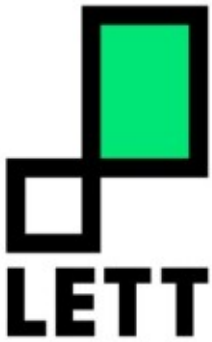
LETT
Estágio em Ciência de Dados
Teste Técnico
Simpsons Dataset

Cleiton Kennedy de Morais Filho



Uberaba
2022

Cleiton Kennedy de Morais Filho

LETT
Estágio em Ciência de Dados
Teste Técnico
Simpsons Dataset

Lista de ilustrações

Figura 1 – Supervised Machine Learning Flow (RASCHKA, 2015). 10
Figura 2 – Formato dos dados. 14
Figura 3 – Dados separados por nome dos personagens (dados de treinamento). . . 14
Figura 4 – Dados separados por nome dos personagens (dados de validação). . . . 14
Figura 5 – Verificação de valores nulos. 15
Figura 6 – Amostragem das **Features**. 15
Figura 7 – Amostragem dos **Labels**. 15
Figura 8 – **Features** após limpeza. 16
Figura 9 – Comparação entre diferentes formas de vetorização. 18
Figura 10 – Comparação entre diferentes formas de implementar o modelo Naive Bayes. 20

Lista de códigos

Código 2.1 – Load do dataset e amostragem dos dados. 13
Código 2.2 – Verificação de valores nulos e análise de amostra. 15
Código 2.3 – Remoção dos nomes da coluna de features. 16
Código 2.4 – Teste de diferentes métodos de vetorização. 17
Código 2.5 – Naive Bayes Multinomial. 19
Código 2.6 – Naive Bayes Bernoulli. 19
Código 2.7 – Naive Bayes Complement 19
Código 2.8 – Naive Bayes Gaussian. 19

Sumário

1 Introdução 9
1.1 Objetivos 9
1.2 Conceitos 9
1.2.1 Processamento Natural de Linguagem 9
1.2.2 Machine Learning Pipeline 10
1.2.3 Abordagens 10
2 Desenvolvimento 13
2.1 Dados 13
2.1.1 Data Loading 13
2.1.2 Data Cleaning 15
2.2 Vetores 16
2.2.1 CountVectorizer 16
2.2.2 TF-IDF 16
2.2.3 Comparativo 17
2.3 Modelos 18
2.3.1 Multinomial 19
2.3.2 Bernoulli 19
2.3.3 Complement 19
2.3.4 Gaussian 19
2.3.5 Comparativo 20
2.4 Análise dos Resultados 20
2.4.1 Hiperparâmetros 20
2.4.2 Abordagens alternativas 20
2.5 Problemas Encontrados 21
3 Conclusão 23
Referências 25

Capítulo 1

Introdução

1.1 Objetivos

O objetivo deste trabalho é implementar um algoritmo de Processamento de Linguagem Natural que será capaz de classificar uma frase qualquer e associá-la à um personagem dos Simpsons. Para isto, será criado um modelo de aprendizagem de máquina que será treinado (aprendizagem supervisionada) em um dataset de referência para em seguida ser validado.

1.2 Conceitos

Para obter uma compreensão mais precisa do escopo do projeto, alguns conceitos serão definidos.

1.2.1 Processamento Natural de Linguagem

Processamento Natural de Linguagem consiste na adoção de ferramentas tecnológicas e métodos computacionais para analisar e representar textos em diversos níveis de abstração para se obter um processamento de linguagem próximo ao nível humano (LIDDY, 2001). Este conceito é amplamente utilizado no desenvolvimento de soluções que pretendem uma interação automatizada com usuários, por exemplo, no reconhecimento de voz, na tradução de textos, no desenvolvimento de chatbots, etc.

Usualmente o desenvolvimento de um algoritmo de processamento natural de linguagem contém alguns passos, considerando-se um dataset limpo:

- Tokenização: O texto é separado em palavras ou blocos de palavras (n-grams). Desta forma, cada item poderá ser analisado de forma independente, assumindo seu devido papel nos cálculos aplicados.

- ❏ Vetorização: O texto, já tokenizado, é então convertido em valores numéricos, tendo assim uma representação mensurável de cada token.
- ❏ Modelagem: Um modelo de aprendizagem será implementado de acordo com a abordagem escolhida.

Estes são itens específicos no escopo do processamento natural de linguagem, em outro nível de abstração estes itens estarão inseridos dentro de um fluxo de aprendizagem, conforme mostrado na seção seguinte.

1.2.2 Machine Learning Pipeline

O desenvolvimento de uma solução utilizando aprendizagem de máquina segue uma sequência pré definida conforme Figura 1.

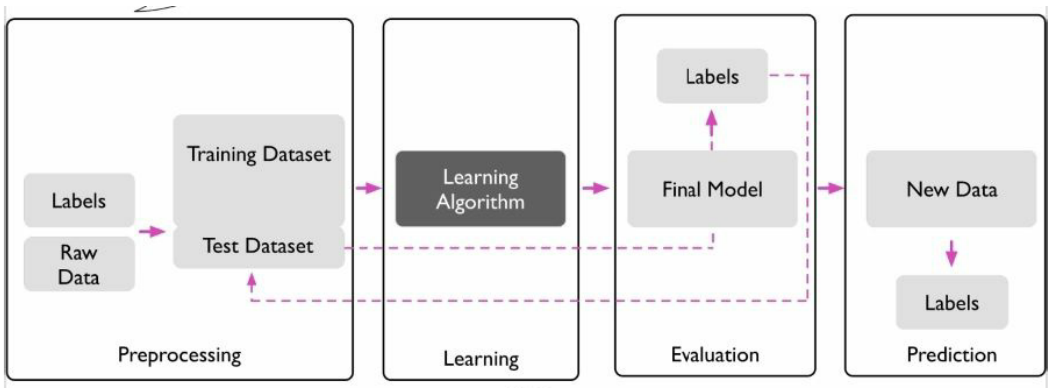


Figura 1 – Supervised Machine Learning Flow (RASCHKA, 2015).

O dataset é separado em **Features** e **Labels**, as Features são parâmetros de uma instância do banco de dados que serão utilizados para se inferir um padrão de comportamento do sistema. Os Labels indicam o parâmetro de saída desejado e que será utilizado posteriormente para mensurar a qualidade das previsões.

Então, no fluxograma do processo de desenvolvimento, o primeiro passo é realizar o treinamento do modelo utilizando dados previamente limpos e adequados ao método pretendido. Este treinamento irá utilizar as Features e Labels previamente associados, conseguindo assim um algoritmo que será capaz de receber novos dados e prever qual o Label adequado para aquele conjunto de novos parâmetros.

1.2.3 Abordagens

Para a escolha do modelo de aprendizagem ideal deve ser feita uma análise do escopo geral do trabalho, de forma que, seja possível identificar as especificações do projeto e escolher uma abordagem adequada.

Entre as abordagens mais conhecidas estão (LIDDY, 2001):

- Simbólica: Nesta abordagem a análise do texto é feita baseada em um conjunto de regras (sintáticas ou semânticas), previamente conhecido. O conhecimento é então manipulado de acordo com a lógica ou com as regras impostas. Exemplos: Árvores de decisão, K-nearest, clustering conceitual, etc.

- Estatística: Esta abordagem emprega métodos matemáticos quantitativos para representar e manipular o texto. A informação conhecida é generalizada e ao ser representada por números e padrões, pode ser utilizada em métodos estatísticos que irão inferir o seu valor representativo.

- Conectiva: Similar à abordagem estatística, esta abordagem também utiliza abstração dos dados para realizar operações matemáticas significativas. Entretanto, as informações são representadas em grupos que compartilham determinados aspectos e os cálculos são realizados baseados na relação (conexão) entre esses grupos.

Neste trabalho será utilizada a abordagem estatística, pois o objetivo é demonstrar como os dados devem ser tratados e quais métricas de eficiência podem ser utilizadas. Após a análise dos resultados, outras abordagens podem ser utilizadas de forma mais assertiva, com objetivos mais específicos.

Desta forma, o escopo do desenvolvimento será dividido em três etapas:

1. Data Loading/Cleaning: Análise dos dados e aplicação de modificações necessárias.
2. Tokenização/Vetorização: Abstração dos dados para representação matemática.
3. Modelagem: Aplicação de um modelo de aprendizagem para treinamento e classificação.
4. Análise dos resultados e propostas de otimização.

Capítulo 2

Desenvolvimento

2.1 Dados

O dataset utilizado contém falas de diversos personagens dos Simpsons e está dividido em dois arquivos, um contendo dados para treinamento e outro contendo dados para validação.

2.1.1 Data Loading

Primeiro, os dados são carregados e sua estrutura é analisada:

```
#Importação de bibliotecas
import pandas as pd

#Definição de visualização de output
pd.set_option('display.max_colwidth', 150)

#Carregando os arquivos
df_treino = pd.read_csv('simpsons_train.csv')
df_valida = pd.read_csv ('simpsons_test.csv')

#Analisando a estrutura dos dados
print(df_treino.columns)
print(df_valida.columns)
print(df_treino.shape)
print(df_valida.shape)
```

Código 2.1 – Load do dataset e amostragem dos dados.

Os seguintes resultados foram obtidos:

```
Index(['character_id', 'character_name', 'spoken_words'], dtype='object')
Index(['character_id', 'character_name', 'spoken_words'], dtype='object')
(50457, 3)
(25228, 3)
```

Figura 2 – Formato dos dados.

```
Homer Simpson          17013
Marge Simpson           8113
Bart Simpson            7817
Lisa Simpson            6554
C. Montgomery Burns     1924
Moe Szyslak             1787
Seymour Skinner         1522
Ned Flanders            1304
Chief Wiggum            1141
Grampa Simpson          1129
Krusty the Clown        1081
Milhouse Van Houten     1072
Name: character_name, dtype: int64
```

Figura 3 – Dados separados por nome dos personagens (dados de treinamento).

```
Homer Simpson          8506
Marge Simpson          4056
Bart Simpson           3909
Lisa Simpson           3278
C. Montgomery Burns     962
Moe Szyslak             894
Seymour Skinner         761
Ned Flanders            652
Chief Wiggum            570
Grampa Simpson          564
Krusty the Clown        541
Milhouse Van Houten     535
Name: character_name, dtype: int64
```

Figura 4 – Dados separados por nome dos personagens (dados de validação).

Desta forma, podemos ter uma noção geral do formato dos dados e de como poderá ser feita a abordagem escolhida. Como nosso dataframe possui apenas duas colunas, a coluna de *spoken_words* será utilizada como **Feature** e a coluna *character_name* será utilizada como **Label**.

2.1.2 Data Cleaning

Neste processo é analisada a qualidade dos dados armazenados e verificada a presença de inconsistências e valores nulos que possam prejudicar a implementação do modelo.

```
#Verificação de valores nulos:
print(df_treino.isnull().sum(axis = 0))
print(df_valida.isnull().sum(axis = 0))

#Amostragem dos dataframes:
print(df_treino.spoken_words.head())
print(df_treino.character_name.head())
print(df_valida.spoken_words.head())
print(df_valida.character_name.head())
```

Código 2.2 – Verificação de valores nulos e análise de amostra.

Os resultados obtidos estão ilustrados nas Figuras 5, 6 e 7.

```
character_id      0
character_name    0
spoken_words      0
dtype: int64
character_id      0
character_name    0
spoken_words      0
dtype: int64
```

Figura 5 – Verificação de valores nulos.

```
0                     Marge Simpson - Homie, it's eleven at night. Have you told him yet?
1    Lisa Simpson - Looks like we're back to traveling on tramp steamers and produce trucks.
2                                 Homer Simpson - Whew, I'd hate to be that kid's father.
3              Homer Simpson - Yes, that's a real pickle. Would you excuse me for a moment?
4                    Milhouse Van Houten - Hey Bart, I finished organizing the stock room.
Name: spoken_words, dtype: object
```

Figura 6 – Amostragem das **Features**.

```
0          Marge Simpson
1           Lisa Simpson
2          Homer Simpson
3          Homer Simpson
4    Milhouse Van Houten
Name: character_name, dtype: object
```

Figura 7 – Amostragem dos **Labels**.

Pode-se notar que não existem valores nulos, porém na coluna *spoken_words* os nomes dos personagens estão inseridos antes da frase em si e isso pode gerar inconsistência no treinamento do modelo. Portanto, será aplicado um método para remoção de tudo que

não consistir especificamente da frase dita pelo personagem. Em seguida o dataframe será separado em dois, sendo o primeiro constituído apenas pelas features e o segundo constituído pelos labels.

```
1 treino_x, treino_y = df_treino['spoken_words'].str.rsplit(pat='-').str
    [-1], df_treino['character_name']
2 valida_x, valida_y = df_valida['spoken_words'].str.rsplit(pat='-').str
    [-1], df_valida['character_name']
3 print(treino_x.sample(5))
```

Código 2.3 – Remoção dos nomes da coluna de features.

Assim, foi obtido o resultado amostrado na Figura 8.

```
47342                                                              Ha!
36003     I like you too, but you broke the law. So I've got to bring you in.
27410                                              You leave my wife alone!
13141                 Why you little... I'll teach you to trick your sister!
15565         Let's kick this up a notch and get you some wiggle in your lap.
Name: spoken_words, dtype: object
```

Figura 8 – **Features** após limpeza.

Agora os dados estão prontos para o processo de vetorização e consequente modelagem.

2.2 Vetores

O processo de transcrever texto em números representativos é chamado de engenharia de atributos (*feature engineering*) (KULKARNI; SHIVANANDA, 2019). Para isto, existem diversos métodos que podem ser empregados, neste trabalho serão implementados os métodos *CountVectorizer* e o método *TF-IDF*, em seguida os resultados serão comparados e analisados. Para a aplicação dos diferentes métodos será utilizado um modelo baseado no método Naive-Bayes Multinomial.

2.2.1 CountVectorizer

Este método consiste em transcrever as palavras do texto em categorias representativas e criar um vetor/matriz de valores constituídos pela contagem da presença daquela categoria no texto. Desta forma é possível obter uma representação baseada na frequência dos termos, resultando em soluções mais robustas.

2.2.2 TF-IDF

TF-IDF significa *Term Frequency-Inverse Document Frequency*, onde *Term Frequency* indica a frequência de determinado termo em uma frase e *Inverse Document Frequency*

indica a frequência de determinada instância comparada com a quantidade total de instâncias. Desta forma, enquanto TF indica a importância de um termo em uma única frase independente do tamanho do documento, IDF indica a "representatividade"daquele termo em relação ao conjunto completo de dados. Assim, é possível uma análise mais completa da importância de cada termo na amostra.

2.2.3 Comparativo

Para implementar cada método é possível determinar qual parâmetro de representação será utilizado. Por exemplo, podem ser utilizadas palavras isoladas (analyzer=word), letras isoladas (analyzer=char), grupo de palavras (n-gram = x,y) e "stop_words", que funcionam ponderando quais palavras possuem menos peso para a análise. Para realizar um comparativos entre esses métodos será utilizado o método CountVectorizer com o parâmetro stop_words='english', que irá diminuir o peso das palavras comumente utilizadas na estruturação de frases em inglês. Também será utilizado o método TF-IDF com os parâmetros word, character e n-gram. Desta forma poderá ser obtido uma visão geral da eficiência de cada método de vetorização para o problema proposto.

```
#Definição do modelo de aprendizagem utilizado
model = MultinomialNB()
print('Precisão para diferentes métodos de vetorização:\n')

#Utilização do método de vetorização CountVectorizer
vec_count = CountVectorizer(stop_words='english')
treinoX_count = vec_count.fit_transform(treino_x).toarray()
validaX_count = vec_count.transform(valida_x).toarray()
model.fit(treinoX_count, treino_y)
eficiencia_count = model.score(validaX_count, valida_y) * 100
print(f'CountVectorizer: {eficiencia_count}%')

#Utilização do método de vetorização TF_IDF com análise de palavras:
vec_tfid1 = TfidfVectorizer(analyzer='word')
treinoX_tfidf = vec_tfid1.fit_transform(treino_x).toarray()
validaX_tfidf = vec_tfid1.transform(valida_x).toarray()
model.fit(treinoX_tfidf, treino_y)
eficiencia_tfidf = model.score(validaX_tfidf, valida_y) * 100
print(f'TF-IDF [Utilizando palavras]: {eficiencia_tfidf}%')

#Utilização do método de vetorização TF-IDF com análise de n-grams:
vec_ngram = TfidfVectorizer(analyzer='word', ngram_range=(2,3),
    max_features=10000)
treinoX_ngram = vec_ngram.fit_transform(treino_x)
validaX_ngram = vec_ngram.transform(valida_x)
model.fit(treinoX_ngram, treino_y)
eficiencia_ngram = model.score(validaX_ngram, valida_y) * 100
```

```
27 print(f'TF-IDF [Utilizando n-gram(2,3)]: {eficiencia_ngram}%')
28
29 ##Utilização do método de vetorização TF-IDF com análise de letras:
30 vec_char = TfidfVectorizer(analyzer='char', ngram_range=(2,3),
       max_features=10000)
31 treinoX_char = vec_char.fit_transform(treino_x)
32 validaX_char = vec_char.transform(valida_x)
33 model.fit(treinoX_char, treino_y)
34 eficiencia_char = model.score(validaX_char, valida_y) * 100
35 print(f'TF-IDF [Utilizando letras]: {eficiencia_char}%')
```

Código 2.4 – Teste de diferentes métodos de vetorização.

Com este código foi obtido o resultado ilustrado na Figura 9.

```
Precisão para diferentes métodos de vetorização:

CountVectorizer: 40.07848422387823%
TF-IDF [Utilizando palavras]: 35.80149040748375%
TF-IDF [Utilizando n-gram(2,3)]: 36.69732043760901%
TF-IDF [Utilizando letras]: 36.958934517203105%
```

Figura 9 – Comparação entre diferentes formas de vetorização.

Pode-se observar que o método CountVectorizer obteve melhores resultados e portanto será utilizado nas etapas seguintes do desenvolvimento.

2.3 Modelos

Como visto na introdução, neste trabalho nós utilizaremos uma abordagem estatística e portanto um modelo deste tipo. Foi escolhido o modelo **Naive Bayes**, pois se trata de um modelo probabilístico de fácil implementação e cujos resultados podem ser analisados rapidamente.

A teoria deste método consiste no cálculo da probabilidade de uma hipótese caso certa evidência seja verdadeira:

$$P(H|E) = \frac{P(E|H).P(H)}{P(E)}$$

Em que $P(H|E)$ = Probabilidade da Hipótese ser verdadeira caso a evidência seja verdadeira, $P(E|H)$ = Probabilidade da Evidência ser verdadeira dado a Hipótese verdadeira, $P(H)$ = Probabilidade da Hipótese ser verdadeira, $P(E)$ = Probabilidade da Evidência ser verdadeira.

Assim, o conceito básico do cálculo da probabilidade é definido. Entretanto existem dentro deste paradigma diversas formas alternativas de realizar essa estimativa, entre

eles o método de Bernoulli, Multinomial, Complemento, Gaussiano, etc. Entretanto, foge do escopo deste trabalho analisar cada um matematicamente e portanto eles serão implementados de forma computacional e os resultados serão comparados.

2.3.1 Multinomial

```
model_multinomial = MultinomialNB()
model_multinomial.fit(treino_x, treino_y)
eficiencia_multinomial = model_multinomial.score(valida_x, valida_y) *
    100
print(f"Multinomial: {eficiencia_multinomial}%")
```

Código 2.5 – Naive Bayes Multinomial.

2.3.2 Bernoulli

```
model_bernoulli = BernoulliNB()
model_bernoulli.fit(treino_x, treino_y)
eficiencia_bernoulli = model_bernoulli.score(valida_x, valida_y) * 100
print(f"Bernoulli: {eficiencia_bernoulli}%")
```

Código 2.6 – Naive Bayes Bernoulli.

2.3.3 Complement

```
model_complement = ComplementNB()
model_complement.fit(treino_x, treino_y)
eficiencia_complement = model_complement.score(valida_x, valida_y) * 100
print(f"Complement: {eficiencia_complement}%")
```

Código 2.7 – Naive Bayes Complement

2.3.4 Gaussian

```
model_gaussian = GaussianNB()
model_gaussian.fit(treino_x, treino_y)
eficiencia_gaussian = model_gaussian.score(valida_x, valida_y) * 100
print(f"Gaussian: {eficiencia_gaussian}%")
```

Código 2.8 – Naive Bayes Gaussian.

2.3.5 Comparativo

A implementação dos códigos acima geraram o resultado ilustrado na Figura X.

```
Precisão para diferentes modelos de classificadores Naive-Bayes:

Multinomial: 40.07848422387823%
Bernoulli: 39.182654193752974%
Complement: 36.1265260821309 65%
Gaussian: 11.935151419058188%
```

Figura 10 – Comparação entre diferentes formas de implementar o modelo Naive Bayes.

2.4 Análise dos Resultados

Foi observado que dentre os métodos utilizados o uso do vetorizador CountVectorizer e do modelo Naive Bayes com distribuição Multinomial obteve a melhor precisão dentre os métodos testados, entretanto, a precisão encontrada ainda é muito baixa (40%). Isto significa que podemos focar a análise apenas nestes métodos porém fazendo alterações nos parâmetros de entrada, de forma que seja possível alcançar uma precisão maior. Algumas modificações que poderiam ser implementadas são:

2.4.1 Hiperparâmetros

Hiperparâmetros são os parâmetros utilizados para configurar os métodos utilizados, neste trabalho nós utilizamos parâmetros genéricos tanto para realizar a vetorização quanto para treinar o modelo. Entretanto é possível realizar um refinamento desses parâmetros, de modo que, seja possível obter melhores resultados. Por exemplo:

- Hiperparâmetros de vetorização: Podem ser testadas diferentes formas de "stop_words", analyzers, n-grams, dicionários customizados, etc.
- Hiperparâmetros de modelagem: Podem ser testadas diferentes formas de suavização e estimação das classes e probabilidades (alpha, fit$_prior, class_prior, etc$).

Ou seja, cada método permite a implementação de diversas técnicas de refinamento que podem ser úteis para a evolução da solução obtida. Além disso, também existem ferramentas que otimizam esse processo de forma automatizada, como a função GridSearchCV(), que testa diferentes combinações de parâmetros e implementa a combinação mais eficiente.

2.4.2 Abordagens alternativas

Visto que, a diferença entre uma classe (personagem) e outra está relacionada com gírias, talvez a abordagem estatística não seja a mais indicada, pois a frequência destes

termos específicos será muito pequena para gerar resultados impactantes. Desta forma, pode-se tentar adotar abordagens semânticas, relacionando determinadas gírias com significados específicos. Assim, seria possível criar um dicionário personalizado, ou utilizar métodos específicos, como por exemplo:

- First-order predicate logic (FOPL): Utiliza lógica booleana para inferir relacionamentos semânticos (Em python pode ser implementado utilizando a biblitoeca aima.utils e aima.logic)
- Semantic Nets: Os termos são representados como nós e o relacionamento entre esses nós é analisado. Assim, é possível criar uma rede baseada na importância semântica de cada termo.
- Conceptual dependency (CD): Os termos são divididos em termos menores que implicam o significado do termo principal. Desta forma é possível abstrair a palavra em si e estimar o valor de seu significado de forma mais relevante.

2.5 Problemas Encontrados

No desenvolvimento do trabalho foram enfrentados alguns problemas, entre eles:

- Memória insuficiente: O tamanho do dataset implica em um volume grande de cálculos, estes dados ultrapassavam a capacidade da memória RAM disponível resultando em erro. **Solução:** Foi alterado o tamanho da paginação da memória da máquina, desta forma foi possível utilizar espaço dos discos locais como memória virtual, possibilitando a continuidade dos cálculos. **Solução alternativa:** Utilizar notebooks ou serviços em nuvem, pois os mesmos oferecem acesso à ferramentas de computação com maiores recursos.
- Tempo de treinamento muito elevado. **Solução:** Para melhorar a velocidade de treinamento o dataset foi dividido em porções menores, porém apesar de mais rápido a precisão fica comprometida. Assim, foi utilizado o dataset menor para comparar as possíveis soluções e dentre estas, apenas as melhores foram selecionadas para o treinamento completo. **Solução alternativa:** Novamente, o uso de notebooks e serviços de nuvem poderiam facilitar o treinamento, pois oferecem maior quantidade de recursos.
- Necessidade de treinar o modelo toda vez que o mesmo fosse ser utilizado. **Solução:** Basta implementar uma ferramenta de serialização que irá salvar o modelo treinado em um arquivo. Desta forma, é possível apenas carregar o modelo já treinado quando for realizar alguma predição. Esta serialização pode ser realizada com as biblitoecas pickle e/ou joblib.

Capítulo 3

Conclusão

Com este trabalho foi possível testar diferentes métodos de vetorização, bem como, diferentes modelos de treinamento. Foram comparados os seguintes métodos:

- Vetorização: CountVector(stop_word='english') e TF-IDF(word, char e n-grams).
- Modelos: Naive Bayes (Multinomial, Bernoulli, Complement e Gaussian).

Foi obtida uma precisão de 40% utilizando CountVector e Naive Bayes Multinomial, sem o tunning de hiperparâmetros. O que indica que uma abordagem semântica para a vetorização pode ser utilizada para melhores resultados.

Além disso, foi realizada uma pequena revisão teórica dos princípios básicos que envolvem o treinamento de um modelo de aprendizagem para que o mesmo possa representar, interpretar, e manipular texto de forma natural.

Referências

KULKARNI, A.; SHIVANANDA, A. **Natural language processing recipes**. [S.l.]: Springer, 2019.

LIDDY, E. D. Natural language processing. 2001.

RASCHKA, S. **Python machine learning**. [S.l.]: Packt publishing ltd, 2015.