UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

IMD0040 - Linguagem de Programação II - 2019.1
Especificação do Trabalho Final

Proposta:

Sistema para detecção de *Fake News* através da análise de similaridade entre conjuntos de palavras.

1. Introdução

Fake News representam um problema atual e muito sério. Há questões de escala continental e até mundial sendo discutidas sob a ótica da distorção promovida por notícias falsas espalhadas em redes sociais. Fatos que antes demoravam meses para se propagar, hoje podem ser propagados em questão de horas, ou até minutos. Se isso é feito de forma tendenciosa ou mesmo fraudulenta, há respaldo para afirmar que os prejuízos econômicos, sociais e humanos são incontáveis.

Neste contexto, este projeto consiste na implementação de um sistema orientado a objetos que permita a detecção de *Fake News* através da análise de similaridade entre conjuntos de palavras. O sistema a ser desenvolvido deverá ser capaz de ler *Fake News* catalogadas em um *Dataset* ou buscá-las na Internet (*online data scraping*).

2. Objetivos Específicos

O projeto apresenta os seguinte objetivos específicos que deverão ser contemplados pelo seu sistema (programa):

1. Ler os dados (*fake news)* do *dataset* disponibilizado;
2. Pré-processar as notícias visando a formatação ideal para posterior análise;
3. Armazená-las em uma estrutura que facilita a busca;
4. Coletar (*web scraping*) notícias da Internet;
5. Escolher um algoritmo de similaridade de *strings* dentre os sugeridos;
6. Aplicar um algoritmo de similaridade para comparar uma nova notícia com as notícias (*fake news*) armazenadas.

3. Metodologia

A partir do dataset fornecido “boatos-de-whatsapp-boatosorg.zip” (disponibilizado no Sigaa no momento da abertura da atividade), recupere cada linha (registro) do mesmo,

formatando-a para posterior armazenamento e análise. Para tal, siga os passos descritos abaixo. Após a linha de cabeçalho, há dados organizados para *Fake News* com os campos: id; conteudo; link; e timestamp.

Assuma que a primeira linha (após o cabeçalho) contém o seguinte conteúdo no campo "conteúdo":

> "Se você enviar para apenas 20 contatos em um minuto... o Brasil inteiro vai desmascarar este Bandido. NÃO quebre essa corrente. Os incautos precisam ser esclarecidos antes que seja tarde demais...". "Essa é a jornalista Patrícia Campos Mello, que fez matéria contra Bolsonaro na Folha. Petista de carteirinha!!"

Dessa forma, será necessário implementar uma rotina de pré-processamento visando remover palavras com 3 caracteres ou menos. Contudo, esse parâmetro (quantidade) deve ser configurável no sistema.

> você enviar para apenas contatos minuto Brasil inteiro desmascarar este Bandido quebre essa corrente incautos precisam esclarecidos antes seja tarde demais Essa jornalista Patrícia Campos Mello matéria contra Bolsonaro Folha Petista carteirinha

Além disso, é importante deixar todas as palavras em minúsculo (*lower case*) e remover todos os acentos encontrados no texto.

> voce enviar para apenas contatos minuto brasil inteiro desmascarar este bandido quebre essa corrente incautos precisam esclarecidos antes seja tarde demais essa jornalista patricia campos mello materia contra bolsonaro folha petista carteirinha

Para facilitar a comparação entre *Strings*, toda e qualquer palavra repetida deverá ser retirada do texto processado.

> voce enviar para apenas contatos minuto brasil inteiro desmascarar este bandido quebre essa corrente incautos precisam esclarecidos antes seja tarde demais jornalista patricia campos mello materia contra bolsonaro folha petista carteirinha

Na sequência, será necessário ordenar todas as palavras do texto, ficando o exemplo da seguinte forma:

> antes apenas bandido bolsonaro brasil campos carteirinha contatos contra corrente demais desmascarar enviar esclarecidos essa este folha incautos inteiro jornalista materia mello minuto para patricia petista precisam quebre seja tarde voce

Por último, aplicar uma função hash (*hash* criptográfica), que para esse projeto poderia ser a função SHA-1 (Wang et al., 2005). Essa função foi projetada pela Agência de Segurança Nacional dos Estados Unidos e é um Padrão Federal de Processamento de Informação dos Estados Unidos (NIST, 2019). A função SHA-1 produz um valor de dispersão de 160 bits (20

bytes) conhecido como resumo da mensagem. Um valor de dispersão SHA-1 é normalmente tratado como um número hexadecimal de 40 dígitos.

Dessa forma, aplicando essa referida função sobre o texto abaixo:

> *TEXTO:* antes apenas bandido bolsonaro brasil campos carteirinha contatos contra corrente demais desmascarar enviar esclarecidos essa este folha incautos inteiro jornalista materia mello minuto para patricia petista precisam quebre seja tarde voce

Teríamos o seguinte resultado:

> *HASH SHA-1:* b3ef530cf955bca20bae74e1aa638a0414819f8a

3.1 Armazenamento

É importante armazenar todos os textos processados do dataset em uma estrutura de dados que facilite a busca e posterior análise. Além dos textos processados, os originais, a data que foi publicado a fake news e a URL deverão também ser armazenados. Como sugestão, um HashMap poderia ser utilizado, tendo a chave sendo gerada pela função hash SHA-1 e o valor sendo um objeto contendo os outros itens (texto original, texto processado, data e URL).

3.2 Análise de Similaridade

Após armazenar adequadamente os textos (fake news) disponibilizados no dataset em uma estrutura de dados, o sistema de detecção deve ser capaz de coletar (*web scraping*) notícias na Internet, processar essas notícias coletadas, e ao final, analisar (comparar) a similaridade delas com todas as outras já armazenadas previamente. Observe que o processamento das novas notícias segue os mesmos passos descritos no item 3.

Na sequência, o sistema deve utilizar a mesma função hash (SHA-1) para gerar um hash sobre a notícia processada. De posse desse novo hash (não armazenado anteriormente), o sistema deve buscar/procurar na estrutura de armazenamento se um valor igual de hash já tenha sido guardado. Caso já tenha sido armazenado, então a notícia que foi coletada é de fato uma fake news.

Caso não haja valor de hash igual (busca sem sucesso), o sistema deve aplicar um algoritmo de similaridade sobre a notícia coletada, assim como em todas as notícias já armazenadas, para que, dessa forma se consiga fazer uma análise comparativa entre a nova notícias e todas as outras.

4. Algoritmos de Similaridade entre Strings

Há vários algoritmos capazes de comparar strings propostos na literatura. Contudo, vamos apresentar um grupo restrito formado por apenas 4 (quatro). Os algoritmos são os seguintes: Cosine, Levenshtein, Trigram e Jaro-Winkler.

- **Algoritmo de similaridade Cosine (Singhal, A., 2001; Sidorov, G. et al., 2014; Perone, C. S., 2019)**: é uma métrica usada para medir a similaridade entre textos (notícias), independentemente de seu tamanho. Matematicamente, mede o cosseno do ângulo entre dois vetores projetados em um espaço multidimensional. A semelhança de cosseno é vantajosa porque, mesmo que os dois documentos semelhantes estejam distantes da distância euclidiana (devido ao tamanho do documento), é provável que ainda possam estar mais próximos uns dos outros. Quanto menor o ângulo, maior a similaridade do cosseno.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

- **Distância Levenshtein (Navarro, G., 2001; Wagner, Robert A. and Fischer, Michael J., 1974)**: em teoria da informação, a distância Levenshtein ou distância de edição entre duas strings (duas sequências de caracteres) é dada pelo número mínimo de operações necessárias para transformar uma string na outra. Entendemos por "operações" a inserção, deleção ou substituição de um carácter. O nome advém do cientista russo Vladimir Levenshtein, que considerou esta distância já em 1965. É muito útil para aplicações que precisam determinar quão semelhantes duas strings são.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

- **Algoritmo Trigram (Dunning, T., 1994)**: no campo da linguística computacional e da probabilidade, um *n*-grama é uma sequência contígua de *n* itens de uma dada amostra de texto ou fala. Um algoritmo de Trigrama é um caso de *n*-grama, uma sequência contígua de *n* (três, neste caso) itens de uma dada amostra.

- **Algoritmo Jaro-Winkler (Jaro, M. A., 1995; Winkler, W. E., 1990)**: a métrica de Jaro-Winkler é uma medida de similaridade entre duas *strings*. É uma variação da métrica Jaro distance d_j, que estabelece a seguinte equação para duas strings S_1 e S_2:

$$d_j = \frac{m}{3a} + \frac{m}{3b} + \frac{m-t}{3m}$$

onde:

- m é o número de correlações entre caracteres;
- a e b são os tamanhos de s_1 e s_2, respectivamente;
- t é o número de transposições.

4.1 Análise de Similaridade

É importante ressaltar que, o sistema aqui especificado, deverá ter uma rotina de análise que faça uso de um ou mais algoritmos descritos acima, e que seja capaz de medir a similaridade entre uma notícia coletada/buscada (*web scraping*) da Internet e todas as outras retiradas do *Fake News* dataset (disponibilizado).

Além de usar os algoritmos como base para calcular a similaridade, será necessário informar qual o percentual (%) de similaridade entre as notícias, facilitando, dessa forma, a identificação de *fake news*. Contudo, um limiar de confiança (*threshold*) deve ser estabelecido (85% por exemplo). Podendo também ser alterado através de um componente gráfico que aumente ou diminua tal limiar (JSlider -> Swing).

5. Modelagem e Implementação

Seu sistema deverá ser modelado de acordo com as melhores práticas de orientação a objetos, e ser implementado com a linguagem de programação Java. Lembre-se de agrupar classes com funcionalidades semelhantes em pacotes. Além disso, será **obrigatório** o uso de:

1. Encapsulamento
   - Utilização de padronização de acesso e modificação com gets e sets
2. Modularização e padronização
   - Nomeação de classes e métodos segundo padrões;
   - Separação das classes em pacotes
   - Design de Classes (responsabilidade e coesão)
3. Herança
4. Polimorfismo
5. Classes abstratas
6. Interfaces
7. Padrões de projeto
   - Utilização de algum(ns) padrão(ões) de projeto de forma justificada
8. Tratamento de exceções
9. Estrutura de dados simples (qualquer coleção)
10. Interface Gráfica
    - Utilização de JavaFx para interface Gráfica

Seu programa também deverá incluir uma interface gráfica que ofereça ao usuário pelo menos as seguintes funcionalidades:

- Abrir o arquivo de fake news;
- Processar as notícias;
- Fazer o armazenamento e busca das notícias;
- Fazer a busca de uma nova notícia da web;
- Analisar novas notícias com as armazenadas.

6. Entrega e Avaliação

Seu grupo deverá submeter, via sigaa, um arquivo compactado contendo:

- Código fonte do sistema desenvolvido, incluindo um documento README.TXT contendo instruções de como se pode compilar o código fonte.

- Relatório Técnico, contendo pelo menos as seguintes seções:

    - Introdução: contendo uma breve descrição do problema abordado e do que será apresentado em seu relato;

    - Descrição da abordagem de solução do problema: Descrevendo o projeto OO (diagramas de classes), e destacando as decisões de projetos tomada, bem como identificando padrões de projeto aplicados;

    - Descrição geral das estruturas de dados utilizadas com explicação dos algoritmos utilizados;

    - Conclusão;

    - Referências;

O relatório deverá ser feito seguindo o template da Sociedade Brasileira de Computação (SBC) que pode ser encontrado no seguinte endereço:

> http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos

6.1 Apresentação

Cada dupla terá 20 minutos para fazer sua apresentação. Nesta apresentação, os grupos deverão demonstrar o funcionamento do programa desenvolvido. Além disso, deverão estar aptos a responder questões sobre o desenvolvimento do projeto.

Importante: não será dada uma única nota ao grupo. Cada componente do grupo receberá uma nota de acordo com seu desempenho durante a apresentação.

O programa será avaliado como um todo, ou seja, os requisitos não receberão pontuação individualmente. Dessa forma, a falta de um ou mais requisitos acarretará na perda de pontos, que poderá ser compensada (não totalmente, claro) através de outros componentes bem desenvolvidos. Componentes adicionais serão muito bem vistos, desde que implementados de maneira racional.

Referências

1. Xiaoyun Wang, Yiqun Lisa Yin and Hongbo Yu. **Finding Collisions in the Full SHA-1**, Crypto 2005.
2. NIST - **Computer Security Division - Computer Security Resource Center**. https://csrc.nist.gov/projects/hash-functions, 2019.
3. Singhal, Amit. **Modern Information Retrieval: A Brief Overview**. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43, 2001.
4. Sidorov, G.; Gelbukh, A.; Gómez-Adorno, H.; and Pinto, D. **Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model**. Computación y Sistemas. 18 (3): 491–504. doi:10.13053/CyS-18-3-2043, 2014.
5. Christian S. Perone. **Machine Learning: Cosine Similarity for Vector Space Models**. http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/, acessado em 07/05/2019.
6. Navarro, Gonzalo. **A guided tour to approximate string matching**. ACM Computing Surveys. 33 (1): 31–88, 2001.
7. Wagner, Robert A.; Fischer, Michael J. The String-to-String Correction Problem. Journal of the ACM, 21 (1): 168–173, 1974.
8. Ted Dunning. **Statistical Identification of Language**. Technical Report MCCS. New Mexico State University: 94–273, 1994.
9. Jaro, M. A. **Probabilistic linkage of large public health data file**. Statistics in Medicine. 14 (5–7): 491–8, 1995.
10. Winkler, W. E. **String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage**. Proceedings of the Section on Survey Research Methods. American Statistical Association: 354–359, 1990.

ANEXO

Tabela de pontuação para correção:

Codificação	Modularização e padronização	0,5
	Herança e Polimorfismo	0,5
	Classes Abstratas ou Interfaces	0,5
	Tratamento de Exceções	0,5
	Padrões de projeto	0,5
	Estrutura de dados (hashMap)	0,5
	Interface Gráfica	1,0
Apresentação	Relatório	1,0
	Apresentação	1,0
Objetivos	Processamento das notícias	1,0
	Web scraping	1,0
	Análise de similaridade	2,0
	Plus (ponto extra)	
Total de pontos		10,0