UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Leandro Nascimento Adegas

1. Introdução

O dataset utilizado neste trabalho é composto por dois arquivos CSV distintos, News_fake.csv e News_notFake.csv, cada um com 300 linhas e 4 colunas (id, news_url, title e tweet_ids), totalizando 600 registros.

No primeiro arquivo estão as notícias rotuladas como falsas e, no segundo, as notícias verdadeiras, e no código essas classes são mapeadas para um rótulo binário simples (label = 1 para fake e label = 0 para not fake), resultando em um conjunto de dados perfeitamente balanceado entre as duas classes. A coluna **id** traz um identificador numérico de cada notícia, **news_url** registra o link da checagem ou da notícia, **title** contém o título em texto livre (que é o principal campo usado como entrada nos modelos e classificações) e **tweet_ids** lista os identificadores dos tweets em que aquele conteúdo circulou.

Uma análise descritiva dos títulos mostra que, entre as 300 notícias falsas, o comprimento médio dos títulos é de aproximadamente 143 caracteres e 24 palavras, enquanto nas 300 notícias verdadeiras os títulos são, em média, um pouco mais curtos, com cerca de 109 caracteres e 19 palavras por título, sugerindo que fake news tendem a usar manchetes mais longas e potencialmente mais chamativas.

Neste projeto, portanto, o que se analisa é como, a partir desses títulos rotulados (texto + label binária), diferentes estratégias de pré-processamento (limpeza, remoção de stopwords, stemming, lematização) e diferentes formas de representação textual (atributos estatísticos, Bag-of-Words, TF-IDF, bigramas e Word2Vec) influenciam o desempenho de um classificador supervisionado na tarefa de distinguir automaticamente entre notícias falsas e verdadeiras.

O dataset utilizado é composto por dois arquivos CSV:

- News_fake.csv – contém títulos de notícias classificadas como falsas;
- News_notFake.csv – contém títulos de notícias classificadas como verdadeiras.

Ambos os arquivos possuem a mesma estrutura de colunas:

- id ⟶ identificador numérico da notícia;

- news_url → URL original da notícia;
- title → título da notícia (texto principal utilizado neste trabalho);
- tweet_ids → identificadores de tweets relacionados àquela notícia.

No código, foi feita a seguinte convenção:

- Todas as instâncias de News_fake.csv recebem a label 1;
- Todas as instâncias de News_notFake.csv recebem a label 0.

Assim, após a leitura e concatenação dos dois arquivos, obtemos um dataset único com:

- 300 notícias falsas;
- 300 notícias verdadeiras;

Total de 600 registros, formando um dataset balanceado (mesmo número de exemplos nas duas classes).

Geração de atributos:

Chama internamente os métodos corretos:

"count" → Bag-of-Words

"tfidf" → TF-IDF

"cooc" → bigramas

"word2vec" → embeddings médios

"stats" → atributos simples

2. Observações

Durante a implementação e execução dos códigos alguns problemas foram evidentes como a necessidade de instalação de bibliotecas como a do nltk, gensim e spacy que, embora já tenha instalado algumas, não consegui executar o código porque estava solicitando funções específicas que não instalou usando o pip install, tendo que baixar manualmente todo o pacote nltk no computador.

3. Resultados e discussão

> Pré-processamento:

Na **Questão 1**, o foco foi a construção do módulo de pré-processamento de texto, implementado no arquivo preprocessing.py. O **objetivo** dessa etapa é transformar os títulos brutos, cheios de ruídos típicos de dados reais (links, menções, pontuação, letras maiúsculas, etc.), em textos mais limpos e padronizados, facilitando o trabalho dos métodos de extração de atributos. O fluxograma dessa parte pode ser descrito da seguinte forma: cada título entra primeiro na função limpar_texto, que converte tudo para minúsculas e remove URLs, menções do tipo @usuario, hashtags, números e pontuação, além de normalizar espaços em branco. Em seguida, o texto passa pela função tokenizar, que quebra a frase em uma lista de palavras. Sobre essa lista de tokens, podem ser aplicadas três operações opcionais: remoção de stopwords (função remover_stopwords), stemming (aplicar_stemming) ou lematização (aplicar_lematizacao).

A função preprocessar_texto coordena todas essas etapas, recebendo o texto original e devolvendo uma string pré-processada, pronta para ser utilizada nos vetorizadores da Questão 2.

Por fim, a função preprocessar_corpus aplica esse fluxo a todos os títulos do conjunto de dados. Na prática, ao comparar visualmente alguns títulos antes e depois do pré-processamento, é possível observar que os textos se tornam mais curtos, com menos símbolos e mais focados em palavras de conteúdo, o que reduz a dimensionalidade e tende a beneficiar as representações baseadas em palavras.

Além disso, funções auxiliares como tem_mencao e contar_stopwords_texto foram fundamentais para gerar atributos estatísticos simples que serão explorados posteriormente, enriquecendo a representação dos documentos.

> Extração de atributos:

Na **Questão 2**, foi desenvolvido o módulo de extração de atributos, presente no arquivo atribute_extraction.py, por meio da classe ExtratorAtributos. Esta classe organiza diferentes estratégias de transformar texto em vetores numéricos, permitindo ligar ou desligar o pré-processamento de forma centralizada. O fluxograma conceitual da Questão 2 é: textos (brutos ou já pré-processados) → método interno preparar_textos (que decide se chama ou não o preprocessamento.py) → método específico de extração de atributos → matriz de atributos que é passada ao classificador. A implementação contempla cinco formas principais de representação: (i) atributos estatísticos simples, (ii) Bag-of-Words, (iii) TF-IDF, (iv) bigramas (coocorrência) e (v) Word2Vec.

No caso dos atributos estatísticos, a função atributos_estatisticos_simples calcula, para cada título, o tamanho em caracteres, o número total de palavras, o número de palavras distintas, um indicador binário de presença de menções @ e a quantidade de stopwords, formando um vetor de cinco dimensões por documento. Esses atributos capturam aspectos de "estilo" dos títulos, como serem mais longos, repetitivos ou cheios de palavras vazias, o que pode diferenciar fake news de notícias reais.

As representações Bag-of-Words e TF-IDF são implementadas via CountVectorizer e TfidfVectorizer, respectivamente: na fase de treino, os métodos ajustar_bow e ajustar_tfidf aprendem o vocabulário a partir dos textos e produzem matrizes esparsas de contagens ou pesos TF-IDF; na fase de teste, transformar_bow e transformar_tfidf aplicam esse vocabulário para gerar atributos consistentes.

A matriz de coocorrência é implementada por meio de bigramas (ngram_range=(2,2)), capturando pares de palavras frequentes, como "fake news" ou "última hora", que podem ser fortes indícios do tipo de notícia.

Por fim, a representação Word2Vec é construída treinando-se um modelo de embeddings no próprio corpus, com o método ajustar_word2vec, e, em seguida, calculando a média dos vetores das palavras de cada título, produzindo representações densas de dimensão fixa.

Essa diversidade de representações permite comparar abordagens baseadas em contagem, importância de termos, coocorrência local e semântica distribuída, como discutido na Questão 3.

> Classificação:

Na **Questão 3**, o arquivo classification.py integra todas as etapas anteriores para treinar e avaliar um classificador de fake news. Inicialmente, a função carregar_dataset_fakenews lê os arquivos News_fake.csv e News_notFake.csv, atribuindo rótulo 1 às notícias falsas e 0 às verdadeiras, e concatenando tudo em um único DataFrame. Em seguida, utiliza-se a coluna title como texto de entrada e a coluna label como alvo da classificação. O conjunto é então dividido em treino (80%) e teste (20%) por meio da função train_test_split, com stratify=labels para manter a proporção de fake/not fake em ambos os conjuntos. O classificador escolhido foi uma Regressão Logística (LogisticRegression), aplicada pela função treinar_avaliar, que recebe X_treino, X_teste, y_treino e y_teste, treina o modelo e retorna a acurácia no conjunto

de teste. A função gerar_atributos conecta o ExtratorAtributos com o classificador, chamando internamente o método de extração adequado para cada tipo de atributo (ajustar_bow, ajustar_tfidf, ajustar_coocorrencia, ajustar_word2vec, atributos_estatisticos_simples, e seus correspondentes de transformação).

Os resultados numéricos de cada experimento foram organizados em DataFrames (df_a, df_b, df_c) e visualizados por meio de gráficos de barras, o que facilita a comparação e a discussão dos efeitos de pré-processamento e representação sobre o desempenho.

De forma geral, os resultados obtidos nas três questões mostram, em conjunto, a importância de tratar o texto antes da classificação, de escolher uma boa forma de representá-lo numericamente e de analisar empiricamente essas escolhas, em vez de assumir que uma técnica será sempre melhor. As tabelas de resultados (df_a, df_b, df_c) e os gráficos de barras gerados pelo código formam a base quantitativa desta discussão, permitindo justificar, com números e visualizações, qual combinação de pré-processamento e extração de atributos mais contribuiu para a detecção automática de fake news a partir dos títulos do dataset utilizado.

4. Conclusões

De forma satisfatória, o resultado esperado foi alcançado mostrando com detalhes a comparação dos resultados que foram contra os que não foram tratados. O pré-processamento de texto tende a melhorar o desempenho em relação ao uso de textos brutos, principalmente para representações como BoW (Bag-of-Words) e TF-IDF.

Entre as formas de extração de atributos, o TF-IDF e o Bag-of-Words apresentaram os melhores resultados, provavelmente por equilibrar frequência local e global das palavras.

Na comparação entre a lematização e stemming, a lematização se mostrou ligeiramente superior, indicando que há uma preservação semântica melhor das palavras e uma redução de dimensionalidade.

5. Próximos passos

Algumas melhorias poderiam ser feitas, tanto no dataset quanto nos classificadores:

- Usar o texto completo da notícia, não apenas o título, como entrada para os modelos, o que provavelmente aumentaria muito a quantidade de informação disponível;
- Experimentar outros classificadores, por exemplo:
    - SVM (Máquinas de Vetores de Suporte)
    - Random Forest
    - Redes neurais simples (MLP) ou modelos pré-treinados como BERT em português
- Aprimorar o pré-processamento:
    - Corrigir emojis, risadas, gírias e abreviações
    - Tratar melhor negações
- Explorar embeddings pré-treinados:
    - Usar modelos Word2Vec ou FastText trainados em grandes corpora em português
    - Usar BERTimbau ou outros modelos Transformer para representar frases
- Analisar interpretabilidade:
    - Ver quais palavras mais contribuem para o modelo classificar uma notícia como fake
    - Identificar padrões de linguagem típicos de notícias falsas vs verdadeiras