Inf1038
Clusterização e Classificação de Texto

Grupo:

- Eduardo Luna: 2111484
- Luca Ribeiro: 2112662
- Lucas Larios: 2020723

Análise Exploratória:

O dataset selecionado para a G2 é uma base de reviews de roupas da Amazon, podendo ser encontrado no Kaggle.

Realizando uma análise exploratória sobre os dados, podemos observar que o dataset possui 49338 samples e 9 colunas. Dentre essas 9, 2 são textuais (título e crítica), uma é categórica (Cloth_class), a qual definimos como o target das predições, enquanto as demais features são numéricas.

	Title	Review	Cons_rating	Cloth_class	Materials	Construction	Color	Finishing	Durability
0	NaN	Absolutely wonderful - silky and sexy and comf...	4.0	Intimates	0.0	0.0	0.0	1.0	0.0
1	NaN	Love this dress! it's sooo pretty. i happene...	5.0	Dresses	0.0	1.0	0.0	0.0	0.0
2	Some major design flaws	I had such high hopes for this dress and reall...	3.0	Dresses	0.0	0.0	0.0	1.0	0.0
3	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5.0	Pants	0.0	0.0	0.0	0.0	0.0
4	Flattering shirt	This shirt is very flattering to all due to th...	5.0	Blouses	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
49333	Dress felt and fit great. I got lots of compl...	Loved the color!!! Dress fit great and I got ...	5.0	Dresses	0.0	0.0	1.0	0.0	0.0
49334	Loved the dress but poor quality	This dress looked great and I loved the materi...	2.0	Dresses	1.0	0.0	0.0	0.0	1.0
49335	Cute dress, didn't fit	Wanted this dress to work it didn't. It is ver...	1.0	Dresses	0.0	1.0	0.0	0.0	0.0
49336	Very cute!	No complaints othe than the zipper gets stuck ...	4.0	Dresses	0.0	0.0	0.0	0.0	1.0
49337	Good fit	The fabric was really nice, I'm a L and it fit...	5.0	Dresses	1.0	1.0	0.0	0.0	0.0

49338 rows × 9 columns

Boa parte dos dados sobre as features numéricas (com exceção da avaliação dos usuários) é nula no dataset.

```
nulls = pd.DataFrame(df.isnull().sum(), columns = ['Nulls'])

display(nulls.T)
```

	Title	Review	Cons_rating	Cloth_class	Materials	Construction	Color	Finishing	Durability
Nulls	3968	831	214	16	43597	43595	43596	43601	43604

Como um dos principais objetivos do trabalho é estudar e analisar sobre NLP, essas colunas numéricas com muitos dados ausentes foram removidas. Para as colunas não numéricas foram removidos os samples em que algum se encontrava nulo.

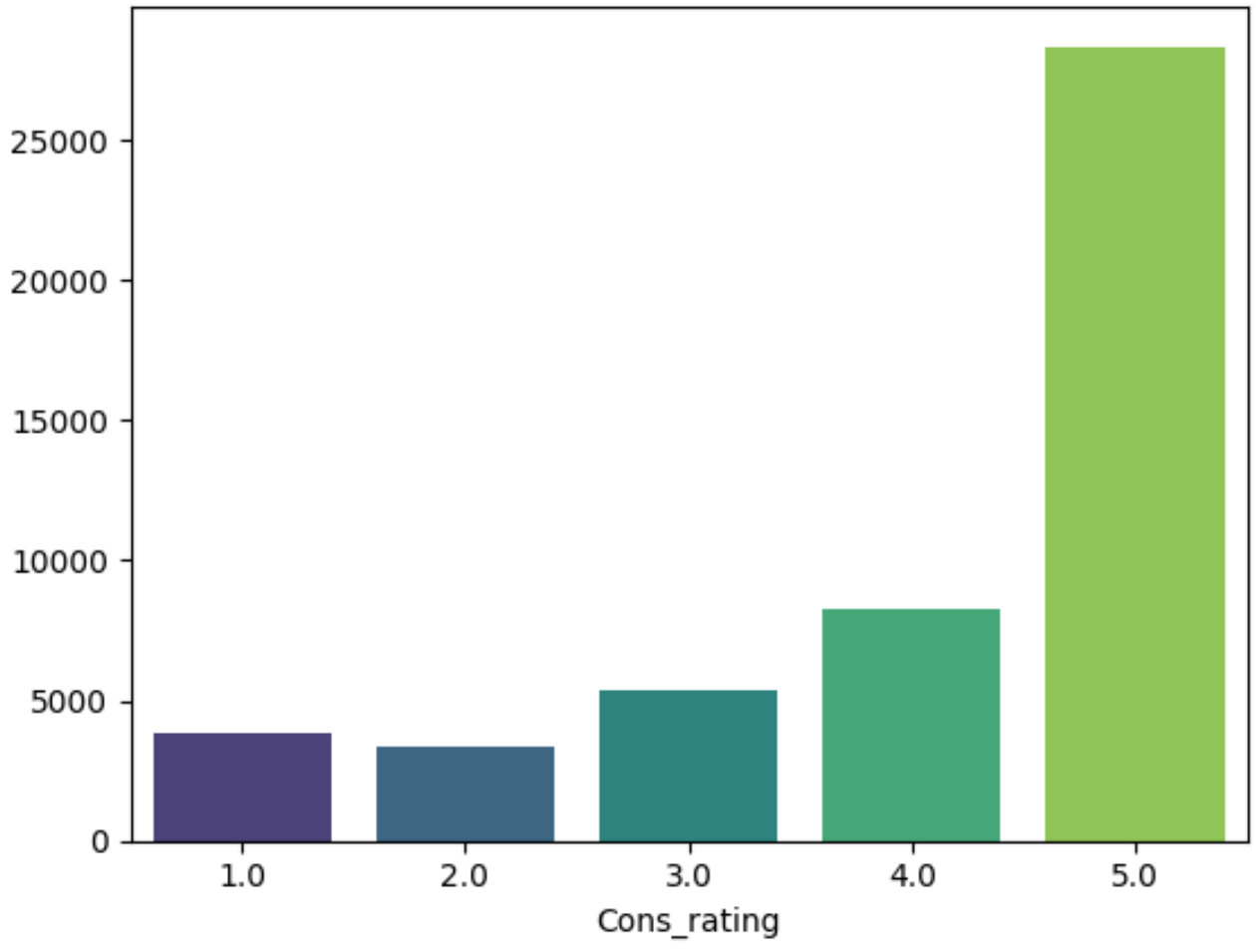
```
clean_df = df.dropna(subset = ['Title', 'Review', 'Cloth_class']).drop(columns = ['Materials', 'Construction', 'Color', 'Finishing', 'Durability'])

display(clean_df)
```

	Title	Review	Cons_rating	Cloth_class
2	Some major design flaws	I had such high hopes for this dress and reall...	3.0	Dresses
3	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5.0	Pants
4	Flattering shirt	This shirt is very flattering to all due to th...	5.0	Blouses
5	Not for the very petite	I love tracy reese dresses, but this one is no...	2.0	Dresses
6	Cagrcoal shimmer fun	I aded this in my basket at hte last mintue to...	5.0	Knits
...	...	...	...	...
49333	Dress felt and fit great. I got lots of compl...	Loved the color!!! Dress fit great and I got ...	5.0	Dresses
49334	Loved the dress but poor quality	This dress looked great and I loved the materi...	2.0	Dresses
49335	Cute dress, didn't fit	Wanted this dress to work it didn't. It is ver...	1.0	Dresses
49336	Very cute!	No complaints othe than the zipper gets stuck ...	4.0	Dresses
49337	Good fit	The fabric was really nice, I'm a L and it fit...	5.0	Dresses

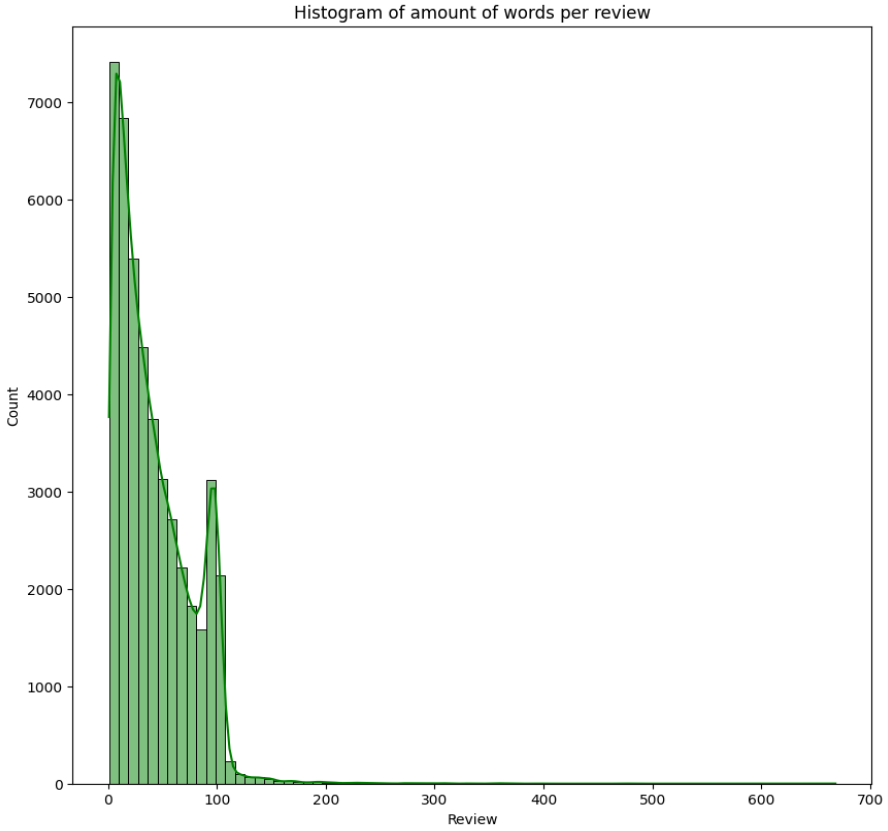
45308 rows × 4 columns

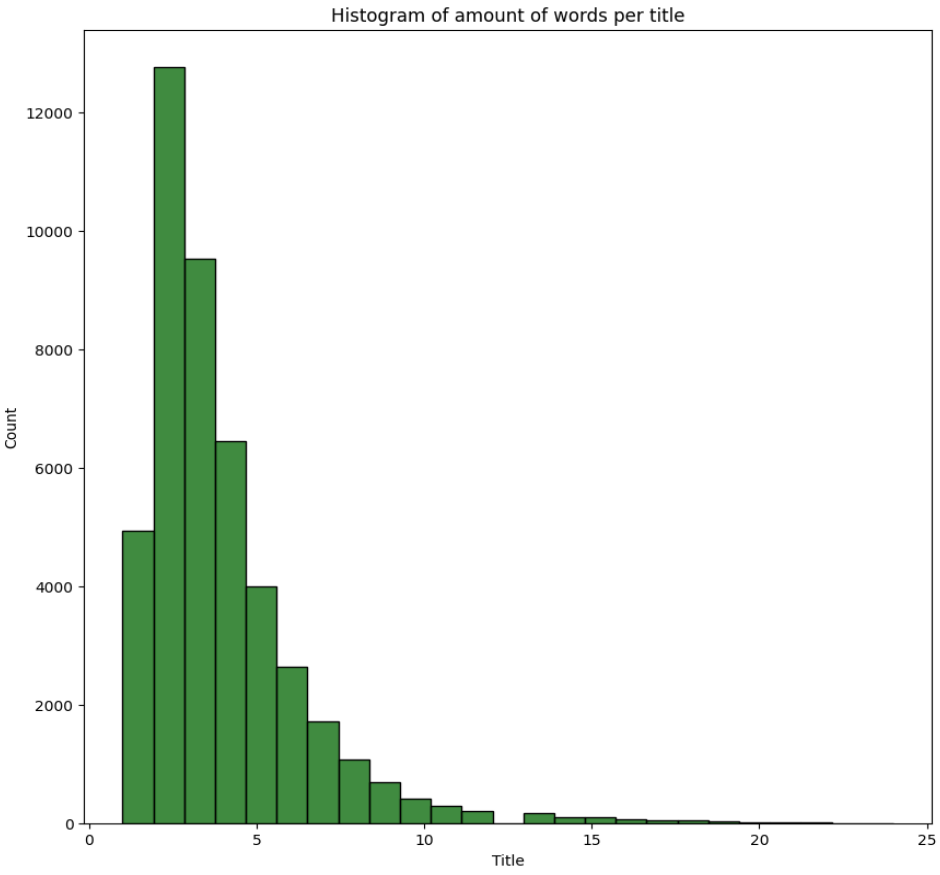
Dessa forma, restaram 45308 samples em que as features textuais e o target se encontram não nulos.

Para a distribuição das avaliações dos usuários:

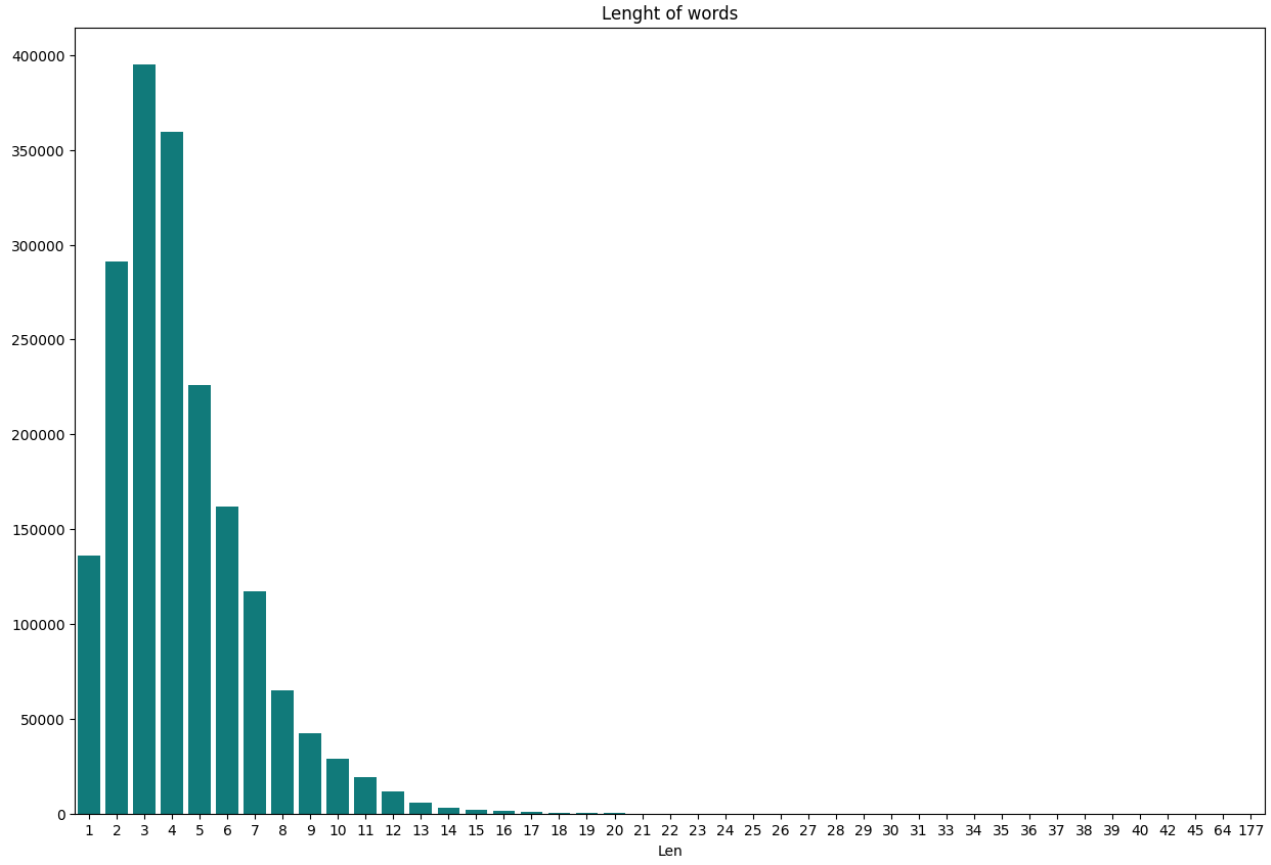


Fazendo um histograma para visualizar a quantidade de palavras por review e por título:

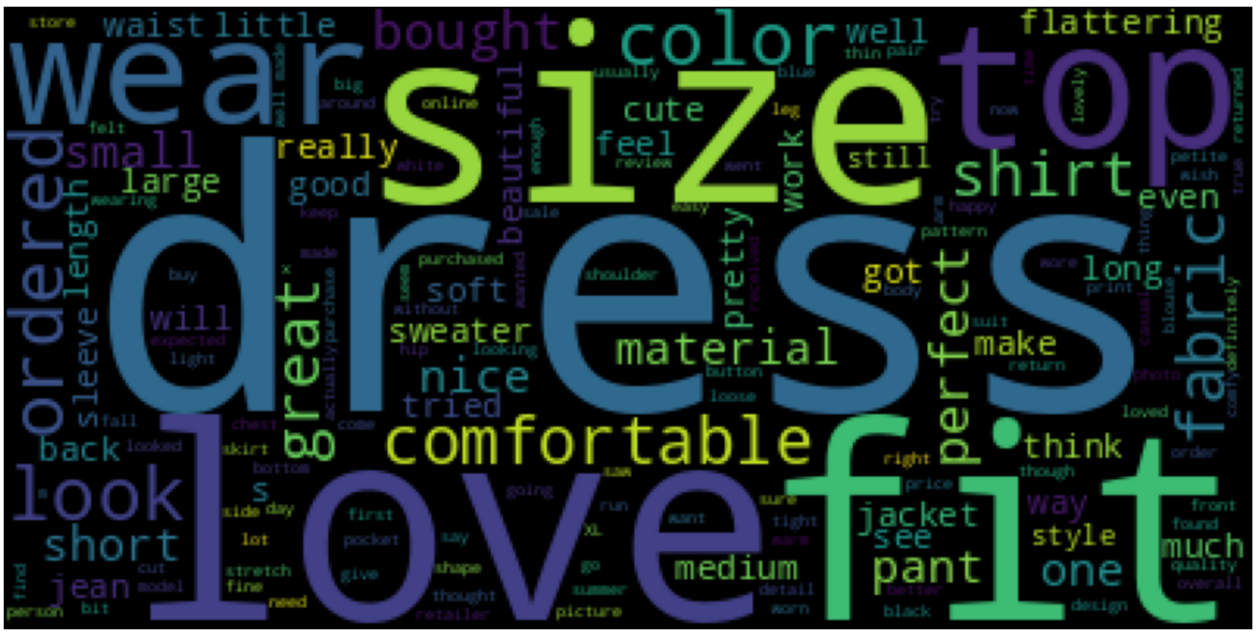




Histograma com distribuição dos tamanhos das palavras nas reviews do dataset:

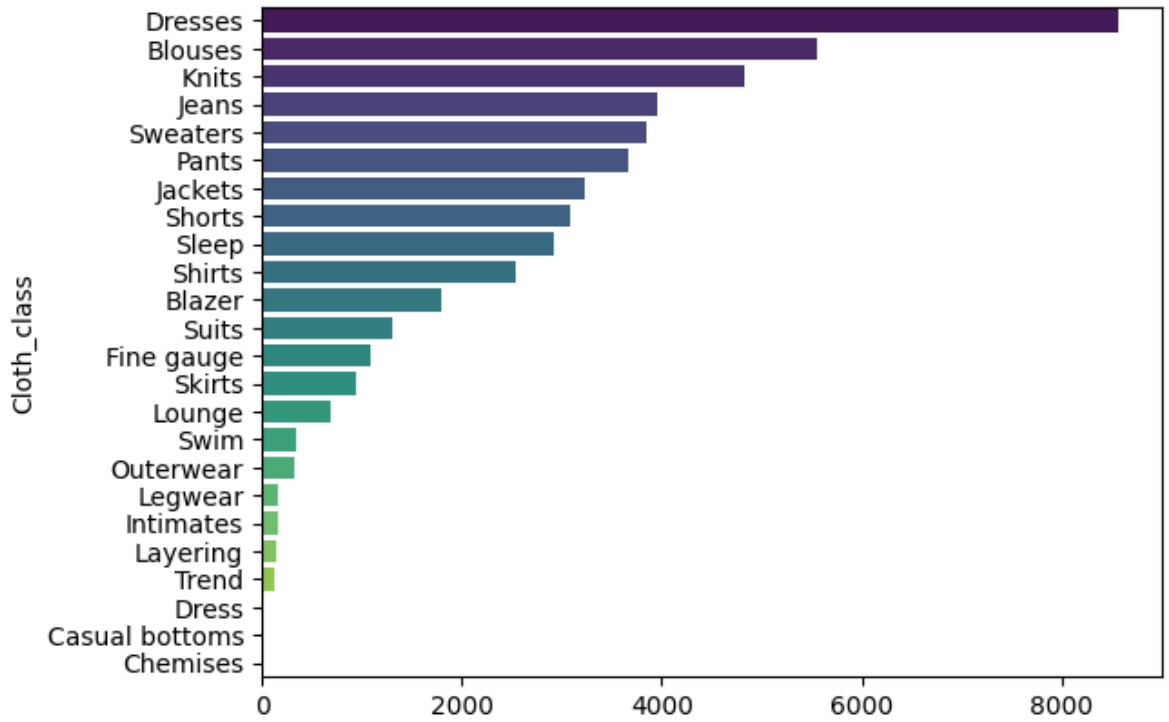


WordCloud com palavras mais frequentes (sem contar stopwords):



Analisando a distribuição dos dados no target (classe de roupas), foi encontrado o problema de que algumas das 24 classes possuem muitos poucos samples, o que tornaria inviável para que os modelos preditivos conseguissem classificá-los.

```
classes = df['Cloth_class'].value_counts()

sns.barplot(y = classes.index, x = classes.values, palette = 'viridis', orient = 'h')

plt.show()
```



Para contornar isso, inicialmente foram realizadas técnicas de aprendizado não supervisionado visando a encontrar classes que fossem próximas umas das outras de acordo com as características textuais, utilizando múltiplas combinações de métodos de redução de dimensionalidade (PCA, Truncated SVD, UMAP e t-SNE) com algoritmos de clusterização (KMeans, Bisecting KMeans e Cluster Aglomerativo). Entretanto, somente isso não retornou bons resultados para encontrar classes similares.

Utilizando a biblioteca SpaCy, voltada para processamento de linguagem natural, foi feita uma similaridade de palavras entre as classes, buscando já reduzir o número. Sendo definido um threshold de 1500, as classes foram separadas em 2 grupos: com número de samples maior ou igual ao threshold e com número menor. A partir disso, foi possível, após fazer a stemização (redução das palavras para o stem ou raiz) dos nomes das classes, compará-las e substituir todas as classes em que houve uma similaridade acima de 68%, reduzindo, assim, o número de classes para 16.

Foram criadas as seguintes funções auxiliares para tal análise, a primeira para gerar a visualização do HeatMap de similaridade de texto das classes e a segunda para fazer a substituição das classes com pouco samples pela classe mais próxima acima do threshold (caso a similaridade seja maior que o valor passado no parâmetro min_replacer):

```
def plot_words_similarity(matrix, under_threshold, over_threshold):
    fig, ax = plt.subplots(figsize = (10, 10))

    sns.heatmap(matrix,
                cmap = 'viridis',
                xticklabels = over_threshold,
                yticklabels = under_threshold,
                annot = True
               ).set(xlabel = 'Classes over threshold', ylabel = 'Classes under threshold')

    plt.show()
```

```
def word_similarity_replacer(cloth_class, max_similarities, min_replacer = 0.5):
    under_threshold = set(max_similarities.keys())

    def replacer(x):
        if x not in under_threshold or max_similarities[x][1] < min_replacer: return x
        else: return max_similarities[x][0]

    return cloth_class.apply(replacer)
```

Definindo o stemmer para inglês (idioma mais predominante nas reviews no dataset), permitindo reduzir as palavras para a raiz:

```
stemmer = SnowballStemmer('english')
```

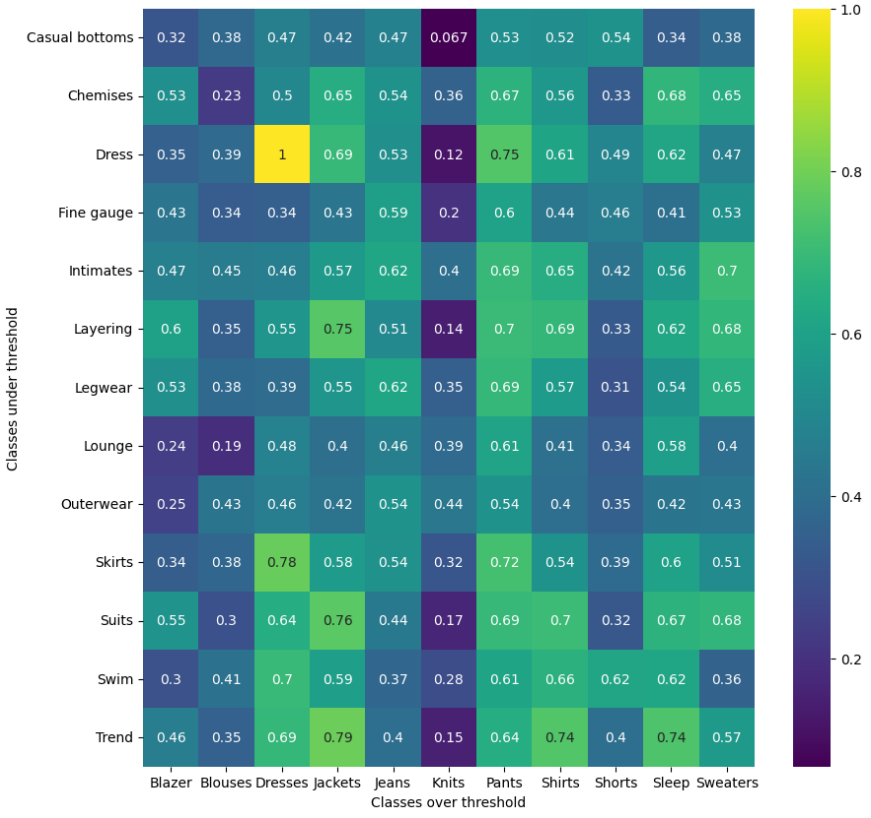
Separando as classes com base no threshold:

```
threshold = 1500

classes_list_over = list(classes[classes.values >= threshold].index)
classes_list_under = list(classes[classes.values < threshold].index)

classes_list_over.sort()
classes_list_under.sort()
```

Gerando a matriz e encontrando a classe mais próxima:

```
matrix = list()
max_similarities = dict()

for under in classes_list_under:
    under_stem = stemmer.stem(under)

    under_token = nlp(under_stem)

    row = list()

    max_sim = float('-inf')
    max_sim_token = ''

    for over in classes_list_over:
        over_stem = stemmer.stem(over)

        over_token = nlp(over_stem)

        similarity = under_token.similarity(over_token)

        if similarity > max_sim:
            max_sim = similarity
            max_sim_token = over

        row.append(similarity)

    matrix.append(row)
    max_similarities[under] = (max_sim_token, max_sim)
```

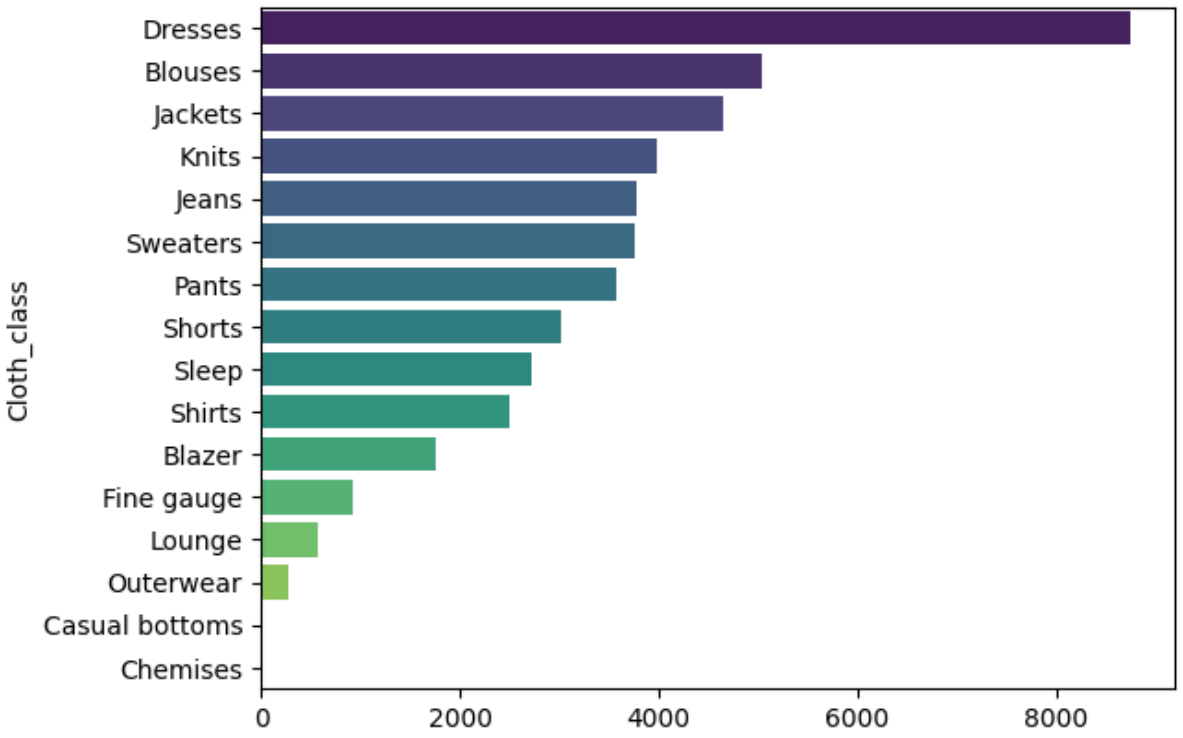
Resultado:



Com o threshold de 68% de similaridade, 8 classes puderam ser substituídas:

```
Casual bottoms: Shorts (0.54)
Chemises: Sleep (0.68)
Dress: Dresses (1.00)
Fine gauge: Pants (0.60)
Intimates: Sweaters (0.70)
Layering: Jackets (0.75)
Legwear: Pants (0.69)
Lounge: Pants (0.61)
Outerwear: Jeans (0.54)
Skirts: Dresses (0.78)
Suits: Jackets (0.76)
Swim: Dresses (0.70)
Trend: Jackets (0.79)
```

Distribuição da quantidade de samples para cada classe após realizar a substituição:



A partir desses resultados, foi buscada a solução anterior de aplicação de redução dimensional somada a um algoritmo de clusterização.

Feature Extraction:

2 métodos foram utilizados para realizar uma engenharia de features, gerando dados numéricos, a partir da coluna review: Count Vectorizer e Tfidf Vectorizer, sendo o primeiro usado para a parte de redução dimensional e clusterização, enquanto o segundo foi utilizado para a classificação.

Enquanto o Count Vectorizer gera o número de frequência de acordo com o índice do corpus (vocabulário), o Tf-idf analisa o peso global das palavras nos documentos (samples) em geral.

Função auxiliar para fazer a transformação da feature textual em uma matriz esparsa numérica:

```
def text_feature_extract(text_feature, model):
    text = model.fit_transform(text_feature)

    return text
```

Com os dois métodos foram utilizadas stopwords em algumas diferentes linguagens:

```
languages = {'english', 'spanish', 'portuguese', 'italian'}

stopwords = list()

for lang in languages:
    stopwords.extend(nltk.corpus.stopwords.words(lang))
```

Stemização dos dados:

```
stemmed = word_features.str.split().apply(lambda x: [stemmer.stem(y) for y in x])

stemmed = stemmed.apply(lambda x: ' '.join(x))
```

Além disso, para ambos foram utilizados valores de 3 (Tf-idf) ou 4 (Count Vectorizer) a 8000 ocorrências das palavras, descartando, assim, palavras que apareçam com frequência muito alta ou muito baixa no dataset.

Para o Count Vectorizer, utilizamos múltiplos n gramas (sequência de n palavras), não somente unigrama:

```
count_vect0 = CountVectorizer(stop_words = stopwords, min_df = 4, max_df = 8000, ngram_range = (1, 2))
```

```
text_all = text_feature_extract(stemmed, count_vect0)

print(f"Number of words = {len(count_vect0.get_feature_names_out())}")
```

```
Number of words = 43833
```

O Tf-Idf Vectorizer foi utilizado da seguinte forma:

```
tfidf_vect = TfidfVectorizer(stop_words = stopwords, min_df = 3, max_df = 8000)

text_all = text_feature_extract(stemmed, tfidf_vect)
```

```
print(f"Number of words = {len(tfidf_vect.get_feature_names_out())}")
```

```
Number of words = 7608
```

Após esses métodos, a matriz text_all gerada foi, então, utilizada para as tarefas de aprendizado supervisionado e não supervisionado.

Modelagem de Tópicos:

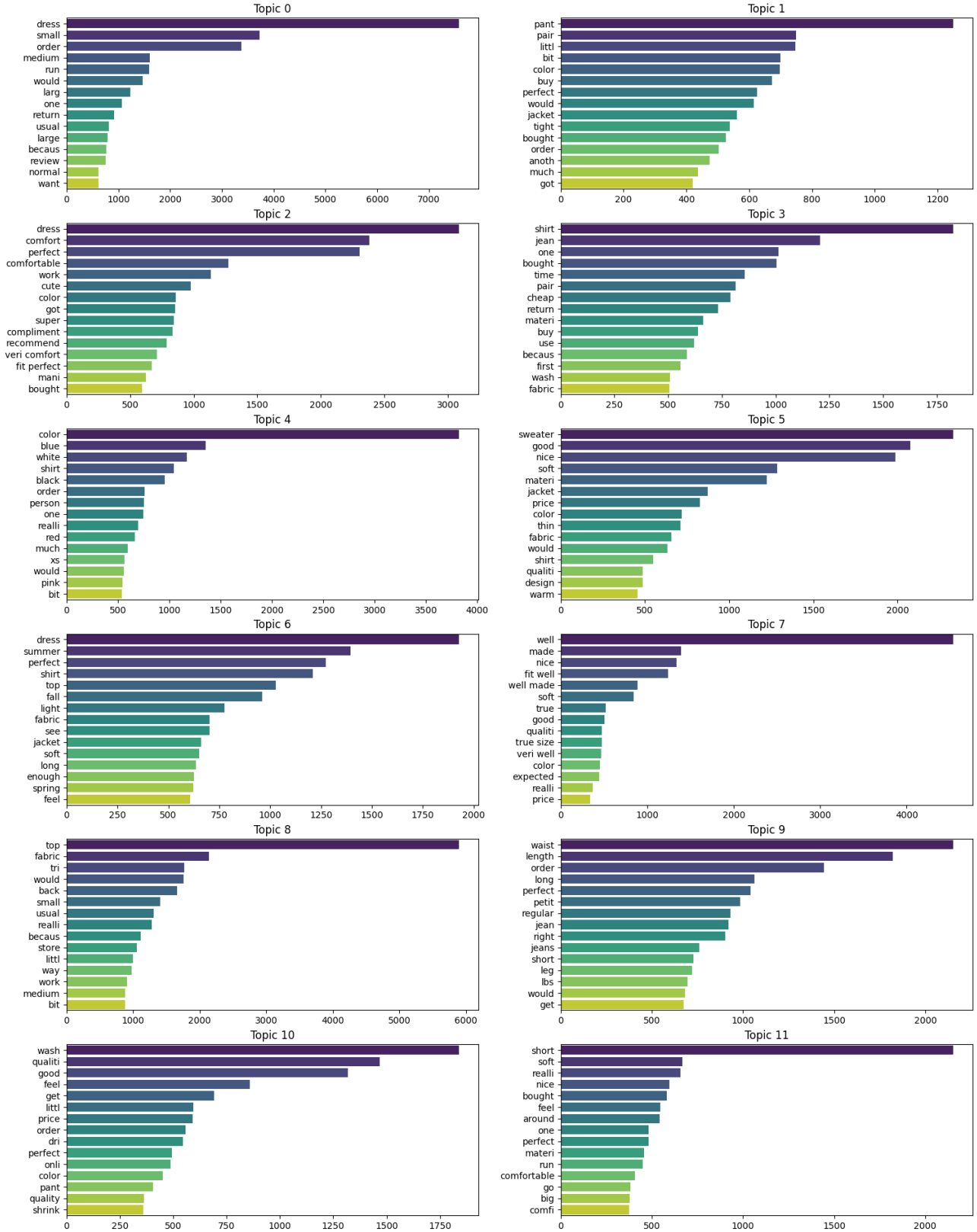
Para explorar mais sobre os dados do dataset, foi realizada uma modelagem de tópicos baseada nas reviews. O algoritmo Latent Dirichlet Allocation é uma rede bayesiana que costuma ser utilizado para tal tarefa. O modelo busca encontrar similaridades entre os conjuntos de dados, atribuindo importância de determinados termos para cada tópico.

Separando em 12 tópicos:

```
lda = LatentDirichletAllocation(n_components = 12, n_jobs = 4, random_state = 1)

topics = lda.fit_transform(text_all)
```

Código para gerar gráficos com as 15 palavras mais importantes de cada tópico:

```
fig, axes = plt.subplots(6, 2, figsize = (18, 24))

n_top_words = 15

for idx, topic in enumerate(lda.components_):
    top_features_idx = topic.argsort()[-n_top_words:]
    top_features = count_vect0.get_feature_names_out()[top_features_idx]
    weights = topic[top_features_idx]

    sns.barplot(x = weights[::-1],
                y = top_features[::-1],
                palette = 'viridis',
                orient = 'h',
                ax = axes[idx // 2][idx % 2]
               ).set_title(f'Topic {idx}')

plt.show()
```


Topic 0
dress
small
order
medium
run
would
larg
one
return
usual
large
becaus
review
normal
want
0 1000 2000 3000 4000 5000 6000 7000
Topic 1
pant
pair
littl
bit
color
buy
perfect
would
jacket
tight
bought
order
anoth
much
got
0 200 400 600 800 1000 1200
Topic 2
dress
comfort
perfect
comfortable
work
cute
color
got
super
compliment
recommend
veri comfort
fit perfect
mani
bought
0 500 1000 1500 2000 2500 3000
Topic 3
shirt
jean
one
bought
time
pair
cheap
return
materi
buy
use
becaus
first
wash
fabric
0 250 500 750 1000 1250 1500 1750
Topic 4
color
blue
white
shirt
black
order
person
one
realli
red
much
xs
would
pink
bit
0 500 1000 1500 2000 2500 3000 3500 4000
Topic 5
sweater
good
nice
soft
materi
jacket
price
color
thin
fabric
would
shirt
qualiti
design
warm
0 500 1000 1500 2000
Topic 6
dress
summer
perfect
shirt
top
fall
light
fabric
see
jacket
soft
long
enough
spring
feel
0 250 500 750 1000 1250 1500 1750 2000
Topic 7
well
made
nice
fit well
well made
soft
true
good
qualiti
true size
veri well
color
expected
realli
price
0 1000 2000 3000 4000
Topic 8
top
fabric
tri
would
back
small
usual
realli
becaus
store
littl
way
work
medium
bit
0 1000 2000 3000 4000 5000 6000
Topic 9
waist
length
order
long
perfect
petit
regular
jean
right
jeans
short
leg
lbs
would
get
0 500 1000 1500 2000
Topic 10
wash
qualiti
good
feel
get
littl
price
order
dri
perfect
onli
color
pant
quality
shrink
0 250 500 750 1000 1250 1500 1750
Topic 11
short
soft
realli
nice
bought
feel
around
one
perfect
materi
run
comfortable
go
big
comfi
0 500 1000 1500 2000

Utilizando a api da OpenAI, foram passadas as palavras mais importantes de cada tópico para que o Chat GPT, com base nisso, gerasse um título para os tópicos:

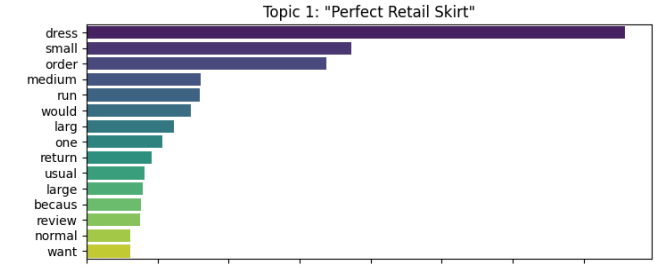
```
client = OpenAI(
    api_key = api_key
)

content = 'I want you to give a 1 to 5 word title to these 12 topic based on the group of words:'
```

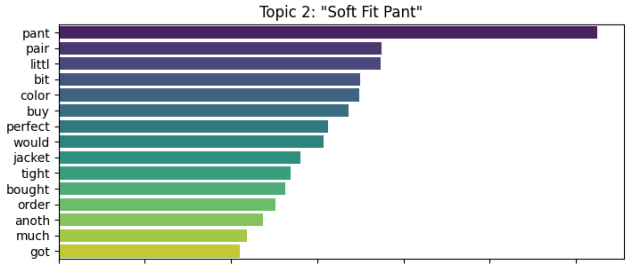
```
n_top_words = 40

topic_dict = dict()

for idx, topic in enumerate(lda.components_):
    top_features_idx = topic.argsort()[-n_top_words:]
    top_features = count_vect0.get_feature_names_out()[top_features_idx]
    weights = topic[top_features_idx]

    topic_dict[idx] = {'top_features': top_features, 'weights': weights}

    content += f'Topic {idx+1}: {top_features}\n'
```

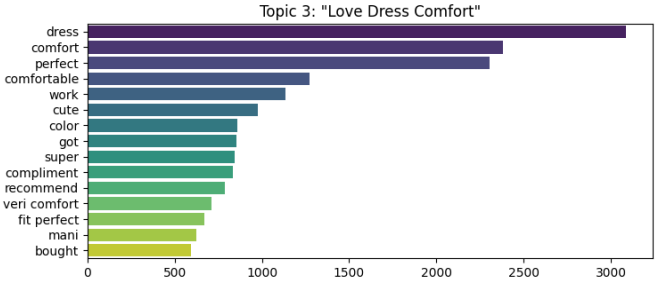
```
completion = client.chat.completions.create(
    model = 'gpt-3.5-turbo',
    messages = [
        {'role': 'user',
         'content': content}
    ],
    temperature = 0.1
)

titles = completion.choices[0].message.content.split('\n')
```

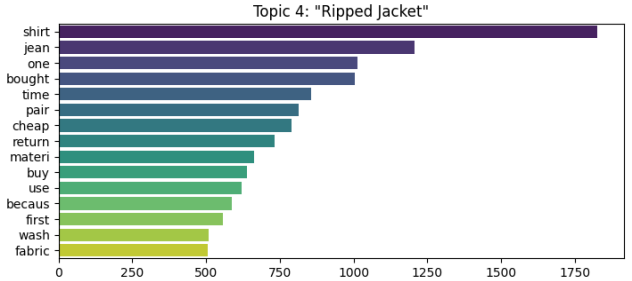
```
fig, axes = plt.subplots(6, 2, figsize = (18, 24))

for idx in topic_dict.keys():
    top_features = topic_dict[idx]['top_features'][::-1][:15]
    weights = topic_dict[idx]['weights'][::-1][:15]
    title = titles[idx]

    sns.barplot(x = weights,
                y = top_features,
                palette = 'viridis',
                ax = axes[idx // 2][idx % 2]
               ).set_title(title)

plt.show()
```

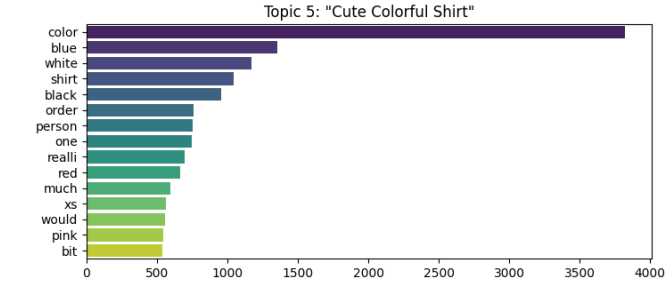
Gráficos com os títulos gerados com a API Chat GPT:

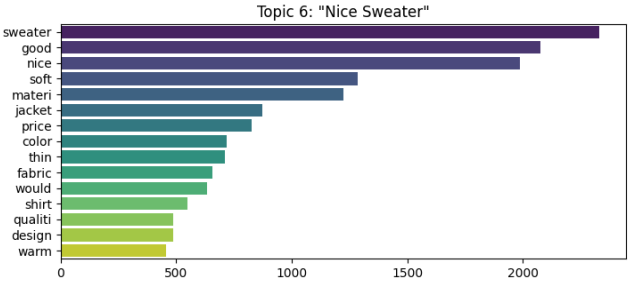


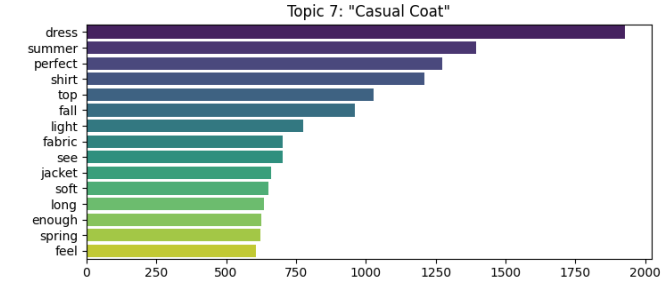


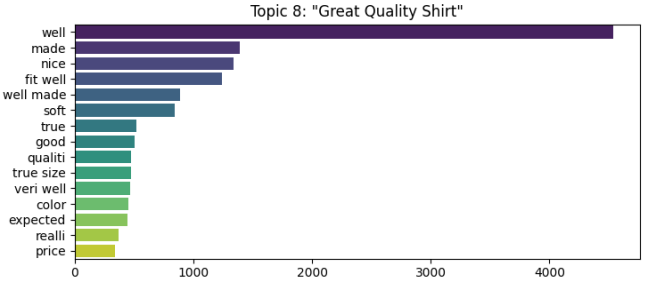


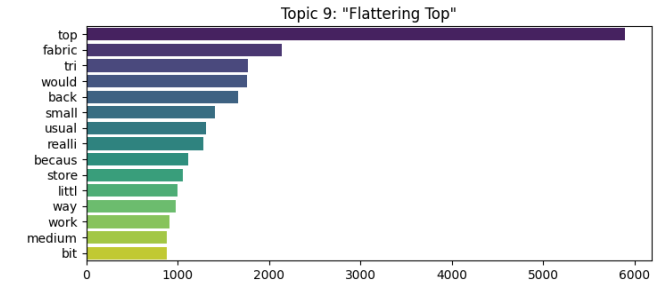


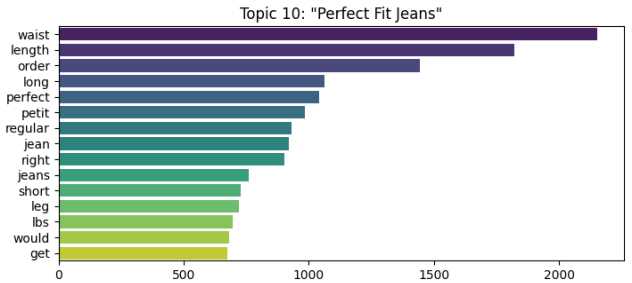


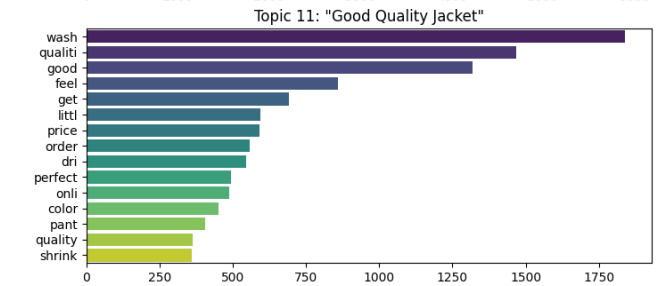


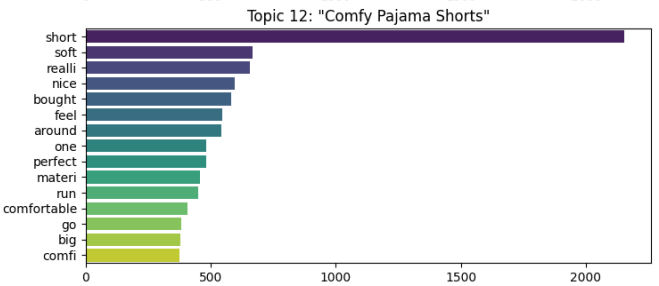












Redução de Dimensionalidade e Algoritmos de Clusterização:

A partir da matriz text_all gerada com o Count Vectorizer, foram combinados algoritmos de redução de dimensionalidade e de clusterização com o objetivo de explorar mais sobre os dados e encontrar classes similares (com base nas features ao invés do nome) para as classes de poucos samples.

Para visualizar os resultados da redução dimensional, foi criada uma função para gerar um gráfico em 2D com a biblioteca Seaborn e outra para gerar em 3D com a plotly:

```
def plot_2d_reduction(x_axis, y_axis, hue, legend_title = 'cloth class'):
    fig, ax = plt.subplots(figsize = (10, 10))

    sns.scatterplot(x = x_axis, y = y_axis, hue = hue, palette = 'viridis')

    plt.legend(bbox_to_anchor = (1.02, 1),
               loc = 'upper left',
               borderaxespad = 0,
               title = legend_title
              )

    plt.show()
```

```
def plot_3d_reduction(data, hue):
    fig = px.scatter_3d(
        data, x = 0, y = 1, z = 2, color = hue,
        labels = {'0': 'Feature 1', '1': 'Feature 2', '2': 'Feature 3'}
    )

    fig.show()
```

Funções auxiliares para visualização dos resultados dos clusters:

```
def cluster_results(cluster_model, text, df):
```

```
def plot_total_cluster(total_cluster):
```

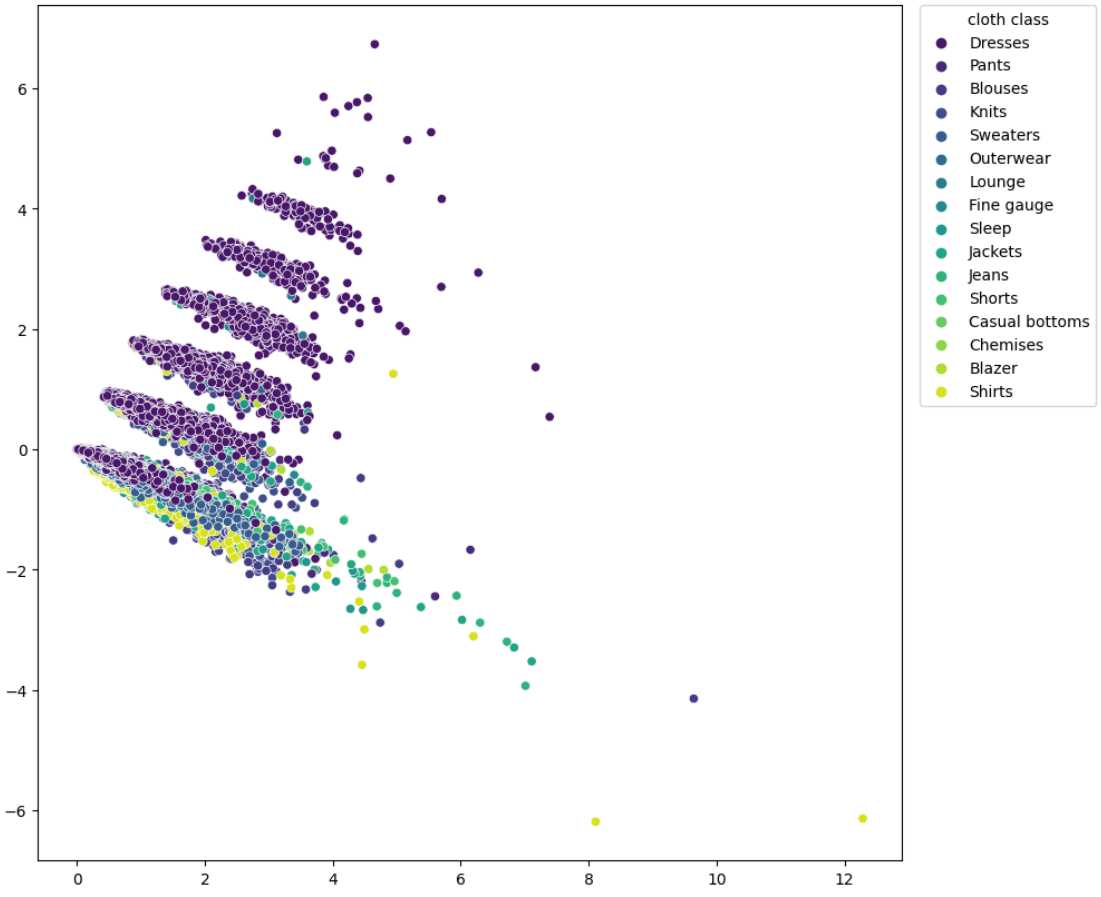
```
def plot_cloth_cluster(model_cloth):
```

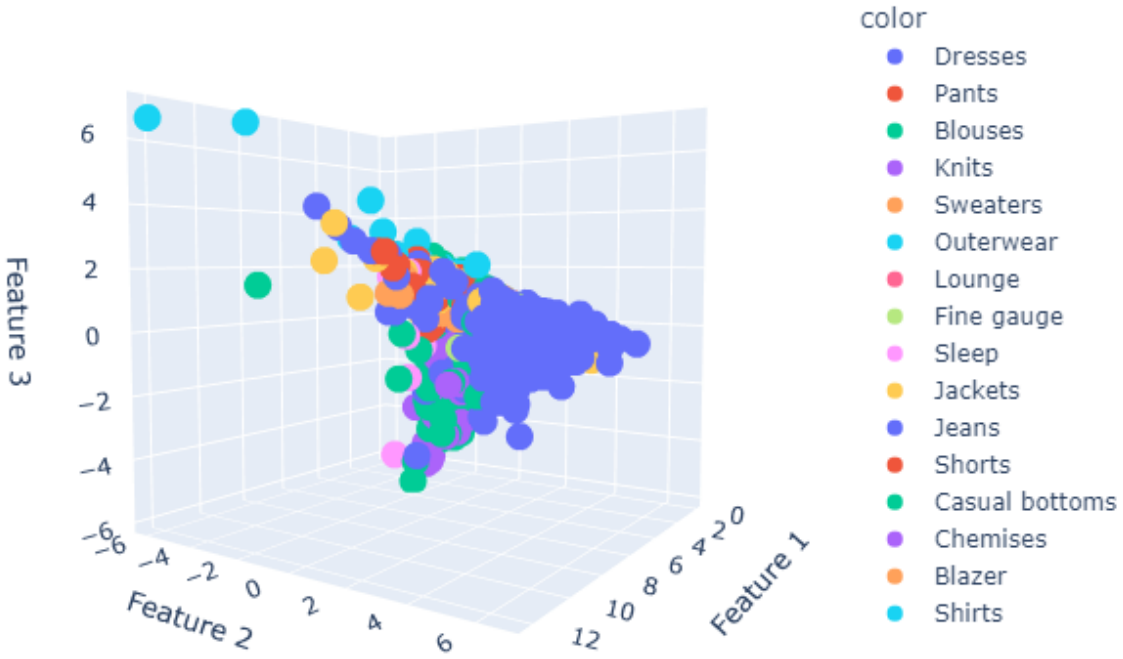
```
def get_cluster_class(model_cloth):
```

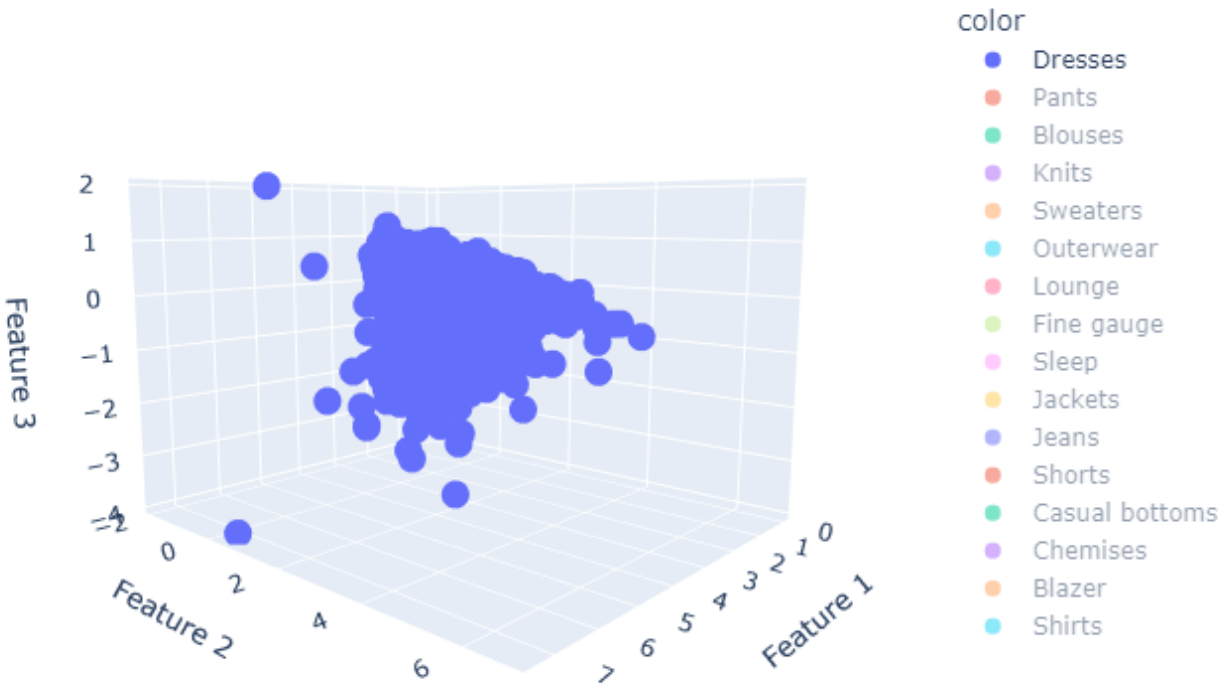
Função para substituir as classes abaixo do threshold de samples (definido previamente como 1500) pela classe com cluster similar de maior presença no cluster:

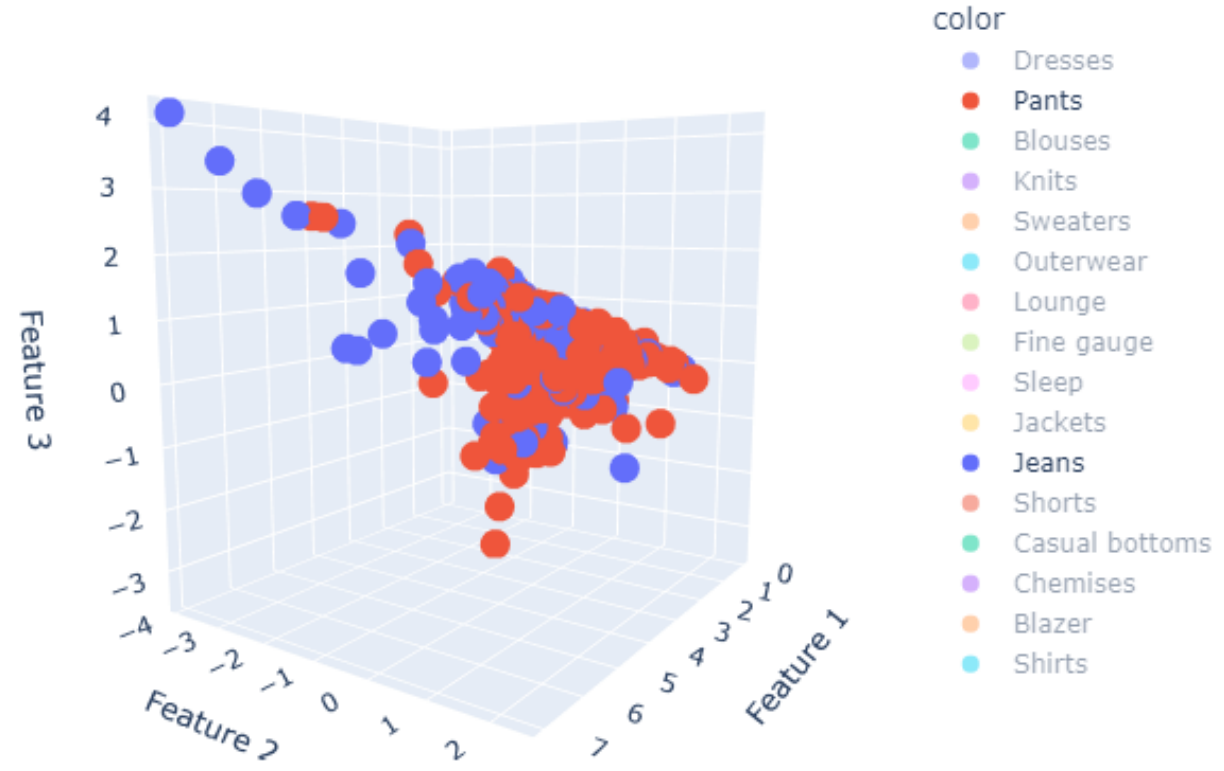
```
def get_replacer_class(cluster_dict, classes_under_threshold, cloth_cluster):
```

O Truncated SVD atua fazendo uma redução de dimensionalidade linear por média de decomposição truncada de valor singular. Somado ao uso de Count Vectorizer ou Tfidf Vectorizer, esse método se torna conhecido como LSA (Latent Semantic Analysis). É um algoritmo eficiente para lidar com matrizes esparsas.

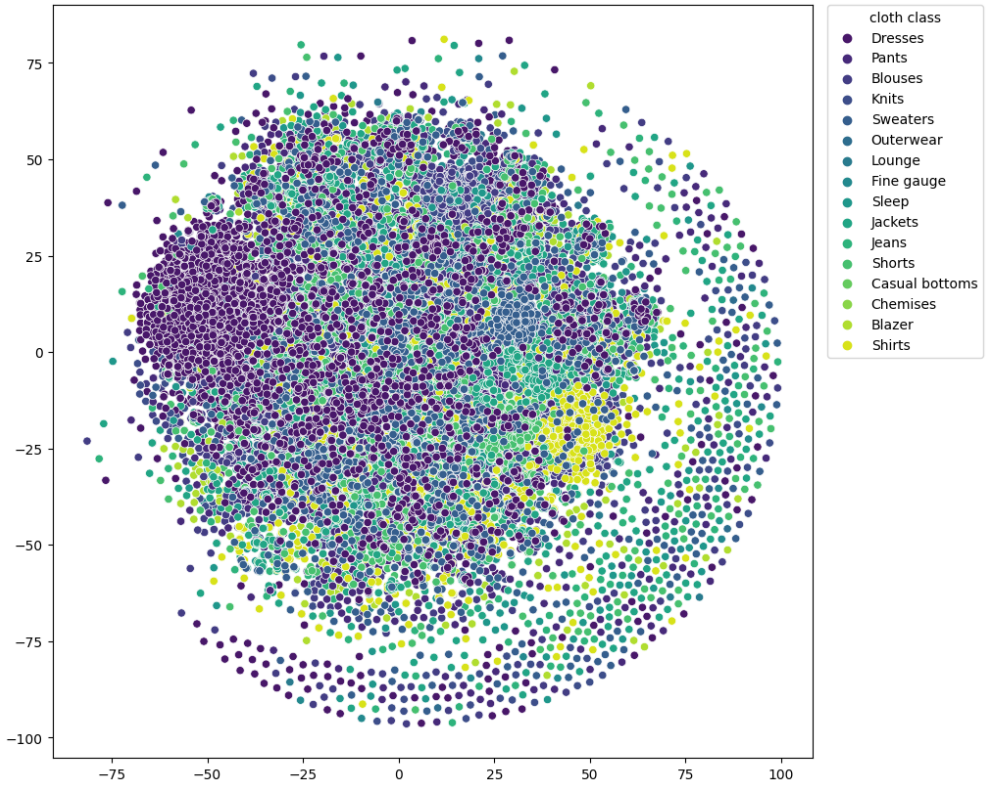




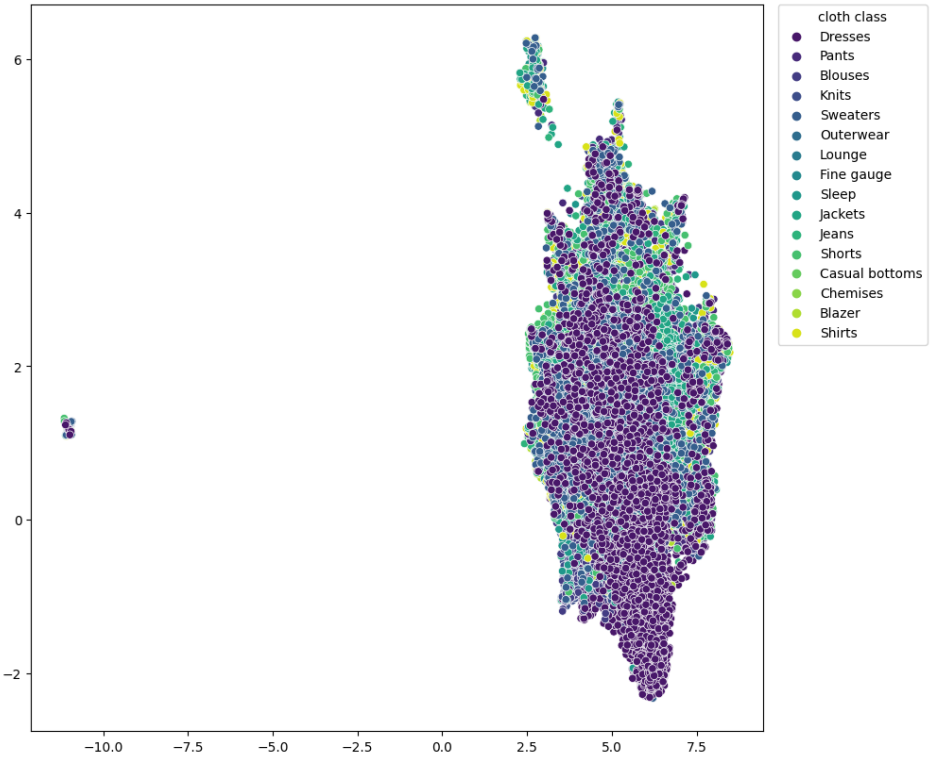

color
Dresses
Pants
Blouses
Knits
Sweaters
Outerwear
Lounge
Fine gauge
Sleep
Jackets
Jeans
Shorts
Casual bottoms
Chemises
Blazer
Shirts
Feature 3
Feature 2
Feature 1


color
Dresses
Pants
Blouses
Knits
Sweaters
Outerwear
Lounge
Fine gauge
Sleep
Jackets
Jeans
Shorts
Casual bottoms
Chemises
Blazer
Shirts
Feature 3
Feature 2
Feature 1

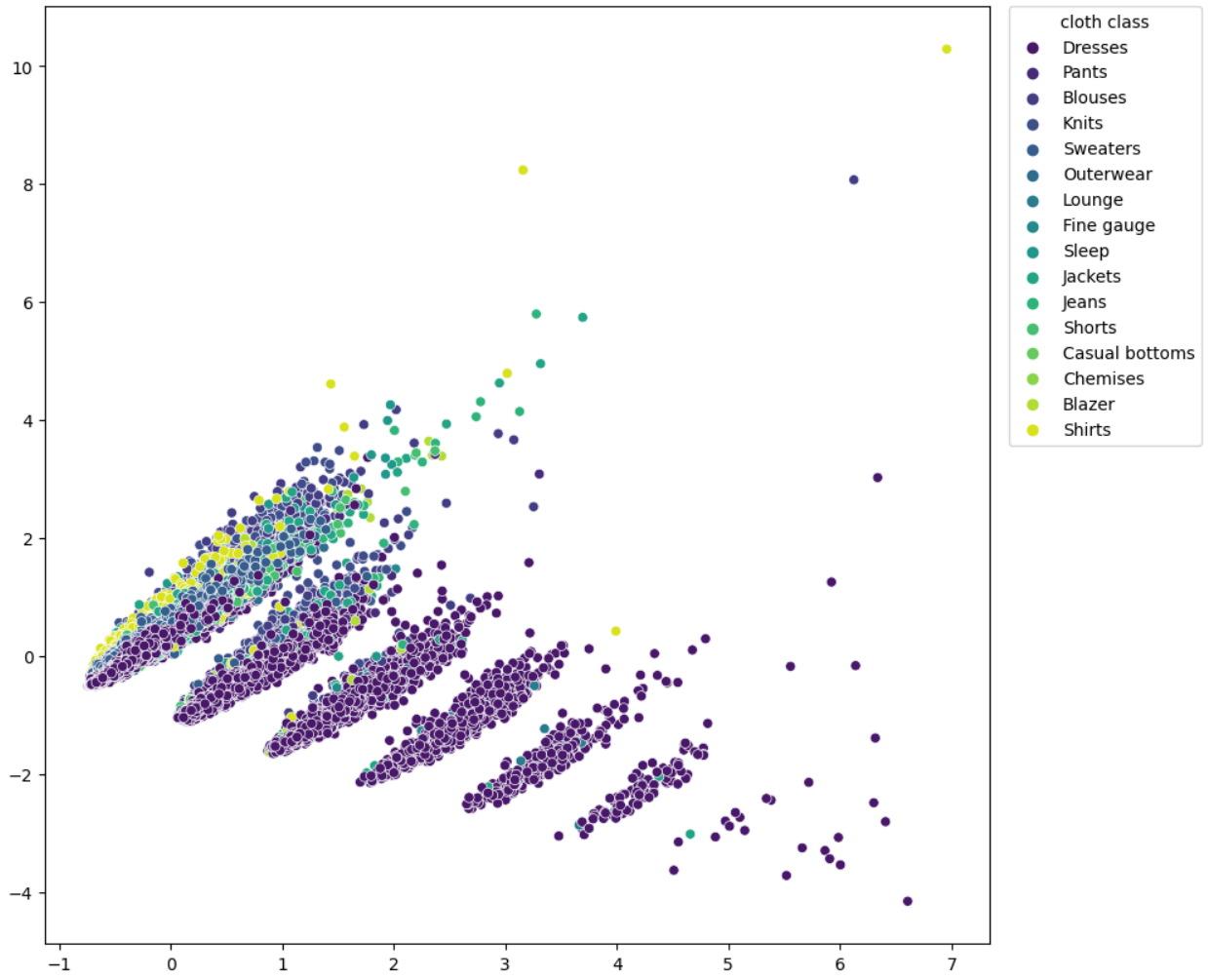
t-SNE (T-distributed Stochastic Neighbor Embedding) é um algoritmo de redução dimensional que busca facilitar a visualização. Diferentemente do Truncated SVD, sua redução dimensional não é linear.



UMAP é outra técnica de redução de dimensionalidade não linear:



Por último, PCA, ou Principal Component Analysis, busca converter features correlacionadas em um alto espaço dimensional em uma série de features não correlacionadas em um espaço de dimensão menor, sendo esse conjunto os componentes principais. A transformação linear é ortogonal, de forma que os componentes principais são perpendiculares entre si.



Clustering é um tipo de aprendizado não supervisionado que busca agrupar objetos similares. Para analisar quais classes são similares, foram utilizados diferentes tipos de algoritmos de cluster, como KMeans, Agglomerative Clustering e Bisecting KMeans.

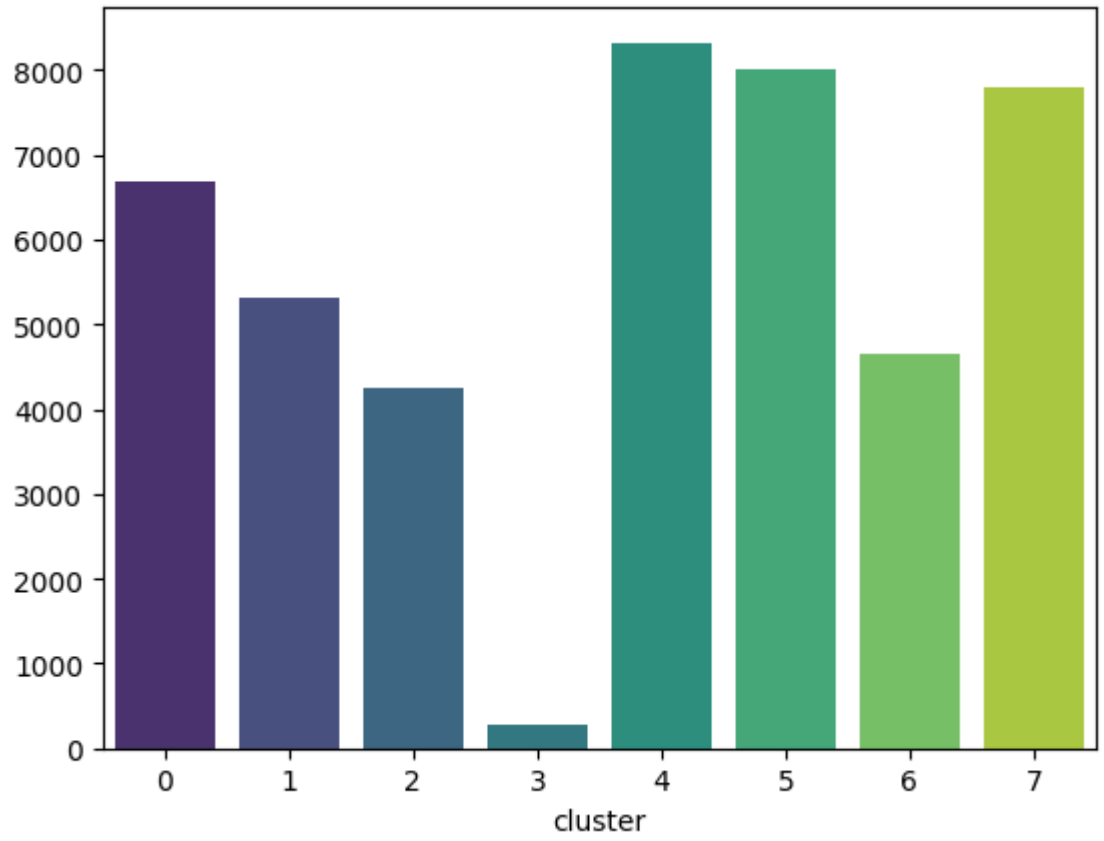
Utilizando as funções auxiliares, foram obtidos os seguintes resultados para o KMeans após a redução dimensional feita com o UMAP:

```
kmeans = KMeans(n_clusters = 8, n_init = 514, random_state = 1, max_iter = 354)

kmeans_df, total_kmeans, kmeans_cloth = cluster_results(kmeans, umap_red, new_df)

display(kmeans_df)
```

	Title	Review	Cons_rating	Cloth_class	cluster
2	Some major design flaws	I had such high hopes for this dress and reall...	3.0	Dresses	5
3	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5.0	Pants	1
4	Flattering shirt	This shirt is very flattering to all due to th...	5.0	Blouses	0
5	Not for the very petite	I love tracy reese dresses, but this one is no...	2.0	Dresses	1
6	Cagrcoal shimmer fun	I aded this in my basket at hte last mintue to...	5.0	Knits	4
...	...	...	...	...	...
49333	Dress felt and fit great. I got lots of compl...	Loved the color!!! Dress fit great and I got ...	5.0	Dresses	1
49334	Loved the dress but poor quality	This dress looked great and I loved the materi...	2.0	Dresses	0
49335	Cute dress, didn't fit	Wanted this dress to work it didn't. It is ver...	1.0	Dresses	4
49336	Very cute!	No complaints othe than the zipper gets stuck ...	4.0	Dresses	7
49337	Good fit	The fabric was really nice, I'm a L and it fit...	5.0	Dresses	4

45308 rows × 5 columns

Número de samples por cluster:



Havendo 5 classes restantes após a utilização da similaridade de texto com a SpaCy, foram encontradas as classes mais similares após agrupar pelo KMeans utilizando UMAP:

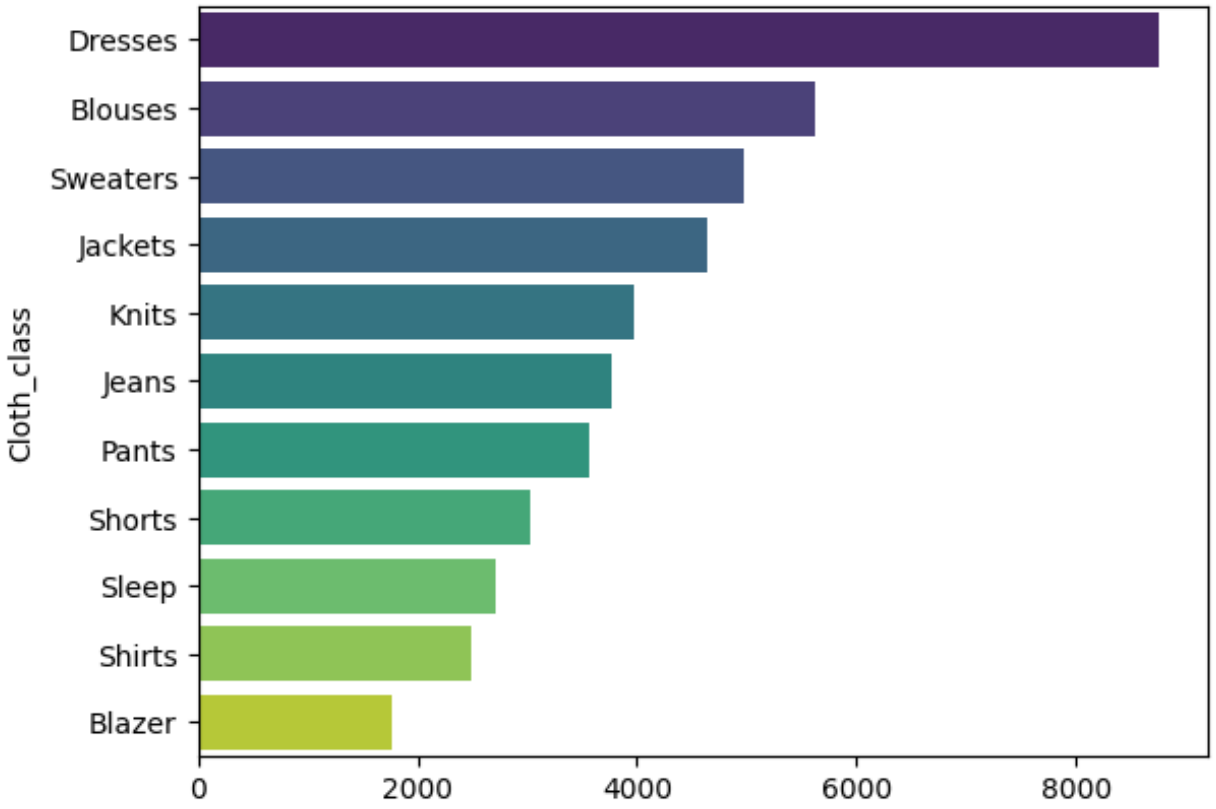
```
kmeans_replacer_map = get_replacer_class(kmeans_dict, classes_list_under, kmeans_cloth)

print(*kmeans_replacer_map.items(), sep = '\n')
```

```
('Casual bottoms', 'Jackets')
('Chemises', 'Dresses')
('Fine gauge', 'Sweaters')
('Lounge', 'Blouses')
('Outerwear', 'Sweaters')
```

Esse foi o resultado ao encontrar, para cada classe, o cluster em que aparece mais frequentemente, substituindo cada uma das classes que estão abaixo do threshold de 1500 samples (Casual bottoms, Chemises, Fine Gauge, Lounge e Outerwear) pela classe do seu cluster que apareceu mais frequentemente (caso o cluster também seja o de maior frequência da classe).

Por exemplo, ao analisar a seguinte imagem (frequência por cluster e classe), é possível observar que o cluster de maior frequência para Sweaters foi o cluster 4, sendo o mesmo para Outerwear e Fine Gauge. Como sweaters é a classe que mais apareceu no cluster, todos os samples previamente considerados como outerwear e fine gauge passam a ser considerados como sweaters.

Visando a facilitar o treinamento de modelos de aprendizado supervisionado, após fazer tal substituição de classes, foi gerado um novo csv (reviews.csv), a partir do qual, foi realizado um tuning para a classificação de classes de roupa. A nova distribuição por classe foi a seguinte:



Modelos de Classificação:

Diferentes modelos foram utilizados para realizar a predição de classe. Inicialmente, sendo utilizada como métrica a acurácia. Entretanto, devido ao desbalanceamento da base, mesmo com o threshold mínimo de 1500 samples, o roc auc score (one vs one) foi a nova métrica utilizada para medir a classificação.

Utilizando dessa vez o text_all gerado pelo Tfidf Vectorizer, há 7608 features com target multiclasse (11).

Para medir com o cross validation score estratificado, foi gerado com o make_scorer o seguinte scoring:

```
scorer = make_scorer(roc_auc_score, needs_proba = True, multi_class = 'ovo')

cv = StratifiedKFold(n_splits = 4)
```

Buscando analisar os resultados com split em train x test, também estratificando:

```
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state = 1, test_size = 0.4, stratify = y)
```

O Logistic Regression, modelo linear de classificação, sem e com pipeline usando redução dimensional (Truncated SVD de 800 componentes), foram obtidos bons resultados com o roc auc score, além de uma eficiente performance:

```
reducer = TruncatedSVD(n_components = 800)

log_reg = LogisticRegression(max_iter = 8000, n_jobs = 4, C = 1)

log_reg_pipe = Pipeline([
                ('reduction', reducer),
                ('logistic_regression', log_reg)
               ])
```

```
%%time

log_reg.fit(train_X, train_y)
```

```
CPU times: user 83.3 ms, sys: 309 ms, total: 392 ms
Wall time: 7.11 s
```

```
LogisticRegression
LogisticRegression(C=1, max_iter=8000, n_jobs=4)
```

```
log_reg0_preds = {'train': log_reg.predict(train_X),
                  'test': log_reg.predict(test_X),
                  'prob_train': log_reg.predict_proba(train_X),
                  'prob_test': log_reg.predict_proba(test_X),
                 }
```

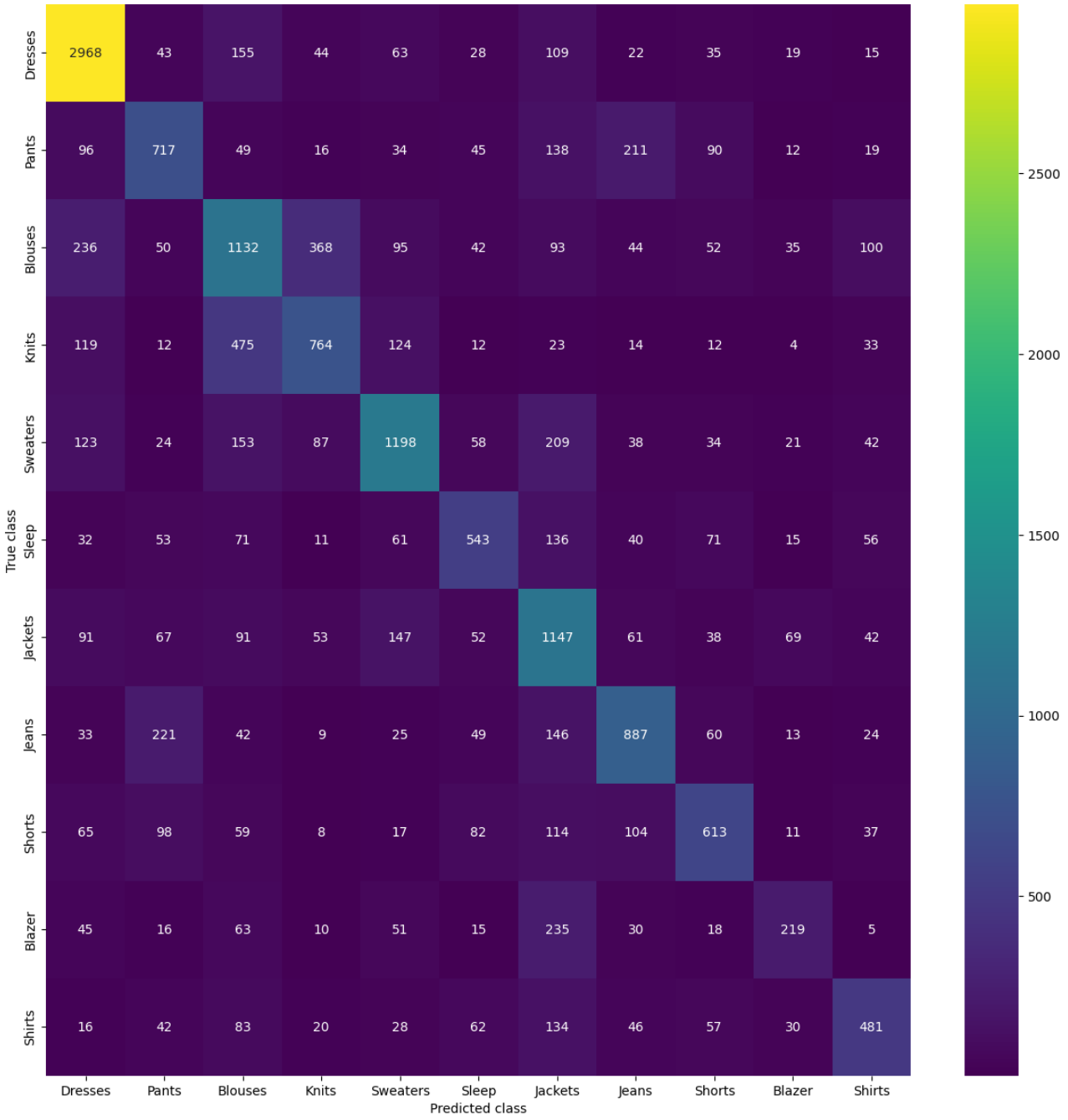
```
print(f'Logistic Regression train accuracy score = {accuracy_score(train_y, log_reg0_preds["train"])}')
print(f'Logistic Regression test accuracy score = {accuracy_score(test_y, log_reg0_preds["test"])}')
```

```
Logistic Regression train accuracy score = 0.7078428487345497
Logistic Regression test accuracy score = 0.5886669609357758
```

```
print(f'Logistic Regression train roc auc score = {roc_auc_score(train_y, log_reg0_preds["prob_train"], multi_class = "ovo")}')
print(f'Logistic Regression test roc auc score = {roc_auc_score(test_y, log_reg0_preds["prob_test"], multi_class = "ovo")}')
```

```
Logistic Regression train roc auc score = 0.9471567482649336
Logistic Regression test roc auc score = 0.9034680214223717
```

Matriz de confusão para o Logistic Regression (sem redução dimensional):



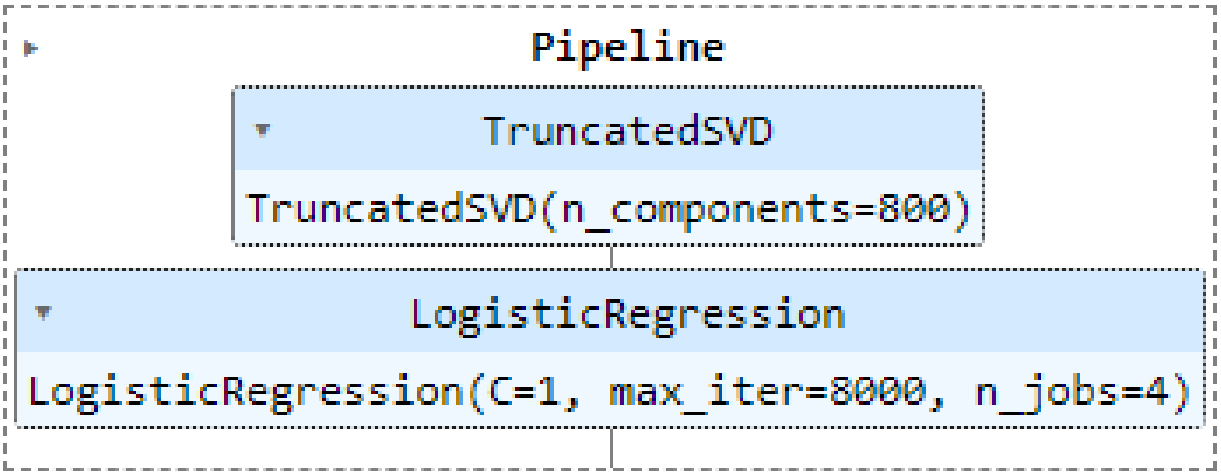

```
log_reg_scores = cross_val_score(log_reg, X, y, cv = cv, scoring = scorer, n_jobs = 4)
```

```
print(f'Logistic Regression cross validation scores = {log_reg_scores}')
print(f'Logistic Regression mean cross validation score = {log_reg_scores.mean()}')
```

```
Logistic Regression cross validation scores = [0.88449269 0.90437692 0.89066565 0.87274459]
Logistic Regression mean cross validation score = 0.8880699616446952
```

```
%%time

log_reg_pipe.fit(train_X, train_y)
```

```
CPU times: user 35.9 s, sys: 9.64 s, total: 45.5 s
Wall time: 38.5 s
```




```
log_reg_preds = {'train': log_reg_pipe.predict(train_X),
                 'test': log_reg_pipe.predict(test_X),
                 'prob_train': log_reg_pipe.predict_proba(train_X),
                 'prob_test': log_reg_pipe.predict_proba(test_X),
                }
```

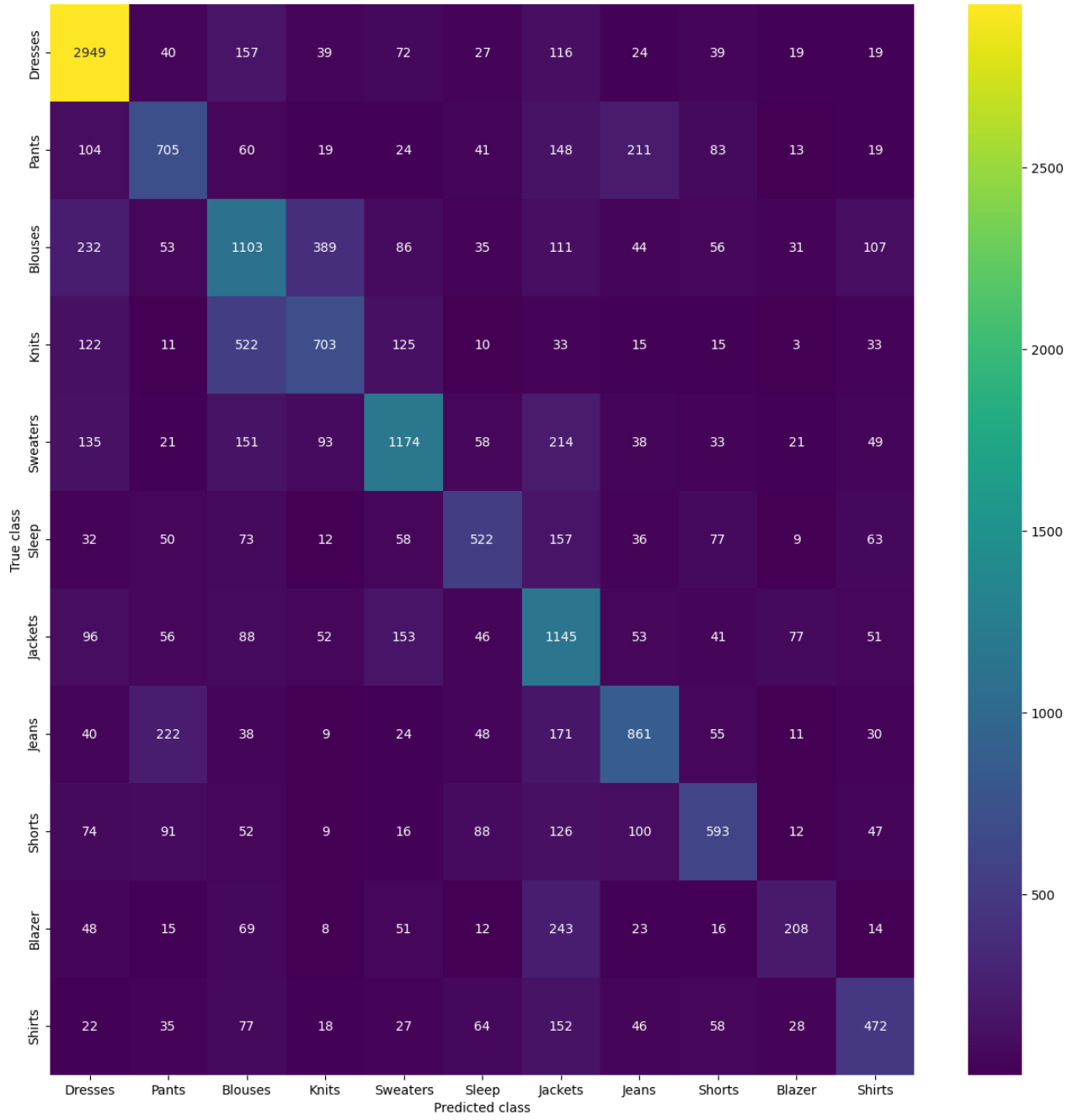
```
print(f'Logistic Regression train accuracy score = {accuracy_score(train_y, log_reg_preds["train"])}')
print(f'Logistic Regression test accuracy score = {accuracy_score(test_y, log_reg_preds["test"])}')
```

```
Logistic Regression train accuracy score = 0.6328722778104767
Logistic Regression test accuracy score = 0.5757559037740013
```

```
print(f'Logistic Regression train roc auc score = {roc_auc_score(train_y, log_reg_preds["prob_train"], multi_class = "ovo")}')
print(f'Logistic Regression test roc auc score = {roc_auc_score(test_y, log_reg_preds["prob_test"], multi_class = "ovo")}')
```

```
Logistic Regression train roc auc score = 0.9255695032496561
Logistic Regression test roc auc score = 0.8997531145089248
```

Matriz de confusão para o Logistic Regression (com redução dimensional):




```
log_reg_pipe_scores = cross_val_score(log_reg_pipe, X, y, cv = cv, scoring = scorer, n_jobs = 4)
```

```
print(f'Logistic Regression cross validation scores = {log_reg_pipe_scores}')
print(f'Logistic Regression mean cross validation score = {log_reg_pipe_scores.mean()}')
```

```
Logistic Regression cross validation scores = [0.87900175 0.90074317 0.88844028 0.87055117]
Logistic Regression mean cross validation score = 0.8846840941700301
```

O Support Vector Machine foi outro modelo que apresentou bons resultados com ou sem pipeline usando redução dimensional, porém tendo uma performance bem mais lenta que o modelo linear. Como parâmetros, foram selecionados o kernel de base radial, soft margin (C) igual a 10 e gamma igual a 0.01, permitindo que o modelo tivesse uma maior capacidade de generalizar.

```
reducer = TruncatedSVD(n_components = 200)

svc = SVC(C = 10, gamma = 0.01, probability = True)

svc_pipe = Pipeline([
                ('reduction', reducer),
                ('svc', svc)
            ])
```

```
%%time

svc.fit(train_X, train_y)
```

```
CPU times: user 13min 2s, sys: 198 ms, total: 13min 2s
Wall time: 13min 2s
```

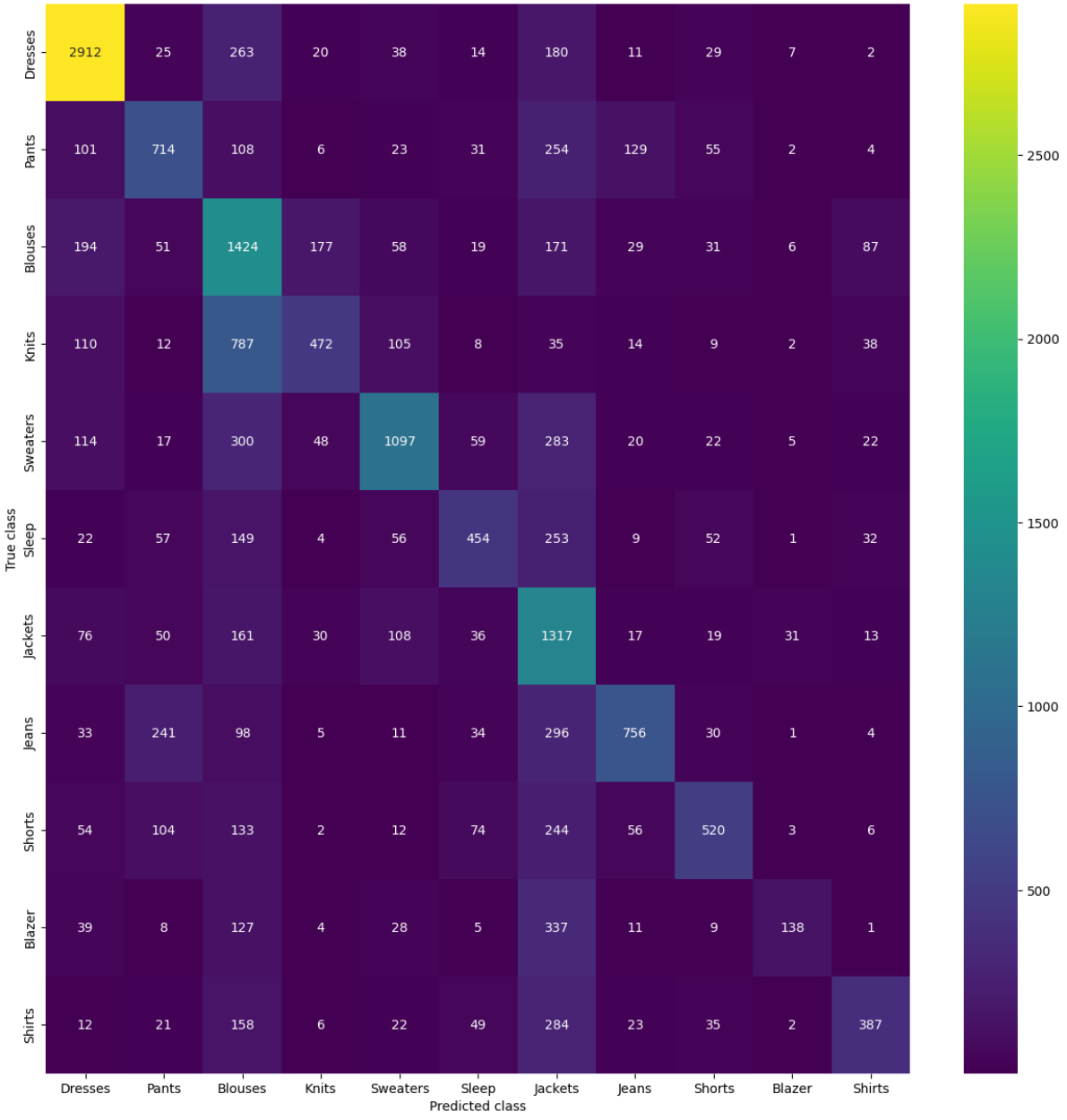
```
SVC(C=10, gamma=0.01, probability=True)
```

```
svc0_preds = {'train': svc.predict(train_X),
              'test': svc.predict(test_X),
              'prob_train': svc.predict_proba(train_X),
              'prob_test': svc.predict_proba(test_X),
             }
```

```
print(f'SVC train roc auc score = {roc_auc_score(train_y, svc0_preds["prob_train"], multi_class = "ovo")}')
print(f'SVC test roc auc score = {roc_auc_score(test_y, svc0_preds["prob_test"], multi_class = "ovo")}')
```

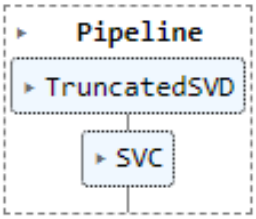
```
SVC train roc auc score = 0.9285465972514073
SVC test roc auc score = 0.8947787025885224
```

Matriz de confusão para o SVC (sem redução dimensional):




```
%%time

svc_pipe.fit(train_X, train_y)
```

```
CPU times: user 14min 55s, sys: 4.63 s, total: 15min
Wall time: 14min 52s
```



```
svc_preds = {'train': svc_pipe.predict(train_X),
             'test': svc_pipe.predict(test_X),
             'prob_train': svc_pipe.predict_proba(train_X),
             'prob_test': svc_pipe.predict_proba(test_X),
            }
```

```
print(f'SVC train roc auc score = {roc_auc_score(train_y, svc_preds["prob_train"], multi_class = "ovo")}')
print(f'SVC test roc auc score = {roc_auc_score(test_y, svc_preds["prob_test"], multi_class = "ovo")}')
```

```
SVC train roc auc score = 0.8914275467371422
SVC test roc auc score = 0.8789137949671915
```

Reduzindo para 200 features utilizadas com Truncated SVD, foi possível manter uma boa performance validando com o roc auc score (apesar de, validando com a acurácia, os resultados foram muito inferiores).

Seguindo com um modelo de árvore de decisão, foi importante fazer um tuning para evitar tanto o overfitting quanto o underfitting da árvore, sendo esse controle feito com o parâmetro max_leaf_nodes.

```
tree = DecisionTreeClassifier(max_leaf_nodes = 374)
```

```
%%time

tree.fit(train_X, train_y)
```

```
CPU times: user 4.76 s, sys: 2.99 ms, total: 4.76 s
Wall time: 4.76 s
```

```
DecisionTreeClassifier(max_leaf_nodes=374)
```

```
Decision Tree train roc auc score = 0.8858740986422214
Decision Tree test roc auc score = 0.8449981132042503
```

```
Decision Tree cross validation scores = [0.82494242 0.85206509 0.84525157 0.82606749]
Decision Tree mean cross validation score = 0.8370816415694806
```

Utilizando floresta aleatória, ensemble de bagging de árvores de decisão, foram obtidos bons resultados validando com um roc auc score. Porém, ao aplicar uma redução dimensional com o UMAP, esse resultado decaiu bastante para o teste.

```
reducer = UMAP(n_components = 404, metric = 'cosine')

forest_clf = RandomForestClassifier(n_jobs = 4, random_state = 1, max_leaf_nodes = 494, n_estimators = 474)

forest_pipe = Pipeline([
                ('reducer', reducer),
                ('forest', forest_clf)
              ])
```

```
%%time

forest_clf.fit(train_X, train_y)
```

```
CPU times: user 1min 34s, sys: 208 ms, total: 1min 35s
Wall time: 24.7 s
```

```
RandomForestClassifier(max_leaf_nodes=494, n_estimators=474, n_jobs=4,
                       random_state=1)
```

```
forest0_preds = {'train': forest_clf.predict(train_X),
                 'test': forest_clf.predict(test_X),
                 'prob_train': forest_clf.predict_proba(train_X),
                 'prob_test': forest_clf.predict_proba(test_X),
                }
```

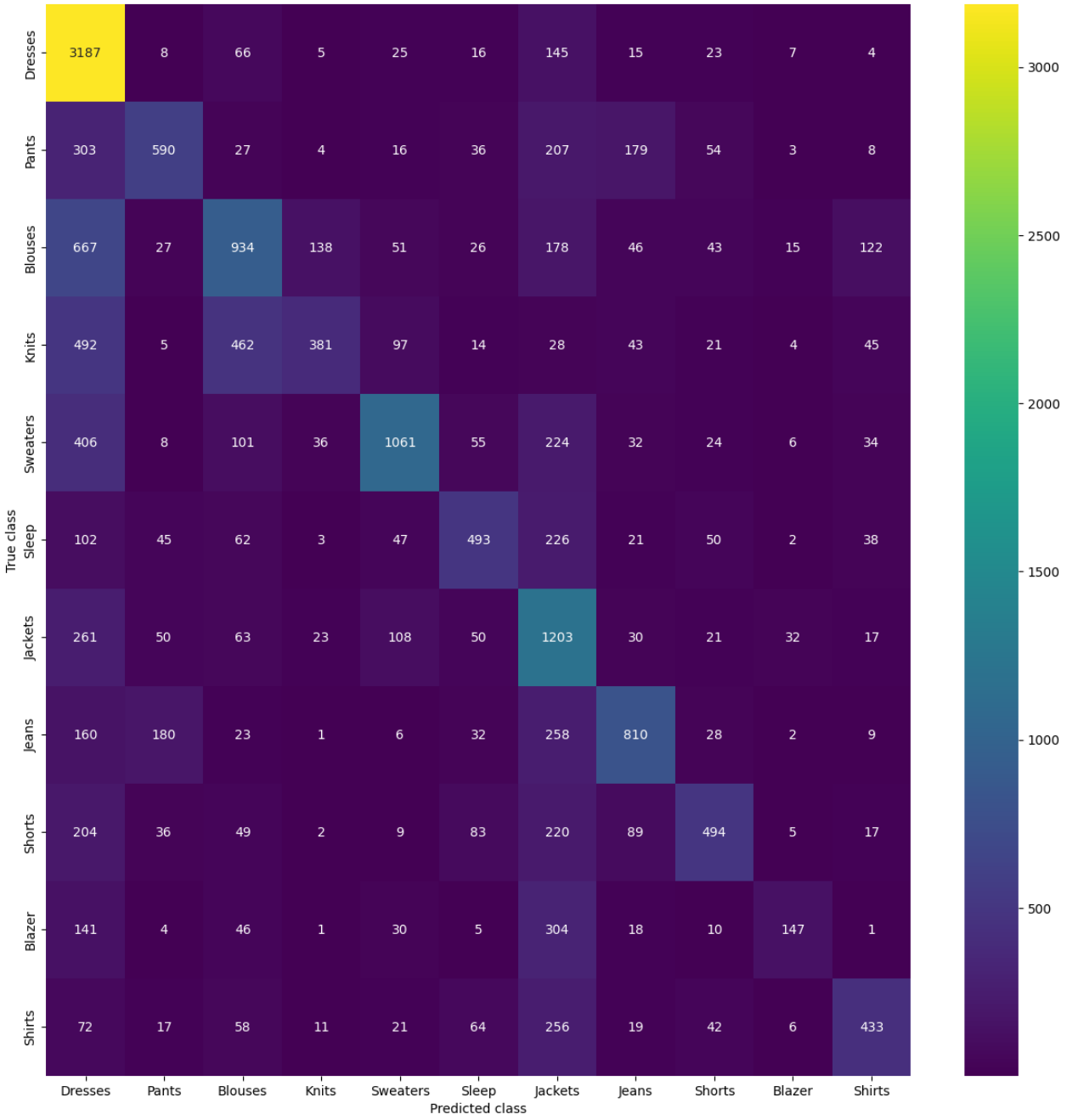
```
print(f'Random Forest train accuracy score = {accuracy_score(train_y, forest0_preds["train"])}')
print(f'Random Forest test accuracy score = {accuracy_score(test_y, forest0_preds["test"])}')
```

```
Random Forest train accuracy score = 0.6408917010005886
Random Forest test accuracy score = 0.537022732288678
```

```
print(f'Random Forest train roc auc score = {roc_auc_score(train_y, forest0_preds["prob_train"], multi_class = "ovo")}')
print(f'Random Forest test roc auc score = {roc_auc_score(test_y, forest0_preds["prob_test"], multi_class = "ovo")}')
```

```
Random Forest train roc auc score = 0.9437921642073157
Random Forest test roc auc score = 0.8904680937614564
```

Matriz de confusão para a Random Forest (sem redução dimensional):



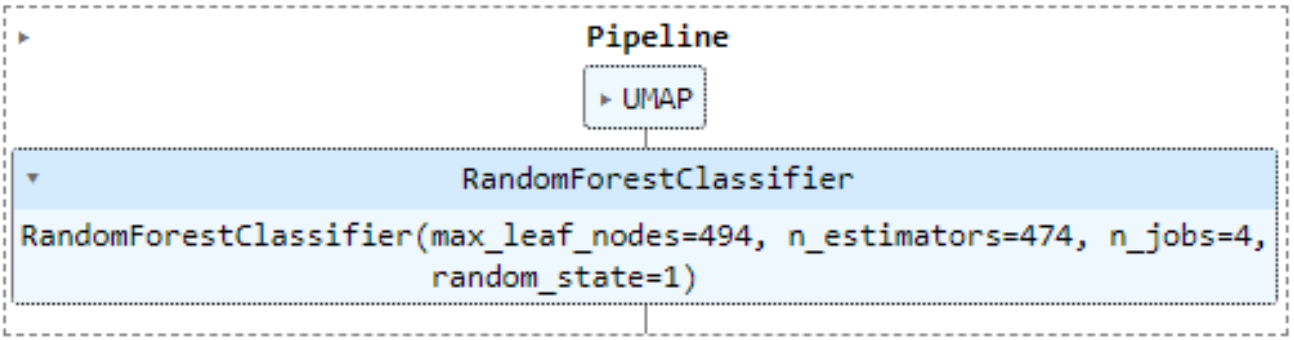

```
forest_scores = cross_val_score(forest_clf, X, y, cv = cv, scoring = scorer, n_jobs = 4)
```

```
print(f'Random Forest cross validation scores = {forest_scores}')
print(f'Random Forest mean cross validation score = {forest_scores.mean()}')
```

```
Random Forest cross validation scores = [0.87430733 0.89584516 0.87866434 0.8546811 ]
Random Forest mean cross validation score = 0.8758744830761936
```

```
%%time

forest_pipe.fit(train_X, train_y)
```

```
CPU times: user 22min 6s, sys: 39.5 s, total: 22min 46s
Wall time: 6min 22s
```




```
forest_preds = {'train': forest_pipe.predict(train_X),
                'test': forest_pipe.predict(test_X),
                'prob_train': forest_pipe.predict_proba(train_X),
                'prob_test': forest_pipe.predict_proba(test_X),
               }
```

```
print(f'Random Forest train roc auc score = {roc_auc_score(train_y, forest_preds["prob_train"], multi_class = "ovo")}')
print(f'Random Forest test roc auc score = {roc_auc_score(test_y, forest_preds["prob_test"], multi_class = "ovo")}')
```

```
Random Forest train roc auc score = 0.9253899339297511
Random Forest test roc auc score = 0.7435570271075572
```

Mesmo aumentando para 800 o número de componentes no UMAP, o resultado não demonstrou melhoras, de forma que o modelo não conseguiu generalizar bem para o conjunto de teste somente com essas features.

Um modelo que costuma ser recomendado para classificação de texto é o Multinomial NB, sendo um modelo probabilístico baseado na aplicação do teorema de Bayes.

Convertendo para array (o modelo não aceita matrizes esparsas como parâmetros para treino e predição):

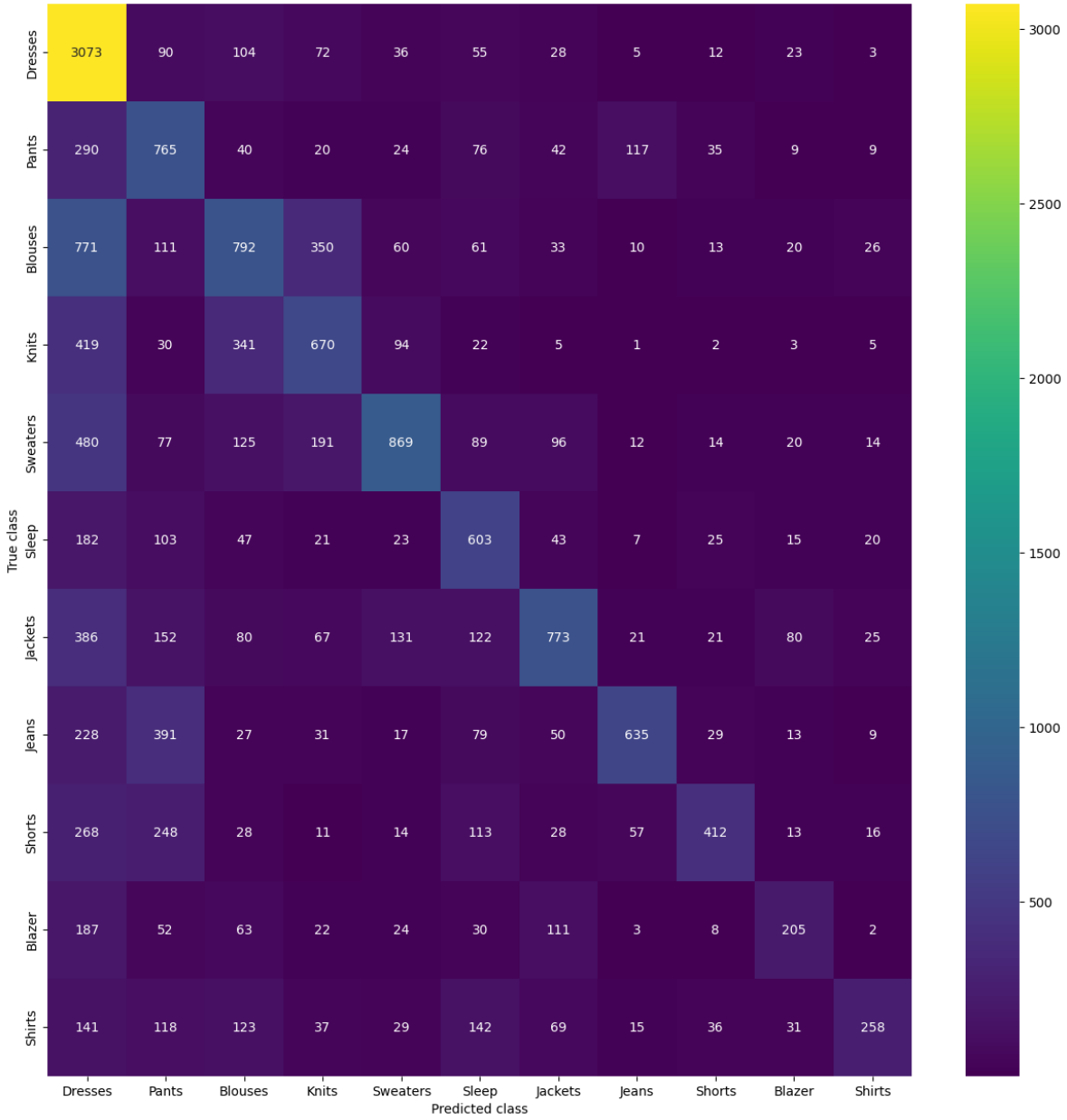
```
train_X_arr = train_X.toarray()
test_X_arr = test_X.toarray()
```

Devido ao desbalanceamento das classes, foi passado como parâmetro um array com as probabilidades das classes no dataset:

```
class_prior = targets.value_counts().values.astype('float32')
class_prior /= class_prior.sum()

multinomial_nb = MultinomialNB(alpha = 0.1, class_prior = class_prior)
```

```
Multomial Naive Bayes train roc auc score = 0.9529433131495095
Multomial Naive Bayes test roc auc score = 0.87933526588734
```

Matriz de confusão para o Multinomial NB:



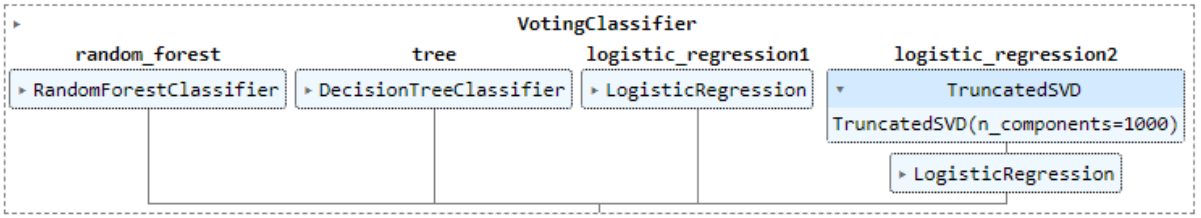
Fazendo o tuning dos parâmetros, ao aumentar muito o alpha (responsável pelo controle de suavização), foi observado que o modelo classificou a maioria dos samples como ‘Dresses’.

Para combinar alguns modelos utilizados, foi gerado um ensemble de Voting Classifier, o qual apresentou os melhores resultados com roc auc score, tanto no cross validation score quanto com o split em treino e teste:

```
rf = RandomForestClassifier(max_leaf_nodes = 504, n_estimators = 474, random_state = 1)
tree = DecisionTreeClassifier(max_leaf_nodes = 504)

log_r = LogisticRegression(max_iter = 8000)
log_r2 = LogisticRegression(max_iter = 8000)

reducer = TruncatedSVD(n_components = 1000)

log_r_pipe = Pipeline([
                ('reducer', reducer),
                ('logistic_regression', log_r2)
             ])

estimators = [
              ('random_forest', rf),
              ('tree', tree),
              ('logistic_regression1', log_r),
              ('logistic_regression2', log_r_pipe)
            ]

voter = VotingClassifier(estimators = estimators, voting = 'soft', n_jobs = 4)
```

```
%%time

voter.fit(train_X, train_y)
```

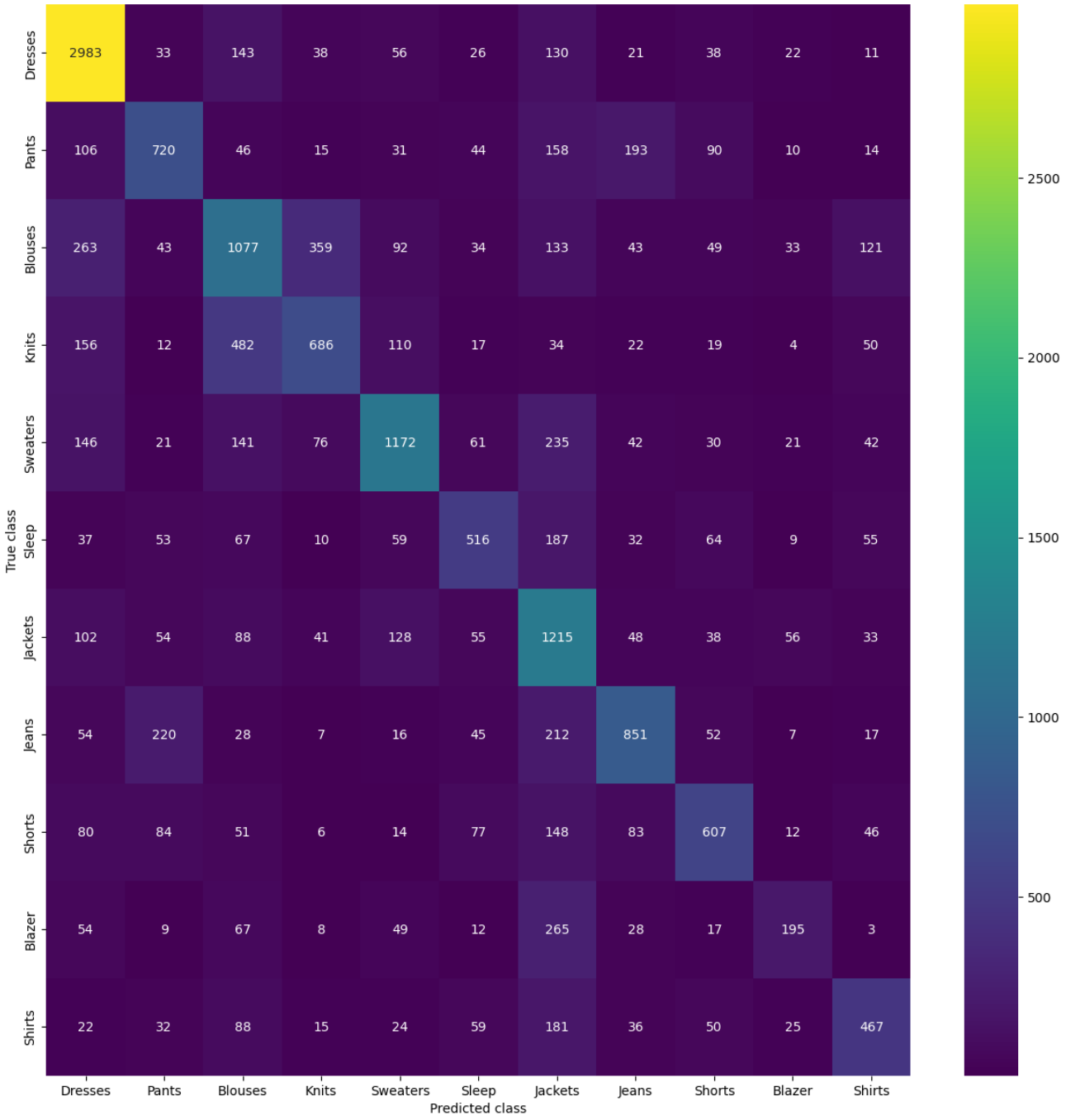
```
CPU times: user 772 ms, sys: 689 ms, total: 1.46 s
Wall time: 1min 16s
```




```
voting_preds = {'train': voter.predict(train_X),
                'test': voter.predict(test_X),
                'prob_train': voter.predict_proba(train_X),
                'prob_test': voter.predict_proba(test_X)
               }
```

```
Voting Ensemble train roc auc score = 0.9449857822626648
Voting Ensemble test roc auc score = 0.9047278033979486
```

Matriz de confusão para o Voting Classifier:




```
voter_scores = cross_val_score(voter, X, y, cv = cv, scoring = scorer, n_jobs = 4)
```

```
print(f'Voting Ensemble cross validation scores = {voter_scores}')
print(f'Voting Ensemble mean cross validation score = {voter_scores.mean()}')
```

```
Voting Ensemble cross validation scores = [0.886578   0.90661891 0.89401284 0.87561878]
Voting Ensemble mean cross validation score = 0.8907071320251179
```

Para comparar alguns dos modelos após um Grid Search, foi utilizada a seguinte função:

```
def train_test(name, model, X, y, grid={}):
    print(name)

    scoring = make_scorer(roc_auc_score, multi_class='ovo',needs_proba=True)

    res_grid = None
    test_size = 0.4
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=test_size,random_state=42, stratify = y)
    res = GridSearchCV(model,grid, scoring = scoring, n_jobs = 4)
    res.fit(X_train,Y_train)
    res_grid = res.best_params_

    prob_train = res.predict_proba(X_train)
    prob_test = res.predict_proba(X_test)

    scores = [roc_auc_score(Y_train, prob_train, multi_class = 'ovo'), roc_auc_score(Y_test, prob_test, multi_class = 'ovo')]

    print("Train score: ", scores[0])
    print("Test score: ", scores[1])
    print("Best params: ", res_grid)
    print("\n\n\n")

    models.append((name,scores[0],scores[1],res_grid))
    return res,res_grid,scores
```

KNN:

```
name = "KNN"
knn_grid = {'n_neighbors':[5,7,10, 15, 20]}
train_test(name,KNeighborsClassifier(),X,y,knn_grid)
```

✓ 4m 54.8s

```
KNN
Train score:  0.9391052797915587
Test score:  0.5640452633619534
Best params:  {'n_neighbors': 10}
```

MultinomialNB:

```
name = "Naive Bayes"
train_test(name,MultinomialNB(),X,y)
```

✓ 0.8s

```
Naive Bayes
Train score:  0.914849467140905
Test score:  0.8684332396182726
Best params:  {}
```

Random Forest:

```
name = "Random Forest Classifier"
random_forest_grid = {'n_estimators':[300,400,500, 600], 'max_depth':[50,100, None], 'max_leaf_nodes': [600,800,1000,None]}
train_test(name,RandomForestClassifier(), X, y, random_forest_grid)
```

✓ 116m 7.0s

```
Random Forest Classifier
Train score:  0.9582361880717837
Test score:  0.8864348411059657
Best params:  {'max_depth': 100, 'max_leaf_nodes': 1000, 'n_estimators': 600}
```

Estudos Futuros:

- Melhor análise para modelos de Voting
- BERTopic
- Análise de sentimento no dataset
- Teste com over sampling
- Outras combinações de Pipeline entre os modelos preditivos e técnicas de redução de dimensionalidade