Universidade Federal do Rio Grande do Sul
Professora: Karin Becker

CLASSIFICAÇÃO E PESQUISA DE DADOS - ANÁLISE DE SENTIMENTOS

Lúcio Pereira Franco 262867
Rodrigo Franzoi Scroferneker 252849

Porto Alegre, 7 de julho de 2017

O Trabalho:

"A análise de sentimentos visa determinar automaticamente o sentimento presente em um texto. Neste trabalho entenderemos sentimento como polaridade, a saber negativa, neutra e positiva. Seu objetivo neste trabalho é utilizar os conhecimentos adquiridos na disciplina para tratar um problema de análise de sentimento em tweets.” Becker, Karin.

Infraestrutura:

Para realizar o projeto, escolhemos a linguagem de programação JAVA pela afinidade dos integrantes do grupo com a linguagem e pela sua simplicidade em trabalhar com arquivos e manuseio de strings.

Com auxilio das bibliotecas:

1) “java.text.Normalizer”: para padronizar as string (maiúsculas/minúsculas, acentuadas) e permitir um sorting mais fácil;

2) “java.util.ArrayList”: para uso de arrays com tamanhos mutáveis;

3) “java.util.StringTokenizer”: para separar os elementos de string de uma maneira mais simples e eficiente;

4) “java.util.Scanner”: para leitura das string no arquivo csv;

5) "java.io": para escrita no terminal.

Estrutura:

Quanto a estrutura de pesquisa, avaliamos o uso conforme os seguintes parâmetros:

- chaves de tamanhos bem variados;
- muitas chaves com prefixos parecidos;
- muitas chaves escritas de maneira enfática. Ex.: GRITOOOOOOO;
- entrada com muitas chaves;
- numero de chaves não previsto;
- novas entradas de chaves.

Com essas considerações, decidimos resolver o problema proposto com uma estrutura de Arvore Trie, com organização das chaves pelos seus prefixos. Alguns motivos que nos levaram à essa escolha:

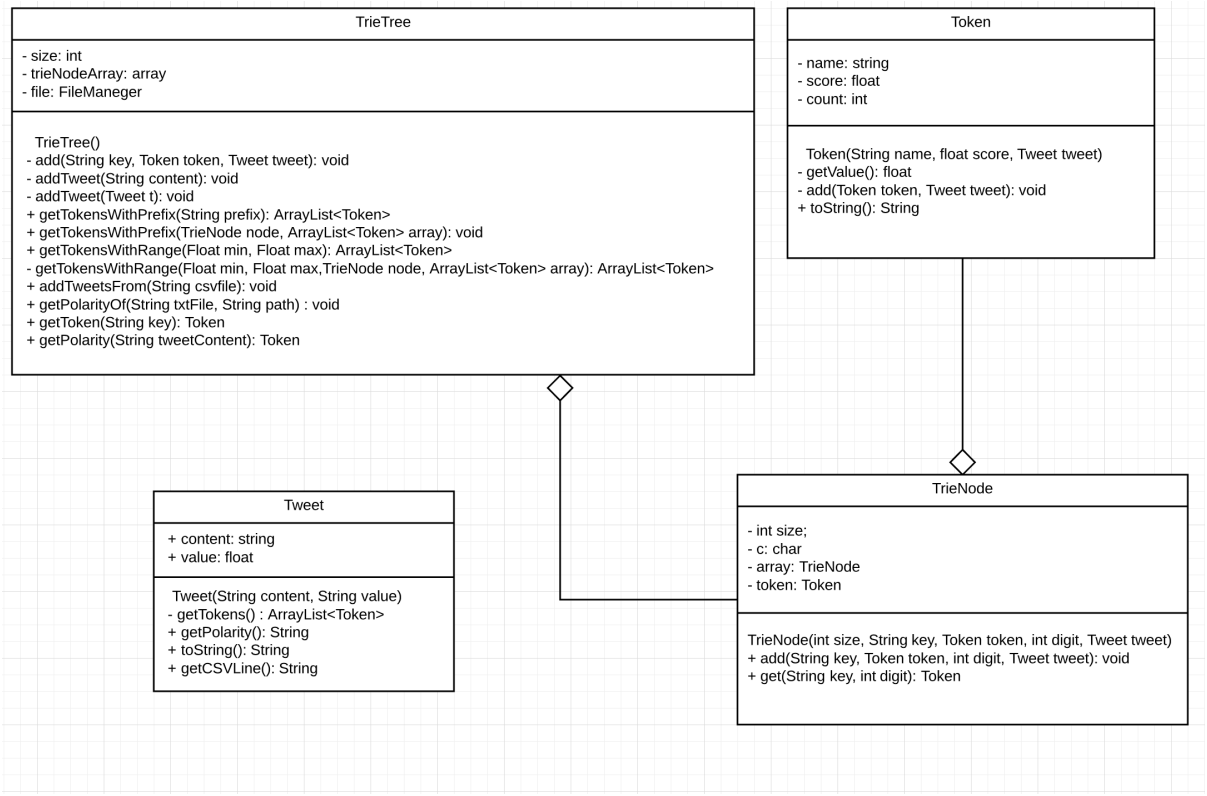
- a eficiência nas buscas não depende do numero de elementos que o dicionário contem;
- a inserção de muitos dados não é um complicador;
- a inserção de novos elementos não requere atualização de todo o dicionário.

A inserção das chaves na TRIE é feita logo após a leitura do arquivo. Como explicado anteriormente, o programa lê o arquivo e cria os objetos Tweets. Seguindo os seguintes passos o programa adiciona os elementos nosso dicionário:

1) Na inicialização da TRIE, passamos todos os objetos Tweets criados;
2) Para cada Tweet é chamada uma função que cria e retorna, um por um, todos os seus Tokens (palavras);
3) Esses Tokens são adicionados de forma recursiva na TRIE;
4) Essa adição é feita transformando cada caracter do Token em seu valor ASCII e caminhando pela TRIE até achar o seu lugar correto.

Método:

Estruturamos o código de forma que cada uma das entidades tivesse a sua casse específica. Tais quais: Trie, TrieNode, Tweet, Token e FileManager.



Criamos uma grande estrutura chamada *TrieTree* para lidar com todas as funções que precisamos. Ela é a que lida com todas as classes criadas, realiza as funções de busca e inserção de novos conteúdos.

Outra grande classe que criamos é a *FileManager*, responsável por todo fluxo de arquivos do programa. Suas funções são amplamente utilizadas pela *TrieTree*, assim, o dicionário abstrai funções de leitura e escrita para essa classe.

Ao ler um arquivo, geramos vários *Tweet* com sentimentos, a partir destes tweets geramos tokens. Para evitar a variação de tokens similares e palavras irrelevantes, filtramos por apenas palavras com 3 dígitos ou mais, removemos os acentos e pontuações.

Os tokes são as chaves da nossa estrutura *TrieTree*.

Previsão de Sentimentos:

Da mesma forma que lemos o arquivo dos tweets iniciais, o novo arquivo é lido linha por linha e o programa faz as seguintes operações:

1) Normaliza o texto lido na linha;
2) Com auxilio da biblioteca “java.util.StringTokenizer”, para cada apalavrado texto, faz uma busca na TRIE para achar o Token correspondente a palavra;
3) Faz a soma de todos os scores dos Tokens pertencentes ao texto;
4) Cria um novo Tweet com o texto e o score calculado;
5) Salva o novo Tweet no arquivo csv com os demais.

Procedimento de Busca:

"Dado um radical de duas ou mais letras, buscar todas as suas variações encontradas nos tweets. Por exemplo, com o radical “am”, poderiamos encontrar diversas conjugações do verbo amar (amei, amamos), variações do substantivo (amor, amorzinho), formas de expressar sentimento (ameeeeeeeei, amooooo), além de outras palavras não relacionadas (amazonas, americano).” Becker, Karin.

Para resolver esse problema o programa faz os seguinte passos:

1) Começa a percorrer a Árvore Trie pelos caracteres da palavra informada;
2) Quando encontrar o nodo que representa o ultimo caracter do radical, começa a percorrer todos os nodos filhos, recursivamente;
3) Todos os Tokens encontrados são palavras com o mesmo radical.

Essa tarefa é especialmente boa de fazer com Arvore Trie, pela trivialidade de sabermos que, encontrando o caminho até o ultimo caracter do radical, todos os tokens, em nodos filhos desse caminho, serão as palavras que procuramos.

"Buscar palavras cujo escore sentimento esteja em algum intervalo (exemplos: palavras com escore de sentimento maior que 0.3, ou entre 0.1 e 0.5)" Becker, Karin.

Para resolver esse problema o programa faz os seguintes passos:

1) Percorre toda a TRIE conferindo se os Tokens encontrados tem score adequado para serem selecionados;

2) Se tiverem o score entre o limite determinando, o token é adicionado em um vetor;

3) Após a passagem da arvore inteira, o vetor tem todos os Tokens cujos scores estão entre os limites predefinidos.

Esse programa não é muito eficiente com a estrutura de Arvore Trie, pois não sabemos os endereços dos nodos que contem os Tokens. Diferentemente de uma estrutura tipo Hash, que cada endereço populado teria seu Token.

Menu:

Assim que o programa for executado, aparecerá no terminal 5 opções para interagir com o dicionário:

1 - Adicionar arquivo CSV ao dicionário

Ao escolher essa função, em seguida será solicitado para inserir o o caminho do diretório do arquivo CSV, por exemplo *Files/entrada.csv*. Em seguida, será executada a função *addTweetsFrom* do dicionário, fazendo com que os valores dele e do arquivo auxiliar sejam atualizados.

2 - Ler arquivo txt e retornar os sentimentos em um outro arquivo

Essa função é responsável de ler um arquivo TXT com tweets em cada linha e criará um documento CSV com os sentimentos dos tweet. Os diretórios são perguntados ao selecionar a opção. Quando os parâmetros forem preenchidos, a função do dicionário *getPolarityOf* será executada.

3 - Retornar palavras com o sufixo

Após inserir um sufixo qualquer, por exemplo *amo,* o programa executará a função do dicionário *getTokensWithPrefix* mostrará no terminal todos os tokens salvos no dicionário com o sufixo escolhido.

4 - Retornar palavras com o score de sentimento em um intervalo

O programa pedirá para ser inserido, um valor mínimo e máximo, respectivamente, e mostrará no terminal todos os tokens do dicionário que possuem o score de sentimentos com os valores mínimos até o máximo, os valores inseridos estarão contidos no intervalo. A função chamada do dicionário é *getTokensWithRange*.

5 -Sair

Encerra a execução do programa.

Classes:

TrieTree: Estrutura de dicionário responsável para montar o dicionário de sentimentos, baseada na *Trie Tree*. Ela é responsável pela lógica de buscas e adição e de chamar a estrutura *FileManager.*

TrieTree: Inicializa a estrutura a partir de um arquivo com tweets e sentimentos já pré estabelecidos.

add: adiciona um novo nodo à arvore ou segue, recursivamente percorrendo a arvore até achar um nodo para adicionar

addTweet: adiciona novos Tweets à estrutura e ao arquivo a partir de uma String ou Tweet.

getTokensWithRange: duas funções utilizadas para percorrer recursivamente a estrutura e retornar um array de tokens que possuem os sentimentos inseridos nos parâmetros.

getTokensWithPrefix: duas funções utilizadas para percorrer recursivamente a estrutura e retornar um array de tokens que possuem os mesmos prefixos.

addTweetsFrom: insere à estrutura tweets vindas de um documento CSV.

getPolarityOf: recebe um arquivo com uma lista de tweets no formato TXT e insere em outro arquivo CSV de saída a polaridade dos tweets.

getToken: retorna um Token a partir de uma String.

getPolarity: retorna um Tweet contendo a polaridade da String inserida no parâmetro.

TrieNode: Nodo da estrutura TrieTree.

TrieNode: inicializa o nodo com seus parâmetros.

add: adiciona o token ao nodo ou atualiza o token, caso já exista um token no nodo encontrado, apenas se o ultimo dígito adicionado for o ultimo dígito do token, caso não for, ela adiciona o próximo dígito a um novo nodo no array de nodos da própria estrutura. Ele se adiciona recursivamente até chegar no ultimo dígito.

get: função chamada pela *TrieTree* para retornar um token. Ela percorre a árvore recursivamente se chamando até encontrar o token desejado, se não encontrar, retorna *null.*

FileManager: Responsável pelo gerenciamento de ler e escrever em arquivos. Suas funções são chamadas exclusivamente pela *TrieTree.*

FileManager: ao inicializar, carrega os tweets iniciais.

loadLinesFrom: lê as linhas de um arquivo txt e retorna um Array de Strings, que posteriormente serão transformadas em tweets.

loadTweetsFrom: lê os tweets de um CSV e retorna um array de Tweets.

addToFile: adiciona um tweet apenas ao arquivo de tweets.

addToExit: adiciona um array de tweets a uma saída.

closeFile: fecha o leitor de arquivos.

Tweet: Estrutura criada para lidar os tweets e seus valores (sentimentos).

Tweet: apenas inicializa a estrutura

getTokens: retorna todos os tokens válidos da frase do tweet.

getPolarity: retorna a polaridade do tweet, sendo a soma dos scores de cada palavra válida. Sendo o score > 0.1, positivo, < -0.1 negativo e, entre esses valores, neutro.

getCSVLine: retorna uma string no formato de CSV

Token: Estrutura criada para lidar com os valores das palavras adicionadas ao dicionário.

Token: inicializa a estrutura.

getValue: retorna a soma dos scores dos tokens dividido pela freqüência em que aparecem.

add: soma os scores e acumula o contador