Detecção de sentimentos usando NLP

- Hildemir Regis
- José Anderson de Souza

Especialização
Deep Learning

Centro de Informática
UFPE

Universidade Federal de Pernambuco

Objetivo

- Treinar classificador usando uma base de dados de revisões de produtos para analisar sentimento dos clientes através dos comentários.
- Apresentar os resultados em:
  - SVM + bow;
  - SVM + embeddings;
  - BERT

Análise dos Dados

Base de dados

- Encontramos uma base de produtos femininos com os dados necessários para trabalhar em cima do treinamento: comentário e nota;
- **23.486** registros;
- Fonte: https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews

Catálogo de dados

- **Unnamed:** índice;
- **Clothing ID:** índice do produto;
- **Age:** idade do cliente avaliador;
- **Title:** título da avaliação;
- **Review Text:** texto do comentário;
- **Rating:** nota;
- **Recommended IND:** índice de recomendação;
- **Positive Feedback Count:** contagem de feedbacks positivos;
- **Division Name:** nome da divisão;
- **Department Name:** nome do departamento;
- **Class Name:** nome da classe;

Pré-processamento

Remover notas e comentários nulos e vazios

```
df = df[df['Rating'].notna()]
df = df[df['Review Text'].notna()]
df = df[df['Rating'] != '']
df = df[df['Review Text'] != '']
```

Transformação das notas em classes categóricas

- Nota <= 2: 0 (negativa);
- Nota = 3: 1 (neutra);
- Nota > 3: 2 (positiva)

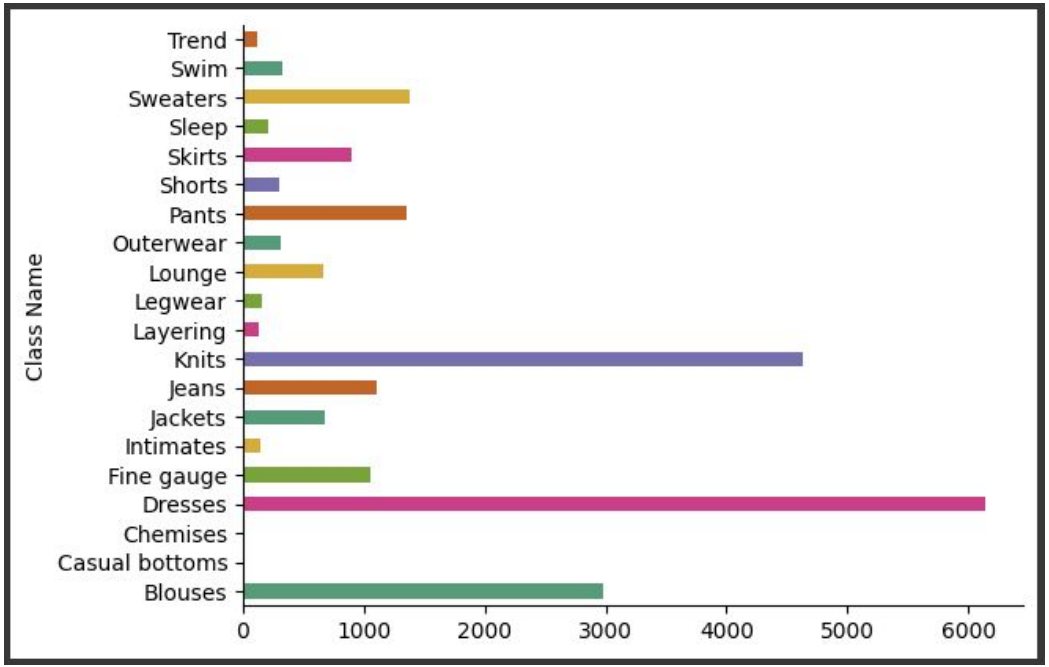
```
def convert_rating(rating):
    if 1 <= rating <= 2:
        return 0
    elif rating == 3:
        return 1
    else:
        return 2

df['Rating'] = df['Rating'].apply(convert_rating)
```

Análise descritiva e exploratória dos dados (EDA)

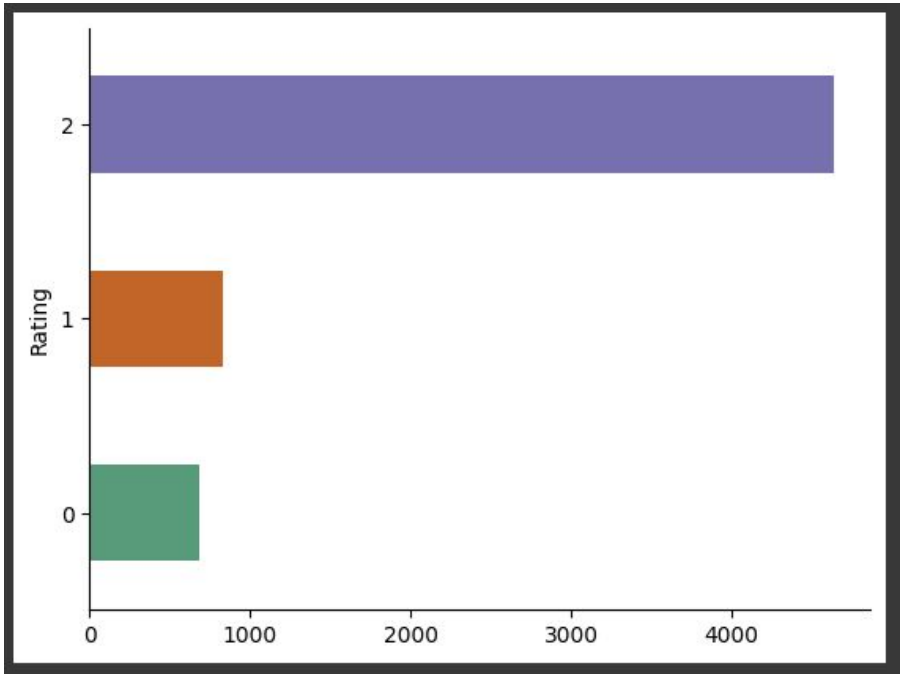
Métricas descritivas

	Unnamed: 0	Clothing ID	Age	Rating	Recommended IND	Positive Feedback Count
count	22641.000000	22641.000000	22641.000000	22641.000000	22641.000000	22641.000000
mean	11740.849035	919.332362	43.280376	1.665960	0.818868	2.630582
std	6781.957509	202.266874	12.326980	0.657139	0.385136	5.786164
min	0.000000	1.000000	18.000000	0.000000	0.000000	0.000000
25%	5872.000000	861.000000	34.000000	2.000000	1.000000	0.000000
50%	11733.000000	936.000000	41.000000	2.000000	1.000000	1.000000
75%	17621.000000	1078.000000	52.000000	2.000000	1.000000	3.000000
max	23485.000000	1205.000000	99.000000	2.000000	1.000000	122.000000

Distribuição das classes de produtos



Distribuição de avaliações da classe dresses



Modelagem

Modelo SVM + Bag of Words

Arquitetura:

- **Entrada:** Dados de texto;
- **Processamento (vetorização BoW):** Transformação do texto em matrizes numéricas para treinamento e teste (Limitação de 5000 características);
- **Modelo de Classificação (SVM):** SVM com kernel linear é treinado com as características extraídas;
- **Saída:** Modelo treinado;

Hiperparâmetros:

- max_features=5000
- kernel='linear'
- probability=True

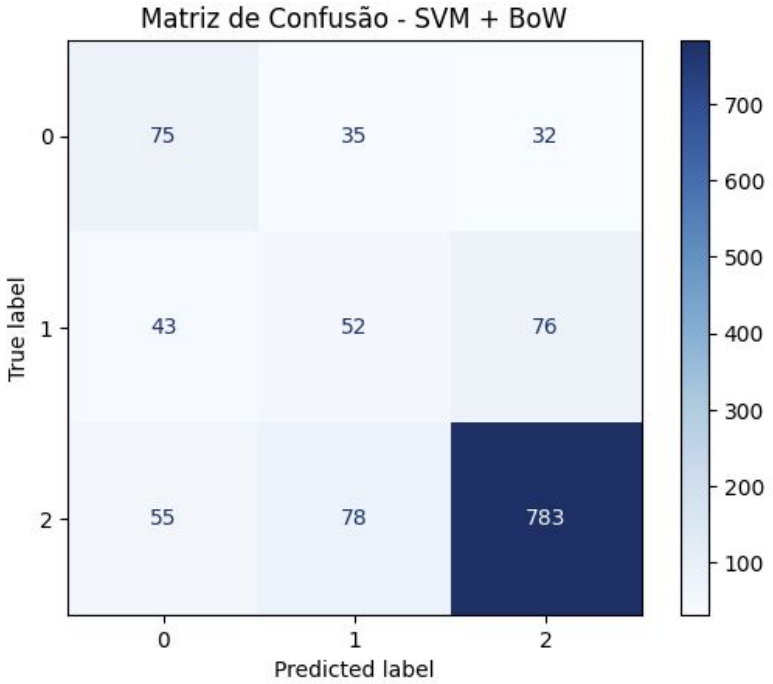
Modelo SVM + Bag of Words

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer
     from sklearn.svm import SVC
     from sklearn.metrics import classification_report

     # Vetorização com BoW
     vectorizer = CountVectorizer(max_features=5000)
     X_train_bow = vectorizer.fit_transform(X_train).toarray()
     X_test_bow = vectorizer.transform(X_test).toarray()

     # Treinar o SVM
     svm_bow = SVC(kernel='linear', probability=True, class_weight='balanced')
     svm_bow.fit(X_train_bow, y_train)

     # Avaliar
     y_pred_bow = svm_bow.predict(X_test_bow)
     print("SVM + BoW")
     print(classification_report(y_test, y_pred_bow))
```

Modelo SVM + Bag of Words

SVM + BoW

	precision	recall	f1-score	support
0	0.44	0.51	0.47	142
1	0.33	0.31	0.32	171
2	0.88	0.86	0.87	916
accuracy			0.74	1229
macro avg	0.55	0.56	0.55	1229
weighted avg	0.75	0.74	0.75	1229



Modelo SVM + Embeddings

Arquitetura:

- **Entrada:** Dados de texto;
- **Processamento (Word2Vec):** Tokenização, treinamento do Word2Vec e geração de embeddings das sentença;
- **Modelo de Classificação (SVM):** SVM com kernel linear é treinado com os embeddings das sentenças como entradas.
- **Saída:** Modelo treinado baseado nos embeddings;

Hiperparâmetros:

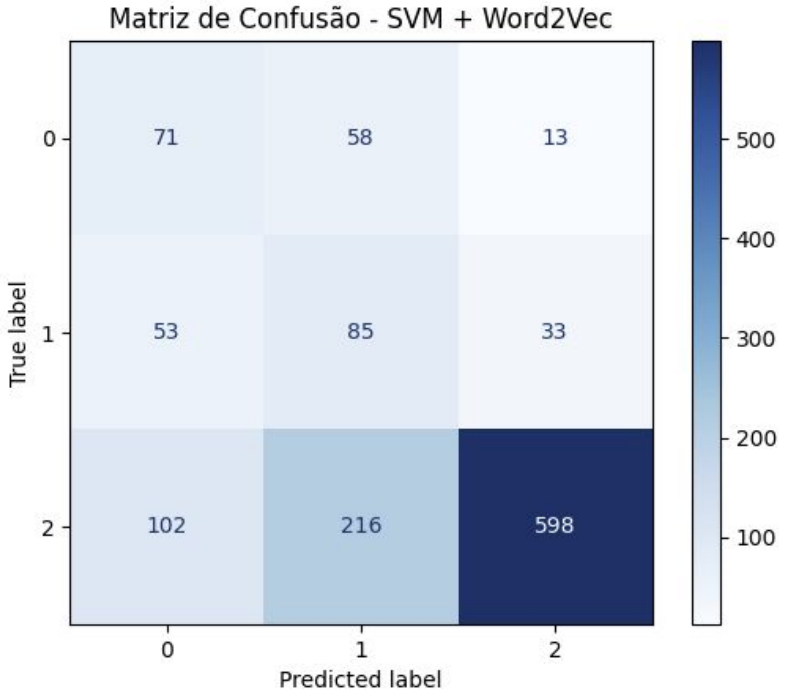
- **Word2Vec:**
  - vector_size=100;
  - window=5;
  - min_count=1;
  - workers=4;
- **SVM:**
  - kernel='linear'
  - probability=True

Modelo SVM + Embeddings

```
[ ] import numpy as np
    import nltk

    from gensim.models import Word2Vec
    nltk.download('punkt_tab')

    # Criando Embeddings com Word2Vec
    tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in X_train]
    word2vec_model = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, min_count=1, workers=4)

    def get_sentence_embedding(sentence):
        tokens = nltk.word_tokenize(sentence)
        embeddings = [word2vec_model.wv[word] for word in tokens if word in word2vec_model.wv]
        return np.mean(embeddings, axis=0) if embeddings else np.zeros(100)

    X_train_emb = np.array([get_sentence_embedding(sentence) for sentence in X_train])
    X_test_emb = np.array([get_sentence_embedding(sentence) for sentence in X_test])

    # Treinar o SVM
    svm_emb = SVC(kernel='linear', probability=True, class_weight='balanced')
    svm_emb.fit(X_train_emb, y_train)

    # Avaliar
    y_pred_emb = svm_emb.predict(X_test_emb)
    print("SVM + Word Embeddings")
    print(classification_report(y_test, y_pred_emb))
```

Modelo SVM + Embeddings

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
SVM + Word Embeddings
              precision    recall  f1-score   support

           0       0.31      0.47      0.37       142
           1       0.24      0.50      0.33       171
           2       0.92      0.66      0.77       916

    accuracy                           0.61      1229
   macro avg       0.49      0.54      0.49      1229
weighted avg       0.75      0.61      0.66      1229
```



Modelo BERT

Arquitetura:

- **Entrada:** Dados de texto;
- **Tokenização**: BERT Tokenizer (max_len=128);
- **Modelo de Classificação (BERT):** BertForSequenceClassification é carregado com o pré-treinamento do BERT (bert-base-uncased);
- **Treinamento:** modelo é treinado utilizando o otimizador AdamW com uma taxa de aprendizado de 5e-5.
- **Saída:** Modelo treinado;

Hiperparâmetros:

- **Tokenização:**
  - max_len=128;
- **Modelo BERT:**
  - bert-base-uncased;
  - num_labels=3;
- **Treinamento:**
  - Número de épocas: 3;
  - batch_size=16;
  - Taxa de aprendizado: 5e-5;
  - Otimizador: AdamW;

Modelo BERT

```python
# Tokenizer e dataset
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_dataset = SentimentDataset(X_train, y_train, tokenizer)
test_dataset = SentimentDataset(X_test, y_test, tokenizer)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Modelo
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)

# Treinamento
model.train()
for epoch in range(3):  # 3 épocas
    for batch in train_loader:
        optimizer.zero_grad()
        input_ids, attention_mask, labels = batch['input_ids'], batch['attention_mask'], batch['label']
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

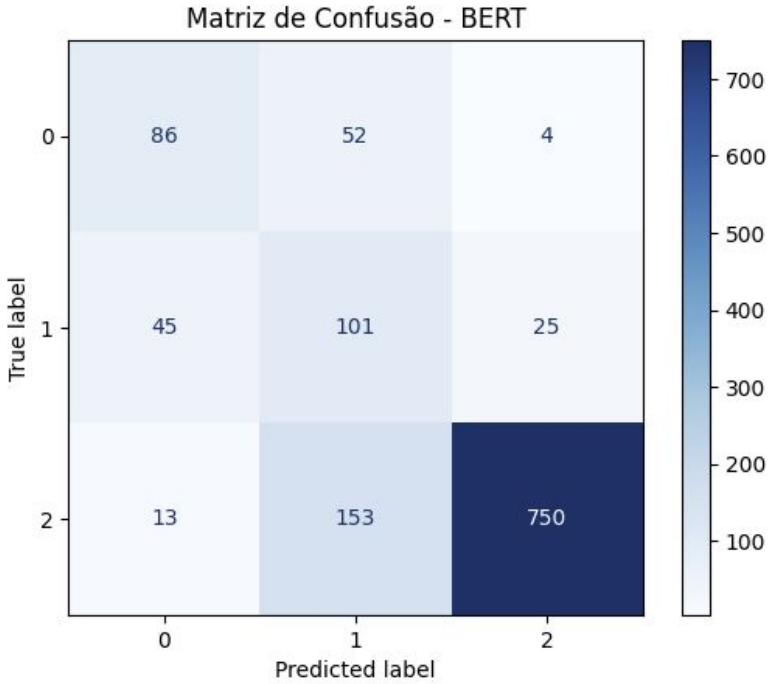
Modelo BERT

```
# Avaliação
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        input_ids, attention_mask, labels = batch['input_ids'], batch['attention_mask'], batch['label']
        outputs = model(input_ids, attention_mask=attention_mask)
        preds = torch.argmax(F.softmax(outputs.logits, dim=1), dim=1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

print("BERT")
print(classification_report(all_labels, all_preds))
```

Modelo BERT

BERT

	precision	recall	f1-score	support
0	0.60	0.63	0.62	142
1	0.37	0.47	0.42	171
2	0.94	0.88	0.91	916
accuracy			0.80	1229
macro avg	0.64	0.66	0.65	1229
weighted avg	0.82	0.80	0.81	1229



Análises

SVM + BoW

	precision	recall	f1-score	support
0	0.44	0.51	0.47	142
1	0.33	0.31	0.32	171
2	0.88	0.86	0.87	916
accuracy			0.74	1229
macro avg	0.55	0.56	0.55	1229
weighted avg	0.75	0.74	0.75	1229

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

SVM + Word Embeddings

	precision	recall	f1-score	support
0	0.31	0.47	0.37	142
1	0.24	0.50	0.33	171
2	0.92	0.66	0.77	916
accuracy			0.61	1229
macro avg	0.49	0.54	0.49	1229
weighted avg	0.75	0.61	0.66	1229

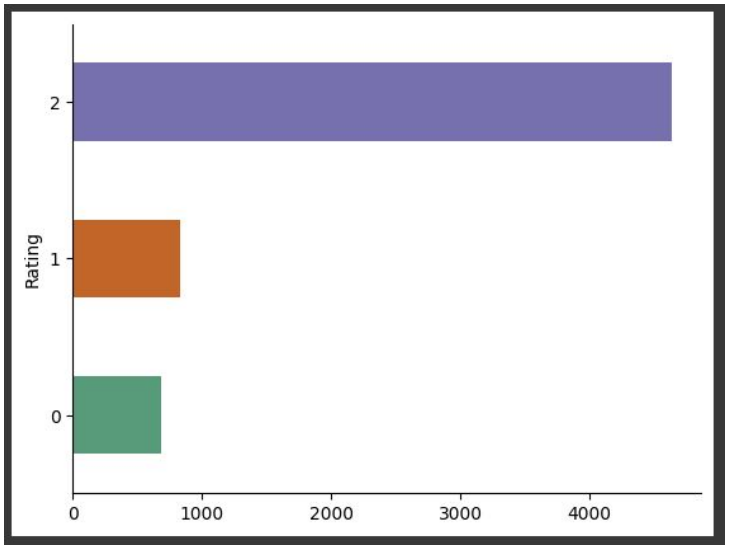
BERT

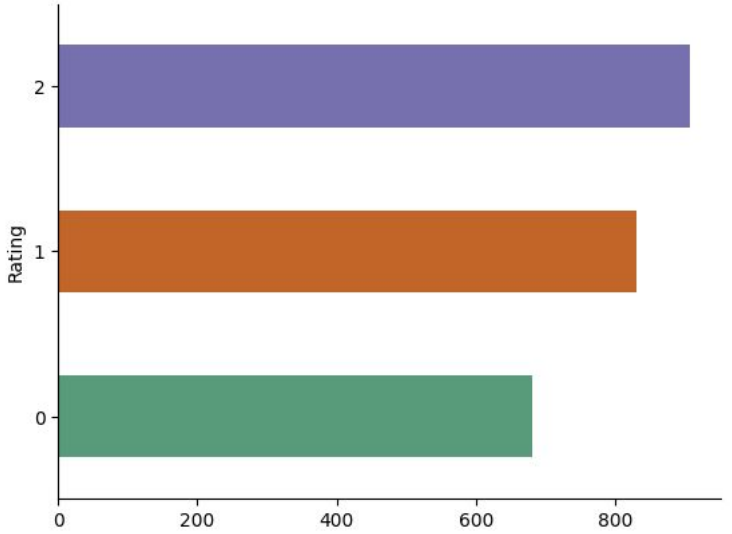
	precision	recall	f1-score	support
0	0.60	0.63	0.62	142
1	0.37	0.47	0.42	171
2	0.94	0.88	0.91	916
accuracy			0.80	1229
macro avg	0.64	0.66	0.65	1229
weighted avg	0.82	0.80	0.81	1229

Balanceamento das classes

```
print("Distribuição original:")
print(df_filtrado['Rating'].value_counts())

df_class_0 = df_filtrado[df_filtrado['Rating'] == 0]
df_class_1 = df_filtrado[df_filtrado['Rating'] == 1]
df_class_2 = df_filtrado[df_filtrado['Rating'] == 2]

mean_minority_size = (len(df_class_0) + len(df_class_1)) // 2
target_size_class_2 = int(mean_minority_size * 1.2)

df_class_2_sampled = df_class_2.sample(n=target_size_class_2, random_state=42)

# Reunir todas as classes
df_balanced = pd.concat([df_class_0, df_class_1, df_class_2_sampled], axis=0)

# Verificar nova distribuição
print("\nDistribuição após ajuste:")
print(df_balanced['Rating'].value_counts())
```

Antes x Depois





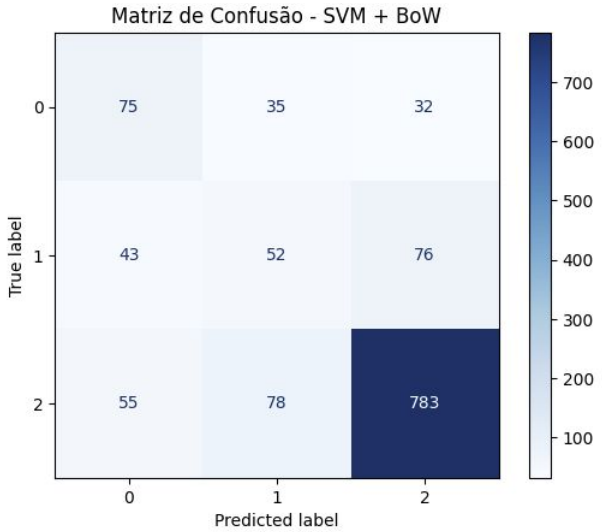
Modelo SVM + Bag of Words (antes x depois)

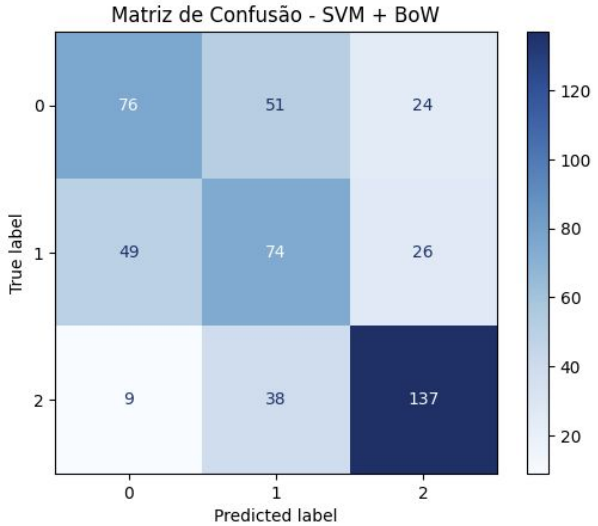
SVM + BoW

	precision	recall	f1-score	support
0	0.44	0.51	0.47	142
1	0.33	0.31	0.32	171
2	0.88	0.86	0.87	916
accuracy			0.74	1229
macro avg	0.55	0.56	0.55	1229
weighted avg	0.75	0.74	0.75	1229

SVM + BoW

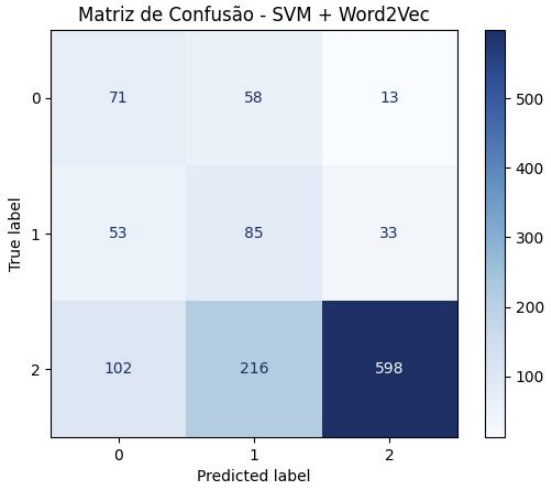
	precision	recall	f1-score	support
0	0.57	0.50	0.53	151
1	0.45	0.50	0.47	149
2	0.73	0.74	0.74	184
accuracy			0.59	484
macro avg	0.58	0.58	0.58	484
weighted avg	0.60	0.59	0.59	484



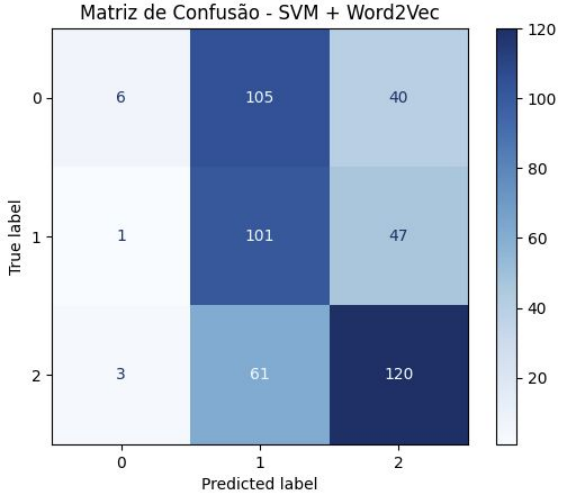


Modelo SVM + Embeddings (antes x depois)

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
SVM + Word Embeddings
              precision    recall  f1-score   support

           0       0.31      0.47      0.37       142
           1       0.24      0.50      0.33       171
           2       0.92      0.66      0.77       916

    accuracy                           0.61      1229
   macro avg       0.49      0.54      0.49      1229
weighted avg       0.75      0.61      0.66      1229
```



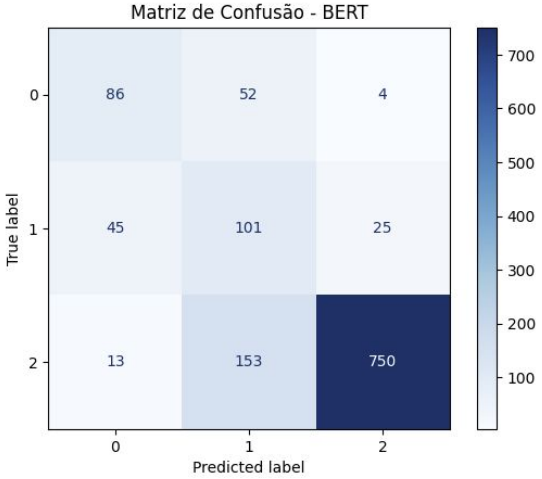

```
SVM + Word Embeddings
              precision    recall  f1-score   support

           0       0.60      0.04      0.07       151
           1       0.38      0.68      0.49       149
           2       0.58      0.65      0.61       184

    accuracy                           0.47       484
   macro avg       0.52      0.46      0.39       484
weighted avg       0.52      0.47      0.41       484
```



Modelo BERT

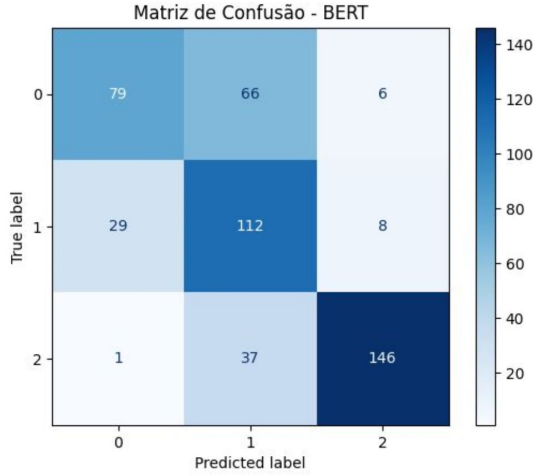
BERT

	precision	recall	f1-score	support
0	0.60	0.63	0.62	142
1	0.37	0.47	0.42	171
2	0.94	0.88	0.91	916
accuracy			0.80	1229
macro avg	0.64	0.66	0.65	1229
weighted avg	0.82	0.80	0.81	1229



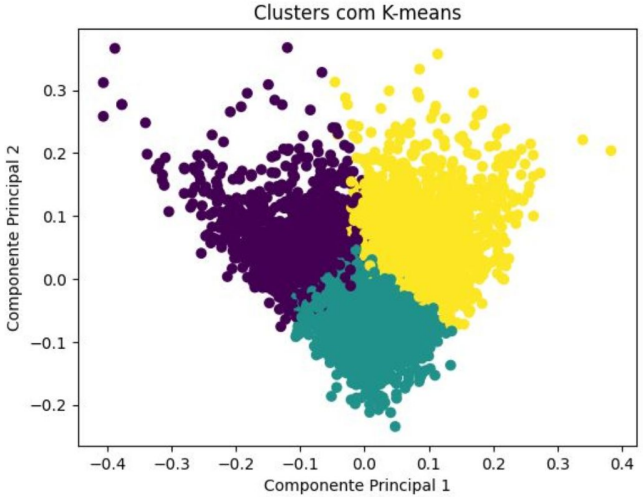
BERT

	precision	recall	f1-score	support
0	0.72	0.52	0.61	151
1	0.52	0.75	0.62	149
2	0.91	0.79	0.85	184
accuracy			0.70	484
macro avg	0.72	0.69	0.69	484
weighted avg	0.73	0.70	0.70	484



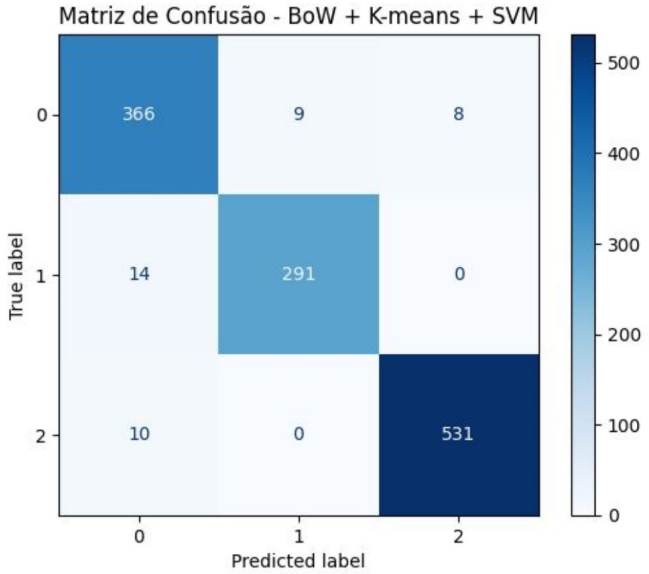
Análises

- Mesmo após o balanceamento e tentativas de separação manual não obtivemos os resultados desejados;
- Logo seguimos com outra estratégia: usar o **K-means** para agrupar melhor as fronteiras de decisão.



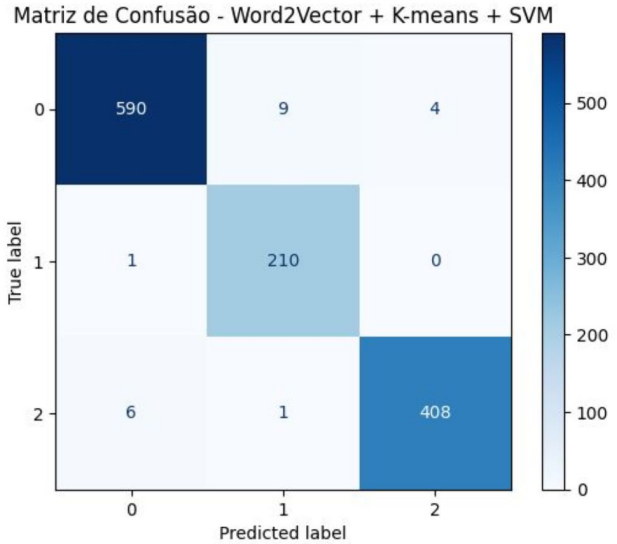
BoW + K-means + SVM

	precision	recall	f1-score	support
0	0.94	0.96	0.95	383
1	0.97	0.95	0.96	305
2	0.99	0.98	0.98	541
accuracy			0.97	1229
macro avg	0.96	0.96	0.96	1229
weighted avg	0.97	0.97	0.97	1229



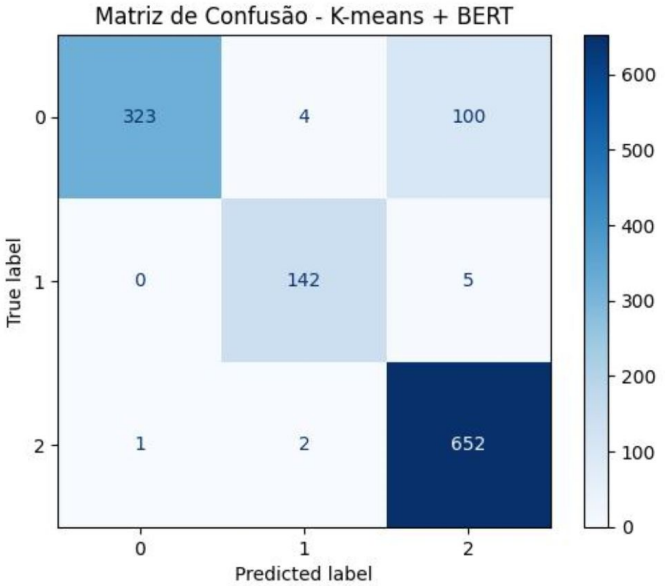
Word2Vector + K-means+ SVM

	precision	recall	f1-score	support
0	0.99	0.98	0.98	603
1	0.95	1.00	0.97	211
2	0.99	0.98	0.99	415
accuracy			0.98	1229
macro avg	0.98	0.99	0.98	1229
weighted avg	0.98	0.98	0.98	1229



K-means + BERT

	precision	recall	f1-score	support
0	1.00	0.76	0.86	427
1	0.96	0.97	0.96	147
2	0.86	1.00	0.92	655
accuracy			0.91	1229
macro avg	0.94	0.91	0.92	1229
weighted avg	0.92	0.91	0.91	1229



Conclusão

- A separação manual das classificações não foi suficiente. Com o K-means conseguimos aprender a fronteira de decisão e usamos ele como input para os modelos.
- O bert é bem mais custoso. E o outros tiveram resultados melhores com menos custo.
- Às vezes o modelo mais simples já resolve problema. Já outras vezes é preciso avaliar melhor a metodologia e combiná-la com outra pode acabar otimizando o trabalho.

Obrigado

Especialização
Deep Learning

Centro de Informática
UFPE

Universidade Federal de Pernambuco