INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS

INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS

RELATÓRIO TÉCNICO – EXAME – MINERAÇÃO DE DADOS – ANÁLISE DE SENTIMENTOS EM AVALIAÇÕES DE PRODUTOS DE E-COMMERCE

INTEGRANTES DO GRUPO 9:

Ernesto Amândio - 20210062

Marcelo Rocha – 20210032

LUANDA

2026

ÍNDICE

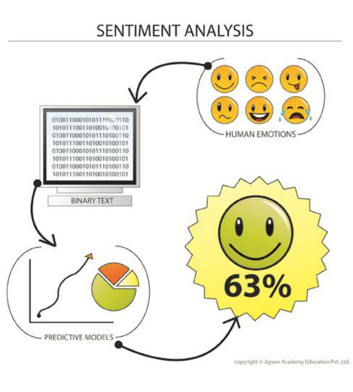
INTRODUÇÃO

Com o crescimento do **comércio eletrónico**, as **avaliações deixadas por consumidores** tornaram-se uma fonte valiosa de informação para empresas e plataformas digitais. Essas avaliações refletem **opiniões, níveis de satisfação e perceções dos utilizadores** em relação a produtos e serviços, influenciando diretamente **decisões de compra** e **estratégias de negócio**. No entanto, devido ao **grande volume de dados textuais** gerados diariamente, torna-se inviável realizar essa análise de forma manual, tornando necessária a utilização de **técnicas automáticas** para o tratamento dessas informações.

Neste contexto, a **Análise de Sentimentos**, aliada ao **Processamento de Linguagem Natural (PNL)** e à **Mineração de Dados**, permite identificar automaticamente o sentimento expresso em textos, geralmente classificado como **positivo, negativo ou neutro**. A utilização de **técnicas de aprendizado de máquina supervisionado** possibilita a construção de modelos capazes de **aprender padrões linguísticos** a partir de dados previamente rotulados, oferecendo resultados **consistentes e escaláveis**.

Este projeto tem como objetivo **analisar avaliações de produtos de e-commerce para identificar o sentimento dos consumidores**, utilizando **modelos de aprendizado de máquina supervisionado aplicados à classificação de texto**. Para tal, é utilizado o **Amazon Product Review Dataset**, um conjunto de dados real **amplamente adotado na área**.

O trabalho segue as principais etapas de um projeto de mineração de dados, incluindo a **preparação dos dados**, **análise exploratória**, **desenvolvimento e avaliação de modelos de classificação**, bem como a **interpretação dos resultados**, resultando na construção de um **modelo de análise de sentimentos** e na **geração de gráficos representativos da distribuição dos sentimentos**.



ESCOLHA DA METO

Para o desenvolvimento deste projeto de **Análise de Sentimentos em Avaliações de Produtos**, optou-se por utilizar a metodologia **CRISP-DM (CRoss Industry Standard Process for Data Mining)**, devido à sua flexibilidade, robustez e ampla aceitação na indústria de dados.

O **CRISP-DM** é uma metodologia estruturada de mineração de dados que guia todo o processo de análise de forma **iterativa e adaptável**, sendo composta por seis fases principais:

1. **Compreensão do Negócio (Business Understanding)**
   - O objetivo desta fase é definir claramente os **objetivos do projeto**, identificar os **critérios de sucesso** e compreender o **contexto do problema**.
   - Para este trabalho, a fase envolve analisar **avaliações de produtos de e-commerce** para identificar automaticamente o **sentimento dos consumidores**, categorizando as opiniões como **positivas, neutras ou negativas**.
   - Inclui também compreender as limitações e características do **Amazon Product Review Dataset**, bem como os requisitos de **escalabilidade** e **precisão** do modelo de classificação.

2. **Compreensão dos Dados (Data Understanding)**
   - Esta fase foca na **coleta, exploração inicial** e **avaliação da qualidade dos dados**.
   - No projeto, envolve analisar a **consistência das avaliações**, detectar **valores ausentes** ou duplicados, compreender a **distribuição das classes de sentimento** e examinar **características textuais relevantes**.
   - É nesta etapa que se identifica a necessidade de **balanceamento das classes** e de **limpeza do texto**, incluindo remoção de **stopwords**, **normalização** e **tokenização**.

3. **Preparação dos Dados (Data Preparation)**

- O objetivo é criar um **conjunto de dados limpo, estruturado e adequado para modelagem**.
- Para este projeto, a fase inclui:
    - **Limpeza e normalização** das avaliações (remoção de símbolos, letras maiúsculas, espaços extras).
    - Transformação do texto em **representações numéricas**, utilizando **Bag of Words, TF-IDF** e **Word Embeddings (GloVe)**.
    - Divisão do **dataset** em **conjuntos de treino e teste**, garantindo **balanceamento entre as classes**.

4. **Modelagem (Modeling)**

- Nesta fase, são aplicadas técnicas de **aprendizado de máquina supervisionado** para **classificar as avaliações**.
- Os algoritmos utilizados para teste incluem **Naive Bayes, Logistic Regression, Support Vector Machines (SVM), Random Forest, Decision Trees e K-Nearest Neighbors (KNN).**
- Cada modelo é **treinado, ajustado** e **avaliado** utilizando métricas como **accuracy, F1-score** e **matriz de confusão**.

5. **Avaliação (Evaluation)**

- Avalia-se o **desempenho dos modelos** em dados **não vistos**, interpretando os resultados e identificando o algoritmo que apresenta melhor **equilíbrio entre precisão e generalização**.
- Esta fase também permite analisar **erros comuns** e propor melhorias no **pré-processamento** ou na **modelagem**.

6. Implementação (Deployment)

- Consiste em consolidar o **modelo final** e disponibilizá-lo para uso em **novas avaliações**.
- Inclui a **exportação dos modelos treinados** e dos **transformadores de texto**, garantindo que **novas entradas** possam ser **classificadas automaticamente** de forma consistente.

DESCRIÇÃO DAS FONTES/DADOS

Neste projeto, os dados utilizados para a **Análise de Sentimentos em Avaliações de Produtos** foram obtidos a partir do **"Amazon Reviews Dataset"**, disponibilizado no **Kaggle** pelo autor **Daniel Ihenacho**. Este conjunto de dados foi construído a partir de **avaliações (reviews) reais de produtos da Amazon**, sendo criado especificamente para tarefas de **classificação de texto e análise de sentimento**.

Este **conjunto de dados** contém **avaliações de produtos da Amazon**, obtidas **parcialmente por web scraping** e **parcialmente a partir do dataset original "Amazon Review Dataset"**, com o objetivo de permitir a construção de **modelos para análise automática de sentimento**. O dataset está licenciado sob **MIT License**, permitindo o uso em **projetos acadêmicos e de pesquisa**.

As principais variáveis presentes nos dados são:

- **“sentiments”**: etiqueta da classe de sentimento associada à review, representando se o conteúdo é **positivo, neutro ou negativo**.
- **“cleaned_review”**: o texto da avaliação, já pré-processado e limpo pelo autor para facilitar a análise de texto.
- **“cleaned_review_length”**: comprimento da avaliação após a limpeza, ou seja, o número de palavras ou tokens que restaram após o pré-processamento.
- **“review_score”**: a pontuação atribuída pelo usuário ao produto (classicamente de 1 a 5 estrelas), que pode refletir a intensidade do sentimento.

Este dataset apresenta um problema típico de **classificação multiclasse**, com três categorias de sentimento (positiva, neutra e negativa), o que é adequado para a aplicação de algoritmos de **aprendizado de máquina supervisionado**.

PRÉ-PROCESSAMENTO DOS DADOS

- **Identificação e tratamento de valores ausentes**

Após o carregamento do **Amazon Reviews Dataset**, foi realizada uma verificação sistemática da existência de valores ausentes nas variáveis do conjunto de dados. Para tal, recorreu-se à função ***isnull().sum()***, permitindo identificar quantitativamente possíveis casos de incompletude.

Os resultados obtidos foram os seguintes:

```
df.isnull().sum()
```

```
sentiments               0
cleaned_review           3
cleaned_review_length    0
review_score             0
dtype: int64
```

A análise revelou, portanto, a existência de **apenas três registos com valores ausentes na variável** “**cleaned_review”**, que representa o texto da avaliação do produto. Com base na natureza do problema e nos conceitos abordados em aula, os valores ausentes identificados foram classificados como **MCAR (Missing Completely At Random)**. Esta classificação justifica-se pelo facto de a ausência do texto da review **não apresentar relação direta com as restantes variáveis do conjunto de dados**, como o sentimento atribuído ou a pontuação da avaliação. É plausível assumir que estes casos resultam de **erros pontuais no processo de recolha, scraping ou pré-processamento do dataset**, não refletindo um padrão sistemático ou dependente de outras características dos dados.

Dado que o número de registos afetados é **extremamente reduzido (3 registos)** quando comparado com a dimensão total do dataset, optou-se pela **exclusão dos registos com valores ausentes** na variável "**cleaned_review**", conforme recomendado para situações de MCAR com baixa proporção de incompletude.

```
df = df.dropna(subset=['cleaned_review'])
df.isnull().sum()
```

```
sentiments               0
cleaned_review           0
cleaned_review_length    0
review_score             0
dtype: int64
```

A escolha deste método é **justificada pelo próprio objetivo do projeto**, que consiste na **classificação de texto**. Registos sem conteúdo textual **não contribuem para a aprendizagem do modelo**, nem permitem a extração de características linguísticas relevantes, tornando a sua manutenção no dataset irrelevante para a análise.

- **Codificação/Mapeamento da variável alvo**

A variável **"sentiments"** presente no conjunto de dados representa a **classe de sentimento associada a cada avaliação (*negative*, *neutral* e *positive*).** No entanto, os **algoritmos de aprendizado de máquina supervisionado utilizados neste projeto** (como Naive Bayes, SVM, Regressão Logística e Random Forest) **não operam diretamente sobre rótulos textuais**, exigindo que a variável alvo esteja representada de forma numérica.

Dessa forma, tornou-se necessário realizar a **codificação da variável categórica de sentimento**, transformando os rótulos textuais em valores numéricos inteiros, mantendo uma correspondência clara e consistente entre cada classe e o seu significado semântico.

```python
def map_sentiment(sentiment):
    if sentiment == "negative":
        return 0    # Negativo
    elif sentiment == "neutral":
        return 1    # Neutro
    else:
        return 2    # Positivo


df['sentiments_final'] = df['sentiments'].apply(map_sentiment)
```

Este mapeamento foi aplicado à variável original **"sentiments"**, resultando na criação de uma nova variável denominada "**sentiments_final"**, que passou a ser utilizada como **variável dependente (target)** nos modelos de classificação.

- **Análise e Balanceamento das Classes**

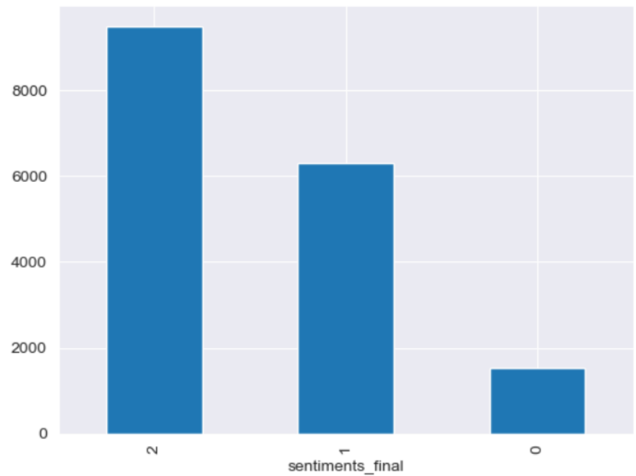
Após a codificação da variável alvo, procedeu-se à **verificação do balanceamento das classes de sentimento** no conjunto de dados. Para tal, foi analisada a distribuição da variável **"sentiments_final"**, tanto de forma numérica como gráfica.

```python
df['sentiments_final'].value_counts()
```

```
sentiments_final
2    9507
1    6304
0    1538
Name: count, dtype: int64
```

```python
df['sentiments_final'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='sentiments_final'>
```



A análise re\ **as classes**, com predominânci **utras**, enquanto a classe **negat** nente inferior de registos. Este desbalanceamento pode comprometer o desempenho dos modelos de classificação, levando-os a favorecer as classes majoritárias e a apresentar fraca capacidade de generalização para classes minoritárias.

Inicialmente, com o objetivo de mitigar esse problema, foi testada a aplicação da técnica de **undersampling**, que consiste em reduzir aleatoriamente o número de registos das classes maioritárias até igualar a dimensão da classe minoritária. Após o balanceamento, cada classe passou a conter **1.538 registos**, resultando num conjunto de dados final com **4.614 avaliações**. Esta distribuição equilibrada foi validada numericamente e visualmente.

```python
# Descobrir tamanho da menor classe
min_class_size = df['sentiments_final'].value_counts().min()

# Fazer undersampling em cada classe
df_balanced = (
    df.groupby('sentiments_final', group_keys=False)
      .apply(lambda x: x.sample(n=min_class_size, random_state=42))
)

# Embaralhar o dataset final
df_balanced = df_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

# Verificar
df_balanced['sentiments_final'].value_counts()
```

```
C:\Users\Marcelo Rocha\AppData\Local\Temp\ipykernel_23568\3466893856.py:7: Deprecati
his behavior is deprecated, and in a future version of pandas the grouping columns w
` to exclude the groupings or explicitly select the grouping columns after groupby t
  .apply(lambda x: x.sample(n=min_class_size, random_state=42))
```

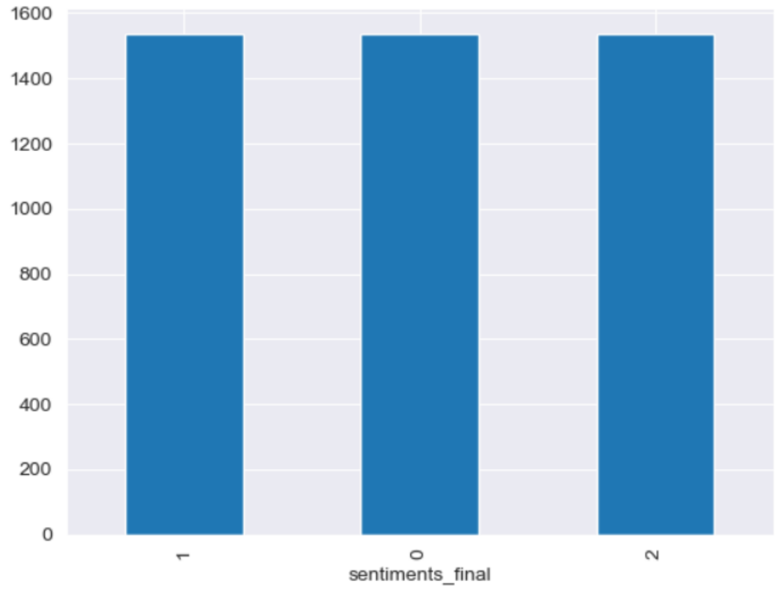
```
sentiments_final
1    1538
0    1538
2    1538
Name: count, dtype: int64
```

```
print("Tamanho do Dataset",df_balanced.shape)
```

```
Tamanho do Dataset (4614, 5)
```

```
print('Distribuição de sentimentos')
df_balanced['sentiments_final'].value_counts().plot(kind='bar')
```

```
Distribuição de sentimentos
<Axes: xlabel='sentiments_final'>
```



No entanto, apesar de esta abordagem ter permitido obter resultados aceitáveis, verificou-se que o undersampling implicou uma **redução significativa do volume total de dados**, causando **perda de informação relevante**. Essa limitação pode afetar a capacidade do modelo em reconhecer padrões mais complexos presentes nas classes maioritárias.

Dessa forma, optou-se por uma abordagem alternativa mais adequada ao contexto do projeto: a utilização de **class weights** nos algoritmos de classificação. Esta técnica permite treinar os modelos diretamente com os **dados originais desbalanceados**, atribuindo maior penalização aos erros cometidos sobre as classes minoritárias, **sem necessidade de remover amostras**.

A utilização de **class weights** demonstrou um desempenho **superior e mais estável** em comparação com o undersampling, sendo por isso adotada como a estratégia principal **ao longo da análise e avaliação dos modelos desenvolvidos**. Assim, a implementação anteriormente utilizada para **undersampling** foi **desativada (comentada no código)**, mantendo-se apenas para fins de comparação e documentação nos anexos.

```python
# # Descobrir tamanho da menor classe
# min_class_size = df['sentiments_final'].value_counts().min()

# # Fazer undersampling em cada classe
# df_balanced = (
#     df.groupby('sentiments_final', group_keys=False)
#       .apply(lambda x: x.sample(n=min_class_size, random_state=42))
# )

# # Embaralhar o dataset final
# df_balanced = df_balanced.sample(frac=1, random_state=42).reset_index(drop=True)

# Manter dataset
df_balanced = df.copy()
df_balanced['sentiments_final'].value_counts()
```

```
sentiments_final
2    9507
1    6304
0    1538
Name: count, dtype: int64
```

- **Seleção das Variáveis Relevantes e Limpeza do Texto**

Após a análise referente ao balanceamento do conjunto de dados, procedeu-se à **seleção das variáveis relevantes** para o problema de classificação de sentimentos. Considerando que o objetivo do projeto é a **classificação de texto**, foram extraídas:

```python
X = df_balanced['cleaned_review'] #o conteúdo textual das avaliações
y = df_balanced['sentiments_final'] #correspondente ao sentimento associado a cada avaliação.
```

Esta separação permite distinguir claramente as **variáveis independentes (texto)** da **variável dependente (sentimento)**, preparando os dados para as etapas subsequentes de pré-processamento e modelagem.

Apesar de o dataset disponibilizar uma coluna denominada *cleaned_review*, optou-se por repetir todo o processo de limpeza textual, por uma questão de **rigor metodológico** e **controlo sobre o pré-processamento**. Em projetos de **Mineração de Dados** e **Processamento de Linguagem Natural**, é boa prática não assumir como definitiva a limpeza realizada por terceiros, garantindo que todas as etapas estejam alinhadas com os **objetivos do estudo**.

Além disso, durante a inspeção dessa coluna, verificou-se alguma **inconsistência no tratamento aplicado**, nomeadamente na **remoção incompleta de stopwords**, o que poderia introduzir **ruído textual** e afetar negativamente o **desempenho dos modelos**. Assim, a limpeza foi refeita de forma **padronizada**, assegurando maior **fiabilidade na preparação dos dados**.

Para este efeito, foi implementada uma **função genérica de limpeza de texto**. O processo de limpeza inclui, de forma resumida:

- **Remoção de conteúdo HTML**, garantindo que apenas o texto relevante é analisado;
- **Normalização para minúsculas**, evitando duplicação artificial de termos;
- **Remoção de caracteres especiais e números**, reduzindo ruído no texto;
- **Tokenização**, transformando o texto em unidades linguísticas básicas;
- **Tratamento de stopwords**, com especial atenção à **preservação de negações**, fundamentais para a análise de sentimentos;
- **Lematização**, permitindo reduzir palavras à sua forma canónica, mantendo o significado semântico.

```python
# Config globais (evita recarregar na função)
STOP_WORDS = set(stopwords.words('english'))
# Preservar mais negações para melhor análise de sentimentos
NEGATIONS = ['not', 'no', 'never', 'neither', 'nor', 'cannot', "can't", 'nothing', 'none', 'nowhere', 'nobody']
for word in NEGATIONS:
    STOP_WORDS.discard(word)

LEMMATIZER = WordNetLemmatizer()
STEMMER = SnowballStemmer('english')


def clean_review(text: str,
                 remove_stopwords: bool = True,
                 lemmatize: bool = True,
                 stem: bool = False,  # Opcional, para comparação com o original
                 remove_numbers: bool = False,
                 return_tokens: bool = False) -> str | list[str]:
    """
    Função de limpeza otimizada para reviews da Amazon.
    Compatível com TF-IDF/Word2Vec e modelos como MNB, LR, SVM, KNN, AdaBoost.

    Parâmetros:
    - text: texto bruto
    - remove_stopwords: remover stopwords (preservando negações)
    - lemmatize: aplicar lematização (recomendado)
    - stem: aplicar stemming (alternativa, mas menos precisa)
    - remove_numbers: remover números (ex: preços)
    - return_tokens: devolver lista de tokens (para Word2Vec) ou string (para TF-IDF)

    Retorna: string limpa ou lista de tokens
    """
```

```python
if pd.isna(text) or not isinstance(text, str):
    return "" if not return_tokens else []

# 1. Remover HTML
text = BeautifulSoup(text, "lxml").get_text()

# 2. Minúsculas
text = text.lower()

# 3. Remover indesejados
pattern = r'[^a-z\s]' if remove_numbers else r'[^a-z0-9\s]'
text = re.sub(pattern, ' ', text)

# 4. Tokenização precisa
tokens = word_tokenize(text)

# 5. Remover stopwords
if remove_stopwords:
    tokens = [t for t in tokens if t not in STOP_WORDS and len(t) > 1]  # Ignora tokens muito curtos

# 6. Lematização ou stemming
if lemmatize:
    tokens = [LEMMATIZER.lemmatize(t) for t in tokens]
elif stem:
    tokens = [STEMMER.stem(t) for t in tokens]

# 7. Retornar
if return_tokens:
    return tokens
return ' '.join(tokens)
```

A função foi aplicada a todas as avaliações do conjunto de dados, originando uma nova representação textual limpa (**X_cleaned**), que serviu de base para a **extração de características** e para o **treino dos modelos de classificação**.

```python
X_cleaned = []
X_cleaned = [clean_review(text, remove_stopwords=True, lemmatize=True, remove_numbers=True)
             for text in X]
```

- **Representação Vetorial do Texto**

Após a limpeza do texto, tornou-se necessário **converter as avaliações textuais em representações numéricas**, uma vez que os algoritmos de aprendizado de máquina **não operam diretamente sobre texto**, mas sim sobre valores numéricos ou vetores.

Neste projeto, foram **testadas três abordagens distintas de representação textual** onde cada uma delas gera um conjunto distinto de características numéricas, que foi utilizado como entrada para os modelos de classificação:

Bag of Words (BoW)

- A abordagem **Bag of Words** representa cada documento/texto como um vetor baseado na **frequência de ocorrência das palavras**, ignorando a ordem em que aparecem no texto. Cada posição do vetor corresponde a um termo do vocabulário, e o valor indica quantas vezes esse termo ocorre na avaliação.

- Neste trabalho, utilizou-se o **CountVectorizer**, com configurações que permitem: ignorar termos muito raros ou excessivamente frequentes , limitar o tamanho do vocabulário e considerar **unigramas e bigramas**, captando expressões simples relevantes para o sentimento.

- Esta abordagem é simples, eficiente e frequentemente eficaz em problemas de classificação de texto, embora não capture relações semânticas entre palavras.

```python
#Classificação de texto usando Bag of Words com CountVectorizer

# Configuração recomendada
countVect = CountVectorizer(
    min_df=5,              # ignora palavras muito raras (<5 documentos)
    max_df=0.85,           # ignora palavras muito frequentes (>85% dos documentos)
    max_features=8000,    # mantém apenas as 10k palavras/ngrams mais frequentes
    ngram_range=(1,2),     # unigrams e bigrams (captura expressões como "not good")
    strip_accents='unicode',# remove acentos
    binary=False,           # presença/ausência da palavra é suficiente para classificação
    token_pattern=r'\b\w+\b' # ignora símbolos isolados, pega apenas palavras
)


X_all_countVect = countVect.fit_transform(X_cleaned)

feature_names = countVect.get_feature_names_out()

print("Number of features : %d\n" % len(feature_names))
print("Show some feature names:\n", feature_names[::1000])
```

```
Number of features : 8000

Show some feature names:
 ['aa' 'choice' 'excellent speaker' 'hold long' 'mentioned' 'pandemic'
 'setup mouse' 'tried another']
```

TF-IDF (Term Frequency–Inverse Document Frequency)

- O **TF-IDF** é uma extensão do Bag of Words que procura medir a **importância relativa de uma palavra** em um documento, considerando também a sua frequência em todo o conjunto de dados. Enquanto palavras muito comuns recebem menor peso, termos mais específicos de determinadas avaliações tornam-se mais relevantes. Isso ajuda a reduzir o impacto de palavras pouco informativas e a destacar termos discriminativos para a classificação de sentimentos.

- Neste projeto, utilizou-se o **TfidfVectorizer**, com parâmetros ajustados para: controlar a dimensionalidade do vocabulário, utilizar n-gramas e normalizar os vetores, tornando-os mais adequados para modelos como SVM e Regressão Logística.

```
#Classificação de texto usando TF-IDF com TfidfVectorizer
tfidf = TfidfVectorizer(
    min_df=5,
    max_df=0.85,
    max_features=12000,      # 🔥 mais adequado para 4600 docs
    ngram_range=(1,2),
    strip_accents='unicode',
    norm='l2',
    use_idf=True,
    smooth_idf=True,
    sublinear_tf=True
)

X_all_tfidf = tfidf.fit_transform(X_cleaned)

print(f"Vocabulário TF-IDF: {X_all_tfidf.shape[1]:,} termos")
print("Exemplos de features:\n", tfidf.get_feature_names_out()[::1500][:20])
```

```
Vocabulário TF-IDF: 10,839 termos
Exemplos de features:
 ['aa' 'color got' 'flaw' 'last around' 'multiple device'
 'quality horrible' 'subdued' 'work love']
```

Word Embeddings (GloVe)

- Diferentemente das abordagens anteriores, os **Word Embeddings** representam palavras como vetores densos que **capturam relações semânticas e contextuais**. Palavras com significados semelhantes tendem a ter representações próximas no espaço vetorial.

- Neste trabalho, foi utilizado o modelo **GloVe (Global Vectors for Word Representation)**, previamente treinado em grandes volumes de texto. Cada palavra é associada a um vetor numérico de dimensão fixa.

- Para representar uma avaliação completa, foi calculada a **média dos vetores das palavras presentes no texto**, resultando em um vetor único por review. Essa abordagem permite capturar informação semântica mais rica, embora dependa da cobertura do vocabulário do modelo pré-treinado.

```python
#WORD EMBEDDING
# Vamos assumir que X_cleaned já é a lista de strings limpas
X_tokens = [word_tokenize(text) for text in X_cleaned]

#print(X_tokens[0][:10])  # exemplo de tokens do primeiro review

# Caminho para o arquivo .txt do GloVe
glove_file = 'glove.6B/glove.6B.100d.txt'

# Carregar embeddings
embeddings_index = {}
with open(glove_file, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector

print(f"Número de palavras no GloVe: {len(embeddings_index)}")
#---------------------------------------------------------------------------
embedding_dim = 100  # depende do GloVe que você baixou

def review_to_vec(tokens, embeddings_index, embedding_dim):
    vecs = []
    for t in tokens:
        if t in embeddings_index:
            vecs.append(embeddings_index[t])
    if len(vecs) > 0:
        return np.mean(vecs, axis=0)  # média das palavras
    else:
        return np.zeros(embedding_dim)

# Transformar todas as reviews
X_embeddings = np.array([review_to_vec(tokens, embeddings_index, embedding_dim) for tokens in X_tokens])

print(X_embeddings.shape)  # deve dar (4602, 100)
```

```
Número de palavras no GloVe: 400000
(4614, 100)
```

- **Divisão dos Dados em Conjuntos de Treino e Teste**

Após a etapa de pré-processamento e da conversão do texto em representações numéricas, o conjunto de dados final, foi dividido em **dados de treino** e **dados de teste**, com o objetivo de **avaliar o desempenho dos modelos em dados não vistos durante o treino**.

A divisão foi realizada utilizando a técnica **train-test split**, adotando-se a proporção de **80% dos dados para treino** e **20% para teste**, prática comum em projetos de aprendizado de máquina supervisionado. Para garantir a **reprodutibilidade dos resultados**, foi definido um valor fixo para o parâmetro *random_state*.

Além disso, foi aplicado o critério de **estratificação das classes** (*stratify*=*y*), assegurando que a distribuição dos sentimentos (negativo, neutro e positivo) fosse preservada tanto no conjunto de treino quanto no conjunto de teste. Esta decisão é particularmente importante em problemas de classificação multiclasse, pois evita enviesamentos na avaliação do modelo.

```python
# Dividir em treino e teste (80% treino, 20% teste)
X_train, X_test, y_train, y_test = train_test_split(
    X_all_countVect,
    y,
    test_size=0.2,         # 20% para teste
    random_state=42,       # para reprodutibilidade
    stratify=y             # mantém proporção de classes
)


target_names = ['Negative', 'Neutral' , 'Positive']

# Verificar distribuição das classes
print("Distribuição no treino:")
print(y_train.value_counts())
print("\nDistribuição no teste:")
print(y_test.value_counts())

print(X_train.shape, X_test.shape)
```

```
Distribuição no treino:
sentiments_final
2    7606
1    5043
0    1230
Name: count, dtype: int64

Distribuição no teste:
sentiments_final
2    1901
1    1261
0     308
Name: count, dtype: int64
(13879, 8000) (3470, 8000)
```

Importa destacar que o conjunto de características (**X**) utilizado na divisão depende da abordagem de representação textual adotada:

- **Bag of Words** → *X_all_countVect*

- **TF-IDF** → *X_all_tfidf*
- **Word Embeddings (GloVe)** → *X_embeddings*

TÉCNICAS DE MINERAÇÃO DE DADOS E METODOLOGIAS UTILIZADAS

Após a conclusão das etapas de **pré-processamento**, **representação textual** e **divisão dos dados em treino e teste**, procedeu-se à aplicação de **técnicas de mineração de dados baseadas em aprendizado de máquina supervisionado**, com o objetivo de **avaliar e comparar o desempenho de diferentes algoritmos de classificação de texto**.

Dado que a Análise de Sentimentos é um problema de **classificação multiclasse**, onde o objetivo é prever o sentimento de uma avaliação (*negativo, neutro ou positivo*), foram testados **diversos algoritmos de classificação supervisionada**, combinados com **diferentes métodos de representação textual**, de forma a identificar a abordagem mais eficiente para o problema em estudo.

- **Metodologia Experimental Adotada**

  A metodologia seguida consistiu em:

  - Aplicar **cada algoritmo de classificação** sobre **cada método de representação textual**;
  - Avaliar o desempenho dos modelos utilizando métricas padronizadas;
  - Comparar os resultados obtidos entre as diferentes combinações, garantindo uma análise justa e consistente.

  As representações textuais testadas foram:

  - **Bag of Words (BoW)**
  - **TF-IDF (Term Frequency–Inverse Document Frequency)**
  - **Word Embeddings (GloVe)**

- **Algoritmos de Classificação Utilizados**

  Os seguintes algoritmos de aprendizado de máquina supervisionado foram implementados e avaliados na fase final do projeto, utilizando **class weights** para lidar com o **desbalanceamento de classes**:

- **Logistic Regression:** Modelo linear de classificação que estima a probabilidade de cada classe, sendo particularmente eficaz em problemas de classificação de texto quando combinado com representações como Bag of Words e TF-IDF.

- **Support Vector Machine (SVM):** Algoritmo que procura maximizar a margem entre as classes no espaço de características, sendo reconhecido pelo seu bom desempenho em problemas de classificação textual de alta dimensionalidade.

- **Random Forest:** Modelo baseado em *ensemble learning*, que combina múltiplas árvores de decisão para reduzir overfitting e melhorar a generalização.

- **Decision Tree:** Modelo de classificação baseado em uma estrutura hierárquica de decisões, onde os dados são divididos sucessivamente de acordo com atributos que melhor separam as classes.

```
lr = LogisticRegression(class_weight='balanced', max_iter=1000)
```

```
clf=svm.SVC(kernel='linear', class_weight='balanced', probability=True)
```

```
clf = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42,
    n_jobs=-1,
    class_weight="balanced"
)
```

```
clf = DecisionTreeClassifier(
    max_depth=None,
    random_state=42,
    class_weight="balanced"
)
```

Nota: Inicialmente, também foram testados **Naive Bayes (Multinomial)** e **K-Nearest Neighbors (KNN)**. No entanto, estes algoritmos **não oferecem suporte direto a class weights**, sendo menos adequados para datasets com **classes desbalanceadas**. Por esse motivo, foram utilizados apenas na fase de **undersampling**, mas não foram considerados na análise final com **class weights**, onde se optou por **modelos supervisionados capazes de lidar com dados textuais esparsos e com suporte a penalização de classes minoritárias**.

- **Avaliação dos Modelos**

  Para cada combinação entre **representação textual** e **algoritmo**, os modelos foram treinados utilizando o conjunto de treino e avaliados no conjunto de teste. As métricas consideradas foram:

  - **F1-Score (ponderado):** principal métrica de desempenho, adequada para **datasets desbalanceados**, refletindo melhor a performance do modelo em todas as classes.

  - **Matriz de Confusão:** permite analisar detalhadamente os acertos e erros do modelo por classe.

  - **Relatório de Classificação (precision, recall e f1-score por classe):** fornece informações detalhadas sobre o desempenho em cada classe específica.

  - **Acurácia (Accuracy):** reportada apenas para referência, **mas não deve ser considerada a métrica principal**, dado que pode ser **enganosa em situações de desbalanceamento de classes**.

ANÁLISE DOS RESULTADOS

Nesta secção são analisados os resultados obtidos a partir da aplicação dos diferentes algoritmos de classificação supervisionada, combinados com os três métodos de representação textual estudados: **Bag of Words**, **TF-IDF** e **Word Embeddings (GloVe)**.

A avaliação baseou-se principalmente nas métricas **F1-Score (ponderado)**, bem como na análise dos **relatórios de classificação**, **matrizes de confusão e acurácia (reportada apenas como referência)**, permitindo uma comparação detalhada entre os modelos.

- **Resultados com Bag of Words**

  A representação **Bag of Words** apresentou, de forma geral, os **melhores resultados globais** entre as três abordagens testadas:

- **Logistic Regression (LR)** apresentou o **melhor desempenho global**, com **F1-score ponderado ≈ 0,889**. O modelo mostrou **equilíbrio consistente entre precisão e recall** nas três classes, evidenciado pelos F1-scores individuais: **Negative (0,73), Neutral (0,86) e Positive (0,93)**. A matriz de confusão indica uma **distribuição equilibrada**, com **menor confusão entre sentimentos opostos**.

- **SVM** também teve **desempenho elevado** (**F1-score ponderado ≈ 0,885**), ligeiramente inferior ao da LR. Mostrou **bom equilíbrio**, especialmente na classe Neutral (**recall 0,87**) e Positive (**recall 0,92**).

- **Random Forest** obteve **F1-score ponderado ≈ 0,875**. Apesar da **alta precisão para Negative (0,98)**, o **recall desta classe foi baixo (0,51)**, indicando **confusão significativa entre Negative e Neutral**.

- **Decision Tree** apresentou **desempenho razoável** (**F1-score ponderado ≈ 0,848**), com **maior confusão nas classes Neutral e Positive**.

Classes	Logistic Regression			Decision Trees		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,74	0,72	0,73	0,60	0,71	0,65
Neutral	0,85	0,88	0,86	0,82	0,82	0,82
Positive	0,94	0,92	0,93	0,91	0,88	0,90

Classes	SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,72	0,73	0,73	0,98	0,51	0,67
Neutral	0,84	0,87	0,85	0,81	0,90	0,86
Positive	0,94	0,92	0,93	0,92	0,92	0,92

Métricas Globais	Logistic Regression		Decision Trees	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,888760807	0,889040033	0,845821326	0,847960219

Métricas Globais	SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,884726225	0,885110536	0,878962536	0,874937307

Com Bag of Words, modelos lineares como **Logistic Regression** e **SVM** mostraram-se particularmente eficazes, indicando que a separação das classes no espaço vetorial é aproximadamente linear.

- **Resultados com TF-IDF**

A abordagem **TF-IDF** apresentou resultados mistos:

  - **Random Forest** foi o modelo com **melhor F1-score ponderado (≈ 0,872)**, mostrando **bom desempenho geral**, embora ainda com **confusão na classe Negative (recall 0,50)**.

  - **SVM** apresentou **F1-score ponderado ≈ 0,850**, equilibrando **precisão e recall** nas três classes, mas ligeiramente inferior ao desempenho com Bag of Words.

  - **Logistic Regression** teve **F1-score ponderado ≈ 0,841**, com **precisão elevada em Positive (0,94)**, mas **recall mais baixo em Neutral (0,82)**.

  - **Decision Tree** obteve **F1-score ponderado ≈ 0,833**, sendo o modelo **menos eficaz** nesta abordagem, com **maior confusão entre Neutral e Positive**.

Classes	Logistic Regression			Decision Trees		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,58	0,75	0,66	0,56	0,68	0,62
Neutral	0,78	0,82	0,80	0,79	0,82	0,80
Positive	0,94	0,86	0,90	0,91	0,86	0,89

Classes	SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,64	0,73	0,68	0,97	0,50	0,66
Neutral	0,78	0,85	0,81	0,83	0,87	0,85

Métricas Globais	Logistic Regression		Decision Trees	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,836887608	0,840842841	0,829682997	0,832576281

Métricas Globais	SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,847550432	0,850197574	0,877233429	0,872271703

Embora o TF-IDF consiga reduzir o impacto de termos muito frequentes, neste conjunto de dados ele não trouxe ganhos significativos face ao Bag of Words, possivelmente devido à natureza relativamente direta das reviews da Amazon.

- **Resultados com Word Embeddings (GloVe)**

  A utilização de **Word Embeddings** apresentou desempenho inferior aos métodos anteriores baseados em contagem:

  - **Random Forest** obteve o **melhor desempenho** (**F1-score ponderado ≈ 0,828**), com **recall alto para Positive (0,94)**, mas **recall baixo para Negative (0,47)**, indicando **confusão significativa entre classes**.

  - **Decision Tree** apresentou **F1-score ponderado ≈ 0,758**, equilibrando melhor as classes, mas ainda **abaixo dos métodos Bag of Words e TF-IDF**.

  - **Logistic Regression e SVM** tiveram **desempenho modesto** (**F1-score ponderado ≈ 0,665 e 0,649**, respetivamente). A **representação média dos vetores das palavras** parece ter causado **perda de contexto importante**, prejudicando a capacidade de generalização dos modelos lineares.

Classes	Logistic Regression			Decision Trees		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,30	0,72	0,42	0,54	0,59	0,56
Neutral	0,58	0,52	0,55	0,74	0,69	0,72
Positive	0,86	0,72	0,78	0,80	0,83	0,82

Classes	SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,29	0,75	0,42	0,87	0,47	0,61
Neutral	0,56	0,49	0,52	0,83	0,76	0,79
Positive	0,85	0,70	0,77	0,84	0,94	0,89

Métricas Globais	Logistic Regression		Decision Trees	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,647262248	0,665044862	0,758213256	0,758111029

Métricas Globais	SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,629682997	0,648849485	0,835158501	0,828314431

  Apesar de os embeddings capturarem semântica, a estratégia de representar cada review pela **média dos vetores das palavras** pode ter causado perda de informação contextual, limitando o desempenho dos modelos clássicos de Machine Learning.

- **Modelo Final**

Analisando de forma global todas as combinações testadas, observa-se que o melhor desempenho global foi alcançado pela combinação **Logistic Regression com Bag of Words**, apresentando um **F1-score ponderado** em torno de **0,889**, evidenciando:

- Elevado equilíbrio entre precisão e recall nas três classes;
- Boa capacidade de generalização no conjunto de teste;
- Estabilidade nos resultados e menor confusão entre sentimentos opostos.

Classes	Logistic Regression		
	Precision	Recall	F1-Score
Negative	0,74	0,72	0,73
Neutral	0,85	0,88	0,86
Positive	0,94	0,92	0,93

Métricas Globais	Logistic Regression	
	Accuracy	F1-Score (Weighted)
	0,888760807	0,889040033

CONCLUSÃO

Os resultados obtidos demonstram que, para o conjunto de dados em estudo, **modelos lineares** combinados com **representações baseadas em frequência de palavras** apresentam o melhor desempenho global. Em particular, a combinação **Logistic Regression + Bag of Words** destacou-se como a abordagem mais eficaz, alcançando um **F1-score ponderado** de aproximadamente **0,889**, evidenciando um bom equilíbrio entre **precisão** e **recall** nas três classes de sentimento (**negativo**, **neutro** e **positivo**). Este resultado confirma que soluções mais simples e bem ajustadas ao problema podem superar abordagens teoricamente mais complexas quando aplicadas ao contexto adequado.

Embora técnicas como **TF-IDF** e **Word Embeddings** ofereçam **representações semanticamente mais ricas**, os seus resultados foram, neste caso, inferiores ou menos consistentes, especialmente quando combinadas com determinados classificadores. Tal comportamento pode estar relacionado com o **tamanho do dataset**, a forma de **agregação dos embeddings** e a natureza do problema de **classificação multiclasse**.

Como possíveis melhorias e direções futuras, o trabalho poderá ser estendido com a utilização de **modelos baseados em redes neurais**, como **LSTM** ou **GRU**, bem como **modelos transformer**, como o **BERT**, que são capazes de capturar de forma mais eficaz o **contexto semântico das frases**. Diferentemente das abordagens baseadas em **Bag of Words**, estes modelos consideram a **ordem** e a **dependência entre palavras**, o que pode reduzir erros em frases onde o sentimento depende do contexto, como por exemplo "este produto não é mau", que pode ser interpretada como **negativa** por modelos baseados apenas na presença da palavra "mau".

Em síntese, o trabalho demonstrou que a **escolha adequada da representação textual** e do **algoritmo de classificação** é determinante para o sucesso da **Análise de Sentimentos**, tendo sido possível identificar uma solução **eficiente**, **interpretável** e **computacionalmente viável** para o problema proposto.

REFERÊNCIAS BIBLIOGRÁFICAS

GANESH. **SentimentalAnalysisofAmazonReviews**.GitHub, 2019. Disponível em: https://github.com/ganesh292/SentimentalAnalysisofAmazonReviews. Acesso em: 3 fev. 2026.

KOKJE, Richa Chunekar; CHOUHAN, Gajendra Singh. **A Supervised Learning Technique for Classifying Amazon Product Reviews based on Buyers Sentiments**. International Journal of Computer Applications, v. 175, n. 38, p. 36-41, dez. 2020. Disponível em: https://www.ijcaonline.org/archives/volume175/number38/kokje-2020-ijca-920956.pdf. Acesso em: 3 fev. 2026.

NABIL, Sana; ELBOUHDIDI, Jaber; CHKOURI, Mohamed Yassin. **Sentiment Analysis of Amazon's Reviews Using Machine Learning Algorithms**. Journal of Theoretical and Applied Information Technology, v. 99, n. 22, p. 5571-5581, 30 nov.

2021. Disponível em: http://www.jatit.org/volumes/Vol99No22/30Vol99No22.pdf. Acesso em: 3 fev. 2026.

ALHARBI, Ibrahim, et al. **Evaluation of Sentiment Analysis via Word Embedding and RNN Variants for Amazon Online Reviews**. Wireless Communications and Mobile Computing, 2021. Disponível em: https://onlinelibrary.wiley.com/doi/10.1155/2021/5536560. Acesso em: 3 fev. 2026.

Classifying the sentiment of product reviews using multiple vectorization methods: a machine learning and deep learning approach. arno.uvt.nl, [s.d.]. Disponível em: http://arno.uvt.nl/show.cgi?fid=171019. Acesso em: 3 fev. 2026.

QORICH, Mohammed. **Text sentiment classification of Amazon reviews using word embeddings and convolutional neural network**. Bohrium, 17 fev. 2023. Disponível em: https://www.bohrium.com/en/paper-details/text-sentiment-classification-of-amazon-reviews-using-word-embeddings-and-convolutional-neural-networks/865003561392537992-2679. Acesso em: 3 fev. 2026.

ANEXOS

- **Avaliação/Desempenho dos modelos quando usado undersampling**

  - **Resultados com Bag of Words**

Classes	KNN			Naive Bayes			Logistic Regression		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,80	0,24	0,37	0,72	0,79	0,75	0,81	0,82	0,81
Neutral	0,45	0,91	0,61	0,64	0,39	0,49	0,70	0,75	0,72
Positive	0,81	0,55	0,66	0,68	0,87	0,77	0,86	0,81	0,83

Classes	Decision Trees			SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,77	0,80	0,78	0,79	0,82	0,81	0,80	0,85	0,82
Neutral	0,71	0,72	0,72	0,73	0,77	0,75	0,79	0,79	0,79
Positive	0,80	0,76	0,78	0,89	0,81	0,84	0,88	0,81	0,85

Métricas Globais	KNN		Naive Bayes		Logistic Regression	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,57	0,55	0,69	0,67	0,79	0,79

Métricas Globais	Decision Trees		SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,76	0,76	0,80	0,80	0,82	0,82

  - **Resultados com TF-IDF**

Classes	KNN			Naive Bayes			Logistic Regression		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	1,00	0,02	0,04	0,68	0,83	0,75	0,75	0,84	0,79
Neutral	0,34	0,97	0,51	0,66	0,40	0,50	0,70	0,68	0,69
Positive	0,80	0,12	0,20	0,72	0,85	0,78	0,87	0,79	0,83

Classes	Decision Trees			SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,74	0,75	0,74	0,75	0,80	0,78	0,76	0,88	0,81
Neutral	0,68	0,69	0,69	0,68	0,70	0,69	0,81	0,73	0,77
Positive	0,76	0,74	0,75	0,89	0,79	0,84	0,86	0,81	0,83

Métricas Globais	KNN		Naive Bayes		Logistic Regression	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,37	0,25	0,69	0,68	0,77	0,77

Métricas Globais	Decision Trees		SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,73	0,73	0,76	0,77	0,81	0,81

- **Resultados com Word Embeddings (GloVe)**

Classes	KNN			Logistic Regression		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,57	0,84	0,68	0,67	0,76	0,71
Neutral	0,72	0,36	0,48	0,54	0,49	0,52
Positive	0,69	0,71	0,70	0,75	0,73	0,74

Classes	Decision Trees			SVM			Random Forest		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Negative	0,64	0,71	0,67	0,65	0,78	0,71	0,75	0,80	0,78
Neutral	0,60	0,55	0,57	0,55	0,49	0,52	0,77	0,74	0,76
Positive	0,68	0,67	0,67	0,77	0,70	0,74	0,81	0,80	0,81

Métricas Globais	KNN		Logistic Regression	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,64	0,62	0,66	0,66

Métricas Globais	Decision Trees		SVM		Random Forest	
	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)	Accuracy	F1-Score (Weighted)
	0,64	0,64	0,66	0,65	0,78	0,78