Programação II - DI/UFES

Trabalho Prático II - NLProg

Informações Gerais

Professor: Filipe Mutz

Professor.: Vinícius Mota

Data de entrega: 02/02/2023

Grupos: Os trabalhos podem ser realizados em dupla.

Forma de Entrega: Os arquivos .h, .c e o Makefile devem ser comprimidos em um arquivo ".zip" e submetidos por todos os integrantes do grupo na atividade do AVA. **Trabalhos entregues fora do prazo definido acima receberão nota zero**.

NÃO ADICIONAR no `.zip` arquivos que não forem código-fonte (e.g., arquivos de teste, diretórios vazios, arquivos gerados pelos programas, etc.).

Entrevistas: Haverá uma entrevista que definirá a nota individual de cada aluno.

Contexto

O trabalho consiste em desenvolver um conjunto de programas para mineração de dados em textos de notícias de jornal.

Objetivo

O trabalho tem como objetivo colocar em prática a utilização de:

- alocação dinâmica de memória
- estruturação de projetos em bibliotecas e programas
- ponteiros de funções
- funções para manipulação de arquivos textos e binários

Descrição

Este trabalho visa a construção de um buscador de notícias simplificado. Para isto, serão oferecidos um conjunto de notícias previamente filtrados e deverão ser implementado três programas:

- Construção de Índices: Cria um índice de documentos a partir dos textos das notícias, processando e organizando os dados de forma que as operações feitas pelos outros programas sejam mais eficientes;
- Programa principal: Programa que será utilizado pelo usuário final para buscar e classificar notícias; e
- Experimentos: realizar experimentos para medir o quão corretas são as análises automáticas feitas pelo sistema.

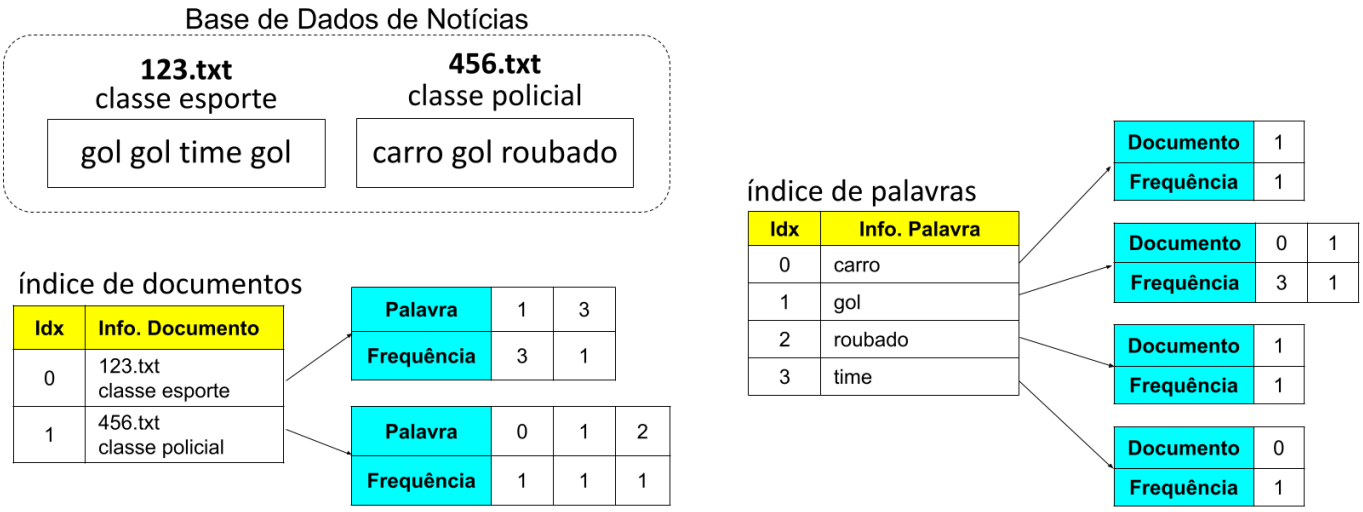
Cada programa deverá definir as estruturas e funções em uma biblioteca para resolver seus problemas específicos. Contudo, a biblioteca será "compartilhada" pelos programas (no sentido de ser usada pelos três quando necessário). Códigos replicados em dois ou mais programas levarão a punições severas na nota.

Serão disponibilizados arquivos .zip contendo dois arquivos, treino.txt e teste.txt, e duas pastas chamadas treino e teste. As pastas contêm vários arquivos texto, cada um com uma notícia de jornal. Os arquivos possuem a lista de arquivos nas respectivas pastas e a classe à qual a notícia pertence (e.g., eco para economia, ept para esportes e assim por diante).

Programa 1: Construção de Índices

A partir dos textos das notícias devem ser criados um índice de documentos (tipicamente chamado na literatura de índice direto ou *forward index*) e um índice de palavras (chamado na literatura de índice invertido ou *inverted index*). Estes índices armazenam as mesmas informações, a frequência e os valores de TF-IDF (explicação mais abaixo) das palavras nos documentos, mas em duas perspectivas diferentes, uma dos documentos e outra das palavras.

A figura abaixo ilustra os índices para uma base de dados que possui dois arquivos, 123.txt e 456.txt, das classes esporte e policial, respectivamente. As caixas abaixo dos nomes dos arquivos descrevem seus conteúdos. O índice de palavras armazena para cada palavra, a lista de documentos em que ela aparece, o número de ocorrências, e o valor de TF-IDF (omitido para tornar a figura mais legível). O índice de documentos armazena para cada documento, as palavras que aparecem nele, o número de ocorrências e os valores de TF-IDF. A título de exemplo, de acordo com o índice de documentos, a palavra gol aparece nos documentos 0 (123.txt) e 1 (456.txt) com frequência de 3 no primeiro e de 1 no segundo. Pelo índice de documentos, o texto 123.txt contém as palavras 1 (gol) e 3 (time) com frequências de 3 e 1, respectivamente.



Observação 1: No índice de documentos, palavras são referenciadas por suas posições no índice de palavras. Da mesma forma, no índice de palavras, documentos são referenciados por suas posições no índice de documentos.

Observação 2: A lista de palavras deve ser ordenada de forma que nos demais programas seja possível recurar as informações da palavra eficientemente usando busca binária (função bsearch).

Observação 3: Considere o índice de documentos. A figura pode dar a entender que as frequências das palavras devem ser armazenadas usando dois vetores, um com as posições das palavras e outro com as

contagens. Contudo, o código ficará mais organizado se for utilizado um vetor em que cada elemento é uma estrutura que contém a posição da palavra e a frequência. As mesmas observações valem para o índice de palavras.

Observação 4: Além da frequência da palavra, nos índices também deve ser armazenado o valor do TF-IDF (explicado mais abaixo).

Entrada e saída do Programa Construção de índice

O programa 1 deverá receber dois argumentos via linha de comando, a saber, o caminho para o arquivo que contém os nomes dos arquivos e as classes (o train.txt) e o nome do arquivo de saída. O programa deverá salvar no arquivo de saída os índices em **formato binário**.

Exemplo: Seja `indice` o nome do programa que gera o índice

Entrada:

```
.\indice data/train.txt indice.bin
```

Saída:

Em caso de sucesso: Arquivo binário com os índices e imprime o número de documentos e de palavras diferentes;

Em caso de erro: mensagem de erro. É responsabilidade do desenvolvedor garantir que o programa não "quebre" caso não sejam passados os argumentos incorretamente.

TF-IDF

O TF-IDF (acrônimo para *term frequency - inverse document frequency*, ou frequência do termo - inversa da frequência nos documentos) é um valor numérico utilizado frequentemente em sistemas de mineração de dados e calculado a partir das frequências de um termo em um documento específico e no *corpus* completo.

Para motivar o uso do TF-IDF, considere que o usuário deseja buscar as notícias que contêm os termos "o time foi vencedor com 4 gols" (imagine uma busca no google). As palavras "o", "foi", "com" e o número "4" aparecem com muita frequência nos mais diversos documentos. Note, contudo, que provavelmente o usuário não estava interessado em recuperar os documentos em que *estes termos* mais apareciam. Os termos "time", "vencedor" e "gols" representam melhor o alvo da consulta. O que estes termos têm de especial é que eles são frequentes em um subconjunto restrito do universo de notícias: as notícias de esportes.

O TF-IDF é uma forma de atribuir um peso à frequência das palavras de forma a reduzir a influência daquelas que são comuns em quase todos os documentos (no exemplo, "o", "foi", "com" e "4"). O TF-IDF de uma palavra p em um documento d é o produto de dois termos:

$$TF\text{-}IDF(p, d) = tf(p, d) * idf(p)$$

onde *tf(p, d)* é a frequência de *p* em *d* e *idf(t, d)* é um fator de ponderação dado por:

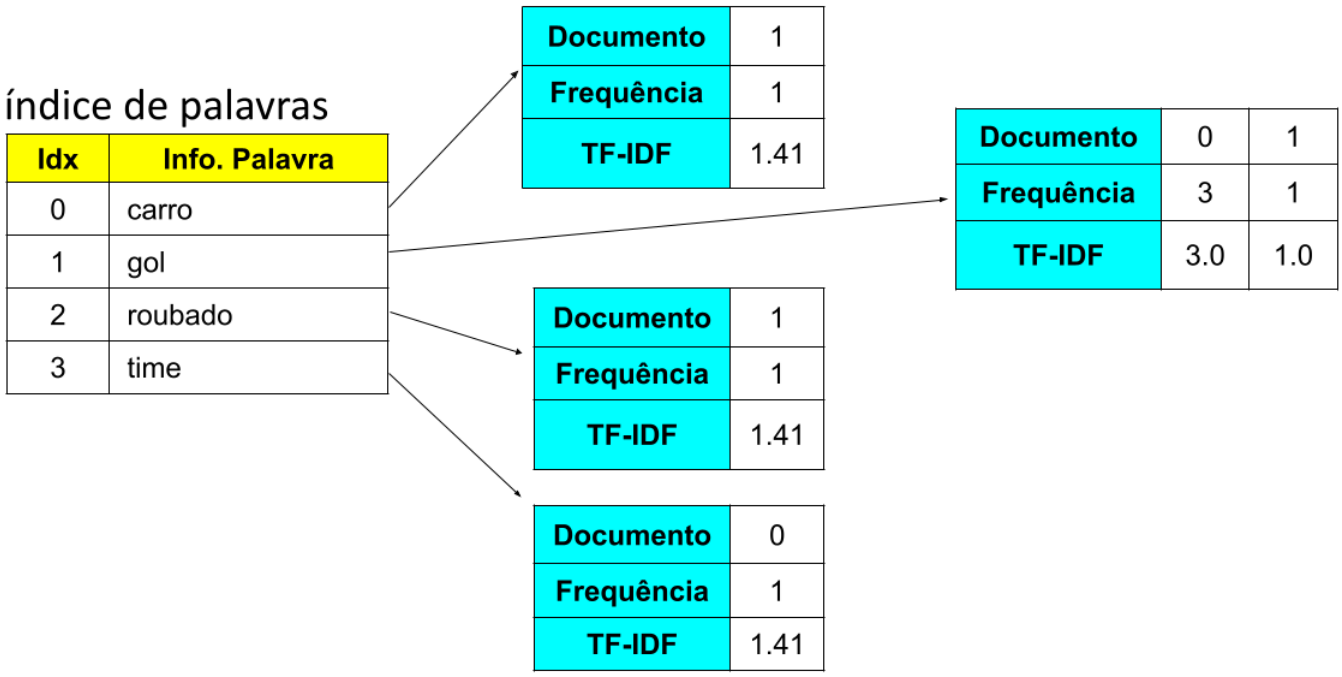
$$idf(p) = ln\left[\frac{1+n}{1+df(p)}\right] + 1$$

em que *n* é o número total de documentos e *df(p)* é o número de documentos em que *p* aparece.

Note como o TF-IDF de um termo em um documento será maior se o termo for muito frequente no documento e pouco frequente nos outros.

Observação: Após a construção dos índices, os valores de TF-IDF devem ser calculados e armazenados nos índices em conjunto com a frequência crua dos termos.

Os valores de TF-IDF para o exemplo da seção anterior é dado na figura abaixo. Como a palavra gol aparece em todas as notícicas do exemplo, cada ocorrência recebe peso (idf) de 1. Já as outras palavras que aparecem em apenas um documento receberam peso (idf) de 1.45 por ocorrência. Assim, neste contexto, a palavra gol seria considerada menos importante que as demais pelo sistema. Note, contudo, que este é um exemplo simplório e que na realidade as palavras que receberiam menos peso seriam, por exemplo artigos ("a", "as", "os", "um") e outros termos comuns ("com", "pouco", "muito", "é" e assim por diante).



Programa 2: Programa Principal

O programa principal receberá via linha de comando o caminho do arquivo gerado pelo programa 1 e um número inteiro K (ver opção 2 abaixo). Ele deverá ter como funcionalidades:

1. Buscar notícias: O usuário digitará uma linha contendo uma ou mais palavras e o programa deverá mostrar na tela as 10 notícias em que as palavras mais aparecem. Para selecionar as notícias a serem exibidas, o programa deverá utilizar o índice de palavras para identificar todas as notícias em que **pelo menos um** dos termos digitados aparece. Para cada notícia encontrada, deverão ser somados os valores de TF-IDF dos termos digitados, de forma que notícias que contém vários termos relevantes da consulta com

uma alta frequência recebam maior precedência. Somar os valores de TF-IDF é similar a somar as frequências das palavras nas notícicas, exceto que o TF-IDF atribui peso maior aos termos com maior "valor semântico". As notícias deverão ser exibidas de forma ordenada pela soma dos TF-IDFs, da maior para a menor.

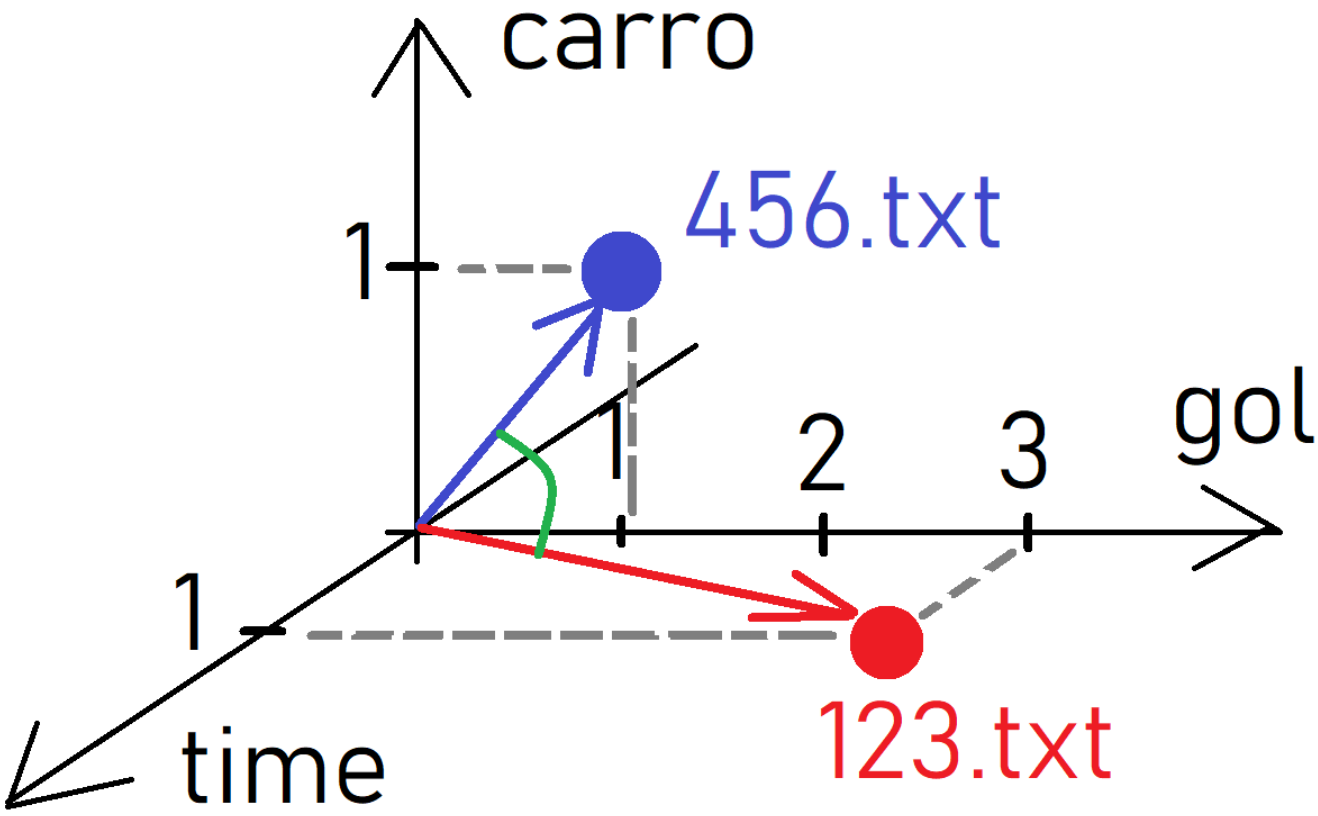
Exemplos de entrada para a opção de busca:

- *Gol*: Ao analisar o índice de palavras, vamos que a palavra gol aparece em dois documentos, o 0 e o 1. Ambos os documentos deveriam ser retornados pelo buscador. Como o TF-IDF no documento 0 (3.0) é maior que o do documento 1 (1.0), o 0 deve aparecer primeiro na lista.
- *carro roubado gol*: As palavras carro, roubado e gol aparecem nos documentos 0 e 1. Portanto, ambos devem ser exibidos no resultado do buscador. A prioridade dos documentos é a soma dos TF-IDFs dos termos da busca que aparecem no documento. O documento 0 contém apenas a palavra gol com TF-IDF de 3 e, logo, este será o valor de prioridade do documento. No documento 1, as três palavras aparecem e a soma de seus TF-IDFs será 1.41 (de carro) + 1.41 (de roubado) + 1.0 (de gol) = 3.81. Assim, o documento 1 deverá aparecer acima na lista. Note que se a consulta fosse *carro gol*, o documento 1 teria prioridade de 2.41 e seria mostrado como segundo da lista. Este segundo exemplo mostra um caso em que o resultado da busca é diferente do que seria esperado por um humano.
- *dolar*: Deve ser exibida uma mensagem dizendo que não existem notícias contendo este texto.
- *carro cachorro*: Se for realizada uma consulta em que parte dos termos existe no dicionário e parte não existe, apenas aqueles que existem devem ser considerados. No exemplo acima, será retornado apenas o documento 1 que contém a palavra carro.

2. Classificar notícias: O usuário digitará um texto e o programa deverá prover uma estimativa de qual classe de notícias o texto pertence. Para isto, será utilizado o classificador por K vizinhos mais próximos (*K-Nearest Neighbors* - KNN). Primeiro, a partir do texto digitado deverá ser construída uma estrutura com a frequência dos termos similar àquela presente no índice de documentos. Depois, usando esta estrutura, deverão ser buscadas as *K* notícias mais similares àquela digitada na base de dados e a classe mais comum neste conjunto será atribuída ao texto digitado (o valor de *K* é aquele passado via linha de comando).

A medida de similaridade entre textos será dada pelo cosseno entre vetores no espaço. Como ilustrado na figura abaixo, notícias podem ser representadas por vetores no espaço. Considere que cada palavra existente no *corpus* define um eixo em um sistema de coordenadas e que a frequência (ou TF-IDF) do termo é o valor da coordenada. Cada notícia será representada um vetor no espaço cujas coordenadas serão dadas pelas frequências (ou TF-IDFs) das palavras no documento.

Considerando a base de dados de exemplo, o vetor da notícia 456.txt é representada pelo vetor (gol=1, carro=1, time=0), enquanto a notícia 123.txt é representada pelo vetor (gol=3, carro=0, time=1). A palavra "roubado" do arquivo 456.txt foi omitida porque não é possível visualizar adequadamente mais de três dimensões.

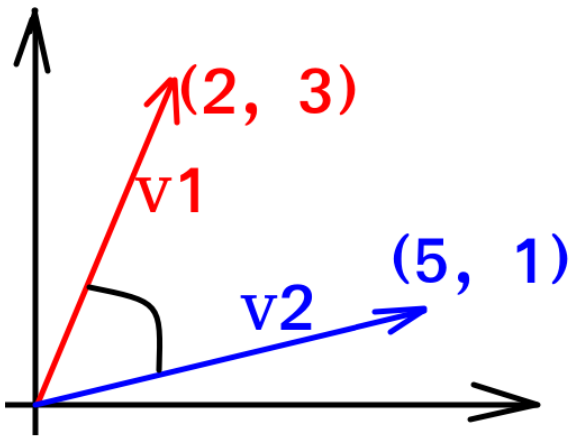


O cosseno é uma forma de medir a similaridade entre documentos analisando o ângulo entre respectivos vetores no espaço (marcado em verde na figura acima). Dois vetores são considerados similares se o ângulo entre eles tender a zero e, neste caso, o cosseno tenderá a 1 (100% de similaridade). Se o ângulo entre dois vetores tender a 90 graus, o cosseno tenderá a 0 (0% de similaridade). O cosseno é dado por:

$$cos(v1, v2) = \frac{v1.v2}{||v1||.||v2||}$$

onde v1 e v2 são vetores contendo a frequência das palavras em dois documentos, v1 . v2 é o produto interno entre eles e ||.|| é a operação de norma.

A figura abaixo ilustra o cálculo do cosseno para dois vetores:



$$\cos(v1,\ v2) = \frac{(2,\ 3) \bullet (5,\ 1)}{||(2,\ 3)||\ \ ||(5,\ 1)||}$$

$$\cos(v1,\ v2) = \frac{2\ *\ 5\ +\ 3\ *\ 1}{\sqrt{2^2\ +\ 3^2}\ *\ \sqrt{5^2\ +\ 1^2}}$$

$$\cos(v1,\ v2) = 0{,}717$$

No trabalho, as coordenadas dos vetores serão os valores de TF-IDF dos termos. Portanto, expandindo a fórmula para o caso de notícicas, temos:

$$cos(d1, d2) = \frac{\sum_{w \in W} [TF\text{-}IDF(w, d1) * TF\text{-}IDF(w, d2)]}{\sqrt{\sum_{w \in W} TF\text{-}IDF(w, d1)^2} * \sqrt{\sum_{w \in W} TF\text{-}IDF(w, d1)^2}}$$

onde d1 e d2 são duas notícias, w é um palavra e W é o conjunto de todas as palavras no *corpus*.

Observação: Apenas os termos existentes **nos dois textos** precisam participar no cálculo do cosseno entre eles. Se um termo não existe em um dos documentos, sua frequência (e, portanto, o TF-IDF) naquele documento será zero, o que faz com que ele seja desconsiderado tanto no numerador quanto no denominador na fórmula. Realizar loops sobre o conjunto de todas as palavras no momento de calcular o cosseno ao invés de considerar apenas aquelas que aparecem nos dois documentos gerará punições graves na nota.

Observação: Note como o sistema de classificação é parecido com o sistema de recomendação do trabalho 1.

Observação: A mesma função usada no índice de documentos para calcular e armazenar a frequência e o TF-IDF dos termos deverá ser utilizada com o texto digitado.

Considere então os seguintes exemplos de entrada de classificacao:

- *Gol*: A consulta será tratada como um documento em que a palavra "gol" aparece uma vez.
- *carro gol roubado*
- *dolar*: << PALAVRA QUE NAO EXISTE NO CORPUS >>

3. Relatório de Palavra: O usuário deve digitar uma palavra e o programa deve exibir o número total de documentos em que a palavra aparece, os 10 em que ela aparece com mais frequência e a frequência da palavra por classe. Ambas as listas devem ser ordenadas com os itens de maior contagem aparecendo primeiro. A ordenação deve ser feita ao selecionar a opção usando a função qsort.

4. Relatório de Documentos: Exibe os 10 documentos mais longos e os 10 mais curtos com o número de palavras e as respectivas classes. As listas devem ser ordenadas, a primeira do maior para o menor e a segunda do menor para o maior. A ordenação deve ser feita ao selecionar a opção usando a função qsort.

Programa 3: Experimentos

O programa 3 deverá receber quatro argumentos via linha de comando:

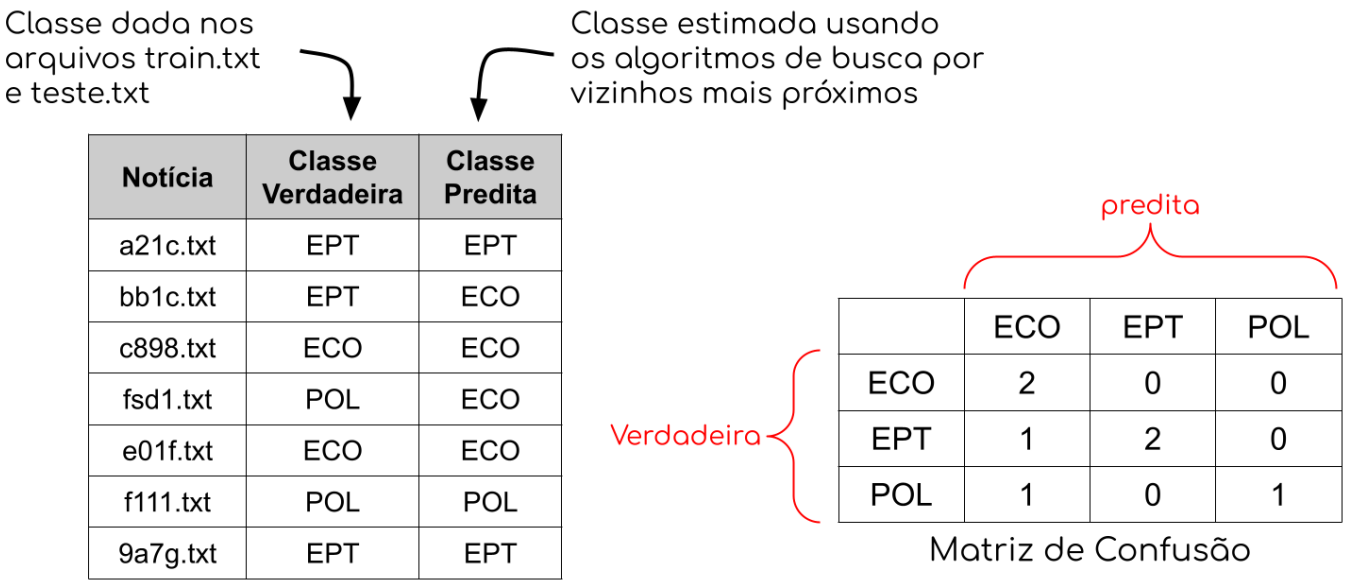
1. O caminho para o arquivo binário contendo os índices gerado pelo programa 1.
2. O caminho para o arquivo contendo as notícias e classes do conjunto de teste (o teste.txt).
3. Um número inteiro K utilizado pelo classificador.
4. O nome do arquivo de saída que o programa deverá escrever.

O programa deve utilizar a técnica de classificação de notícias descrito na seção anterior para classificar todas as notícias do conjunto de teste e salvar em um arquivo texto especificado via linha de comando a acurácia (percentual de acerto) do método e a matriz de confusão.

Na matriz de confusão, linhas representam a classe verdadeira da notícia e colunas representam a classe estimada pelo sistema. Assim, a matriz de confusão é uma matriz quadrada cuja largura e altura são iguais ao número de classes. A célula (i, j) da matriz contém o número de notícias do conjunto de teste que são da classe i, mas foram classificados como sendo da classe j.

Um classificador ideal retornaria a classe verdadeira para todas as amostras, o que levaria a uma matriz de confusão diagonal (é improvável que este caso ideal seja alcançado no trabalho). Além da matriz de confusão e da acurácia, deve ser salvo no arquivo uma sequência de linhas indicando à qual classe as linhas/colunas da matriz de confusão se referem.

A título de exemplo, a figura abaixo mostra a matriz de confusão para uma base de dados que contém 7 notícias das classes EPT, ECO e POL. A tabela da esquerda descreve para cada notícia as classes verdadeiras e estimadas usando os algoritmos descritos mais acima. A tabela direita é a matriz de confusão. A primeira linha contendo os valores (2, 0, 0) indica que existiam 2 documentos da classe ECO e ambos foram classificados corretamente. Os valores (1, 2, 0) na segunda linha indicam que existiam 3 documentos da classe EPT e 2 deles foram classificados corretamente e 1 foi incorretamente classificado como sendo da classe ECO. O fato da soma dos elementos da primeira coluna ser muito maior que das demais (4, enquanto as outras têm valores 2 e 1), indica que o classificador tem uma tendência de atribuir a classe ECO para a maioria das notícias.



Notícia	Classe Verdadeira	Classe Predita
a21c.txt	EPT	EPT
bb1c.txt	EPT	ECO
c898.txt	ECO	ECO
fsd1.txt	POL	ECO
e01f.txt	ECO	ECO
f111.txt	POL	POL
9a7g.txt	EPT	EPT

	ECO	EPT	POL
ECO	2	0	0
EPT	1	2	0
POL	1	0	1

Restrições

- Devem ser usadas as funções `qsort` e `bsearch` sempre que for necessário ordenar vetores ou buscar itens em vetores ordenados.
- Não será permitido que as estatísticas nos relatórios do programa principal exibidas sejam pré-computadas e armazenadas nas estruturas. Elas deverão ser derivadas dos índices quando solicitado pelo usuário.

Avaliação

Critérios Funcionais

1. Programa 1: 20%

1.a. Indice de palavras correto: 7%

1.b. Indice de documentos correto: 7%

1.c. Arquivo binário escrito corretamente: 6%

2. Programa 2: 60%

2.a. Buscador: 20%

2.b. Classificador: 30%

2.c. Relatório de Palavra: 5%

2.d. Relatório de Documentos: 5%

3. Programa 3: 20%

3.a. Classes dos documentos de teste produzidas corretamente: 5%

3.a. Acurácia correta: 2%

3.b. Matriz de confusão correta: 10%

3.c. Escrita do arquivo texto: 3%

Pontos Extras

- [+10%] Implementar o classificador por centróide mais próximo. Este classificador primeiro calcula o valor médio de frequência/TF-IDF de todas as notícias de cada classe. Estes vetores médios são chamados de cetróides e eles representam as características mais comuns dos documentos que compõe aquela classe (observe a semelhança à media das playlists no TP1). Cada classe terá um centróide associado. Para classificar uma nova notícia, é buscado qual é o centróide mais similar (usando o cosseno) à nova notícia e é retornada a classe associada. Este classificador tende a ser mais rápido que o classificador por K vizinhos mais próximos dado que ao invés de medir a distância para todos os documentos, será calculada a distância apenas para os centróides. Contudo, ele tende a levar à mais erros quando os vetores no espaço não se aproximam de uma distribuição Gaussiana. Para ganhar o ponto extra, o grupo deve:
    - Salvar os centróides no arquivo produzido pelo programa 1.
    - Adicionar uma opção extra no programa 2 para realizar a classificação usando o algoritmo de centróide mais próximo.
    - Adicionar no programa 3 a acurácia e a matriz de confusão usando o novo algoritmo.
- [+10%] Comparar a performance de variações do algoritmo de classificação: No programa 3, calcular a acurácia e a matriz de confusão para as seguintes variações dos algoritmos propostos:
    - Frequência dos termos ao invés do TF-IDF e similaridade por cosseno.
    - Presença dos termos nos documentos ao invés do TF-IDF e similaridade por cosseno. Neste caso, cada palavra receberá valor 1 se existir em um documento e 0 se não existir, independente da frequência.

    - Peso dos termos por TF-IDF e distância euclidiana ao invés do cosseno.
    - Presença dos termos nos documentos ao invés do TF-IDF e distância euclidiana ao invés do cosseno.

[+10%] Implementar uma tabela hash e utilizá-la como índice de palavras. Para obter a pontuação, o grupo deve implementar um TAD Tabela Hash e utilizá-lo para armazenar o índice de palavras. Ele deve ser utilizado inclusive no processo de construção do índice. Para obter a pontuação, o aluno deverá explicar durante a entrevista como a estrutura de dados funciona e como a peformance (em termos de tempo) das operações de inserção, verificação de pertinência, atualização, remoção e iteração sobre elementos se compara com a de um vetor.

Critérios de Qualidade

- Deve ser provido um makefile que permita a compilação do código usando "make" e a remoção de arquivos objeto, bilbiotecas e executáveis usando "make clean". Warnings durante a compilação serão punidos com 25% da nota se não resultarem em funcionamento incorreto do programa e em 50% se resultarem (e.g., uso de variáveis não inicializadas será punido com 50%, enquanto variáveis não utilizadas serão punidas com 25%). Não serão considerados na punição warnings por não utilização do retorno de funções (e.g., não uso do retorno do scanf). Códigos que não compilarem receberão nota 0.

- As punições por erros do valgrind no trabalho 1 foram amenas (máx de 10%) dado que era o primeiro contato com a ferramenta e com os conhecimentos de gestão de memória. Neste trabalho, não liberar todos os espaços alocados gerará uma punição de 30% da nota, enquanto outros erros do valgrind gerarão uma punição de 50% da nota.

- Variáveis e funções com nomes inadequados poderão receber punições de até 30% da nota.

- O código deve ser organizado em pastas de forma que os headers (.h), sources (.c) e objetos (.o) sejam armazenados em pastas diferentes. A pasta raíz do projeto deve armazenar apenas os arquivos main.c, Makefile e os binários finais de bibliotecas (.a) e do programa (.exe no windows ou ELF no linux). Não seguir esta estrutura gerará uma punição de até 30% da nota.

- Não seguir a interface de utilização dos programas (argumentos de linha de comando e opções) resultará em punições de até 30% da nota.

- O programa deve utilizar TADs opacas e as funções main dos programas devem ser escritas predominantemente invocando funções definidas na biblioteca. Funções excessivamente longas (mais de uma tela) e a má organização do código em TADs resultará em punições de até 20%.