# Mineração de Dados
# Trabalho Prático 3

Prof. Wagner Meira Jr.

Data de entrega: $1^o$ de fevereiro de 2013

---

## 1 Descrição do Problema

Você chegou ao auge da sua carreira, e até Hollywood já ouviu da sua fama. Sabendo disso, o consagrado site de filmes, o *IMDb*, resolveu te contratar para uma importante tarefa.

O *IMDb* é a maior base de dados de filmes, seriados, programas de televisão e semelhantes existente na internet. Sua reputação vem do fato de ela possuir não apenas dados técnicos sobre os filmes, mas também de os usuários poderem postar comentários e dar notas a cada filme da base, auxiliando assim outras pessoas a decidirem se vale ou não a pena assistir o filme em questão.

Sua base contém milhares de *reviews*, e é atualizada constantemente. Entretanto, como os *reviews* deixados pelas pessoas são textos, ela precisa saber se eles são positivos ou negativos em relação ao filme. Isso é um grande problema, dada a enorme quantidade de dados, e logo não pode ser feito manualmente. Mas classificar os *reviews* é extremamente importante para sistemas de recomendação de filmes, e não há informação mais valiosa para eles que extrair diretamente o que os próprios usuários estão comentando sobre os filmes que assistiram.

Para resolver esse problema, o *IMDb* forneceu a você uma amostra da base de dados deles contendo mais de mil *reviews* diferentes, que já foram classificados como positivos ou negativos, para servir como treino. Ele entregou ainda uma outra amostra, que é de *reviews* ainda não classificados. Cada *review* é formado por um texto (conjunto de palavras) que expressam o sentimento do usuário em relação a um filme em questão.

Para isso, sua tarefa é classificar os novos *reviews* que chegam no sitema do *IMDb* diariamente.

## 2 Avaliação

Você deve implementar o algoritmo **KNN** (*k-Nearest Neighbors*) de forma a classificar as entradas do arquivo de teste.

Você deve ainda resolver uma forma de escolher o melhor $k$ para a base. Dica: cuidado com *overfitting*!

Compare ainda os resultados gerados com e sem a utilização de stopwords.

Analise a qualidade das soluções geradas usando precisão e revocação, e qualquer outra métrica de acurácia que você julgar importante/necessária.

## 3 Formato

Os arquivos necessários estão disponíveis em [1]. A pasta contém o arquivo de treino, o arquivo de teste e um arquivo contendo uma lista de stopwords.

### 3.1 Formato de Entrada

O programa receberá dois arquivos de entrada: um para treino, outro para teste. Entretanto, o formato de ambos os arquivos será o mesmo. Cada linha tanto do arquivo de treino como do de teste consiste em uma instância, na forma:

```
<Classe> <uma ou mais palavras separadas por espaço>
```

Um exemplo de treino seria:

```
1 filme muito bom
1 excelente atuação dos atores
0 nunca vi filme pior em toda minha vida
0 lamentável , muito ruim mesmo
1 brilhante !
```

E um de teste:

```
0 filme muito ruim
1 o melhor filme da minha vida !
```

Note que apesar de o arquivo de teste também conter as classes reais de cada instância, essa informação não deve ser levada em conta na hora da classificação, mas apenas para a avaliação da mesma.

O programa deve também receber como entrada o número $k$ de vizinhos mais próximos a serem analizados a fim de gerar a comparação. O formato de execução deve seguir o formato:

```
$> comando -i <arquivo treino> -t <arquivo teste> -k <número de vizinhos>
```

Você pode incluir outros parâmetros, caso sinta necessidade. Entretanto, eles devem ser opcionais, logo, escolha um valor *default* para que, apenas o com o comando acima, seu programa seja executado corretamente. Não se esqueça de especificar os novos parâmetros no arquivo README.txt.

### 3.2 Formato de Saída

Para cada instância do teste, imprimir a qual classe ela foi atribuída pelo classificador, uma por linha. Para os exemplos de entrada acima, usando $k = 3$, a saída deveria ser:

```
0
1
```

## 4 O que entregar

Você deverá entregar o código com a implementação, juntamente com um arquivo README.txt contendo, resumidamente, os comandos necessários para a execução do seu programa. O comando contido neste arquivo será o utilizado durante a correção e, se ele falhar, serão desconsiderados os pontos da parte de implementação. Logo, **confira bem** se os comandos estão certos.

Você poderá implementar o código na linguagem de sua escolha, mas **não** pode utilizar nenhuma biblioteca nem ferramenta de mineração de dados para tal.

As implementações devem ser testadas em uma das máquinas de graduação do DCC de livre acesso via acesso remoto. Alguns exemplos de máquinas:

```
cipo.grad.dcc.ufmg.br
claro.grad.dcc.ufmg.br
```

Você deve ainda submeter a documentação do trabalho, em formato *pdf*, com no máximo 10 páginas. A documentação deve abordar, pelo menos, os seguintes pontos:

- Uma breve descrição do algoritmo implementado.
- A ordem de complexidade dele.
- Uma análise de como o algoritmo se comporta quando variamos os parâmetros e o tamanho da entrada.
- Discussão sobre a forma de escolha do $k$.
- Análise da base de dados fornecida.
- Discussão sobre a qualidade da solução, com e sem o uso de stopwords.
- A máquina na qual seu tp foi testado.

Crie uma pasta no formato {*seu login*}*_tp3.tar.gz*, contendo apenas o código fonte, o arquivo README.txt e o *pdf* da documentação. Não inclua nenhum executável. Submeta o *.tar.gz* no moodle. Não é necessário entregar a documentação impressa.

# 5 Referências


1. `www.dcc.ufmg.br/~sara/mineracao/tp3_data.tar.gz`