Detectando sentimentos em uma frase utilizando machine learning clássico

Grupo branco

May 23, 2022

1 Introdução

Para nós, humanos, identificar ironia ou humor em uma frase, seja ela escrita ou falada, é uma tarefa relativamente simples, basta entender o contexto e a entonação em que ela se encontra, entretanto, como explicar para uma máquina o que é ironia ou sarcasmo?

O presente documento procura encontrar, através de algoritmos de inteligência artificial, padrões que possam classificar uma frase entre "contém humor" ou "não contém humor". Esse projeto é baseado em um desafio que pode ser encontrado no kaggle, onde diversas pessoas tentam resolver o problema com redes neurais, a maioria com sucesso, mas para fins de estudo, decidimos trabalhar somente com machine learning clássico, apesar de algumas limitações, conseguimos tratar do problema de forma clara.

2 Metodologia

Para uma melhor organização da solução, dividimos o problema em algumas etapas:

- Obtenção dos dados
- Tratamento nos dados brutos
- Escolha de um ou mais modelos
- Criação de um pipeline
- Treino do(s) modelo(s)
- Análise dos resultados

2.1 Obtenção dos dados

Os dados utilizados foram baseados em um desafio do kaggle e podem ser obtidos através desse link. O csv original contém 200 mil frases.

2.2 Tratamento do dataset

Utilizamos a biblioteca pandas para ler o arquivo csv e dropar os dados nulos, assim, ficamos mais livres para realizar operações na tabela sem muitas dificuldades.

O próximo passo foi criar uma função que tratasse cada string presente no dataset, removendo tudo que não agregasse no modelo final, várias técnicas foram utilizadas.

2.2.1 Substituição das abreviações

O dataset bruto possuí diversas palavras abreviadas que inviesariam o modelo, são alguns exemplo: "ain't", "can't've", "could've", "how'd" entre outros, todas foram substituidas pela forma correta de acordo com a lingua inglesa.

2.2.2 Remoção das pontuações

Pontuação não é algo que agrega na hora da criação do modelo, mesmo que pareça estranho afirmar isso a princípio, é verdade, existem várias outras formas de encontrar significado em uma frase sem o uso de pontuações para enfatizar.

2.2.3 Work tokenize

A tokenização serve para dividir uma string em partes menores, O retorno da função é uma lista com cada palavra da frase passada por parâmetro, isso facilita futuras manipulações. A biblioteca utilizada foi a Natural Language Toolkit (NLTK), essa biblioteca é muito utilizada para tratamento de textos pela quantidade de alterações que ela nos permite fazer sem muito esforço.

2.2.4 Stop words

As stopwords são palavras irrelevantes para o modelo, tais podem ser removidas de diversas formas, a abordagem que escolhemos foi utilizando a biblioteca Natural Language Toolkit (NLTK) , você pode ver alguns exemplos de stop words acessando esse repositório.

2.2.5 Stemming

Para realizar o Stemming, a biblioteca Natural Language Toolkit (NLTK) foi utilizada, essa técnica consiste em extrair o radical de cada palavra, assim, o significado fica o mesmo independente do tempo verbal/conjugação

2.3 Escolha dos modelos

O objetivo do projeto é tentar encontrar humor em frases utilizando machine learning clássico, é possível chegar em resultados melhores utilizando redes neurais e aprendizado profundo, mas esse não é o intuíto.

A fins de teste, decidimos escolher alguns modelos e analisar qual performava melhor, dentre eles temos:

- **Multinomial Naive Bayes:** Ótimo para classificação de fetures discretas , bastante usado na comunidade para classificação de textos.
- **Random Forest:** Como o nome sugere, o random forest é uma floresta composta por árvores de decisão treinadas em subconjuntos do dataset, onde a média dos resultados é utilizado para melhorar a accuracy e evitar o overfitting.
- **Logistic Regression:** Apesar do nome causar estranhesa pelo fato de remeter a um modelo de regressão, a regressão logistica é um modelo muito utilizado para classificação de um modo geral.

2.4 Pipeline

O propósito do pipeline é juntar várias etapas em um só lugar para a criação de um estimador que pode ser treinado posteriormente, para isso, utilizamos a biblioteca scikit-learn, focada em aprendizado de máquina. O pipeline seguiu as seguintes etapas:

```
random_forest_pipe = Pipeline([
    ('CountVectorizer', CountVectorizer(analyzer=textProcess)),
    ('TDFID', TfidfTransformer()),
    ('RandomForest', RandomForestClassifier())
])

multinomialNB_pipe = Pipeline([
    ('CountVectorizer', CountVectorizer(analyzer=textProcess)),
    ('TDFID', TfidfTransformer()),
    ('RandomForest', MultinomialNB())
])

logistic_regression_pipe = Pipeline([
    ('CountVectorizer', CountVectorizer(analyzer=textProcess)),
    ('TDFID', TfidfTransformer()),
    ('RandomForest', LogisticRegression())
])
```

Figure 1: Criação do pipeline

2.4.1 CountVectorizer:

Apesar do problema girar em torno de frases, os modelos de aprendizado de máquina não conseguem interpretar textos, contudo, utilizamos o sklearn count vectorizer para criar uma matriz de contagem de token, essa matriz é esparsa e possui tantas dimensões quanto palavras exclusívas no texto.

2.4.2 TfidfTransformer:

o Tfidf é uma técnica utilizada para avaliar a importancia de uma palavra para um documento em uma coleção de texto, a importancia aumenta proporcionalmente ao número de vezes que uma palavra aparece no documento, mas é compensada pela frequência de palavra no texto.

2.5 Treinamento

Como mencionado anteriormente, o projeto visa resolver o problema utilizando aprendizado de máquina clássico, contudo, três modelos foram utilizados. O dataset foi dividido em dois conjuntos, onde 70% compõe o conjunto de treino e 30% para o conjunto de teste/validação, para isso, a biblioteca scikit-learn train test split foi utilizada.

2.6 Análise dos resultados

Para as métricas de avaliação, nós escolhemos a accuracy, f1 score e a curva AUC.

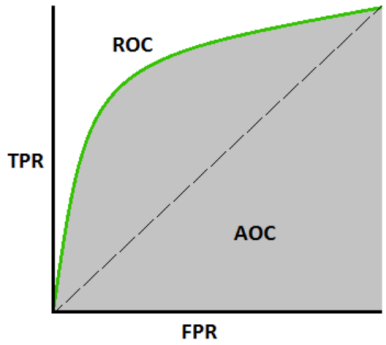
- **Accuracy** é a proporção de predições corretas entre o total de predições, conforme a fórmula a seguir.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **f1 score** é calculado a partir do precision e do recall, onde precision é a divisão de true positives pelo total de positivos, enquanto recall é a divisão de true positives pelo total numero total de exemplos, o f1-score é a média harmonica entre os dois valores.

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

- **AUC** basicamente mensura quão bem o modelo está conseguindo distinguir as classes ao passar do tempo de treinamento, quanto maior melhor.



Para uma melhor visualização, uma tabela foi criada com o resultado de cada modelo treinado, essas métricas foram extraídas do conjunto de teste criado pela melhor iteração de cada modelo criado pelo cross validation, confira:

	accuracy	f1
logistic	0.790477	0.780058
random_forest	0.806659	0.796214
multinomialNB	0.757076	0.743770

Figure 2: Tabela com resultados

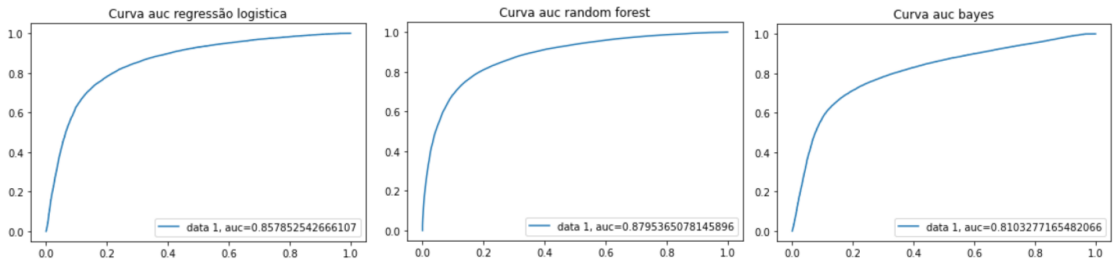


Figure 3: Curvas AUC dos modelos

3 Comparando resultados

Com machine learning clássico foi possível alcançar um resultado aceitável, mas ainda há muito o que fazer para aumentar essas métricas. Procurando o que seria o "estado da arte" para identificação de humor em sentenças, chegamos em duas redes neurais que se destacam e resolvem muito bem o problema, são elas [1] e [**?**].

Method	Configuration	Accuracy	Precision	Recall	F1
XLNET	large	0.916	0.872	0.973	0.920
COLBERT		0.982	0.990	0.974	0.982

Figure 4: XLNet e BERT metrics

Ambos os modelos foram treinados utilizando a mesma base de dados, o processamento foi muito parecido, a diferença é que a rede neural é mais robusta e lida melhor com problemas como esse.

4 Conclusão

Detectar sentimentos em uma frase não é uma tarefa tão fácil, além de escolher um modelo apropriado para o problema, é de extrema importancia quea base de dados seja bem tratada, neste caso, conseguimos resolver o problema com machine learning clássico utilizando uma técnica de aprendizado supervisionado. Após vários testes e modelos criados, o que obteve melhor resultado foi o Random Forest Classifier, com 0.80 de acuracia e 0.79 f1 score.

References

[1] Issa Annamoradnejad and Gohar Zoghi. Colbert: Using bert sentence embedding for humor detection, 2021.