Relatório do Trabalho Prático

UFRGS, 2018/1

Ciência da Computação

Classificação e Pesquisa de Dados, turma A

Integrantes:

Felipe Girardi – 00264098

Lucca Strelow Milano – 00287683

Wellington Nascente Hirsch – 00287715

1) Infraestrutura

A linguagem de programação utilizada é Python 3.6.5. O código é dividido em 3 arquivos:

- **dicio.py** contém a classe 'Tweet', que guarda os tweets lidos com sua polaridade, e as seguintes funções:
    - 'csv2trie', que lê os tweets, simplifica-os, extrai suas palavras e chama a função 'insertTrie';
    - 'reduceTT', que simplifica o tweet, eliminando acentos e palavras pouco significativas e reduzindo-as quando possível, visando facilitar o cálculo do valor de sentimento;
    - 'polarizeTweet', que polariza os tweets contidos em um dado arquivo de acordo com as polaridades das palavras do dicionário e salva-os em um arquivo;
    - 'reduce2radical', função que auxilia na simplificação de palavras em 'reduceTT';
    - 'funcionalidadeA', que executa a funcionalidade A (encontrar tweets com a palavra digitada);
    - 'funcionalidadeB', que executa a funcionalidade B (encontrar palavras com o prefixo digitado).

- **rTrie.py** contém as seguintes classes:
    - 'Trie', que é a estrutura de dados (R-Trie) utilizada para armazenar as informações necessárias (cada nodo possui um char, uma palavra - a classe Word - se for folha, seus filhos e o nível da árvore);
    - 'Word', que contém as informações associadas à a cada palavra em um nodo folha:
        - valor de sentimento médio;
        - número de tweets em que aparece;
        - valor de sentimento acumulado;

        - uma flag para detectar uma palavra repetida em um tweet;
        - índice para a lista de postings.

    E as seguintes funções:

    - ‘insertTrie’, que insere a palavra na árvore Trie, verificando se ela já existe ou não, atualizando as informações da palavra se necessário, além de atualizar os postings para cada palavra;
    - ‘searchTrie’, que procura uma palavra e suas informações na Trie;
    - ‘dataInTrie’, que retorna um vetor com todas as palavras do dicionário de palavras (a fim de imprimi-las);
    - ‘getByPrefix’, que procura uma palavra com dado prefixo na Trie (para a funcionalidade B);
    - ‘dataPrefix’, que imprime e salva as palavras com dado prefixo em um arquivo.

- **main.py** faz uso de um loop - que para quando o usuário não desejar inserir mais arquivos de tweets - e das funções criadas para solucionar a tarefa. Ele inicia lendo um arquivo de tweets polarizados informado pelo usuário e os salva - em forma simplificada pela função ‘reduceTT’ - na arvore Trie, fazendo uso da função ‘csv2Trie’. Após isso, ele salva a lista de postings em um arquivo e imprime o dicionário de palavras resultante da leitura de tweets, utilizando a função ‘dataInTrie’. Posteriormente, ele polariza um arquivo composto por tweets (sem scores de sentimentos) informado pelo usuário, fazendo uso da função polarizeTweet. Em seguida, ele realiza a funcionalidade A do trabalho segundo a palavra informada pelo usuário, usando a função ‘funcionalidadeA’ e, logo após, realiza a funcionalidade B do trabalho - com o prefixo informado também pelo usuário - utilizando a função ‘funcionalidadeB’. Por último, caso o usuário deseje, ele insere mais tweets de um arquivo (informado pelo usuário) no arquivo de tweets original, atualizando a árvore e as demais estruturas, continuando o loop até o usuário não desejar inserir novos tweets.

Apenas uma biblioteca extra foi utilizada: ‘unicodedata’, em dicio.py, cuja função ‘normalize’ auxiliou na simplificação dos tweets. Nenhum framework foi utilizado.

Mais informações serão mencionadas mais adiante no relatório.

2) Acesso ao arquivo de tweets

Tratando-se do arquivo de "alimentação" do dicionário, a aplicação pede um nome de arquivo .csv no formato dado na especificação do trabalho (tweet; polaridade em cada linha). Assim, para cada um dos tweets do arquivo, lendo linha após linha, o tweet é simplificado (as palavras nele contidas eliminando acentos, pontuação, letras maiúsculas e palavras com tamanho menor que 3) e a polaridade informada no arquivo é atribuída a todas as palavras restantes do tweet após a simplificação. Em seguida adicionamos essas palavras no dicionário.

Agora tratando-se do arquivo a ser polarizado, a aplicação da mesma forma pede um nome de arquivo .csv no formato dado na especificação do trabalho (um tweet em cada linha). Assim, para cada um dos tweets do arquivo o tweet é simplificado (sem alterar o original) e é efetuada a busca de cada uma das palavras restantes no dicionário. Ao encontrar a palavra, usamos a sua polaridade média para descobrir a polaridade do tweet. Caso não encontre a palavra, ela simplesmente não é contabilizada. Com isso é calculada a média de todas as polaridades medias das palavras presentes no tweet e atribuída uma classificação (negativa, neutra ou positiva) para o tweet. Com tudo calculado escrevemos em um novo arquivo no formato (tweet original; polaridade) todos os tweets do arquivo de entrada com seus resultados.

Essa forma de buscar pelas palavras, tanto no arquivo de alimentação quanto no que será polarizado, o acesso às palavras mencionado acima foi a alternativa mais adequada para armazenar a palavra e suas informações no dicionário, que será explicado a seguir.

3) Representação do dicionário

A árvore R-Trie foi utilizada para representar o dicionário, pois dado o número de informações a serem armazenadas, é necessária uma estrutura cujo tempo de execução não dependa do número total de elementos. A Trie possui essa propriedade, sendo que quanto maior seu tamanho mais eficiente torna-se o uso do espaço. A busca na Trie restringe o número de elementos a serem observados e comparados a cada char analisado - percorrendo o caminho indicado por eles, diminuindo assim o tempo de procura das palavras dos tweets - provando ser muito útil com um grande número de tweets e inserção, por conta de sua análise por caminho de caracteres. A Trie também pode associar valores às suas folhas, armazenando assim, os dados de cada palavra - outra propriedade que foi utilizada na resolução da tarefa para armazenar os dados relacionados aos sentimentos, formando assim o dicionário de palavras. Em suma, a R-Trie foi escolhida devido à fácil manipulação, baixa complexidade e grande poder de trabalho com strings.

A aplicação possui uma única Trie vazia da qual, após ler o arquivo de "alimentação" do dicionário como explicado na seção anterior, é preenchida com diversas palavras criando assim um dicionário. O nodo da Trie pode ser de 3 tipos:

- um nodo raiz que é apenas usado como controle da árvore;

- um nodo vazio que é um caminho para uma palavra válida do dicionário da qual possui apenas um caractere. Não possui valor;

- um nodo com palavra que é uma palavra reconhecida no dicionário da qual é constituída pela concatenação de todas os caracteres anteriores e do caractere do nodo atual de forma sequencial a partir da raiz até esse nodo. O nodo possui um objeto do tipo ‘Word’ (uma estrutura que armazena uma palavra, informações estatísticas sobre a sua polaridade e outras informações, mencionadas na seção 1) como valor.

A inserção e atualização da Trie é feita com uma única função, ‘csv2trie’, que recebe um objeto do tipo ‘Word’ como entrada e começa a percorrer a

árvore iterativamente buscando os nodos vazios dado cada caractere da palavra contida na Word até o penúltimo caractere. Se terminar antes disso significa que não há nodos vazios na árvore que cheguem até a palavra válida, então adicionamos esses nodos do ponto que paramos até o penúltimo caractere da palavra. Com isso finalizamos a inserção efetuando os seguintes testes:

- Se o último caractere da string não for filho do nodo anterior aonde a palavra deve ser inserida (ou seja, o nodo não está na árvore ainda), então um nodo com palavra e informações associadas é inserido, significando que a palavra não estava no dicionário;

- Caso ele já estiver, mas for um nodo vazio, então ele é convertido para um nodo com palavra e informações atribuindo um valor ao nodo vazio, significando que a palavra não estava no dicionário;

- Porém caso já estiver e for um nodo com palavra, significando que a palavra já estava no dicionário, então é efetuada uma operação de "soma" dos dados estatísticos do objeto Word que já estava no dicionário com o que não estava e inserindo o resultado atualizado na árvore. Aqui também é decrescentado em 1 o contador de tweets para a palavra (no código é ‘appears’) caso ela apareca 2 ou mais vezes no mesmo tweet.

Nessa função também é sempre atualizado o dicionário chave/postings, que será utilizado na funcionalidade A e explicado melhor a seguir.

4) Funcionalidades

Ambas as funcionalidades utilizam a R-Trie, sendo que a funcionalidade A também utiliza um dicionário chave/postings, que contém uma lista de postings.

A) Funcionalidade A

A funcionalidade A utiliza arquivo invertido para localizar as palavras solicitadas. O arquivo invertido é um dicionário chave/posting (lista de postings) que contém a chave de uma palavra e a lista de todos os tweets em que a palavra aparece. Como cada linha do arquivo de tweets possui um tweet, a lista de postings é uma lista de todas as linhas em que a palavra se encontra. A chave/index de cada palavra está contida no dicionário de palavras na árvore Trie. Logo, ao encontrar a palavra na árvore Trie, tem-se seu índice, que aponta para a lista de postings, onde estão todos os tweets que contém a palavra, bastando abrir o arquivo de tweets e encontrar a linha que corresponde ao posting.

O arquivo invertido/lista de postings foi utilizado na funcionalidade A visando obter um registro que permita eficiente acesso ao tweet indicado por uma palavra no arquivo, já que procurá-la diretamente no tweet do arquivo principal seria uma tarefa muito custosa, dado o número de tweets que possam ser inseridos e seu tamanhos. O dicionário chave/posting, formado ao construir o dicionário de palavras na árvore Trie, permite acesso direto à posição desses tweets, o que facilita a busca por eles ao não precisar percorrer todas as palavras dos tweets novamente.

Durante a inserção de uma palavra na Trie, é também feita a inserção de um índice em ordem crescente (sendo 0, o valor da primeira palavra, 1 o da segunda, seguindo adiante), que servirá para representar a palavra na lista de postings, e no dicionário chave/postings é adicionado um posting que indicará a linha ocupada pelo tweet contendo a palavra em questão no arquivo. Com essas informações, é feito um arquivo a partir do dicionário chave/postings,

contendo o índice da palavra e as linhas dos tweets que a contém em cada linha.

A função ‘funcionalidadeA’, em dicio.py, procura pela palavra pedida (de 3 ou mais letras após simplificação) na Trie. Se ela existir, recolhe seus postings, abre o arquivo de tweets original e percorre as linhas do tweet até encontrar a linha indicada pelo posting. Nesse momento, o tweet é impresso na tela e salvo num arquivo cujo nome contém a palavra pedida. O algoritmo também quebra o loop caso exista apenas 1 posting para a palavra, evitando ir até o fim do arquivo sem necessidade. Se a palavra pedida não existir, informa o usuário.

Sobre a atualização, é importante mencionar que, cada vez que um arquivo de tweets novo é adicionado aos tweets antigos, os tweets novos são escritos no fim do arquivo original. Nesse processo, um índice é criado para a palavra nova no dicionário de palavras e o dicionário chave/postings é atualizado, adicionando uma nova chave para as palavras que surgirem nos tweets novos e um posting para apontar o tweet em que ela se encontra. Realiza-se a criação de mais um arquivo - referente a palavra que o usuário deseja procurar - e armazena-se nele todos os tweets referentes àquela palavra, tendo assim, com o passar das atualizações, vários arquivos de tweets referentes a palavras. Com isso, a busca por mais palavras não é afetada pela inserção de novos tweets.

B) Funcionalidade B

A partir da função ‘funcionalidadeB’, em ‘dicio.py’, a funcionalidade B utiliza a árvore Trie para encontrar as palavras que iniciam com o prefixo dado (de 2 ou mais letras). A árvore é percorrida até encontrar (ou não, nesse caso informa o usuário) o último caracter do prefixo - isso é feito pela função ‘getByPrefix’. Se esse prefixo possuir filhos (ou seja, há palavras com esse prefixo), cria-se um arquivo para salvar todas as palavras (com suas informações). Com isso, a função ‘dataPrefix’ é chamada, e ela recursivamente encontra, imprime na tela e escreve no arquivo a palavra com o prefixo.

A árvore Trie é utilizada por ser eficiente na busca de palavras com determinada condição - no caso, a busca de palavras com dado prefixo. Após

passar pelas letras do prefixo, apenas os nodos filhos da última letra do prefixo são percorridos, evitando assim a consulta de elementos desnecessários e encontrando objetivamente as palavras adequadas. Por essa razão a Trie foi usada nesta funcionalidade.

Em relação à atualização, à medida que palavras novas são acrescentadas à árvore, elas podem conter prefixos que outras palavras não possuíam. Dado tal prefixo, a execução encontra essas novas palavras e as armazena em um novo arquivo.