diferentes bases de dados

Bekkerman, McCallum et al. (2004) apresentaram um extenso estudo comparativo de classificação de emails por pastas utilizando dois conjuntos de mensagens reais disponíveis: Enron Corporation (500.000 mensagens e 150 utilizadores) e CALO DARPA (22.000 mensagens e 196 utilizadores) que pertence a um projecto de pesquisa SRI. Na experiência foram utilizados um conjunto de métodos de classificação automáticos e metodologias de avaliação, proporcionando um alto nível de precisão com uma fracção de tempo de treino de métodos alternativos. Os classificadores utilizados foram o Naive Bayes, Máxima Entropia, Máquina de Suporte Vectorial e uma variante do Winnow que foram aplicados em dois tipos diferentes de experiências e em ambos os conjuntos de dados. A primeira experiência consistiu em considerar uma linha de tempo, calcularam a precisão para cada divisão de treino e teste e traçaram uma curva de precisão sobre o número de mensagens de treino na divisão. Se o conjunto de teste contém mensagens de uma paste recém-criada, e portanto nenhuma mensagem desta pasta pode ser encontrada no conjunto de treino, então estas mensagens serão ignoradas no cálculo da precisão. A segunda experiência está relacionada com a cobertura. A precisão é traçada sobre o percentual de cobertura do conjunto de teste. Para cada divisão treino/teste, após a acção do classificador, as mensagens de teste são ordenadas de acordo com a pontuação obtida na classificação, de seguida é atribuído um limiar à lista ordenada e os primeiros 10%, 20%, ..., 100% das mensagens são escolhidas e por fim calcula-se a precisão em cada um dos 10 limiares e a média de precisão em cada limiar sobre todas as divisões do conjunto de treino/teste. Após a aplicação dos diferentes classificadores concluíram que o Winnow apresenta uma boa performance, às vezes até superior aos restantes classificadores mais populares. No entanto, a maior parte dos resultados obtidos apresentam valores de precisão relativamente baixos, abaixo dos 70%.

2.5.5. Factor temporal

Kiritchenko, Matwin et al. (2004) propuseram uma nova solução para o problema de classificação integrando informações temporais com as abordagens tradicionais de atributos com base no conteúdo. É sabido que cada mensagem tem um carimbo de tempo, ou seja, existe uma sequência de itens em que cada um contem um factor temporal e conteúdo. Esta informação temporal está sempre presente mas por norma é ignorada pela maioria dos investigadores. Os autores investigaram a potencial relevância que a informação temporal tem nas tarefas comuns no domínio do email. A incorporação da informação temporal passa por extrair os atributos temporais como o dia da semana e a hora do dia em que a mensagem foi recebida através de padrões sequenciais. Um novo algoritmo MINTS (*MINing Temporal Sequential patterns*) foi definido para descobrir padrões sequenciais através dos intervalos de tempo entre os eventos dos padrões. Para adicionar os intervalos de tempo ao algoritmo são necessárias duas acções. Primeiro, quando se combinam pares de padrões de 1-evento para fazer padrões de 2-eventos é preciso considerar todas as possibilidades de intervalos de tempo entre os eventos. Segundo, quando se combina um par de padrões de n-eventos para produzir padrões de (n+1)-eventos, os padrões de n-eventos devem ter o mesmo prefixo, o que inclui não apenas o mesmo tipo de evento mas também o mesmo tempo de intervalo entre os eventos. Para a transformação da informação temporal em atributos existem três métodos: "Classe prevista" em que uma classe prevista através dos padrões de tempo é adicionada ao conjunto de atributos, treina o classificador e por fim é usada para prever a classe de uma nova mensagem; "*Cascading*" que utiliza a probabilidade da classe prevista através dos padrões de tempo e junta-a ao conjunto de atributos; "Padrões", neste caso todos os padrões temporais são adicionados ao conjunto de atributos e o sistema de classificação irá aprender quais os padrões e em que contexto serão utilizados. Na experiência realizada, a classificação ficou a cargo de três métodos: "*Simple Vote*" que consistiu em treinar o classificador Naive Bayes e obter um conjunto de probabilidades para as classes de cada elemento do conjunto de teste, de seguida é aplicado o classificador baseado nos padrões temporais ao mesmo conjunto de teste, obtendo assim um segundo conjunto de probabilidade para as classes, por fim as probabilidades são comparadas e escolhe-se a que tiver maior probabilidade; outro método é o "*replacing Uncertainties*" que é igual ao "*Simple Vote*" no entanto aplica

um limiar ao classificador Naive Bayes, caso a probabilidade não ultrapasse esse limiar escolhe-se a classe prevista pelo classificador temporal; o ultimo método é o "*Replacing A Priori Probabilities*" que simplesmente altera as probabilidades a priori do classificador Naive Bayes pelas probabilidades calculadas pelos padrões temporais. A experiência foi aplicada a diversos conjuntos de emails e concluíram que foi possível reduzir o erro de classificação em 10% utilizando as relações temporais.

2.5.6. Representação com base em expressões

Crawford, Koprinska et al. (2004) apresentaram um estudo sobre a efectividade da utilização de uma representação com base em expressões na classificação de emails e o efeito desta abordagem em algoritmos de aprendizagem. Avaliaram também vários métodos de selecção de atributos e da redução dos níveis de representação de *bag of words* nos diversos algoritmos. A construção deste tipo de representação passou pela selecção e remoção de *stop-words.* De seguida as expressões são estatisticamente seleccionadas usando o método de ganho de informação ou frequência nas mensagens. Na experiência utilizaram o email de quatro diferentes utilizadores, havendo um critério de classificação diferente para cada um. As mensagens dos utilizadores foram ordenadas por data considerando os primeiros 2/3 como dados de treino e o restante como teste. Para cada utilizador 5%, 10% e 30% dos atributos dos *bag of words* foram calculados e este número foi utilizado para definir o número de atributos tanto para a representação em *bag of words* como para a representação por expressões. Os algoritmos implementados foram a Maquina de Suporte Vectorial, k-Vizinhos Mais Próximos, Árvore de Decisão, Perceptron, Widrow-Hoff e Naive Bayes, através do software WEKA. O estudo levou à conclusão que existe uma forte diferença na dificuldade de categorizar automaticamente os emails dos diferentes utilizadores. A combinação dos classificadores e dos métodos de selecção de atributos afectam a performance, revelando ser necessário avaliar os métodos de selecção de atributos através dos diferentes classificadores. A representação com base em expressões revelou ser eficiente apresentando uma boa performance. Os melhores classificadores para ambas as diferentes representações das mensagens foram a Máquina de Suporte Vectorial, o Widrow-Hoff e o k-Vizinhos Mais Próximos. A selacção de atributos

mostrou-se útil para todos os classificadores com a excepção da Máquina de Suporte Vectorial e do Widrow-Hoff.

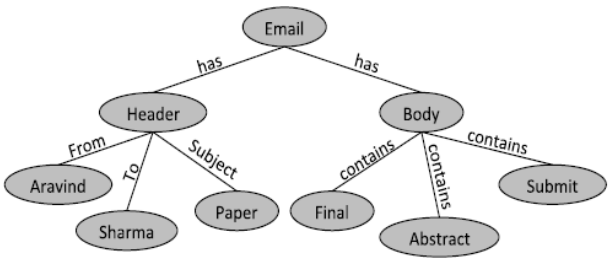
2.5.7. TaskPredictor

Shen, Li et al. (2006) construíram um sistema chamado TaskTracer que reconhece as tarefas realizadas pelos utilizadores e ajuda a gerir os seus recursos. O sistema necessita de uma notificação, por parte do utilizador, cada vez que altera as suas actividades, mas muitas vezes os utilizadores esquecem-se de fazer a notificação e, por isso, os autores introduzem um novo sistema de machine learning, TaskPredictor, que tenta prever a actividade actual do utilizador. Este sistema tem duas componentes: uma para a actividade do ambiente de trabalho e outra específica para o email. A elevada precisão do sistema é alcançada combinando três técnicas: selecção de recursos através de informação mútua; classificação com base num limiar de confiança; e um projecto hibrido em que um classificador Naive Bayes estima a confiança para a classificação mas onde a decisão real de classificação é feita pelo classificador Máquina de Suporte Vectorial. O sistema produzido para a classificação de email é o TaskPredictor.email. Neste sistema o processamento dos dados é feito criando um atributo para cada remetente, outro atributo para o destinatário e outro atributo para o assunto da mensagem, cada conjunto de destinatários é tratado como um atributo independente. Consideraram que o corpo da mensagem não tem qualquer valor preditivo adicional. Relativamente ao limiar de precisão, o sistema considera que se a probabilidade prevista se encontrar acima do limiar de precisão, então é feita uma previsão, caso contrário nada é feito. Também nesta abordagem é tido em conta duas medidas de performance, a cobertura e a precisão. No entanto o objectivo é sempre o de atingir a máxima precisão mesmo tendo como custo uma baixa cobertura. Relativamente à escolha dos atributos, os autores aplicaram três diferentes métodos: eliminação de palavras com baixo nível de informação; aplicação de um algoritmo simples com base em regras para converter as palavras às suas raízes; e aplicação de informação mútua (ganho de informação), uma medida de relevância que mede a redução da entropia na previsão da distribuição da classe através do valor do atributo, para seleccionar os 200 atributos com maior poder preditivo que irão integrar no conjunto de treino. Quanto à classificação, o sistema aplica uma combinação de dois classificadores, Naive Bayes e SVM, em que o primeiro

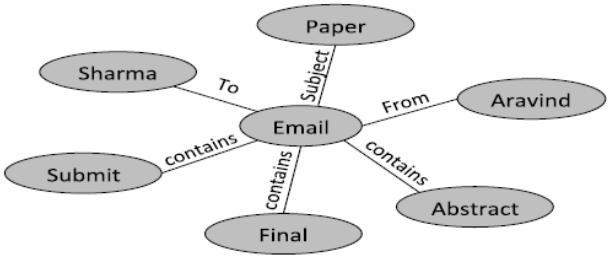
calcula a probabilidade condicionada de ser determinada classe dado determinada mensagem e o segundo diz que se existem k classes então k(k-1)/2 classificadores têm de ser treinados. Por fim, para prever a classe de uma nova mensagem cada classificador faz uma previsão e essas previsões são combinados de forma a produzir uma probabilidade estimada. Os resultados da experiência revelaram que é possível obter uma precisão de 90% com 65% de cobertura.

2.5.8. Graph Mining

A grande maioria das técnicas desenvolvidas para a classificação de emails por pastas dependem apenas da extracção de palavras-chave de alta frequência, ignorando assim os aspectos estruturais inerentes a uma mensagem de email, que pode ter um papel importante na classificação. Desta forma, Chakravarthy, Venkatachalam et al. (2010) apresentaram uma nova técnica de classificação baseada em grafos, *graph mining*. A ideia principal é que uma pasta pré-classificada pode ser usada para extrair subestruturas que representem uma pasta com precisão. Relativamente ao pré-processamento dos dados, o objectivo é facilitar a retenção de subestruturas frequentes ao longo da mensagem. Assim, todas as palavras que compõem as subsestruturas têm de ser retidas na mensagem também. As palavras têm de ocorrer com frequência através de todas as mensagens e não apenas de uma. Desta forma é controlada a disparidade de umas mensagens serem maiores do que outras. Em primeiro lugar as *stop-words* são eliminadas. De seguida as palavras pertencentes à mesma família são transformadas numa única palavra, a palavra raiz. Por fim a selecção de atributos é feita classificando as palavras em classes com base nas suas frequências em todos os documentos. As palavras que ficam retidas são as que cujas frequências representem mais de *f*% da soma de todas as frequências. O parâmetro *f* foi ajustado através de experiências de



Email: Tree Representation



Email: Star Representation

forma a assegurar que as palavras escolhidas são frequentes não apenas numa única mensagem. As representações em grafos são escolhidas com base no conhecimento do domínio. Os autores propuseram duas diferentes representações, em árvore e em estrela. A representação em árvore inicia-se com o tipo de documento como raiz e em seguida ramifica-se com base no domínio. Este esquema considera toda a informação em cada mensagem com cada palavra conectada à raiz. A representação em estrela é constituída por uma raiz central. As palavras escolhidas formam os restantes vértices, juntamente com as arestas que os conectam à raiz central. Após ambas as representações terem sido construídas, os autores afirmam que a representação mais rica é a representação em árvore. Para a experiência decidiram comparar o *graph-mining* ao Naive Bayes. Os resultados revelaram uma melhoria significativa em relação ao Naive Bayes,

3. Base de dados e pré-processamento

A base de dados utilizada para a realização deste trabalho foi cedida pela empresa Idryl. Os dados são referentes a mensagens de email de uma instituição de ensino e são compostos por 2240 mensagens distribuídas por 24 pastas.

Nome da pasta	Número de mensagens por pasta
p145	1
p154	1
p155	1
p102	2
p202	2
p156	9
p206	15
p5	16
p158	16
p190	17
p137	34
p189	37
p191	61
p207	61
p160	62
p431	72
p157	88
p501	96
p159	175
p148	184
p188	192
p203	213
p205	248
p101	637

Os dados apresentam-se em dois formatos, um documento de excel que contem a informação de cabeçalho dos emails como “de”, “para”, “data” e “assunto”, assim como

o código da mensagem que corresponde aos ficheiros .dat onde se encontra o corpo da mensagem.

O objectivo consiste em verificar se é possível, através da análise do texto de cada mensagem, atribuir cada mensagem a uma pasta. Para isso é necessária a construção de um classificador.

3.1. Organização e estrutura dos dados

O primeiro passo para a organização dos dados passou por estabelecer uma relação entre todos os documentos, ou seja, como foi dito acima, relacionar o documento excel com os documentos .dat. Assim, foi criada uma função em R que faz a ligação do código da mensagem que se encontra no documento excel ao documento .dat com o mesmo código. Desta forma, ao corpo da mensagem que está em formato .dat é lhe atribuído uma classe, ou seja, a pasta em que está inserido.

Códigos R utilizados:

Em primeiro lugar foi criada a função para ler o ficheiro excel:

```
FichBase<-read.csv("MFB.csv",sep=";")
```

```
> FichBase[1:5,]
  Código CodAnexo            Pasta            Data                                                                  Assunto        De.Converted.       Para.Converted.
1  17439     3585 email:/203/saidas 27/02/2013 15:09 Informação de reabertura de aula para registo de sumário servdoc@gooportal.com antmelo@gooportal.com
2  17438     3584 email:/203/saidas 27/02/2013 14:45 Informação de reabertura de aula para registo de sumário servdoc@gooportal.com mgarcia@gooportal.com
3  17411     3583 email:/205/saidas 26/02/2013 21:09 Informação de reabertura de aula para registo de sumário servdoc@gooportal.com epanyik@gooportal.com
4  17410     3582 email:/205/saidas 26/02/2013 21:08 Informação de reabertura de aula para registo de sumário servdoc@gooportal.com  apalha@gooportal.com
5  17409     3581 email:/205/saidas 26/02/2013 21:06 Informação de reabertura de aula para registo de sumário servdoc@gooportal.com  florbm@gooportal.com
  Anexo
1 email
2 email
3 email
4 email
5 email
  '
```

De seguida foi criada uma função que seleccionou apenas a coluna “Pasta”, e daí restringiu-se apenas aos números das pastas, ou seja, pôs-se de parte as palavras “email” e “saídas” que apareciam nessa coluna, seleccionando apenas a coluna 2 da coluna “Pastas”:

```
nomespastas<-strsplit(as.character(FichBase$Pasta),'/',fixed=TRUE)
as.data.frame(nomespastas[1:10])[2,]
```

A organização dos cabeçalhos das mensagens do documento excel foi feita através da criação de listas consoante a pasta a que pertenciam. O código a seguir

revelado demonstra a criação de listas de uma classificação binária, ou seja, se uma mensagem pertence à pasta X vai para a lista X, caso contrário vai para a lista Xn (X corresponde ao nome da pasta):

```
mudar.listas <- function (nomespastas, pasta) {
        lista1 <- list()
        lista2 <- list()
linha <- 1
poslista1<-1
poslista2<-1

#
for (i in linha : nrow(FichBase)) {

if (as.data.frame(nomespastas[linha])[2,]==pasta)
{
lista1[[poslista1]]<-FichBase[linha,]
 poslista1 <- poslista1 + 1}
 else {
 lista2[[poslista2]]<-FichBase[linha,]
 poslista2<-poslista2 + 1}
linha<-linha + 1
}

#
listas <- list()
listas[[1]]<-lista1
listas[[2]]<-lista2
return(listas) }


lista.res<-mudar.listas(nomespastas, "101") – #101 é o exemplo de uma pasta
```

O passo seguinte foi criar, dentro do computador, pastas com os ficheiros .dat organizados, ou seja, os ficheiros classificados:

```
dat.listas <- function (lista1,lista2) {
        lista.dat1 <- list()
        poslista1<-1

        for (i in (1:length(lista1))) {
        nome.fich1 <- lista101[[poslista1]][1,"CodAnexo"]
        poslista1 <- poslista1 + 1
        nome.fich11 <- paste(nome.fich1,"dat",sep=".")
        nome.pasta.ant <- paste("DAT files/", nome.fich11, sep="")
        nome.pasta.nova <- paste("Datpos/", nome.fich11, sep="")
        if (file.exists(nome.fich11)) {
```

```
        lista.dat1[[poslista1]] <- file.copy(nome.pasta.ant, nome.pasta.nova)
        }
                }

                lista.dat2 <- list()
                poslista2<-1
        for (i in (1:length(lista2))) {
        nome.fich2 <- lista101n[[poslista2]][1,"CodAnexo"]
        poslista2 <- poslista2 + 1
        nome.fich22 <- paste(nome.fich2,"dat",sep=".")
        nome.pasta.ant <- paste("DAT files/", nome.fich22, sep="")
        nome.pasta.nova <- paste("Datneg/", nome.fich22, sep="")
        if (file.exists(nome.fich22)) {
        lista.dat2 [[poslista2]] <- file.copy(nome.pasta.ant, nome.pasta.nova)
        }
        }

        listas <- list()
        listas[[1]] <- lista.dat1
        listas[[2]] <- lista.dat2
        return(listas)
}
```

Este código criou pastas consoante a função "mudar.listas", isto é, criou apenas duas pastas, a pasta "datpos" e a pasta "datneg". A pasta "datpos" contem as mensagens que pertencem à pasta escolhida, no caso acima demonstrado é a pasta p101, e a "datneg" contem as restantes mensagens. Manualmente criou-se uma pasta no computador chamada "101" e para lá moveram-se os ficheiros da pasta "datpos". Este processo foi executado para todas as pastas até estarem todas organizadas.

A criação do corpus foi feita através do seguinte código R:

```
p145 <- Corpus(DirSource("145",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p154 <- Corpus(DirSource("154",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p155 <- Corpus(DirSource("155",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p202 <- Corpus(DirSource("202",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p102 <- Corpus(DirSource("102",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p156 <- Corpus(DirSource("156",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p206 <- Corpus(DirSource("206",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p158 <- Corpus(DirSource("158",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p190 <- Corpus(DirSource("190",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p137 <- Corpus(DirSource("137",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p189 <- Corpus(DirSource("189",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p191 <- Corpus(DirSource("191",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p207 <- Corpus(DirSource("207",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p160 <- Corpus(DirSource("160",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p431 <- Corpus(DirSource("431",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p157 <- Corpus(DirSource("157",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p501 <- Corpus(DirSource("501",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
```

```
p159 <- Corpus(DirSource("159",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p148 <- Corpus(DirSource("148",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p188 <- Corpus(DirSource("188",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p203 <- Corpus(DirSource("203",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p205 <- Corpus(DirSource("205",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p101 <- Corpus(DirSource("101",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
p5 <- Corpus(DirSource("5",encoding="Latin1"), readControl=list(reader=readPlain, language="portuguese"))
```

Exemplo de leitura de um documento:

```
inspect(p101[1])

<div>Exmo(a) &amp;nbsp;Prof. Doutor JOSÃ‰ MANUEL MARTINS LOPES,&amp;nbsp;</div>
<div>&amp;nbsp;</div>
<div>Serve o presente para comunicar que foi aprovado o seu acesso Ã &amp;nbsp;&amp;nbsp;plataforma de GestÃ£o Letiva (Registo de sumÃ¡rios).</div>
<div>Para o efeito foi-lhe atribuÃ-da a seguinte senha: </div>
<div>&amp;nbsp;</div>
<div>Com os melhores cumprimentos</div>
<div>&amp;nbsp;</div>
<div>GestÃ£o Letiva (UCP - CRBR)</div>
```

Com este exemplo é possível verificar que, com a utilização do *encoding* "Latin1", algumas palavras não aparecem escritas na sua forma original. Por exemplo, o palavra original "Gestão" foi lida como "GestÃ£o". No entanto, é de realçar que esta situação não afecta a classificação pois a transformação foi aplicada a todas as palavras e, portanto, a palavra "Gestão" foi transformada da mesma forma em todos os documentos, ou seja, no caso de esta ser uma palavra representativa de uma classe, o sistema irá reconhece-la dessa forma também.

A escolha do *encoding* "Latin1" deve-se ao facto de o *encoding* "UTF-8", escolhido em primeiro lugar e que mantinha a originalidade das palavras, apresentar um erro quando os dados foram transformados para a matriz documento-termo.

3.2. Pré-processamento dos dados

Para extrair informação de grandes conjuntos de textos, como é o caso de mensagens de email, é necessário fazer um pré-processamento desses textos e organizar a informação extraída de forma estruturada. Esta etapa visa essencialmente fazer a remoção de dados desnecessários facilitando uma melhor classificação dos emails.

3.2.1. Remoção das *stopwords*

As *stopwords* são palavras cujo teor de informação é bastante baixo mas que apresentam elevada frequência, como é o caso de palavras como “de”, “a” e “mas”. A sua eliminação é importante pois reduz o tamanho dos documentos mantendo apenas as palavras essenciais para uma extracção mais eficaz da informação.

3.2.2. Remoção de caracteres HTML

No mesmo código utilizado para a remoção das *stopwords* foram introduzidas as palavras “div”, “br” e “nbsp” que, em conjunto com o código que elimina a pontuação, removeram os caracteres de HTML presentes nos ficheiros de dados fornecidos. Apesar de existirem mais caracteres de HTML que poderiam ser introduzidos no código, neste caso não foi necessário por não aparecerem nos dados.

Para a remoção dos caracteres HTML haviam outros códigos que poderiam ter sido implementados, no entanto, estes código apresentavam erros quando aplicados a dados do tipo corpus. Os códigos utilizados encontram-se na secção 8.3.3.

Código R utilizado para a remoção de *stopwords* e HTML:

```
tm_map(docs.p, removeWords, c("div","br","nbsp",stopwords('portuguese')))
```

3.2.3. Remoção de espaços em branco

As mensagens de email são documentos escritos muitas vezes em contexto mais apressado e, desta forma, a probabilidade de erros como a inserção de espaços em branco extras torna-se mais elevada. Portanto, surgiu a necessidade de os eliminar, mantendo apenas os espaços em branco necessários para separar as palavram e manter a organização do texto uniformizada.

Código R utilizado para remoção de espaços em branco:

```
tm_map(docs.p, stripWhitespace)
```

3.2.4. Transformação de letras maiúsculas em minúsculas

Os algoritmos reconhecem palavras como sendo iguais se estiverem escritas da mesma forma, ou seja, se "Maria" e "maria" pertencerem ao mesmo *corpus*, o algoritmo irá reconhece-las como palavras diferentes.

Para manter todas as palavras iguais escritas da mesma forma foi utilizado um código que transformou todas as letras maiúsculas em minúsculas.

Código R utilizado para transformação de letras maiúsculas em minúsculas:

```
tm_map(docs.p, tolower)
```

3.2.5. Remoção de pontuação e números

A pontuação e os números foram removidos por não conterem informação relevante para a classificação.

Código R utilizado para remoção de pontuação e números:

```
tm_map(docs.p, removePunctuation)
tm_map(docs.p, removeNumbers)
```

3.2.6. Exemplo

Mensagem antes do pré-processamento	Mensagem após o pré-processamento
  Bom dia AgradeÃ§o que complete a ficha de colaborador. Sem todos os dados nÃ£o poderemos proceder ao pagamento. verifico por exemplo que nÃ£o tem os dados bancÃ¡rios para se proceder Ã transferÃªncia. com estima, NatÃ¡lia Costa Diretora Financeira 	bom dia agradeã complete ficha colaborador sem dados nã poderemos proceder pagamento verifico exemplo nã dados bancãrios proceder ã transferãncia estima natãlia costa diretora financeira

4. Processamento dos dados

4.1. Matriz Documento-Termo

Nesta etapa cada documento é transformado na sua forma numérica, ou seja, cada documento é decomposto em termos e frequências, é criado um *Bag of Words*. O *Bag of Words* é o mesmo que a Matriz Documento-Termo em que cada linha dessa matriz representa um documento e cada coluna representa um termo, os restantes elementos indicam a frequência de cada termo em cada documento.

Exemplo:

Utilizando os documentos da pasta p202:

```
A document-term matrix (2 documents, 20 terms)

Non-/sparse entries: 28/12
Sparsity           : 30%
Maximal term length: 16
Weighting          : term frequency (tf)

          Terms
Docs       antãnio automaticamente celina consulting efectuar enviado envio
  1504.dat       1               1      1          1        1       1     1
  1658.dat       0               1      0          1        0       1     0
          Terms
Docs       estou favor gooportal graã idassinaturamail idryl mensagem obrigado
  1504.dat     1     1         1    1                1     1        1        1
  1658.dat     0     0         1    0                1     1        0        0
          Terms
Docs       olã plataforma por responde teste
  1504.dat   1          1   1        1     1
  1658.dat   0          1   0        0     1
```

Neste caso temos uma matriz com duas linhas (2 documentos) e 20 colunas (20 termos).

Utilizando os dados de todas as pastas:

```
A document-term matrix (2240 documents, 3629 terms)

Non-/sparse entries: 66866/8062094
Sparsity           : 99%
Maximal term length: 171
Weighting          : term frequency (tf)
```

Obtemos uma matriz com 2240 linhas (2240 documentos) e 3629 colunas (3629 termos).

4.2. Termos frequentes

O cálculo da frequência de termos é bastante importante para a classificação de documentos, ajuda a remover termos desnecessários, desta forma diminui o tamanho da matriz Documento-Termo e, por conseguinte, melhora a performance dos classificadores.

Os termos com mais frequência do conjunto de dados deste trabalho são os seguintes: (Termos com pelo menos 250 ocorrências.)

```
> findFreqTerms( dtm.mx, 250)
 [1] "ã\u0081rea"         "acesso"             "agradece"
 [4] "alteraãães"         "ano"                "aprovaãã"
 [7] "aprovado"           "atividade"          "aula"
[10] "aulas"              "caso"               "classmsonormalspan"
[13] "colaborador"        "com"                "comunicar"
[16] "conforme"           "consulta"           "contabilidade"
[19] "controlo"           "crbr"               "cumprimentos"
[22] "curricular"         "curso"              "definitivo"
[25] "deverã"             "dezembro"           "dia"
[28] "disponãvel"         "docente"            "docentes"
[31] "documento"          "efeito"             "efetuou"
[34] "estado"             "exmo"               "ficha"
[37] "gestã"              "indicaãã"           "indicado"
[40] "informa"            "letiva"             "licenciatura"
[43] "mail"               "maria"              "mãs"
[46] "melhores"           "mensal"             "menu"
[49] "mestrado"           "normal"             "opãã"
[52] "parãmetro"          "para"               "pelo"
[55] "plataforma"         "preenchimento"      "presente"
[58] "proceda"            "proceder"           "psicologia"
[61] "reaberta"           "registado"          "registo"
[64] "relativa"           "respetivo"          "seguinte"
[67] "serve"              "serviã"             "silva"
[70] "situaãã"            "sumãrio"            "sumãrios"
[73] "trabalho"           "ucp"                "ucpcrbr"
[76] "unidade"            "utilizar"           "verificaãã"
[79] "verificar"
```

4.3. Remoção de termos de baixa frequência

Por fim eliminaram-se os termos com pouca frequência utilizando a função r*emoveSparseTerms* com um máximo de 0.95 de *sparsity* mantendo apenas 148 termos significativos.

```
A document-term matrix (2240 documents, 148 terms)

Non-/sparse entries: 46531/284989
Sparsity           : 86%
Maximal term length: 33
Weighting          : term frequency (tf)
```

Código R utilizado:

```
removeSparseTerms( dtm.mx, 0.95)
```

5. Classificação

O primeiro passo para a classificação é escolher o melhor classificador a utilizar e, para isso, é necessário testar diferentes classificadores. O teste aos classificadores consiste em treinar cada classificador com um conjunto de treino e depois aplica-lo a um conjunto de teste.

Os resultados dessa classificação irão devolver uma matriz confusão onde constam os dados bem classificados em cada classe assim como os dados mal classificados em cada classe.

As experiências de classificação foram feitas para classificação binária e classificação multi-classe. Apesar de o objectivo ser a classificação multi-classe, a opção de realizar também a classificação binária deve-se ao facto de, desta forma, ser mais fácil avaliar a performance dos classificadores.

A matriz que se segue representa uma matriz confusão para classificação binária, mas é também aplicada a classificação multi-classe.

Matriz confusão:

Classes	A - Prevista	B - Prevista
A - Verdadeira	VP	FN
B - Verdadeira	FP	VN

VP – Verdadeiro Positivos FN – Falsos Negativos FP – Falsos Positivos VN – Verdadeiros Negativos

Para comparar os classificadores são usados a taxa de erro de classificação (% de casos mal classificados) para classificação binária e multi-classe e a precisão (proporção de exemplos positivos classificados correctamente entre todos aqueles preditos como positivos), a sensibilidade (recall, taxa de acerto da classe positiva) e a medida F1 (medida de teste de precisão) para classificação binária. Todas estas medidas são calculadas a partir da matriz confusão.

Taxa de erro = $\frac{FN+FP}{Total}$

Precisão = $\frac{VP}{VP+FP}$

Sensibilidade = $\frac{VP}{VP+FN}$

F1 = $2 * \frac{Precisão * Sensibilidade}{Precisão+Sensibilidade}$

Os classificadores são treinados com base num conjunto de treino criado a parir dos dados. Neste caso, o conjunto de treino representa 70% dos dados totais, os restantes 30% vão ser utilizados para testar os classificadores e calcular as medidas de comparação.

Nas experiências realizadas foram testados quatros algorítmos: Máquina de Suporte Vectorial (*Support Vector Machine*), k-Vizinhos mais próximos (k-Nearest Neighbors), Naive Bayes (do WEKA) e Redes Neuronais. Estes algorítmos foram escolhidos através das conclusões retiradas do estado da arte onde se verifica que estes são os classificadores com melhores resultados nesta área.

As experiências dividiram-se em classificação binária e classificação multi-classe. A classificação binária foi explorada para testar os classificadores em problemas simples de *text-mining*. Na classificação binária foi também testada entre dados equilibrados e não-equilibrados, ou seja, classes com e sem o mesmo número de exemplos.

5.1. Classificadores

5.1.1. Redes Neuronais

O algoritmo Redes Neuronais é um método de classificação baseado em optimização. Consiste num modelo inspirado pela estrutura biológica do cérebro humano, o responsável pelo processamento de diversas informações e geração de respostas. Esta é a motivação que levou a que o algoritmo fosse desenvolvido tendo em conta a estrutura e funcionamento do sistema nervoso, sendo o objectivo simular a capacidade de aprendizagem do cérebro humano na aquisição de conhecimentos.

As Redes Neuronais são sistemas compostos de unidades de processamento simples densamente interconectadas. Estas unidades são conhecidas como neurónios artificiais e estão dispostas por uma ou mais camadas e interligadas por um elevado número de conexões. Essas conexões possuem pesos que ponderam a entrada recebida

por cada neurónio. Os pesos podem ter valores positivos ou negativos que são obtidos através de um processo de aprendizagem.

Uma Rede Neuronal é caracterizada por duas características básicas: arquitectura e aprendizagem. A arquitectura está relacionada com o tipo e número de unidades de processamento e à forma como os neurónios estão conectados, a aprendizagem consiste nas regras utilizadas para o ajuste dos pesos e que informação é utilizada pelas regras (Gama, Carvalho et al. 2012).

Código utilizado na classificação com Redes Neuronais:

```
library(nnet)
nnet(class~., data=dtm.tr, size=2, rang=0.1, decay=5e-4, maxit=200)
```

Legenda:
class – classse
data – dados de treino
size – número de unidades nas camadas intermédias
rang – pesos aleatórios iniciais
decay – parâmetro para a deterioração dos pesos (valor perto de zero)
maxit – número máximo de iterações

5.1.2. k-Vizinhos mais próximos

O algoritmo dos *k*-vizinhos mais próximos (*k*-NN) é um método com base em distâncias. Este algoritmo varia consoante o número de vizinhos considerados, ou seja, varia consoante o valor de *k*.

Cada objecto representa um ponto num espaço que é definido pelos atributos, chamado espaço de entrada. Definindo uma métrica nesse espaço é possível calcular a distância entre dois objectos. A distância mais comum é a distância euclidiana, usada neste trabalho.

A fase de treino deste algoritmo passa pela memorização dos exemplos de treino, ou seja, objectos já classificados. Para classificar um novo objecto é calculada a distância entre o vector de valores dos atributos e cada exemplo de treino em memória. No caso de $k=1$, a classe associada ao exemplo de treino mais próximo do exemplo de teste é utilizada para classificar esse novo objecto. Relativamente ao caso de $k>1$, cada objecto de teste obtém *k* vizinhos. Cada vizinho vota numa classe e o objecto de teste é classificado consoante a classe mais votada (Gama, Carvalho et al. 2012).

Código utilizado na classificação com *k*-NN:

```
library(class)
knn(dtm.tr, dtm.ts, dtm.tr$class, k=1)
```

Legenda:
dtm.tr – dados de treino
dtm.ts – dados de teste
dtm.tr$class – classes dos dados de treino
k – número de vizinhos considerados

5.1.3. Máquina de Suporte Vectorial (SVM)

Tal como as Redes Neuronais, a Máquina de Suporte Vectorial é um método baseado em optimização. Este algoritmo resolve problemas de classificação lineares e não-lineares. As SVM's linares são utilizadas na classificação de conjuntos de dados linearmente separáveis ou que possuam uma distribuição aproximadamente linear. Mas na maioria dos casos, os dados não são lineares e por isso é necessária a SVM não linear.

As SVM's não lineares lidam com problemas não lineares mapeando o conjunto de treino do seu espaço original para um novo espaço de maior dimensão (espaço de características). Este procedimento é realizado com base no teorema de Cover. Dado um conjunto de dados não linear num espaço de entradas *X*, esse teorema afirma que *X* pode ser transformado num espaço de características no qual os objectos são linearmente separáveis com alta probabilidade. Para isso é necessário satisfazer duas condições: a transformação não pode ser linear e a dimensão do espaço de características deve sser suficientemente alta.

Mapeiam-se inicialmente os objectos para um espaço de dimensão maior e aplica-se a SVM linear sobre esse espaço. Ela encontra o hiperplano com maior margem de separação. A informação necessária para o mapeamento é a forma como é realizado o cálculo de produtos escalares entre os objectos no espaço de características. Isso é obtido através do uso de funções chamadas kernels.

Um kernel é uma função que recebe dois pontos no espaço de entradas e calcula o produto escalar desses objectos no espaço de características. Os kernels mais utilizados são os polinomiais, os de base radial (RBF) e os sigmoidais (Gama, Carvalho et al. 2012). O kernel RBF é, por norma, uma boa escolha inicial dado que o kernel linear é considerado como um caso especial de função RBF (Keerthi and Lin 2003).

Código utilizado na classificação com SVM:

```
library(e1071)
svm.classifier <- svm(class~., dtm.tr)
```

5.1.4. Naive Bayes

O classificador Naive Bayes é um método probabilístico com base no Teorema de Bayes. Este classificador assume que os valores dos atributos de um exemplo são independentes da sua classe, ou seja, a probabilidade de um evento *A* (que pode ser uma classe), dado um evento *B* (que pode ser o valor do conjunto de atributos de entrada), não depende apenas da relação entre *A* e *B*, mas também da probabilidade de observar *A* independentemente de observar *B*.

A probabilidade de ocorrência do evento *B* pode ser estimada pela observação da frequência com que esse evento ocorre. Da mesma forma, é possível estimar a probabilidade de ocorrência de um evento *B* para cada classe *A*, P(*B*|*A*). A dúvida assenta no calculo da probabilidade de ocorrer *A* dado *B*, P(*A*|*B*). O Teorema de Bayes resolve este problema utilizando a probabilidade *a priori* da classe, P(*A*). Para calcular a probabilidade *a priori* da classe é necessário manter um contador para cada classe e assumir uma distribuição norma para os atributos.

Código utilizado na classificação com Naive Bayes:

```
library(RWeka)
NB<-make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
```

5.2. Experiências e resultados

5.2.1. Classificação binária

Dados não-equilibrados

Para a classificação binária foi escolhida uma pasta para a classe positiva e as restantes pastas para a classe negativa. Na primeira experiência, a classe positiva foi composta pela pasta p101 por ser a pasta com mais documentos. Os documentos foram classificados segundo se pertenciam à pasta p101 ou não.

Classe	Número de mensagens
Positiva - p101	637
Negativa - Restantes	1603

Como referido acima, foram utilizados quatro classificadores. Em seguida estão representados os resultados de classificação obtidos assim como as medidas de comparação.

	Redes Neuronais	k - NN	Naive Bayes	SVM
Matriz confusão	neg pos neg 461 30 pos 36 145	neg pos neg 464 27 pos 33 148	neg pos neg 472 19 pos 73 108	neg pos neg 447 44 pos 27 154
Taxa de erro	0.098	**0.089**	0.137	0.106
Sensibilidade	0.939	0.945	0.961	0.910
Precisão	0.928	0.934	0.866	0.943
F1	0.933	**0.939**	0.911	0.926

Os resultados revelam uma boa performance de todos classificadores sendo que todos apresentam taxas de erro reduzidas e valores de F1 próximos de 1. O classificador que se destaca é o classificador Redes Neuronais.

Na segunda experiência de classificação binária foi escolhida a pasta p205 por ser a segunda pasta com mais mensagens.

Classe	Número de mensagens
Positiva – p205	248
Negativa - Restantes	1992

Resuldados obtidos:

	Redes Neuronais	k - NN	Naive Bayes	SVM
Matriz confusão	neg pos neg 580 6 pos 15 71	neg pos neg 577 9 pos 17 69	neg pos neg 549 37 pos 10 76	neg pos neg 583 3 pos 22 64
Taxa de erro	**0.031**	0.039	0.069	0.037
Sensibilidade	0.989	0.979	0.937	0.995
Precisão	0.975	0.971	0.982	0.964
F1	**0.982**	0.975	0.959	0.979

Novamente foram obtidos bons resultados para os classificadores. Neste caso houve uma melhoria notória em relação ao caso anterior por terem sido utilizados dados ainda menos balanceados. Os resultados apontam de novo para o classificador Redes Neuronais como o classificador com melhor performance.

A experiência foi repetida para a pasta p203. Neste caso os dados encontram-se ainda menos equilibrados, obtendo assim a classe positiva com 213 mensagens e a classe negativa com 2027 mensagens. Os classificadores Redes Neuronais, k-NN e SVM revelaram uma boa performance com taxas de erro a rondar os 3% e a medida F1 com valor bastante perto de 1. Relativamente ao Classificador Naive Bayes verifica-se uma diminuição na sua performance com valor de taxa de erro perto de 20% e a medida F1 com valor igual a 0.89.

Apesar de os testes aos dados não-equilibrados revelarem bons resultados foram também realizados testes aos parâmetros dos classifcadores k-NN e Redes Neuronais para verificar se era possível melhorar a sua performance. No classificador k-NN foi aumentado o número de vizinhos mais próximos (k) para três, quanto às Redes Neuronais foi aumentado o número de iterações para 500. Os resultados obtidos encontram-se na seguinte tabela:

p101	Redes Neuronais	k-NN
Taxa de erro	0.110 (0.098)	**0.095** (0.089)
F1	0.926 (0.933)	**0.936** (0.939)

p205	Redes Neuronais	k-NN
Taxa de erro	**0.031** (0.031)	0.042 (0.039)
F1	**0.982** (0.982)	0.976 (0.975)

Entre parênteses estão os resultados anteriores.

Em geral, os resultados mostraram uma pequena diminuição na performance, no entanto estas alterações não são significativas. Os classificadores que se destacaram em cada experiência mantêm-se igual à primeira experiência.

Em suma, os testes aos dados não-equilibrados revelam bons resultados, no entanto não são bons indicadores da performance dos classificadores. Para poderem ser retiradas melhores conclusões foram feitos novos testes, desta vez com dados equilibrados.

Dados equilibrados

Nesta nova experiência foi escolhida a pasta p101 para ser a classe positiva. Na classe negativa foram incluidas pastas até atingir um número de mensagens próximo da classe positiva.

Classe	Pasta	Número de mensagens
Positiva	p101	637
Negativa	p205 + p203 + p188	653

	Redes Neuronais	k - NN	Naive Bayes	SVM
Matriz confusão	neg pos neg 171 29 pos 24 164	neg pos neg 184 16 pos 31 157	neg pos neg 191 9 pos 65 123	neg pos neg 172 28 pos 24 164
Taxa de erro	0.137	**0.121**	0.191	0.134
Sensibilidade	0.855	0.92	0.955	0.86
Precisão	0.877	0.856	0.746	0.878
F1	0.866	**0.887**	0.838	0.869

Os resultados indicam uma ligeira diminuição na performance dos classificadores. As taxas de erro aumentaram e a medida F1 diminuiu, no entanto estes valores representam ainda uma boa performance. Neste caso o classificador a destacar é o k-NN, com taxa de erro mais baixa e F1 mais elevado.

Como foi feito com os dados não-equilibrados, também neste caso foram realizadas experiências aos parâmetros dos classificadores Redes Neuronais e k-NN, sendo que as alterações foram as mesmas que utilizadas anteriormente. As experiências obtiveram os seguintes reultados:

	Redes Neuronais	k-NN
Taxa de erro	0.113 (0.137)	**0.106** (0.121)
F1	0.899 (0.866)	**0.908** (0.887)

Entre parênteses estão os resultados anteriores.

Neste caso verifica-se uma melhoria na performance de ambos os classificadores, no entanto esta melhoria não é muito significativa, havendo apenas melhorias de cerca de 1%. O classificador destacado com melhor performance mantem-se o mesmo que na primeira experiência (k-NN).

5.2.2. Classificação multi-classe

A classificação multi-classe é o objectivo da classificação de emails. A ideia base é, através da análise do texto de várias mensagens de email, classificar correctamente uma nova mensagem de email, atribuindo-lhe uma pasta.

Esta classificação pode ser feita através de dois diferentes processos: construir um único classificador para multi-classe ou construir um conjunto de classificadores binários e combina-los de forma a obter a previsão desejada. O segundo processo é bastante complexo tendo-se optado pela utilização do primeiro processo.

Como foi dito anteriormente, as caixas de email estão organizadas por diversas pastas, assim vamos obter uma matriz confusão 24x24. Nesta experiência, a comparação entre classificadores utilizando a matriz confusão com todas as classes é feita através da taxa de erro.

As matrizes confusão multi-classe são apresentadas em baixo:

Redes Neuronais

	101	137	148	156	159	188	203	205	207	5	501
101	166	0	5	0	2	16	1	6	0	0	1
102	0	0	0	0	0	0	0	0	0	0	0
137	2	9	0	0	0	0	0	0	0	0	0
145	0	0	0	0	0	0	0	0	0	0	0
148	0	0	43	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0
156	1	1	0	1	0	1	0	0	0	0	0
157	1	0	26	0	0	0	0	0	0	0	0
158	2	0	0	0	0	4	0	0	0	0	0
159	5	0	1	0	2	45	0	1	0	0	0
160	1	0	0	0	0	16	0	1	0	0	1
188	7	7	0	0	4	40	1	1	5	0	0
189	0	0	10	0	0	0	0	0	0	0	0
190	0	6	0	0	0	0	0	0	0	0	0
191	0	0	13	0	0	0	0	0	0	0	0
202	0	0	0	0	0	0	0	0	0	0	0
203	4	0	1	0	1	0	20	43	0	0	0
205	2	0	0	0	1	1	2	59	0	0	0
206	2	0	0	0	0	1	0	3	0	0	0
207	18	1	0	0	0	0	0	0	0	0	0
431	2	1	0	0	5	17	0	0	0	0	0
5	0	1	0	0	0	0	0	1	0	2	1
501	0	0	0	0	0	0	0	0	0	0	28

Máquina de Suporte Vectorial (SVM)

	101	102	137	145	148	154	155	156	157	158	159	160	188	189	190	191	202	203	205	206	207	431	5	501
101	177	0	0	0	0	0	0	0	0	0	1	0	18	0	0	0	0	0	1	0	0	0	0	0
102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
137	2	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
156	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
157	1	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
158	2	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0
159	6	0	0	0	0	0	0	0	0	0	3	0	45	0	0	0	0	0	0	0	0	0	0	0
160	3	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0
188	9	0	5	0	0	0	0	0	0	0	3	0	41	0	0	0	0	0	0	0	6	0	1	0
189	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
190	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
191	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
202	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
203	7	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	56	4	0	0	0	0	0
205	4	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	7	51	0	0	0	0	0
206	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	2	1	2	0	0	0	0
207	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
431	1	0	0	0	0	0	0	0	0	0	5	0	19	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3	0
501	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28

k- Vizinhos mais próximos (k-NN)

	101	102	137	145	148	154	155	156	157	158	159	160	188	189	190	191	202	203	205	206	207	431	5	501
101	166	0	0	0	0	0	0	0	0	0	3	4	13	0	0	0	0	1	5	0	4	0	0	1
102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
137	2	0	4	0	0	0	0	0	0	0	0	0	1	0	4	0	0	0	0	0	0	0	0	0
145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	19	0	0	0	9	0	0	0	0	5	0	10	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
157	1	0	0	0	21	0	0	0	4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
158	2	0	0	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0
159	7	0	0	0	0	0	0	0	0	0	16	2	23	0	0	0	0	1	1	0	0	4	0	0
160	1	0	0	0	0	0	0	0	0	0	1	2	13	0	0	0	0	1	1	0	0	0	0	0
188	5	0	4	0	0	0	0	0	0	1	11	6	27	0	1	0	1	1	1	0	5	2	0	0
189	0	0	0	0	7	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
190	0	0	5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
191	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
202	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
203	3	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	61	3	0	0	0	1	0
205	1	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	11	50	0	0	0	0	0
206	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	4	0	0	0	0
207	15	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	2	0	0	0
431	2	0	0	1	0	0	0	0	0	0	10	3	9	0	0	0	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	2	0
501	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28

Naive Bayes

	101	102	137	145	148	154	155	156	157	158	159	160	188	189	190	191	202	203	205	206	207	431	5	501
101	130	0	0	0	2	0	9	0	0	0	1	4	0	0	0	0	0	0	1	0	28	15	7	0
102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
137	0	0	6	0	0	0	1	0	0	0	0	0	0	0	3	0	0	0	0	0	1	0	0	0
145	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	0	16	0	0	0	0	8	0	19	0	0	0	0	0	0	0	0
154	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
156	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
157	1	0	0	0	0	0	0	0	24	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
158	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0
159	2	0	0	0	0	0	2	0	0	0	1	5	0	0	0	0	0	0	0	0	1	42	1	0
160	1	0	0	0	0	0	1	0	0	0	1	5	0	0	0	0	0	0	0	0	0	11	0	0
188	4	0	6	0	0	0	0	0	0	0	3	8	0	0	0	0	1	0	0	0	6	36	1	0
189	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	3	0	0	0	0	0	0	0	0
190	0	0	4	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
191	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	11	0	0	0	0	0	0	0	0
202	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
203	2	0	0	0	0	0	2	0	1	0	3	0	0	0	0	0	0	1	46	14	0	0	0	0
205	3	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	49	8	0	2	0	0
206	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	0	1	0	0
207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0
431	0	0	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	21	0	0
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	1	0
501	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28

Resultados obtidos:

	Redes Neuronais	k - NN	Naive Bayes	SVM
Taxa de erro (multi-classe)	0.753	0.417	0.537	**0.381**

A classificação em multi-classe revelou uma diminuição significativa na performance dos classificadores relativamente à classificação binária. As taxas de erro apresentam valores superiores a 30%, enquanto que na classificação binária o valor mais elevado rondava os 10%. Isso acontece porque o problema de classificação multi-classe é mais difícil. O classificador SVM destaca-se com a taxa de erro mais baixa.

É ainda possível estudar a performance dos classificadores relativamente a uma determinada classe, transformando a matriz confusão numa matriz confusão binária. Para isso é necessário escolher uma classe, neste caso foi utilizada a classe p101, e construir uma nova matriz confusão binária.

Como o classificador Máquina de Suporte Vectorial foi o que apresentou melhor taxa de erro na classificação multi-classe, será este o classificador avaliado nesta experiência. A matriz confusão binária para o classificador SVM, tendo como classe positiva a pasta p101, é a seguinte:

SVM	p101 - prevista	p101n - prevista
p101 - verdadeira	177	20
p101n - verdadeira	56	239

Os resultados obtidos foram os seguinte:

Medidas de avaliação	SVM – p101
Taxa de erro	0.154
Sensibilidade	0.885
Precisão	0.76
F1	0.818

Os resultados evidenciam uma boa performance do classificador, sendo que medida F1 apresenta um valor acima de 0.8. No entanto, comparando com os resultados obtidos através da classificação binária para a pasta p101 (F1=0.926), existe uma pequena diminuição da performance do classificador quando aplicado a um problema de multi-classe.

A experiência foi também realizada para a pasta p205. Na tabela seguinte encontram-se os resultados obtidos:

Medidas de avaliação	SVM – p205
Taxa de erro	0.048
Sensibilidade	0.785
Precisão	0.879
F1	0.829

Novamente, os resultados apresentam um valor de F1 superior a 0.8, revelando assim uma boa performance do classificador para a pasta p205. Comparando com os resultados obtidos através da classificação binária (F1=0.979), verifica-se que houve uma diminuição na performance. Tal como aconteceu na classificação binária, foram obtidos melhores resultados para a pasta p205 do que para a p101.

O estudo das alterações de alguns parâmetros foi também aplicado à classificação multi-classe. Os valores dos parâmetros alterados foram os mesmos que aplicados à classificação binária.

Os resultados obtidos para a classificação multi-classe foram os seguintes:

	Redes Neuronais	k-NN
Taxa de erro	0.766 (0.753)	**0.408** (0.417)

Entre parênteses estão os resultados anteriores.

Os resultados revelam uma ligeira melhoria na taxa de erro de ambos os classificadores, no entanto, estamos perante uma alteração pouco significativa, por isso não é possível fazer conclusões sobre a modificação dos parâmetros.

6. Conclusões

O trabalho realizado teve como objectivo verificar se a classificação multi-classe poderia ser utilizada para a classificação de emails e, caso fosse possível, encontrar o melhor classificador para estes dados. Os resultados obtidos revelaram uma boa performance dos classificadores para classificação binária, no entanto, a performance reduziu bastante na classificação multi-classe, tendo apenas o algorítmo SVM demonstrado resultados significativos. Esta situação seria de esperar devido à heterogeneidade das pastas. No 1º caso, os dados estavam não-equilibrados, por isso, a probabilidade de classificar correctamente uma nova mensagem era elevada. No caso da classificação multi-classe, haviam várias pastas e todas com número de mensagens diferentes. Por exemplo, nas pastas que só continham uma mensagem, quando foi feita a divisão dos dados para treino e teste, caso uma dessas mensagens calhasse no conjunto de teste, a pasta onde essa mensagem pertence não teria como ensinar ao classificador que mensagens lá inserir. Assim, seria de esperar que essas mensagens fossem sempre mal classificadas.

Em conclusão, os objectivos previstos foram atingidos podendo assumir-se que a classificação multi-classe pode ser aplicada a dados de email e que o melhor classificador individual para classificar as mensagens em várias pastas seria o SVM.

6.1. Trabalhos futuros

Considerando que a classificação automática de emails surge, hoje em dia, como uma necessidade geral, esta é, portanto, uma área a ter em conta para desenvolvimento futuro. Com os resultados do trabalho surgiu a ideia de, futuramente, ser feita uma classificação multi-classe utilizando classificação binária combinada. A ideia assenta em utilizar a classificação binária para cada pasta, ou seja, como foi feito nas primeiras experiências, verificava-se se uma determinada mensagem pertencia à pasta X, caso não pertencesse ia para a pasta Xn. E repetia-se o processo até a mensagem ser atribuída a uma determinada pasta. No caso de ser atribuída a mais do que uma pasta, seria

necessário utilizar, por exemplo, probabilidades para decidir em que pasta essa mensagem seria inserida.

Outros factores que poderiam ser explorados seriam os parâmetros dos classificadores a utilizar. Tentar encontrar a melhor combinação de parâmetros de forma a obter uma melhor performance dos classificadores.

Relativamente aos dados, seria uma experiência interessante aplicar o que foi feito neste trabalho a um conjunto de dados diferente.

7. Referências

Bekkerman, R., A. McCallum, et al. (2004). Automatic Categorization of Email into Folders: Benchmark Experiments on Enron and SRI Corpora. *Computer Science Department Faculty Publication Series.* University of Massachusetts, Amherst, USA.

Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the eleventh annual conference on Computational learning theory*. Madison, Wisconsin, USA, ACM**:** 92-100.

Brazdil, Pavel - *Classification of Document susing Text Mining Package "tm".* [Projecção visual]. [2011]. 43 diapositivos.

Brutlag, J. D. and C. Meek (2000). Challenges of the email domain for text classification. *Machine Learming - International Workshop then Conference.*

Chakravarthy, S., A. Venkatachalam, et al. (2010). A Graph-Based Approach for Multi-folder Email Classification. 2010 *IEEE 10th International Conference on Data Mining (ICDM).*

Cohen, W. W. (1996). Learning Rules that classify E-mail. *AAAI Spring Symposium in Information Access.*

Crawford, E., I. Koprinska, et al. (2004). Phrases and Feature Selection in E-Mail Classification. *Proceedings of the 9th Australasian Document Computing Symposium.* Melbourne, Australia.

Gama, J., A. Carvalho, M. Oliveira, K. Faceli, A. Lorena. (2012). *Extração de Conhecimento de Dados - Data Mining.* Edições Silabo.

Gomez, J. and M.-F. Moens (2010). Using Biased Discriminant Analysis for Email Filtering. *Knowledge-Based and Intelligent Information and Engineering Systems*. R. Setchi, I. Jordanov, R. Howlett and L. Jain, Springer Berlin Heidelberg. **6276:** 566-575.

Hotho, A., A. Nürnberger, et al. (2005). "A brief survey of text mining." *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology.*

I. Feinerer. (2010, Jan.) Introduction to the *tm* Package – Text Mining in R. http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

I. Feinerer, K.Hornik, and D. Mayer (2008), "Text mining infrastructure in R," *Journal of Statistical Software*, vol. 25, pp. 1-54.

Katakis, I., G. Tsoumakas, et al. (2006). Email Mining: Emerging Techniques for Email Management Department of Informatics, Aristotle University of Thessaloniki, Greece.

Keerthi, S. S. and C. J. Lin (2003). "Asymptotic behaviors of support vector machines with Gaussian kernel." *Neural Comput* **15**(7): 1667-1689.

Kiritchenko, S. and S. Matwin (2001). Email classification with co-training. *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research.* Toronto, Ontario, Canada, IBM Press**:** 8.

Kiritchenko, S., S. Matwin, et al. (2004). "Email classification with temporal features." *Proceedings of the International Intelligent Information Systems* 523-533.

M. Moens (2008), "Information extraction: algorithms and prospects in a retrieval context", *Computational Linguistics*, vol. 34, pp. 315-317.

Patel, F. N. and N. R. Soni (2012). "Text Mining: A brief survey." *International Journal of Advanced Computer Research* **2**(6): 243-248.

Provost, J. (1999). Naive-Bayes vs. Rule-Learning in Classification of Email. University of Texas at Austin.

Sahami, M., S. Dumais, et al. (1998). A Bayesian Approach to Filtering Junk E-Mail. Learning for Text Categorization: *Papers from the 1998 Workshop, AAAI Technical Report WS-98-05*.

Segal, R. B. and J. O. Kephart (1999). MailCat: an intelligent assistant for organizing e-mail. *Proceedings of the third annual conference on Autonomous Agents*. Seattle, Washington, USA, ACM**:** 276-282.

Shen, J., L. Li, et al. (2006). A hybrid learning system for recognizing user tasks from desktop activities and email messages. *Proceedings of the 11th international conference on Intelligent user interfaces*. Sydney, Australia, ACM**:** 86-92.

Xiao-Lin, W. and I. Cloete (2005). Learning to classify email: a survey. Machine Learning and Cybernetics, 2005. *Proceedings of 2005 International Conference on*.

Youn, S. and D. McLeod (2007). A Comparative Study for Email Classification. *Advances and Innovations in Systems, Computing Sciences and Software Engineering*. K. Elleithy. University of Southern California, Los Angeles, CA, USA, Springer Netherlands**:** 387-391.

8. Anexos

8.1. Package de *Text Mining* instalado

```
library(tm)
```

8.2. Código de leitura do ficheiro excel

```
FichBase<-read.csv("MFB.csv",sep=";")
```

8.3. Código para classificação binária

8.3.1. Funções para criar listas em R e no computador

```
#Criação de listas em R


mudar.listas <- function (nomespastas, pasta) {
       lista1 <- list()
       lista2 <- list()
linha <- 1
poslista1<-1
poslista2<-1

#
for (i in linha : nrow(FichBase)) {

if (as.data.frame(nomespastas[linha])[2,]==pasta)
{
lista1[[poslista1]]<-FichBase[linha,]
 poslista1 <- poslista1 + 1}
 else {
 lista2[[poslista2]]<-FichBase[linha,]
 poslista2<-poslista2 + 1}
linha<-linha + 1
}

#
listas <- list()
listas[[1]]<-lista1
listas[[2]]<-lista2
return(listas) }


       lista.res<-mudar.listas(nomespastas, "101")


       lista101<-lista.res[[1]]
       lista101n<-lista.res[[2]]



#Criação de pastas no computador

       dat.listas <- function (lista1,lista2) {
       lista.dat1 <- list()
       poslista1<-1

       for (i in (1:length(lista1))) {
       nome.fich1 <- lista101[[poslista1]][1,"CodAnexo"]
```

```
poslista1 <- poslista1 + 1
nome.fich11 <- paste(nome.fich1,"dat",sep=".")
nome.pasta.ant <- paste("DAT files/", nome.fich11, sep="")
nome.pasta.nova <- paste("Datpos/", nome.fich11, sep="")
if (file.exists(nome.fich11)) {
lista.dat1[[poslista1]] <- file.copy(nome.pasta.ant, nome.pasta.nova)
}
      }

      lista.dat2 <- list()
      poslista2<-1
for (i in (1:length(lista2))) {
nome.fich2 <- lista101n[[poslista2]][1,"CodAnexo"]
poslista2 <- poslista2 + 1
nome.fich22 <- paste(nome.fich2,"dat",sep=".")
nome.pasta.ant <- paste("DAT files/", nome.fich22, sep="")
nome.pasta.nova <- paste("Datneg/", nome.fich22, sep="")
if (file.exists(nome.fich22)) {
lista.dat2 [[poslista2]] <- file.copy(nome.pasta.ant, nome.pasta.nova)
}
}

listas <- list()
listas[[1]] <- lista.dat1
listas[[2]] <- lista.dat2
return(listas)
}
```

8.3.2. Função para ler os ficheiros .dat

```
datpos <- Corpus(DirSource("Datpos",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))

datneg <- Corpus(DirSource("Datneg",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))

docs <- c(datpos, datneg)
```

8.3.3. Funções de pré-processamento

```
docs.p <- docs
docs.p <- tm_map(docs.p, removeWords,
c("div","br","nbsp",stopwords('portuguese'))) #remove stopwords e html
docs.p <- tm_map(docs.p, stripWhitespace) #remove espaços em branco a mais
docs.p <- tm_map(docs.p, tolower) #transforma letras maiúsculas em minúsculas
docs.p <- tm_map(docs.p, removePunctuation) #remove a pontuação
docs.p <- tm_map(docs.p, removeNumbers) #remove números
```

Código de remoção de HTML – não aplicado devido a erro:

```
library(RCurl)
library(XML)

# assign input (could be a html file, a URL, html text, or some
combination of all three is the form of a vector)

input <-

# evaluate input and convert to text
txt <- htmlToText(input)
txt
```

8.3.4. Funções de processamento

Criação de matriz documento-termo:

```
DocumentTermMatrix(docs.p)
dtm.mx <- DocumentTermMatrix( docs.p, control=list(minWordLength=3,
minDocFreq=2))
```

Remoção de termos de baixa frequência:

```
dtm.mx.aux <- removeSparseTerms( dtm.mx, 0.95) #elimina termos que aparecem
poucas vezes

dtm <- as.data.frame(inspect( dtm.mx.aux )) #transformar em dataframe
```

Identificação de termos informativos:

```
info.terms <- vector()


find.info.terms <- function(dtm.tr,min.info) {
                 ix.class <- ncol(dtm.tr)
                 default.info<-info(table(dtm.tr[,ix.class]))
cat("default.info: ", default.info, "\n")
n.atr <- ncol(dtm.tr)-1
n.info.terms <- 0
info.term.ixs <- vector()
col.names <- names(dtm.tr)

for (atri in 1:n.atr) {
if (sum(dtm.tr[,atri])>0)
{ # begin if
no.dif.atr.val<-length(table(dtm.tr[,atri]))
atr.class.table<-table(dtm.tr[,atri],dtm.tr[,ix.class])
n.rows<-nrow(dtm.tr)
atr.info <- 0
for (atr.val in 1: no.dif.atr.val)
{ # begin for
atr.peso <- sum( atr.class.table[atr.val,]) / n.rows
atr.info1 <- atr.peso * info(atr.class.table[atr.val,])
atr.info<-atr.info + atr.info1 }
info.gain <- default.info - atr.info
if (info.gain > min.info)
{ info.term.ixs[n.info.terms] <- atri
n.info.terms <- n.info.terms+1 }
} #end for
} # end if
cat("\n", "Vão ser mantidos ", n.info.terms, " atributos: ", "\n")
cat( col.names[info.term.ixs[1:10]], " etc. ")
cat( col.names[info.term.ixs[n.info.terms-1]],"\n")
cat("Vão ser eliminados ", n.atr-n.info.terms, " atributos", "\n")
return(col.names[info.term.ixs])
}

info <- function(x){
inf <- 0
sumx <- sum(x)
for (i in x) {
pi <- i/sumx
infi <- (pi)*log2(pi)
if (is.na(infi)) infi <- 0
inf <- inf - infi }
return(inf)
}
info.terms <- find.info.terms(dtm,0.005)
```

8.3.5. Funções de classificação

Criação da coluna com a classe:

```
l1 <- length(datpos)
l2 <- length(datneg)
class <- c(rep("pos",l1),rep("neg",l2))
dtm <- cbind(dtm, class)
```

Separação de dados de treino e de teste:

```
permute.index <- sample(1:nrow(dtm), as.integer(0.7*nrow(dtm)))
dtm.tr <- dtm[ permute.index, ] #treino
dtm.ts <- dtm[ -permute.index, ] #teste
```

Classificadores e medidas de avaliação de performance:

Redes Neuronais

```
library(nnet)
nnet.classifier <- nnet(class~., data=dtm.tr, size=2, rang=0.1, decay=5e-4,
maxit=100)
preds.nn <- predict(nnet.classifier, dtm.ts, type="class")
conf.mx.nn <- table(dtm.ts$class, preds.nn) #dtm.ts$class = clas.formula
error.rate.nn <- (sum(conf.mx.nn) - sum(diag(conf.mx.nn))) / sum(conf.mx.nn)

recall <- conf.mx.nn[1]/(conf.mx.nn[1]+conf.mx.nn[3])
precisao <- conf.mx.nn[1]/(conf.mx.nn[1]+conf.mx.nn[2])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

k-NN

```
library(class)
preds.knn <- knn(dtm.tr[, info.terms], dtm.ts[, info.terms], dtm.tr$class,
k=1)
conf.mx.knn <- table(dtm.ts$class, preds.knn)
error.rate.knn <- (sum(conf.mx.knn) - sum(diag(conf.mx.knn))) /
sum(conf.mx.knn)

recall <- conf.mx.knn[1]/(conf.mx.knn[1]+conf.mx.knn[3])
precisao <- conf.mx.knn[1]/(conf.mx.knn[1]+conf.mx.knn[2])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

SVM

```
library(e1071)
svm.classifier <- svm(class~., dtm.tr)
preds.svm <- predict(svm.classifier, dtm.ts)
conf.mx.svm <- table(dtm.ts$class, preds.svm)
error.rate.svm <- (sum(conf.mx.svm) - sum(diag(conf.mx.svm))) /
sum(conf.mx.svm)

recall <- conf.mx.svm[1]/(conf.mx.svm[1]+conf.mx.svm[3])
precisao <- conf.mx.svm[1]/(conf.mx.svm[1]+conf.mx.svm[2])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

Naive Bayes

```
library(RWeka)
NB<-make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
nb.classifier<-NB(class~., dtm.tr)
preds.nb<-predict(nb.classifier, dtm.ts)
conf.mx.nb<-table(dtm.ts$class, preds.nb)
error.rate.nb <- (sum(conf.mx.nb) - sum(diag(conf.mx.nb))) / sum(conf.mx.nb)

recall <- conf.mx.nb[1]/(conf.mx.nb[1]+conf.mx.nb[3])
precisao <- conf.mx.nb[1]/(conf.mx.nb[1]+conf.mx.nb[2])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

8.4. Código para classificação multi-classe

8.4.1. Função para ler os ficheiros .dat

```
p145 <- Corpus(DirSource("145",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p154 <- Corpus(DirSource("154",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p155 <- Corpus(DirSource("155",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p202 <- Corpus(DirSource("202",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p102 <- Corpus(DirSource("102",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p156 <- Corpus(DirSource("156",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p206 <- Corpus(DirSource("206",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p158 <- Corpus(DirSource("158",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p190 <- Corpus(DirSource("190",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p137 <- Corpus(DirSource("137",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p189 <- Corpus(DirSource("189",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p191 <- Corpus(DirSource("191",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p207 <- Corpus(DirSource("207",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p160 <- Corpus(DirSource("160",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p431 <- Corpus(DirSource("431",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p157 <- Corpus(DirSource("157",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p501 <- Corpus(DirSource("501",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p159 <- Corpus(DirSource("159",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p148 <- Corpus(DirSource("148",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p188 <- Corpus(DirSource("188",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p203 <- Corpus(DirSource("203",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p205 <- Corpus(DirSource("205",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
```

```
p101 <- Corpus(DirSource("101",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))
p5 <- Corpus(DirSource("5",encoding="Latin1"),
readControl=list(reader=readPlain, language="portuguese"))

docs <- c(p145, p154, p155, p202, p102, p156, p206, p158, p190, p137, p189,
p191, p207, p160, p431, p157, p501, p159, p148, p188, p203, p205, p101, p5)
```

As funções de pré-processamento e processamento são iguais às de classificação binária.

8.4.2. Funções de classificação

Criação da coluna com a classe:

```
l1 <- length(p145)
l2 <- length(p154)
l3 <- length(p155)
l4 <- length(p202)
l5 <- length(p102)
l6 <- length(p156)
l7 <- length(p206)
l8 <- length(p158)
l9 <- length(p190)
l10 <- length(p137)
l11 <- length(p189)
l12 <- length(p191)
l13 <- length(p207)
l14 <- length(p160)
l15 <- length(p431)
l16 <- length(p157)
l17 <- length(p501)
l18 <- length(p159)
l19 <- length(p148)
l20 <- length(p188)
l21 <- length(p203)
l22 <- length(p205)
l23 <- length(p101)
l24 <- length(p5)
class <-
c(rep("145",l1),rep("154",l2),rep("155",l3),rep("202",l4),rep("102",l5),rep("1
56",l6),rep("206",l7),rep("158",l8),rep("190",l9),rep("137",l10),rep("189",l11
),rep("191",l12),rep("207",l13),rep("160",l14),rep("431",l15),rep("157",l16),r
ep("501",l17),rep("159",l18),rep("148",l19),rep("188",l20),rep("203",l21),rep(
"205",l22),rep("101",l23),rep("5",l24)) #criar a coluna com a classe
dtm <- cbind(dtm, class)
```

Separação de dados de treino e de teste:

```
permute.index <- sample(1:nrow(dtm), as.integer(0.7*nrow(dtm)))
dtm.tr <- dtm[ permute.index, ] #treino
dtm.ts <- dtm[ -permute.index, ] #teste
```

Classificadores e medidas de avaliação de performance:

SVM

```
library(e1071)
svm.classifier <- svm(class~., dtm.tr)
preds.svm <- predict(svm.classifier, dtm.ts)
conf.mx.svm <- table(dtm.ts$class, preds.svm)
error.rate.svm <- (sum(conf.mx.svm) - sum(diag(conf.mx.svm))) /
sum(conf.mx.svm)

Classificação binária p101
recall <- conf.mx.svm[1]/(conf.mx.svm[1]+(sum(conf.mx.svm)-
sum(diag(conf.mx.svm))-sum(conf.mx.svm[,1])))
precisao <- conf.mx.svm[1]/sum(conf.mx.svm[,1])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

k-NN

```
library(class)
preds.knn <- knn(dtm.tr[, info.terms], dtm.ts[, info.terms], dtm.tr$class,
k=1)
conf.mx.knn <- table(dtm.ts$class, preds.knn)
error.rate.knn <- (sum(conf.mx.knn) - sum(diag(conf.mx.knn))) /
sum(conf.mx.knn)

Classificação binária p101
recall <- conf.mx.knn[1]/(conf.mx.knn[1]+(sum(conf.mx.knn)-
sum(diag(conf.mx.knn))-sum(conf.mx.knn[,1])))
precisao <- conf.mx.knn[1]/sum(conf.mx.knn[,1])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

Naive Bayes

```
library(RWeka)
NB<-make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
nb.classifier<-NB(class~., dtm.tr)
preds.nb<-predict(nb.classifier, dtm.ts)
conf.mx.nb<-table(dtm.ts$class, preds.nb)
error.rate.nb <- (sum(conf.mx.nb) - sum(diag(conf.mx.nb))) / sum(conf.mx.nb)

Classificação binária p101
recall <- conf.mx.nb[1]/(conf.mx.nb[1]+(sum(conf.mx.nb)-sum(diag(conf.mx.nb))-
sum(conf.mx.nb[,1])))
precisao <- conf.mx.nb[1]/sum(conf.mx.nb[,1])
f1 <- 2*(precisao*recall)/(precisao+recall)
```

Redes Neuronais

```
library(nnet)
nnet.classifier <- nnet(class~., data=dtm.tr, size=3, rang=0.1, decay=5e-4,
maxit=200)
preds.nn <- predict(nnet.classifier, dtm.ts, type="class")
conf.mx.nn <- table(dtm.ts$class, preds.nn) #dtm.ts$class = clas.formula
error.rate.nn <- (sum(conf.mx.nn) - sum(diag(conf.mx.nn))) / sum(conf.mx.nn)

Classificação binária p101
recall <- conf.mx.nn[1]/(conf.mx.nn[1]+(sum(conf.mx.nn)-sum(diag(conf.mx.nn))-
sum(conf.mx.nn[,1])))
precisao <- conf.mx.nn[1]/sum(conf.mx.nn[,1])
f1 <- 2*(precisao*recall)/(precisao+recall)
```