Relatório

# Classes escolhidas

A escolha final das classes, para este trabalho, foi realizada somente após diversos testes consecutivos, alguns deles após a conclusão dos algoritmos que compõem o programa. Durante esses testes, foram utilizadas várias combinações de classes e, como a implementação do programa garantia a flexibilidade da escolha do número de classes, alguns testes foram efetuadas sobre todas as classes existentes no pacote de mensagens, o que nos permitiu avaliar o nível de erro durante a classificação de todas as classes. Foi interessante notar como a tarefa de classificação de classes muito próximas, ou seja, que contém palavras-chave familiares entre si, gerou uma alta taxa de erros, fazendo com que os testes envolvendo todas as classes gerassem um erro compreendido entre vinte e trinta por cento.

Foram escolhidas, finalmente, as classes “alt.atheism”, “sci.crypt” e “soc.religion.christian”, por apresentarem uma pequena proximidade entre os assuntos da primeira e da terceira classes e uma total (ou quase total) disparidade entre estes e o assunto da segunda classe. As classes escolhidas contêm, respectivamente, 799, 991 e 997 mensagens (arquivos).

Para a criação do vocabulário, segundo orientações da professora, em sala de aula, primeiramente foi realizada uma etapa de preparação dos arquivos, onde cada arquivo era limpo segundo os seguintes passos:

1. Remover cabeçalho (e, possivelmente, rodapé): ignorar linhas que contenham as seguintes palavras-chave: “subject:”, “expires:”, “distribution:”, “lines:”, “from:”, “article-i.d.:”, “in article”, “reply-to:”, “summary:”, “supersedes:”, “archive-name:”, “keywords:”, “last-modified:”, “version:”, “write to:”, “telephone:”, “fax:”, “organization:”, “nntp-posting-host:” e “organization:”.
2. Remover palavras inválidas: remover palavras com menos de três ou mais de 44 caracteres (remove palavras muito grandes, possivelmente inválidas, ou muito pequenas, que não são relevantes para o vocabulário), remover palavras com caracteres repetidos por mais de duas vezes, remover palavras que contenham caracteres inválidos (diferentes de “a..z”). É interessante ressaltar que, neste caso, as palavras foram primeiramente limpas de pontuações, a fim de que *tokens* acompanhados por pontuação (e.g. “God.”) não fossem ignorados. A diferença entre fazer ou não fazer essa limpeza faz crescer aproximadamente três pontos percentuais o erro global da classificação.

Ao final da limpeza de cada arquivo, o conteúdo é salvo em um arquivo com o mesmo nome do arquivo original, em uma pasta “output”, criada na pasta de cada classe. A criação do vocabulário é então iniciada, iterando-se palavra a palavra do conteúdo limpo concatenado de todos os arquivos. O vocabulário foi implementado com o uso de um *HashMap*, de forma que se pudesse contar com a agilidade do acesso direto às posições de palavras e, ao mesmo tempo, com a contagem de cada *token* no conteúdo concatenado.

Após a criação do vocabulário, mais uma etapa de limpeza é realizada, desta vez sobre o próprio vocabulário, segundo os seguintes passos:

1. Remover *tokens* que aparecem menos de três vezes no texto concatenado.
2. Remover os cem *tokens* mais freqüentes: ajuda na precisão da classificação, uma vez que retira palavras que aparecem muito mas que, ao mesmo tempo, dizem pouco sobre a classe onde aparece (e.g. “*the*”, “*and*”, etc).

# Discussão sobre os experimentos realizados

Os primeiros testes realizados foram executados sobre várias combinações de classes, como dito anteriormente. A combinação de três classes próximas, como “alt.atheism”, “soc.religion.christian” e “talk.religion.misc” produz um erro de 27%. O erro global, que hoje está em torno de 10%, para as classes escolhidas (“alt.atheism”, “sci.crypt” e “soc.religion.christian”), foi atingido somente após diversos testes sobre as etapas de treinamento e classificação.

O principal problema que incidia sobre o resultado da classificação era a presença de palavras (tokens) que acabavam poluindo o vocabulário, sem ter muita relevância. Algo que fez muito efeito sobre o desempenho do programa foi a remoção das linhas de cabeçalho, que são comuns à maioria das mensagens. Utilizando um algoritmo para verificar se determinada linha é cabeçalho, o erro do algoritmo diminuiu sensivelmente. Verificações posteriores mostraram que havia uma presença recorrente da palavra “article” nos textos concatenados. A adição da verificação do termo “in article” a esse algoritmo melhorou um pouco o resultado da classificação.

Digno de destaque é o impacto da ausência da limpeza do vocabulário: o erro global salta de 10% para mais de 65%.

# Classificando com Holdout

O desempenho do classificador, após a execução do método Holdout, se mostrou bastante satisfatório, gerando um erro que esteve entre 9,3% e 10,84%, após diversos testes. A matriz de confusão detalha o erro do classificador como segue:

Class: soc.religion.christian. Hits: 299

----> Classified as sci.crypt: 16

----> Classified as alt.atheism: 17

..... Class accuracy: 33/(299+33) = 0.9006024096385542

Class: sci.crypt. Hits: 326

----> Classified as soc.religion.christian: 2

----> Classified as alt.atheism: 2

..... Class accuracy: 4/(326+4) = 0.9878787878787879

Class: alt.atheism. Hits: 220

----> Classified as soc.religion.christian: 33

----> Classified as sci.crypt: 13

..... Class accuracy: 46/(220+46) = 0.8270676691729324

Total accuracy: 0.9105603448275862

É possível observar que a acurácia do classificador é maior para as duas primeiras classes, mas um pouco menor para a última, que tem maior número de classificações errôneas como “soc.religion.christian”, por ser mais próxima da primeira.

# Classificando com Cross-Validation

A classificação com Cross-Validation foi efetuada sobre através da sucessiva iteração sobre os *folds* de treino e teste, onde o array de arquivos de origem foi embaralhado aleatoriamente e segmentado em 10 *folds*, onde uma iteração de dez passos selecionou o k-ésimo *fold* da iteração como *fold* de teste e os demais como *folds* de treino.

O erro médio produzido nas dez iterações (lembrando que este número vai variar minimamente em outras execuções, dependendo da forma como os arquivos forem escolhidos) foi de 10,55%, com um desvio-padrão de apenas 0,02%, o que demonstra uma classificação com erro quase constante sobre todas as iterações. É possível observar, deste modo, que a estimativa de erro global do programa, avaliado através deste método, está próxima dos 10%, ou seja, sua acurácia, para as classes escolhidas, gira em torno de 90%, o que é um número bastante alto.

# Conclusões

O emprego dos métodos *Holdout* e *Cross-Validation* permitiu verificar, com grande exatidão, que o desempenho do modelo, para as classes escolhidas, é “realmente” próximo de apenas 10%. Os mesmos testes, posteriormente realizados sobre **todas** as classes existentes, demonstrou a mesma constância, embora o erro global tenha subido para 27,5% o que, apesar de não ser muito pequeno, ainda nos dá 72,5% de acertos na classificação de novos exemplos, algo satisfatório na maioria dos casos. É preciso lembrar que, englobado neste erro, estão as altas taxas de acerto de algumas classes bastante estratificadas e os erros um pouco maiores das classes que se aproximam mais ou menos por assunto. Acreditamos que o erro de 27,5% para uma classificação sobre todas as classes é bastante aceitável, pois, para melhorar esse erro, seria necessário fazer com que o classificador conseguisse distinguir bem mesmo as classes com assuntos próximos, o que poderia viciá-lo.

O desenvolvimento deste programa nos deu um entendimento muito melhor sobre os algoritmos de treinamento e de classificação do Naive Bayes para classificação textual, além de nos permitir compreender, com clareza, o propósito do emprego dos métodos de avaliação *Holdout* e *Cross-Validation*, que é simplesmente avaliar a eficácia do modelo desenvolvido.