Mineração de Dados
Trabalho Prático 2

Prof. Wagner Meira Jr.

Data de entrega: 16 de novembro de 2012

---

1 Descrição do Problema

Seu desempenho trabalhando para o governo brasileiro foi tão surpreendente e satisfatório que sua habilidade em análises envolvendo Mineração de Dados possui agora uma repercussão internacional. Sabendo disso, uma relevante comunidade de música *online*, o *Last.fm*, resolveu contratar seus serviços para uma importante tarefa.

O *Last.fm* possui em seu sistema milhões de músicas distintas e um grande problema que eles precisam tratar é o de agrupar semanticamente tais músicas. Uma vez que os usuários podem fazer *upload* de músicas próprias no sistema, a diversidade de músicas existentes é gigantesca. Dessa forma, não há como especialistas definirem para cada música gêneros tradicionais tais como sertanejo, pop, rock, dentre outros. Por outro lado, definir grupos de itens semelhantes é de extrema importância para o sistema de recomendação do *Last.fm*, bem como para os usuários, uma vez que eles podem navegar por itens pertencentes a um mesmo gênero.

De forma a tentar resolver este problema, o *Last.fm* forneceu a você uma amostra da base deles contendo 5.000 músicas distintas. Cada música deste conjunto possui um conjunto de tags que representam as tags mais frequentemente atribuídas a tal música pelos usuários do sistema.

Para isso, sua tarefa é agrupar as novas músicas que chegam no sitema do *Last.fm* diariamente, a partir das tags assinaladas a elas.

2 Avaliação

Você deve implementar o algoritmo **K-means** de forma a agrupar semanticamente as músicas contidas na base de dados.

Faça uma análise de como o resultado varia de acordo com a escolha dos centróides iniciais, e apresente uma forma de escolhê-los.

Proponha uma estratégia de avaliação da qualidade dos grupos encontrados, e analise os clusters gerados usando a técnica proposta.

3 Formato

3.1 Formato de Entrada

O programa receberá um arquivo de entrada, em que cada linha desse arquivo corresponde à uma instância, na forma:

```
<ID> <uma ou mais tags separadas por espaço>
```

Um exemplo de arquivo de entrada seria:

```
0 alternative hard american idol rock
1 ska jazz punk oi rock
2 vocalists freestyle pop female rock electronic math
3 trance electronic
4 techno electronic finnish minimal idm
5 swedish trance techno electronic soul
6 spanish punk asturies rock astur folk
```

O programa deve ainda receber, *opcionalmente*, um arquivo de entrada contendo os centróides iniciais. O arquivo deve ter k linhas, e em cada linha, o ID do elemento corresponte à posição inicial do centróide de cada cluster. Os clusters devem ser numerados de 0 a $k-1$, de acordo com a ordem que aparecem neste arquivo.

```
<ID do elemento que corresponderá ao centróide do Cluster 0>
:
<ID do elemento que corresponderá ao centróide do Cluster k-1>
```

Um exemplo do arquivo descrito seria:

```
2
5
```

Observe que embora esse arquivo não seja necessário para a execução *todas* as vezes - ou seja, seu programa tem que ser capaz de propor centróides iniciais mesmo se esse arquivo não for dado - é imprescindível a implementação desta opção no seu programa.

O programa deve também receber como entrada o número k de clusters a serem gerados. A execução deve seguir o formato:

```
$> comando -i <arquivo entrada> -k <número de clusters> [ -c <arquivo centroides> ]
```

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 o cluster ao que ela pertence, uma por linha.

```
<ID> <Cluster>
:
<ID> <Cluster>
```

Observe que ainda que não utilizemos a opção do arquivo de centróides iniciais, os clusters devem ser numerados de 0 a $k-1$.

Para os exemplos de entrada acima, usando $k = 2$, a saída deveria ser:

```
0 0
1 0
2 0
3 1
4 1
5 1
6 0
```

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 dos centróides iniciais.
- Proposta e implementação da avaliação de qualidade dos clusters.
- Análise da base de dados fornecida.
- Discussão sobre a qualidade da solução.
- A máquina na qual seu tp foi testado.

Crie uma pasta no formato {*seu login*}*_tp2.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/tp2_data.tar.gz`