Técnicas Clássicas de Reconhecimento de Padrões (2020/01)

Exercício 06 - Classificação de Textos

Aluno: Ramon Gomes Durães de Oliveira (2019720188)

O objetivo deste exercício é classificador de texto para reviews (avaliações). Para isso, será necessário pré-processar os dados e transformá-los em representações numéricas que permitam a utilização dos classificadores.

Bibliotecas

In [1]:
```
import itertools
import re
import string
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils.np_utils import to_categorical
```

Funções utilitárias para explorar o modelo de tokenização TF-IDF:

In [2]:
```
# Decodifica uma review
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding
feature names.'''
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

# Obtém as palavras com mais peso em uma review
def top_feats_in_doc(Xtr, features, row_id, top_n=25):
    ''' Top tfidf features in specific document (matrix row) '''
    row = np.squeeze(Xtr[row_id].toarray())
    return top_tfidf_feats(row, features, top_n)

# Retorna as n features (palavras) que, em média, são as mais importantes nas
 reviews desejadas
def top_mean_feats(Xtr, features, grp_ids=None, min_tfidf=0.1, top_n=25):
    ''' Return the top n features that on average are most important amongst d
ocuments in rows
        indentified by indices in grp_ids. '''
    if grp_ids:
        D = Xtr[grp_ids].toarray()
    else:
        D = Xtr.toarray()

    D[D < min_tfidf] = 0
    tfidf_means = np.mean(D, axis=0)
    return top_tfidf_feats(tfidf_means, features, top_n)

# Retorna uma lista de DataFrames na qual cada DF tem as top N features (palav
ras) e seu
# valor TFIDF médio, calculado em todos os documentos da mesma classe.
def top_feats_by_class(Xtr, y, features, min_tfidf=0.1, top_n=25):
    ''' Return a list of dfs, where each df holds top_n features and their mea
n tfidf value
        calculated across documents with the same class label. '''
    dfs = []
    labels = np.unique(y)
    for label in labels:
        ids = np.where(y==label)
        feats_df = top_mean_feats(Xtr, features, ids, min_tfidf=min_tfidf, top
_n=top_n)
        feats_df.label = label
        dfs.append(feats_df)
    return dfs

# Função utilitária para plotar resulados
def plot_tfidf_classfeats_h(dfs):
    ''' Plot the data frames returned by the function top_feats_by_class().
 '''
    fig = plt.figure(figsize=(12, 9), facecolor="w")
    x = np.arange(len(dfs[0]))
    for i, df in enumerate(dfs):
        ax = fig.add_subplot(1, len(dfs), i+1)
```

```
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.set_frame_on(False)
        ax.get_xaxis().tick_bottom()
        ax.get_yaxis().tick_left()
        ax.set_xlabel("Mean Tf-Idf Score", labelpad=16, fontsize=14)
        ax.set_title("label = " + str(df.label), fontsize=16)
        ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        ax.barh(x, df.tfidf, align='center', color='#3F5D7D')
        ax.set_yticks(x)
        ax.set_ylim([-1, x[-1]+1])
        yticks = ax.set_yticklabels(df.feature)
        plt.subplots_adjust(bottom=0.09, right=0.97, left=0.15, top=0.95, wspa
ce=0.52)
    plt.show()
```

Carregando e pré-processando os dados

Os dados que serão utilizados abaixo são provenientes da competição do Kaggle intitulada "Brazilian E-Commerce Public Dataset by Olist", disponível em https://www.kaggle.com/olistbr/brazilian-ecommerce (https://www.kaggle.com/olistbr/brazilian-ecommerce).

Os dados são avaliações de produtos na loja Olist feitas por consumidores brasileiros. Os dados serão carregados para visualização abaixo:

In [3]:
```
data = pd.read_csv('./data/olist_order_reviews_dataset.csv')
print(data.shape)
data.head()
```

```
(100000, 7)
```

Out[3]:

	review_id	order_id	review_score	review_
0	7bc2406110b926393aa56f80a40eba40	73fc7af87114b39712e6da79b0a377eb	4	
1	80e641a11e56f04c1ad469d5645fdfde	a548910a1c6147796b98fdf73dbeba33	5	
2	228ce5500dc1d8e020d8d1322874b6f0	f9e4b658b201a9f2ecdecbb34bed034b	5	
3	e64fb393e7b32834bb789ff8bb30750e	658677c97b385a9be170737859d3511b	5	
4	f7c4243c7fe1938f181bec41a392bdeb	8e6bfb81e283fa7e4f11123a3fb894f1	5	

Nota-se que as colunas mais relevantes para a análise são `review_score`, onde está a nota da avaliação de 1 a 5 estrelas, e `review_comment_message`, onde está o texto da avaliação em si.

Como estamos interessados em classificar novas avaliações, serão removidas as avaliações sem texto (NaN):

In [4]:
```
data.dropna(axis='rows', subset=['review_comment_message'], inplace=True)
print(data.shape)
```

```
(41753, 7)
```

Nota-se que a base de dados passa de 100.000 para 41753 observações. Agora serão checados valores faltantes na segunda coluna de interesse: `review_score` :

In [5]:
```
sum(data['review_score'].isna())
```

Out[5]:
```
0
```

Não há valores faltantes na coluna `review_score` .

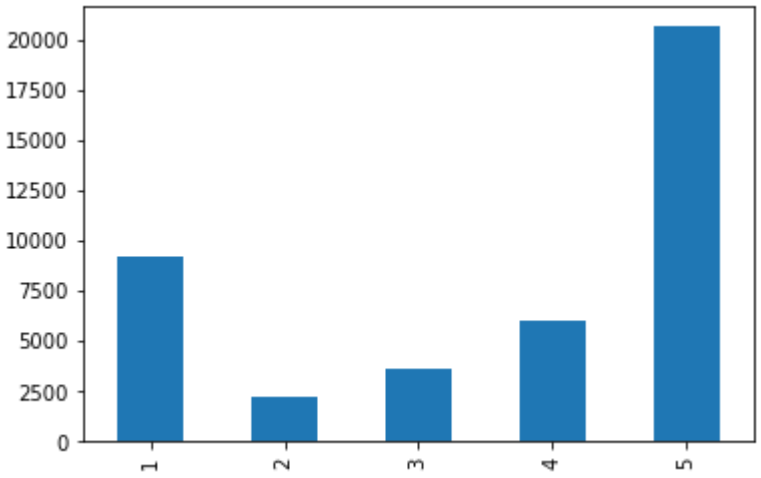
In [6]:
```
data['review_score'].head()
```

Out[6]:
```
3     5
4     5
9     4
12    4
15    5
Name: review_score, dtype: int64
```

Visualizando a distribuição de notas

In [7]:
```
data['review_score'].value_counts(sort=False).plot(kind='bar')
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x265118a2348>
```



Para este trabalho, estamos interessados apenas em distinguir avaliações positivas de negativas, por isso elas serão agregadas de forma que as notas de 1 a 3 sejam consideradas negativas (-1) e as notas 4 e 5 positivas (1).

In [8]:
```
data['grouped_review'] = data['review_score'].where(data['review_score']>3, -1
)
data['grouped_review'] = data['grouped_review'].where(data['grouped_review']<=
3, 1)
data[['review_comment_message', 'review_score', 'grouped_review']].head(15)
```

Out[8]:

	review_comment_message	review_score	grouped_review
3	Recebi bem antes do prazo estipulado.	5	1
4	Parabéns lojas lannister adorei comprar pela I...	5	1
9	aparelho eficiente. no site a marca do aparelh...	4	1
12	Mas um pouco ,travando...pelo valor ta Boa.\r\n	4	1
15	Vendedor confiável, produto ok e entrega antes...	5	1
16	GOSTARIA DE SABER O QUE HOUVE, SEMPRE RECEBI E...	2	-1
19	Péssimo	1	-1
22	Loja nota 10	5	1
24	obrigado pela atençao amim dispensada	5	1
27	A compra foi realizada facilmente.\r\nA entreg...	5	1
28	relógio muito bonito e barato.	5	1
29	Não gostei ! Comprei gato por lebre	1	-1
32	Sempre compro pela Internet e a entrega ocorre...	1	-1
34	Recebi exatamente o que esperava. As demais en...	4	1
36	Recomendo ,	5	1

Limpeza do texto

Dados textuais crus contém uma série de caracteres indesejáveis, além de letras maiúsculas e minúsculas que deveriam ser tratadas como a mesma palavra. A função abaixo removerá:

- pontuação
- letras maiúsculas
- colchetes
- padrões especiais como \r e \n
- aspas

In [9]:
```
def text_clean(text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\r\n|\r|\n', '', text)
    text = re.sub('[‘’“”…]', '', text)
    return text

clean = lambda x: text_clean(x)
```

Exibindo algumas avaliações pós-processamento:

In [10]:
```
data['clean_review_message'] = pd.DataFrame(data.review_comment_message.apply(
clean))
data['clean_review_message'].head(10)
```

Out[10]:
```
3                  recebi bem antes do prazo estipulado
4     parabéns lojas lannister adorei comprar pela i...
9     aparelho eficiente no site a marca do aparelho...
12               mas um pouco travandopelo valor ta boa
15    vendedor confiável produto ok e entrega antes ...
16    gostaria de saber o que houve sempre recebi e ...
19                                              péssimo
22                                          loja nota
24                obrigado pela atençao amim dispensada
27    a compra foi realizada facilmentea entrega foi...
Name: clean_review_message, dtype: object
```

Separando conjuntos de treinamento e teste

Serão utilizados 20% dos dados fornecidos como dados de teste.

In [11]:
```
from sklearn.model_selection import train_test_split

X = data['clean_review_message']
Y = data['grouped_review']

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size = 0.2, random_
state=1)

print("Tamanho do conjunto de treinamento: {}\nTamanho do conjunto de teste:
{}".format(len(Ytrain), len(Ytest)))
```

```
Tamanho do conjunto de treinamento: 33402
Tamanho do conjunto de teste: 8351
```

Modelo 1: TF-IDF + Regressão Logística

Tokenização: TF-IDF:

Para este modelo, a forma escolhida de representação do texto foi a transformação (vetorização) pelo método TF-IDF (Term Frequency - Inverse Document Frequency).

O objetivo deste método é aumentar o peso dado a uma palavra (feature) quanto maior o número de ocorrências dela em uma review, mas atenuar este peso de acordo com o número total de ocorrências dela em todo o conjutno de dados. Com isso, é possível remover a importância dada a palavras extremamente comuns como artigos ("o", "a") e outras palavras muito comuns que não ajudariam a discernir uma review positiva de uma review negativa ("é", "eu", ...).

Olhando mais a fundo para o significado de cada termo:

- TF (Term Frequency): frequência de uma palavra na review em questão. Sendo t uma palavra e d uma review, a função é definida por:

$$tf(t,d) = \frac{n_{t,d}}{\sum_k n_{t,d}}$$

- IDF (Inverse Document Frequency): número total de reviews dividido pelo número de reviews que contém o termo em questão. Sendo N o número total de reviews e D o conjunto contendo todas as reviews do banco de dados:

$$idf(t,D) = log(\frac{N}{|\{d \in D | t \in d\}|})$$

- TF-IDF (Inverse Document Frequency): a combinação dessas funções:

$$tfidf(t,D) = tf(t,D) * idf(t,D)$$

Ajustando este modelo aos dados e exibindo parte do dicionário gerado:

In [12]:
```
vectorizer = TfidfVectorizer()
classifier = LogisticRegression(solver = "lbfgs")
model = Pipeline([('vectorizer',vectorizer),('classifier',classifier)])
model.fit(Xtrain, Ytrain);
tfidf = model.steps[0][1]
dict(itertools.islice(tfidf.vocabulary_.items(), 10))
```

Out[12]:
```
{'produto': 13119,
 'não': 11127,
 'foi': 7647,
 'entregue': 6228,
 'gostaria': 8177,
 'de': 4456,
 'saber': 14909,
 'nova': 11066,
 'previsão': 12978,
 'entrega': 6129}
```

Para visualizar um exemplo do resultado da tokenização, será utilizada a frase:

In [13]:
```
print(Xtrain.iloc[0])
```

```
o produto não foi entregue gostaria de saber a nova previsão de entrega o pro
cesso de entrega está na situação de nota fiscal emitida desde  sem ter nunca
ido para transportadora
```

Após a tokenização, este é o peso dado às 10 palavras mais relevantes desta frase:

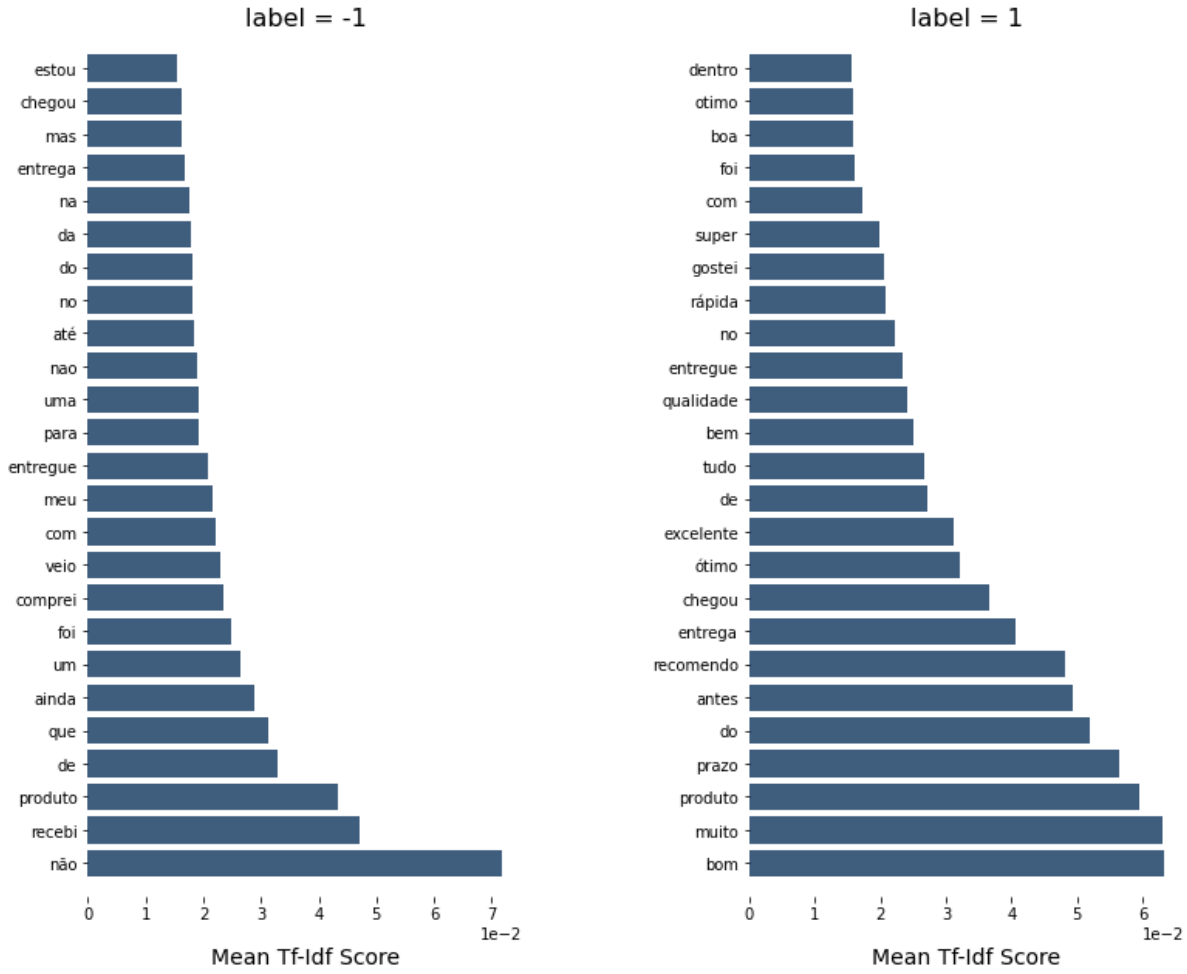
In [14]:
```
transformed_Xtrain = tfidf.transform(Xtrain)
all_features = tfidf.get_feature_names()
row1 = np.squeeze(transformed_Xtrain[0].toarray())
top_tfidf_feats(row = row1, features=all_features, top_n=10)
```

Out[14]:

	feature	tfidf
0	de	0.336112
1	ido	0.324013
2	nova	0.269831
3	emitida	0.260999
4	situação	0.255441
5	transportadora	0.237235
6	processo	0.236812
7	previsão	0.227022
8	desde	0.226091
9	entrega	0.194867

Exibindo as palavras mais relevantes por classe (lembrando que -1 representa avaliações negativas e 1 representa avaliações positivas).

In [15]:
```
dfs = top_feats_by_class(transformed_Xtrain, Ytrain, tfidf.get_feature_names
())
plot_tfidf_classfeats_h(dfs)
```



Predição do Modelo: Regressão Logística

Para classificar os dados, novamente será utilizado um simples modelo de regressão logística. Abaixo é mostrada a matriz de confusão obtida:

In [16]:
```
predictions = model.predict(Xtest)
confusion_matrix(predictions, Ytest)
```

Out[16]:
```
array([[2552,  446],
       [ 463, 4890]], dtype=int64)
```

In [17]:
```
scores = cross_val_score(model, Xtrain, Ytrain, cv=5)
cv_accuracy = 100 * np.mean(scores)
print("Acurácia obtida por validações cruzada com 5 partições: {}%".format(rou
nd(cv_accuracy, 2)))
```

```
C:\Users\ramon\Anaconda3\envs\tf\lib\site-packages\sklearn\linear_model\_logi
stic.py:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Acurácia obtida por validações cruzada com 5 partições: 89.26%
```

Testando com novas frases:

In [18]:
```
model.predict(['estou feliz com a compra'])
```

Out[18]:
```
array([1], dtype=int64)
```

In [19]:
```
model.predict(['fiquei chateado'])
```

Out[19]:
```
array([-1], dtype=int64)
```

In [20]:
```
model.predict(['estou triste com a compra'])
```

Out[20]:
```
array([-1], dtype=int64)
```

In [21]:
```
model.predict(['me agradou muito'])
```

Out[21]:
```
array([1], dtype=int64)
```

In [22]:
```
model.predict(['demorou a chegar'])
```

Out[22]:
```
array([-1], dtype=int64)
```

A pipeline de processamento de dados, juntamente ao modelo de tokenização TF-IDF e à classificação por regressão logística simples se mostraram eficientes na solução do problema, obtendo 89% de acurácia. Como esperado, o modelo classifica melhor as avaliações positivas devido ao desbalanceamento das classes.

O modelo foi capaz de classificar corretamente as frases de teste fornecidas.

Modelo 2: Bag of Words + Regressão Logística

Para avaliar se a tokenização pelo método TF-IDF gerou resultados melhores que o Bag of Words, um modelo similar será treinado para esta forma de tokenização.

In [23]:
```
vectorizer2 = CountVectorizer()
classifier2 = LogisticRegression(solver = "liblinear")
model2 = Pipeline([('vectorizer',vectorizer2),('classifier',classifier2)])
model2.fit(Xtrain, Ytrain)
predictions2 = model2.predict(Xtest)
confusion_matrix(predictions2, Ytest)
```

Out[23]:
```
array([[2462,  388],
       [ 553, 4948]], dtype=int64)
```

In [24]:
```
scores2 = cross_val_score(model2, Xtrain, Ytrain, cv=5)
cv_accuracy2 = 100 * np.mean(scores2)
print("Acurácia obtida por validações cruzada com 5 partições: {}%".format(rou
nd(cv_accuracy2, 2)))
```

```
Acurácia obtida por validações cruzada com 5 partições: 89.03%
```

A média da acurácia do modelo utilizando TF-IDF através da validação cruzada é de 89.26%, enquanto a média do modelo que utiliza Bag of Words é de 89.03%. A diferença é muito pequena em favor do método TF-IDF.

Modelo 3: Tokenização simples + LSTM

Uma outra forma de realizar a mesma tarefa acima é através de redes profundas. Como a tarefa de compreensão semântica de textos depende da sequência (da ordem) em que as palavras aparecem, as redes recorrentes são bastante adequadas. Ao contrário das redes "feed-forward", elas também contém cadamadas recorrentes, com pesos que conectam uma camada a si mesma.

As LSTMs (Long-Short Term Memory) são um tipo de rede recorrente que lidam com o problema do "vanishing/exploding gradient" (desaparecimento/explosão do gradiente) que impedem que redes recorrentes simples aprendam relações entre palavras muito distantes entre si.

A implementação da tokenização e das redes LSTM a ser utilizada é a da biblioteca Keras (utilizando o TensorFlow como backend).

Tokenização:

Bem como para os modelos anteriores, é preciso representar o texto de forma numérica através da tokenização. Neste caso, vamos limitar o vocabulário para um máximo de 2000 palavras: as 2000 palavras mais frequentes.

Tendo cada palavra representada por um número diferente, cada avaliação do conjunto de dados será transformada em uma sequência de números na ordem em que as respectivas palavras aparecem.

Como as redes têm um tamanho de entrada fixo, é preciso fazer com que as frases tenham tamanhos fixos. Isso pode ser feito preenchendo as sequências mais curtas e/ou truncando as mais longas. Neste caso, todas as frases menores que a frase mais longa dos dados serão preenchidas com zeros.

In [39]:
```
# limitando o tamanho do vocabulário
max_features = 2000
# codificando as palavras
tokenizer = Tokenizer(num_words=max_features, split=' ')
# ajustando aos dados e transformando-os em sequências de tokens
tokenizer.fit_on_texts(data['clean_review_message'].values)
X = tokenizer.texts_to_sequences(data['clean_review_message'].values)
# preenchimento das sequências para equalizar o tamanho da entrada
X = pad_sequences(X)
```

LSTM

Abaixo será construída a arquitetura da rede a ser utilizada, com as camadas:

- Embedding (43, 128): codifica as frases tokenizadas em vetores de tamanho 128;
- Dropout (43, 128): uma camada de regularização;
- LSTM (128, 196): a camada recorrente principal;
- Dense (196, 2): a camada de saída com dois neurônios (um para cada classe).

Como este modelo demora bastante a ser treinado, não utilizarei validação cruzada como nos modelos anteriores. Ao invés disso, adicionarei uma técnica de regularização chamada "dropout". Em resumo, ela adiciona uma probabilidade de zerar as ativações da rede, impedindo o sobreajuste.

In [40]:
```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_features, embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='softmax'))
```

In [41]:
```
model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = [
'accuracy'])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 43, 128)           256000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 43, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 394
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

Separação entre treinamento e teste:

In [42]:
```
Ydummies = pd.get_dummies(data['grouped_review']).values

X_train, X_test, Y_train, Y_test = train_test_split(X,Ydummies, test_size = 0.
33, random_state = 42)
```

Ajuste do modelo. Novamente, serão utilizadas apenas 7 épocas devido ao tempo que leva para treinara a rede.

In [45]:
```
batch_size = 32
model.fit(X_train, Y_train, epochs = 7, batch_size=batch_size, verbose = 2)
```

```
C:\Users\ramon\AppData\Roaming\Python\Python37\site-packages\tensorflow\pytho
n\framework\indexed_slices.py:434: UserWarning: Converting sparse IndexedSlic
es to a dense Tensor of unknown shape. This may consume a large amount of mem
ory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Epoch 1/7
 - 112s - loss: 0.3349 - accuracy: 0.8663
Epoch 2/7
 - 100s - loss: 0.2728 - accuracy: 0.8994
Epoch 3/7
 - 102s - loss: 0.2543 - accuracy: 0.9063
Epoch 4/7
 - 102s - loss: 0.2369 - accuracy: 0.9131
Epoch 5/7
 - 100s - loss: 0.2243 - accuracy: 0.9184
Epoch 6/7
 - 101s - loss: 0.2122 - accuracy: 0.9230
Epoch 7/7
 - 100s - loss: 0.2023 - accuracy: 0.9259
```

Out[45]:
```
<keras.callbacks.callbacks.History at 0x2651bc55b48>
```

É possível perceber que a acurácia no conjunto de treinamento aumenta com o passar das épocas. Abaixo a performance da rede será avaliada no conjunto de teste.

In [54]:
```
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = batch_siz
e)
print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
score: 0.29
acc: 0.90
```

A acurácia obtida para este modelo é de 90%. Testando novas reviews para validar os resultados:

In [55]:
```
# Review e tokenização
review = ['odiei esse produto']
review = tokenizer.texts_to_sequences(review)
review = pad_sequences(review, maxlen=43, dtype='int32', value=0)
print(review)

# Predição do modelo
sentiment = model.predict(review,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("review negativa")
elif (np.argmax(sentiment) == 1):
    print("review positiva")
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0 166   2]]
review negativa
```

In [56]:
```
# Review e tokenização
review = ['o produto produto chegou no prazo esperado']
review = tokenizer.texts_to_sequences(review)
review = pad_sequences(review, maxlen=43, dtype='int32', value=0)
print(review)

# Predição do modelo
sentiment = model.predict(review,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("review negativa")
elif (np.argmax(sentiment) == 1):
    print("review positiva")
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    1   2   2  14  16   9 158]]
review positiva
```

Conclusão

Neste trabalho, foram utilizadas 3 abordagens para solucionar o problema de classificação de avaliações de clientes no site de e-commerce da Olist Brasil, cujas acurácias obtidas foram:

- TF-IDF + Regressão Logística: 89.26%
- Bag of Words + Regressão Logística : 89.03%
- Tokenização simples + LSTM: 89.56%

Em relação à performance, a rede LSTM foi a melhor. Apesar de ter obtido resultados apenas ligeiramente superiores às demais abordagens, ressalta-se que não foi realizado nenhum tipo de otimização dos parâmetros dessa rede, e que ela foi treinada por apenas 7 épocas devido ao tempo. Logo, ela parece ser a de maior potencial para obter acurácias ainda maiores dentre as abordagens testadas.

Os resultados mostram que as três formas de resolução do problema são eficazes. O objetivo do trabalho era explorar e validar diferentes abordagens de análise de sentimento em textos, e foi cumprido com sucesso.