bagsofpopcorn_jul29

July 29, 2019

[74]:
```
# Load libraries

import re
import csv

import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns

from wordcloud import WordCloud, STOPWORDS

import pandas as pd

import numpy as np

from scipy import stats

from bs4 import BeautifulSoup

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from textblob import TextBlob
from textblob.classifiers import NaiveBayesClassifier, NLTKClassifier,␣
 ↪DecisionTreeClassifier

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer,␣
 ↪TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.svm import LinearSVC
```

[86]:
```python
# Load data

def load_tsv_data(file_path):
    return pd.read_csv(file_path, delimiter='\t', quoting=csv.QUOTE_NONE,␣
 ↪header=0)

train_df = load_tsv_data("data/labeledTrainData.tsv")

test_df = load_tsv_data("data/testData.tsv")

unlabeled_df = load_tsv_data("data/unlabeledTrainData.tsv")
```

[3]:
```python
# Overview loaded data

def overview_dataset(dataset_df):
    # Data inside
    display(dataset_df.head(3))
    display(dataset_df.tail(3))
    # Dimensions and size
    display(dataset_df.shape)
    # Columns names
    display(dataset_df.columns.values)
    # Duplicated values
    display(dataset_df[dataset_df.duplicated(keep=False)])
    # .describe()
    display(dataset_df.describe(include='all').T)
```

[4]:
```python
overview_dataset(train_df)

overview_dataset(test_df)

overview_dataset(unlabeled_df)
```

	id	sentiment	review
0	"5814_8"	1	"With all this stuff going down at the moment ...
1	"2381_9"	1	"\"The Classic War of the Worlds\" by Timothy ...
2	"7759_3"	0	"The film starts with a manager (Nicholas Bell...

	id	sentiment	review
24997	"10905_3"	0	"Guy is a loser. Can't get girls, needs to bui...
24998	"10194_3"	0	"This 30 minute documentary Buñuel made in the...
24999	"8478_8"	1	"I saw this movie as a child and it broke my h...

```
(25000, 3)
```

```
array(['id', 'sentiment', 'review'], dtype=object)
```

```
Empty DataFrame
Columns: [id, sentiment, review]
Index: []
```

```
           count unique                                                top  \
id         25000  25000                                          "7585_3"
sentiment  25000    NaN                                                NaN
review     25000  24904  "You do realize that you've been watching the ...

          freq mean      std  min  25%  50%  75%  max
id           1  NaN      NaN  NaN  NaN  NaN  NaN  NaN
sentiment  NaN  0.5  0.50001    0    0  0.5    1    1
review       3  NaN      NaN  NaN  NaN  NaN  NaN  NaN
```

```
           id                                             review
0  "12311_10"  "Naturally in a film who's main themes are of ...
1    "8348_2"  "This movie is a disaster within a disaster fi...
2    "5828_4"  "All in all, this is a movie for kids. We saw ...
```

```
               id                                             review
24997    "2531_1"  "I was so disappointed in this movie. I am ver...
24998    "7772_8"  "From the opening sequence, filled with black ...
24999  "11465_10"  "This is a great horror film for people who do...
```

```
(25000, 2)
```

```
array(['id', 'review'], dtype=object)
```

```
Empty DataFrame
Columns: [id, review]
Index: []
```

```
        count unique                                                top freq
id      25000  25000                                           "7585_3"    1
review  25000  24801  "Loved today's show!!! It was a variety and no...    5
```

```
          id                                             review
0   "9999_0"  "Watching Time Chasers, it obvious that it was...
1  "45057_0"  "I saw this film about 20 years ago and rememb...
2  "15561_0"  "Minor Spoilers<br /><br />In New York, Joan B...
```

```
                 id                                             review
49997  "16006_0"  "Griffin Dunne was born into a cultural family...
49998  "40155_0"  "Not a bad story, but the low budget rears its...
49999  "35270_0"  "This not-very-good mummy-alien flick does fea...
```

```
(50000, 2)
```

```
array(['id', 'review'], dtype=object)
```

```
Empty DataFrame
Columns: [id, review]
Index: []
```

```
        count unique                                               top freq
id      50000  50000                                         "47629_0"    1
review  50000  49507  "Am not from America, I usually watch this sho...    5
```

[5]:
```
# Explore the data

# Explore sentiments of the reviews

display(
    train_df['sentiment'].value_counts()
)

display(
    train_df.groupby('sentiment')['review'].describe()
)
```

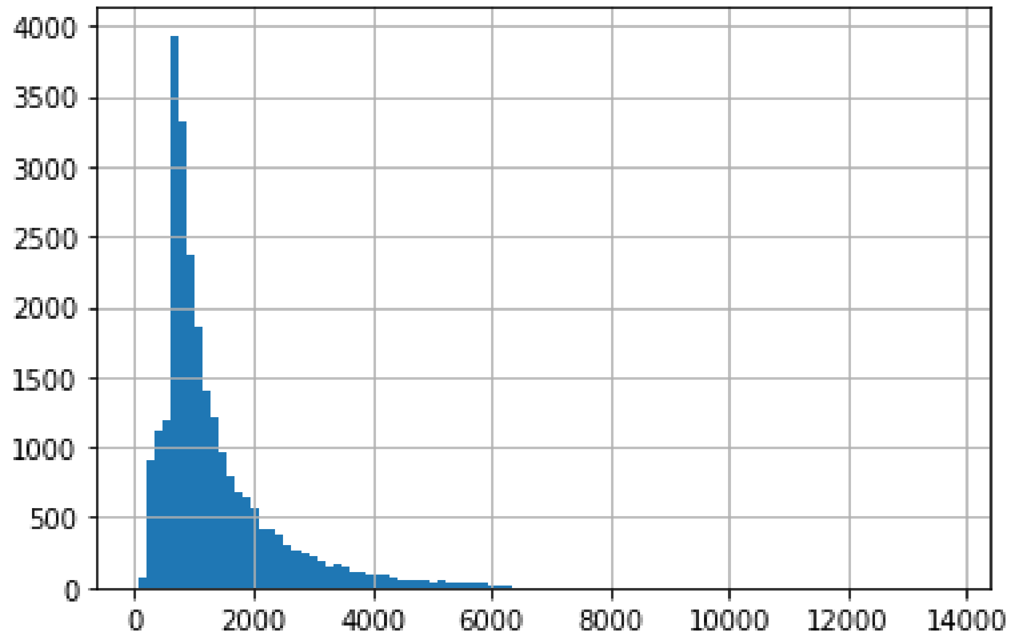
```
1    12500
0    12500
Name: sentiment, dtype: int64
```

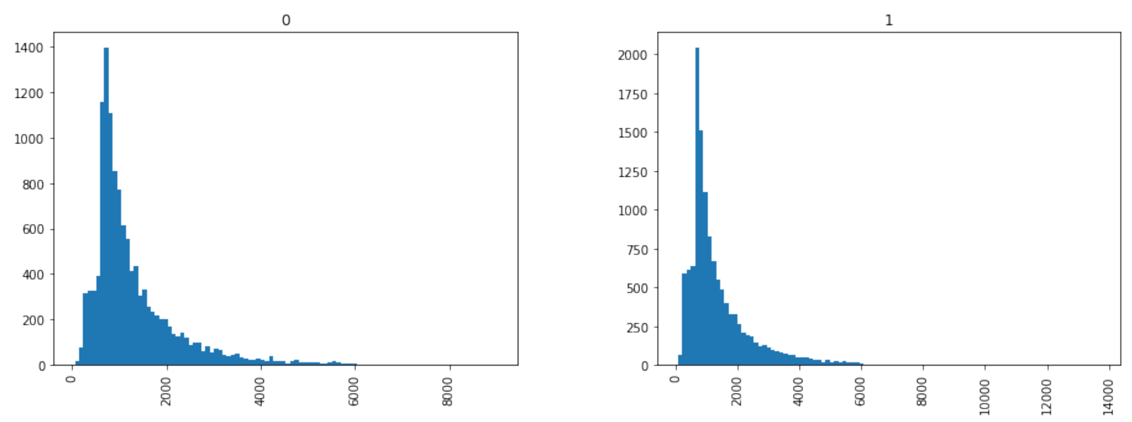
```
           count unique                                                top  \
sentiment
0          12500  12432  "When i got this movie free from my job, along...
1          12500  12472  "I'm gonna tip the scales here a bit and say I...

          freq
sentiment
0            3
1            2
```

[6]:
```python
# Explore length of the reviews

train_df['rev_len'] = train_df['review'].apply(len)
```

[7]:
```python
display(train_df['rev_len'].describe())

# Display distribution of the reviews by review length
train_df['rev_len'].hist(bins=100)
plt.show()

# Display distribution of the reviews by review length and by sentiment score
train_df.hist(column='rev_len', by='sentiment', bins=100, figsize=(15, 5))
plt.show()

# Display Kolmogorov-Smirnov statistic
# From scipy docs:
    # If the K-S statistic is small or the p-value is high,
    # then we cannot reject the hypothesis that
    # the distributions of the two samples are the same.
grouped_by_sentiment = train_df.groupby('sentiment')['rev_len']
display(
    stats.ks_2samp(
        grouped_by_sentiment.get_group(0),
        grouped_by_sentiment.get_group(1)
    )
)  # statistic=0.027760000000000007, pvalue=0.0001310970303242206

# Conclusion: reject the hypothesis that the distributions are the same.
```

```
count    25000.000000
mean      1329.710560
std       1005.239246
min         54.000000
25%        705.000000
50%        983.000000
75%       1619.000000
max      13710.000000
Name: rev_len, dtype: float64
```






```
Ks_2sampResult(statistic=0.027760000000000007, pvalue=0.0001310970303242206)
```

[40]:
```
# Clean the data

wnlemmatizer = WordNetLemmatizer()

def clean_review(raw_text, to_lower, lemmatize, remove_numbers,␣
↪remove_stopwords, return_tokens=False):
    # 1
```

```
    text_nohtml = BeautifulSoup(raw_text).get_text()
    # 2
    if remove_numbers:
        re_clean_pattern = "[^a-zA-Z]"
    else:
        re_clean_pattern = "[^a-zA-Z0-9]"
    text_regexclean = re.sub(re_clean_pattern, " ", text_nohtml)
    # 3
    if to_lower:
        text_tokens = text_regexclean.lower().split(" ")
    else:
        text_tokens = text_regexclean.split(" ")
    # 4
    if remove_stopwords:
        nltk_stopwords = set(stopwords.words("english"))
        text_tokens_nostopwords = [
            token for token in text_tokens
            if token not in nltk_stopwords
        ]
        text_tokens = text_tokens_nostopwords
    # 5
    if lemmatize:
        text_lemmatized_tokens = [wnlemmatizer.lemmatize(token) for token in␣
↪text_tokens]
        text_lemmatized_tokens = [wnlemmatizer.lemmatize(token, "v") for token␣
↪in text_lemmatized_tokens]
        text_tokens = text_lemmatized_tokens
    # 6
    text_cleaned = " ".join(text_tokens)
    if return_tokens:
        return text_tokens
    return text_cleaned
```

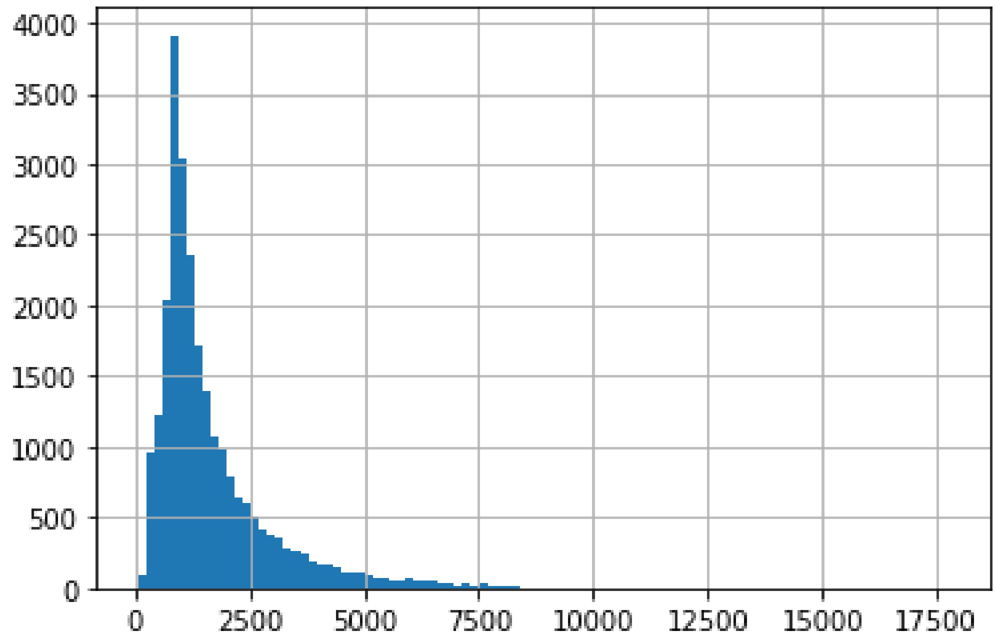
[9]:
```
# Apply data cleaning to datasets;
# Create columns to describe amount of tokens and length of cleaned review

to_lower = True
lemmatize = True
remove_numbers = True
remove_stopwords = True

train_df['cleaned_review'] = train_df['review'].apply(
    lambda x: clean_review(
        x,
        to_lower=to_lower,
        lemmatize=lemmatize,
        remove_numbers=remove_numbers,
```

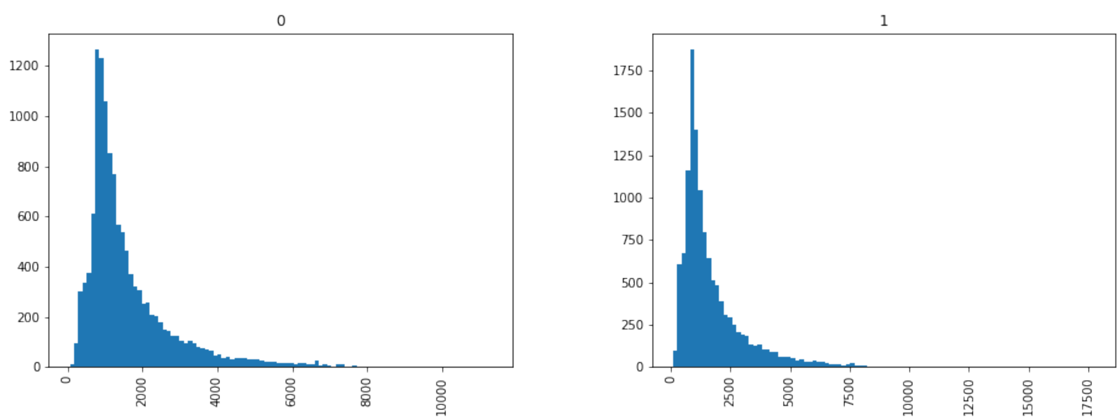
```
        remove_stopwords=remove_stopwords
    )
)

train_df['cln_rev_len'] = train_df['cleaned_review'].apply(
    lambda x: len(' '.join(x))
)

train_df['cln_rev_tokens_len'] = train_df['cleaned_review'].apply(len)
```

[10]:

```
display(train_df.describe())

# Explore created clb_rev_len feature

# Display distribution of the reviews by cleaned review length
train_df['cln_rev_len'].hist(bins=100)
plt.show()
# Display distribution of the reviews by cleaned review length and by sentiment␣
↪score
train_df.hist(column='cln_rev_len', by='sentiment', bins=100, figsize=(15, 5))
plt.show()
# Display Kolmogorov-Smirnov statistic
grouped_by_sentiment = train_df.groupby('sentiment')['cln_rev_len']
display(
    stats.ks_2samp(
        grouped_by_sentiment.get_group(0),
        grouped_by_sentiment.get_group(1)
    )
)  # statistic=0.030240000000000045, pvalue=2.1713577117769O4e-05

# Conclusion: reject the hypothesis that the distributions are the same.
```

	sentiment	rev_len	cln_rev_len	cln_rev_tokens_len
count	25000.00000	25000.000000	25000.00000	25000.00000
mean	0.50000	1329.710560	1646.59552	823.79776
std	0.50001	1005.239246	1271.59652	635.79826
min	0.00000	54.000000	57.00000	29.00000
25%	0.00000	705.000000	855.00000	428.00000
50%	0.50000	983.000000	1209.00000	605.00000
75%	1.00000	1619.000000	2003.00000	1002.00000
max	1.00000	13710.000000	17783.00000	8892.00000

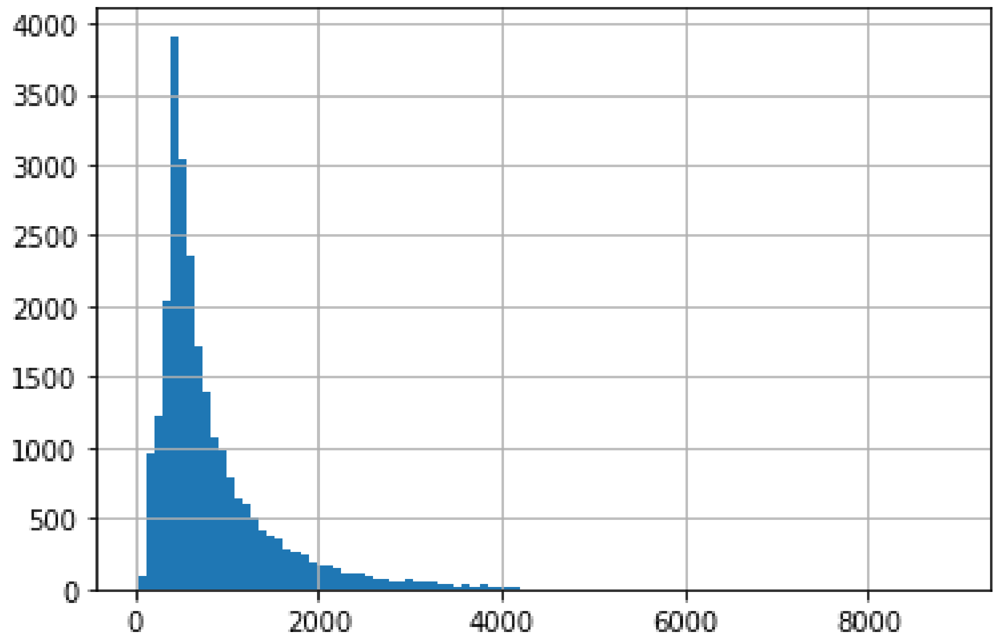


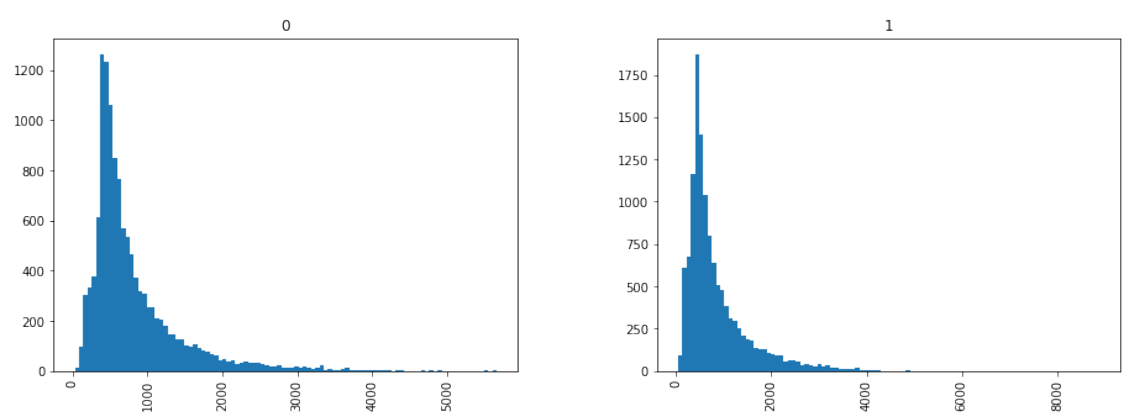



```
Ks_2sampResult(statistic=0.030240000000000045, pvalue=2.171357711776904e-05)
```

[11]:
```
# Explore created cln_rev_tokens_len feature

# Display distribution of the reviews by tokens cnt from cleaned review length
train_df['cln_rev_tokens_len'].hist(bins=100)
plt.show()
# Display distribution of the reviews by tokens cnt and by sentiment score
```

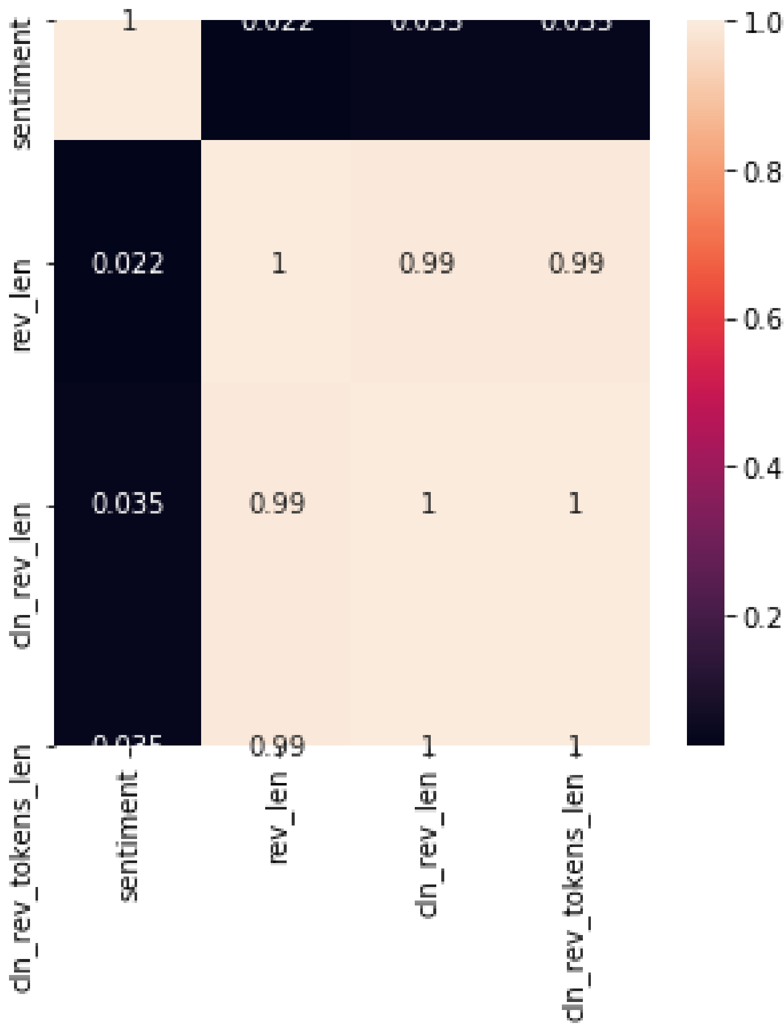
```
train_df.hist(column='cln_rev_tokens_len', by='sentiment', bins=100,␣
 ↪figsize=(15, 5))
plt.show()
# Display Kolmogorov-Smirnov statistic
grouped_by_sentiment = train_df.groupby('sentiment')['cln_rev_tokens_len']
display(
    stats.ks_2samp(
        grouped_by_sentiment.get_group(0),
        grouped_by_sentiment.get_group(1)
    )
)  # statistic=0.030240000000000045, pvalue=2.171357711776904e-05

# Conclusion: reject the hypothesis that the distributions are the same.
```





```
Ks_2sampResult(statistic=0.030240000000000045, pvalue=2.171357711776904e-05)
```

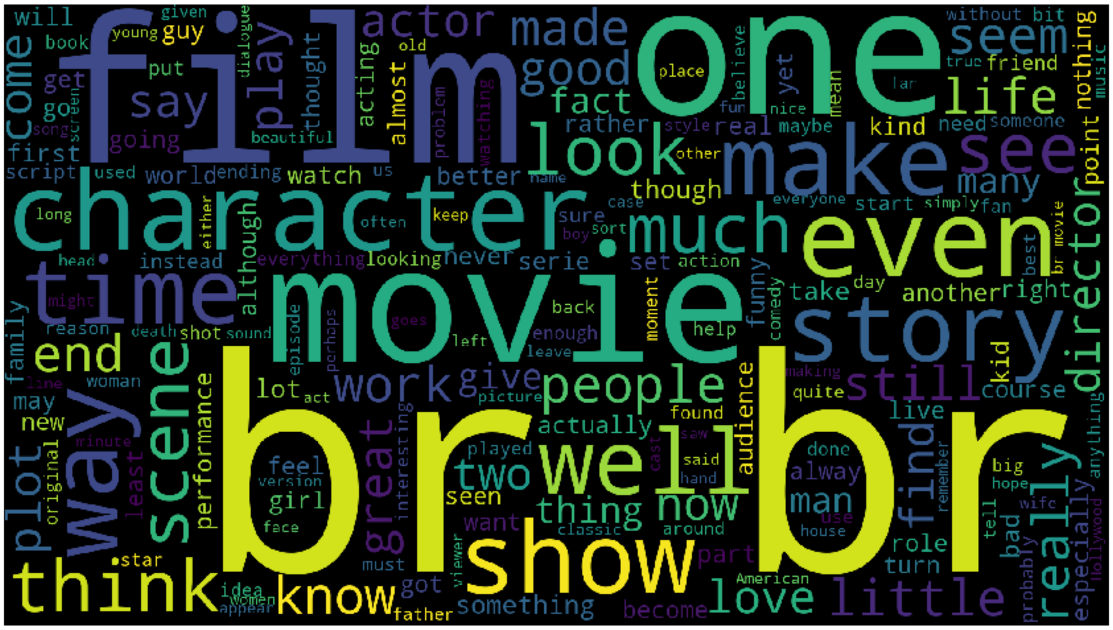
[12]:
```python
plt.figure(figsize=(5, 5))
sns.heatmap(train_df.corr(), annot=True)
plt.show()
```

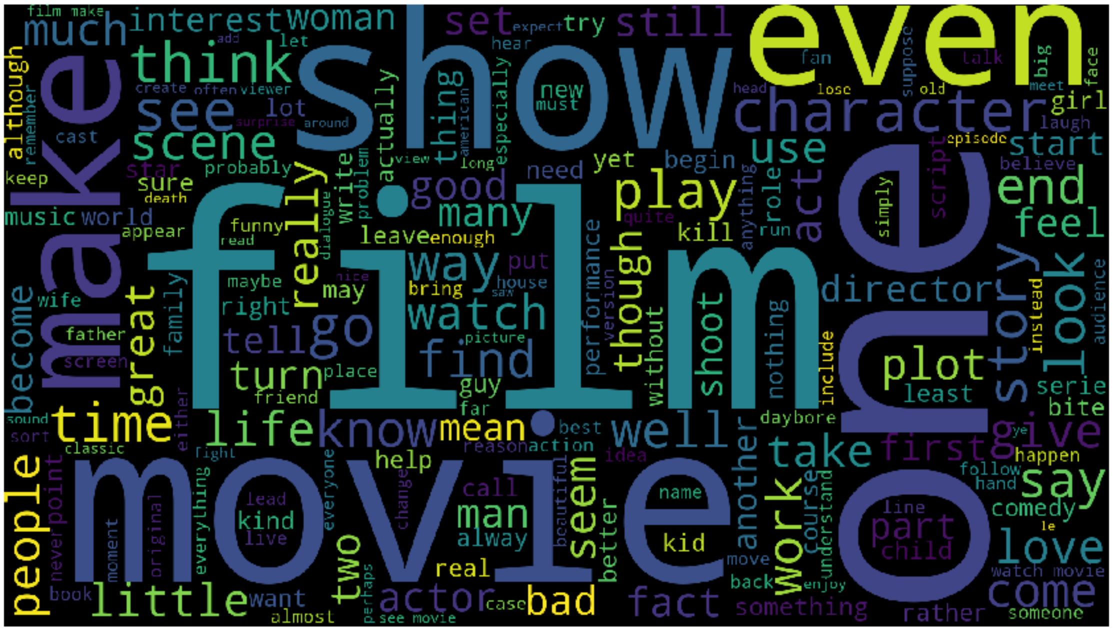


[13]:
```python
# Display cloud of words for the datasets

def display_word_cloud(dataset_df, col_name):
    wordcloud_obj = WordCloud(width=1920, height=1080)
    wordcloud_img = wordcloud_obj.generate(
        ' '.join(dataset_df.loc[:, col_name])
```

```
    )
    plt.figure(figsize=(15, 10))
    plt.imshow(wordcloud_img)
    plt.axis('off')
    plt.show()
```

[14]:
```
display_word_cloud(train_df, 'review')

display_word_cloud(train_df, 'cleaned_review')
```





[15]:
```
# Function to display statistics on predicted values

def display_y_pred_stats(y_true, y_pred):
    # Display confusion matrix
    cm = pd.crosstab(y_true, y_pred)
    TN = cm.iloc[0, 0]
    FN = cm.iloc[1, 0]
    TP = cm.iloc[1, 1]
    FP = cm.iloc[0 ,1]
    display("Confusion matrix", cm)
    # Display accuracy metrics
    display("Accuracy (custom) is {0}".format(round(((TP+TN)*100)/
 ↪(TP+TN+FP+FN), 2)))
    display("Accuracy (sklearn) is {0}".format(accuracy_score(y_true, y_pred)))
    display("FN rate: {0}".format(round((FN*100)/(FN+TP), 2)))
    display("FP rate: {0}".format(round((FP*100)/(FP+TN), 2)))
    display("F1 score is {0}".format(f1_score(y_true, y_pred)))
    # Display classification report
    print(classification_report(y_true, y_pred))
```

[16]:
```
# Functions to try out BOW and TfIdf

def vectorize_df_col(dataset_df, col_name, perform_tfidf=False,␣
 ↪cv_max_features=None):
    """Vectorize dataset_df[col_name] using BOW algorithm.
    Apply Tf-idf transofrmation after that
    """
```

```
    vectorized_words = CountVectorizer(max_features=cv_max_features).
 ↪fit_transform(dataset_df[col_name])
    if perform_tfidf:
        normalized_words = TfidfTransformer().fit_transform(vectorized_words)
        return normalized_words
    return vectorized_words

def apply_model(X_train_full, y_train_full, model, display_stats=False,␣
 ↪return_validat=False):
    X_train, X_validat, y_train, y_validat = train_test_split(
        X_train_full, y_train_full,
        test_size=0.35, random_state=42
    )
    model = model.fit(X_train, y_train)
    y_pred = model.predict(X_validat)
    if display_stats:
        display_y_pred_stats(y_validat, y_pred)
    if return_validat:
        return (y_validat, y_pred)
    return y_pred
```

[17]:

```
# Approach 1: use raw reviews (don't clean them) to predict sentiment
# NOTE: try out both BOW+tfidf and BOW (== no tf-idf) approaches

models = [
    LogisticRegression(solver='saga', max_iter=10000),
#     MultinomialNB(),
#     RandomForestClassifier(n_estimators=500, n_jobs=-1, verbose=2)
]
X_train_full = vectorize_df_col(train_df, 'review', perform_tfidf=False,␣
 ↪cv_max_features=10000)
y_train_full = train_df['sentiment']
for model in models:
    y_pred = apply_model(X_train_full, y_train_full, model, display_stats=True)

# Results: accuracy with tf-idf
# LogReg, solver=saga: 89.03
# MultinomialNB: 85.82
# RandomForestClf: n_est=100: 84.02; n_est=500: 85.04

# Results: accuracy without tf-idf
# LogReg: solver=saga: 88.64; solver=liblinear: 87.82;
# MultinomialNB: 84.43
# RandomForestClf: n_estimators=100: 84.65; n_estimators=500: 86.06
```

```
'Confusion matrix'
```

```
col_0         0     1
sentiment
0          3838   516
1           478  3918
```

```
'Accuracy (custom) is 88.64'
```

```
'Accuracy (sklearn) is 0.8864'
```

```
'FN rate: 10.87'
```

```
'FP rate: 11.85'
```

```
'F1 score is 0.8874292185730465'
```

```
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      4354
           1       0.88      0.89      0.89      4396

    accuracy                           0.89      8750
   macro avg       0.89      0.89      0.89      8750
weighted avg       0.89      0.89      0.89      8750
```

[18]:
```
# Approach 2: apply BOW + tf-idf transformation to the cleaned text
# NOTE: cleaned data == all params True

models = [
    LogisticRegression(solver='saga', max_iter=10000),
#     MultinomialNB(),
#     RandomForestClassifier(n_estimators=100, n_jobs=-1, verbose=2)
]
X_train_full = vectorize_df_col(train_df, 'cleaned_review', perform_tfidf=True,␣
 ↪cv_max_features=8000)
y_train_full = train_df['sentiment']
for model in models:
    y_pred = apply_model(X_train_full, y_train_full, model, display_stats=True)

# Results: accuracy with 0.35 of train_set size for validation set
# LogisticRegression lbfgs, newton-cg, liblinear, sag, saga: 89.01
# MultinomialNB: 86.95
# RandomForestClassifier: 86.77
```

```
'Confusion matrix'
```

```
col_0         0     1
sentiment
0          3798   556
1           443  3953
```

```
'Accuracy (custom) is 88.58'
```

```
'Accuracy (sklearn) is 0.8858285714285714'
```

```
'FN rate: 10.08'
```

```
'FP rate: 12.77'
```

```
'F1 score is 0.8878158338012353'
```

```
              precision    recall  f1-score   support

           0       0.90      0.87      0.88      4354
           1       0.88      0.90      0.89      4396

    accuracy                           0.89      8750
   macro avg       0.89      0.89      0.89      8750
weighted avg       0.89      0.89      0.89      8750
```

[19]:
```
# Approach 3: use length to predict text sentiment
# NOTE: cleaned data == all params True

X_train_full = train_df.loc[:, ['rev_len', 'cln_rev_len', 'cln_rev_tokens_len']]
y_train_full = train_df['sentiment']
model = LogisticRegression(solver='saga')
y_pred = apply_model(X_train_full, y_train_full, model, display_stats=True)

# Results: accuracy with 0.35 of train_set size for validation set
# LogisticRegression: ~[56.30; 56.4]
# MultinomialNB: 56.43
# RandomForestClassifier: 52.18
```

```
'Confusion matrix'
```

```
col_0         0     1
sentiment
0          2451  1903
1          1983  2413
```

```
'Accuracy (custom) is 55.59'
```

```
'Accuracy (sklearn) is 0.5558857142857143'
```

```
'FN rate: 45.11'
```

```
'FP rate: 43.71'
```

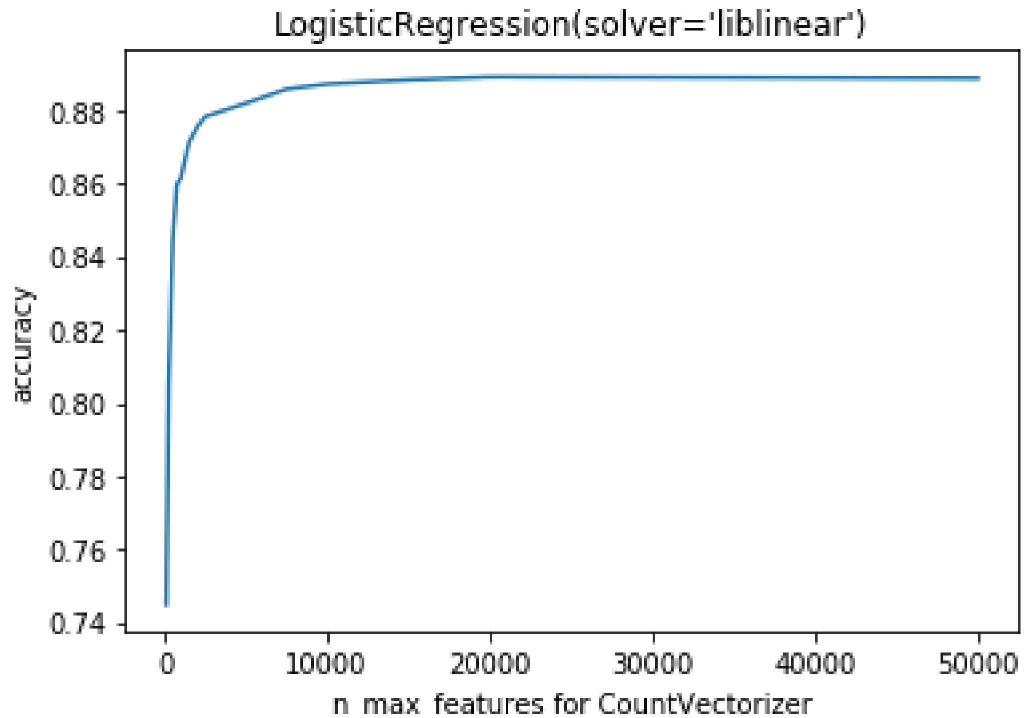
```
'F1 score is 0.5539485766758493'
```

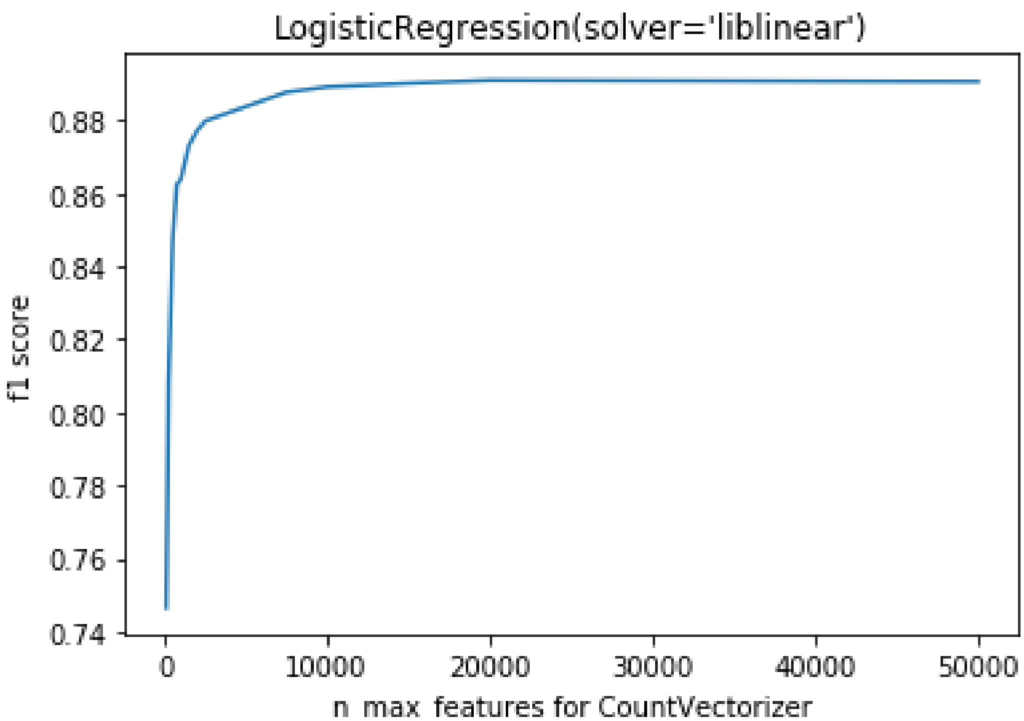
```
              precision    recall  f1-score   support

           0       0.55      0.56      0.56      4354
           1       0.56      0.55      0.55      4396

    accuracy                           0.56      8750
   macro avg       0.56      0.56      0.56      8750
weighted avg       0.56      0.56      0.56      8750
```

[20]:
```
# Approach 4: play with BOW hyperparameters, no TfIdf
# NOTE: cleaned data == all params True
# NOTE: tuning TfidfTransformer didn't have any positive outcome

n_max_features = [100, 250, 500, 750, 1000, 1500, 2000, 2500, 5000, 7500,␣
 ↪10000, 15000, 20000, 50000]
accuracy_values = []
f1_score_values = []

for n_max_features_value in n_max_features:
    X_train_full = vectorize_df_col(train_df, 'cleaned_review',
                                    perform_tfidf=True,
                                    cv_max_features=n_max_features_value)
    y_train_full = train_df['sentiment']
    model = LogisticRegression(solver='liblinear')
    y_true, y_pred = apply_model(X_train_full, y_train_full, model,
                                 display_stats=False, return_validat=True)
    accuracy_values.append(accuracy_score(y_true, y_pred))
```

```python
    f1_score_values.append(f1_score(y_true, y_pred))
    print("dbg: solved for {0} param".format(n_max_features_value))

plt.plot(n_max_features, accuracy_values)
plt.title("LogisticRegression(solver='liblinear')")
plt.xlabel("n_max_features for CountVectorizer")
plt.ylabel("accuracy")
plt.show()

plt.plot(n_max_features, f1_score_values)
plt.title("LogisticRegression(solver='liblinear')")
plt.xlabel("n_max_features for CountVectorizer")
plt.ylabel("f1 score")
plt.show()

display(max(accuracy_values), max(f1_score_values))

# Conclusion: n_max_features=10000 & turned on tf-idf transformation is fine
```

```
dbg: solved for 100 param
dbg: solved for 250 param
dbg: solved for 500 param
dbg: solved for 750 param
dbg: solved for 1000 param
dbg: solved for 1500 param
dbg: solved for 2000 param
dbg: solved for 2500 param
dbg: solved for 5000 param
dbg: solved for 7500 param
dbg: solved for 10000 param
dbg: solved for 15000 param
dbg: solved for 20000 param
dbg: solved for 50000 param
```


LogisticRegression(solver='liblinear')
accuracy
0.74
0.76
0.78
0.80
0.82
0.84
0.86
0.88
0
10000
20000
30000
40000
50000
n_max_features for CountVectorizer


LogisticRegression(solver='liblinear')
f1 score
0.74
0.76
0.78
0.80
0.82
0.84
0.86
0.88
0
10000
20000
30000
40000
50000
n_max_features for CountVectorizer

```
0.8891428571428571
```

```
0.8910112359550562
```

[21]:
```
# Approach 5: use VADER

def get_discrete_sentiment_score_vader(text):
    """Return 0 or 1, depending on compound score.
    Note: positive sentiment: score>=0.05;
          negative sentiment: score<=-0.05;
          use value random from {0, 1} for neutral sentiment: -0.05<=score<=0.05
    """
    compound_score = vader_analyzer.polarity_scores(text).get('compound')
    if compound_score >= 0.05:
        return 1
    elif compound_score <= -0.05:
        return 0
    else:
        return np.random.randint(0, 2)

def try_vader():
    # Use VADER to predict sentiment for data in train set
    vader_analyzer = SentimentIntensityAnalyzer()

    train_df['vader_sentiment_raw'] = train_df['review'].apply(
        lambda x: get_discrete_sentiment_score_vader(x)
    )
    train_df['vader_sentiment_cln'] = train_df['cleaned_review'].apply(
        lambda x: get_discrete_sentiment_score_vader(x)
    )

    # Estimate VADER accuracy

    display_y_pred_stats(train_df['sentiment'],␣
 ↪train_df['vader_sentiment_raw'])  # acc: 69.25

    display_y_pred_stats(train_df['sentiment'],␣
 ↪train_df['vader_sentiment_cln'])  # acc: 67.33

# Conclusion: VADER doesn't perform well for train set - don't use it in final␣
 ↪submission
```

[22]:
```
# Approach 6: use default version of TextBlob

def get_discrete_sentiment_score_textblob(text):
    """Return 0 or 1, depending on sentiment score.
```

```
    Note: positive sentiment: score>=0;
          negative sentiment: score<0;
    """
    sentiment_score = TextBlob(text).sentiment.polarity
    return 1 if sentiment_score >= 0 else 0

def try_textblob():
    # Use TextBlob to predict sentiment for data in train set
    train_df['textblob_sentiment_raw'] = train_df['review'].apply(
        lambda x: get_discrete_sentiment_score_textblob(x)
    )
    train_df['textblob_sentiment_cln'] = train_df['cleaned_review'].apply(
        lambda x: get_discrete_sentiment_score_textblob(x)
    )

    display_y_pred_stats(train_df['sentiment'],␣
 ↪train_df['textblob_sentiment_raw'])  # acc: 68.5

    display_y_pred_stats(train_df['sentiment'],␣
 ↪train_df['textblob_sentiment_cln'])  # acc: 68.59

# Conclusion: default TextBlob doesn't perform well for train set - don't use␣
 ↪it in final submission
```

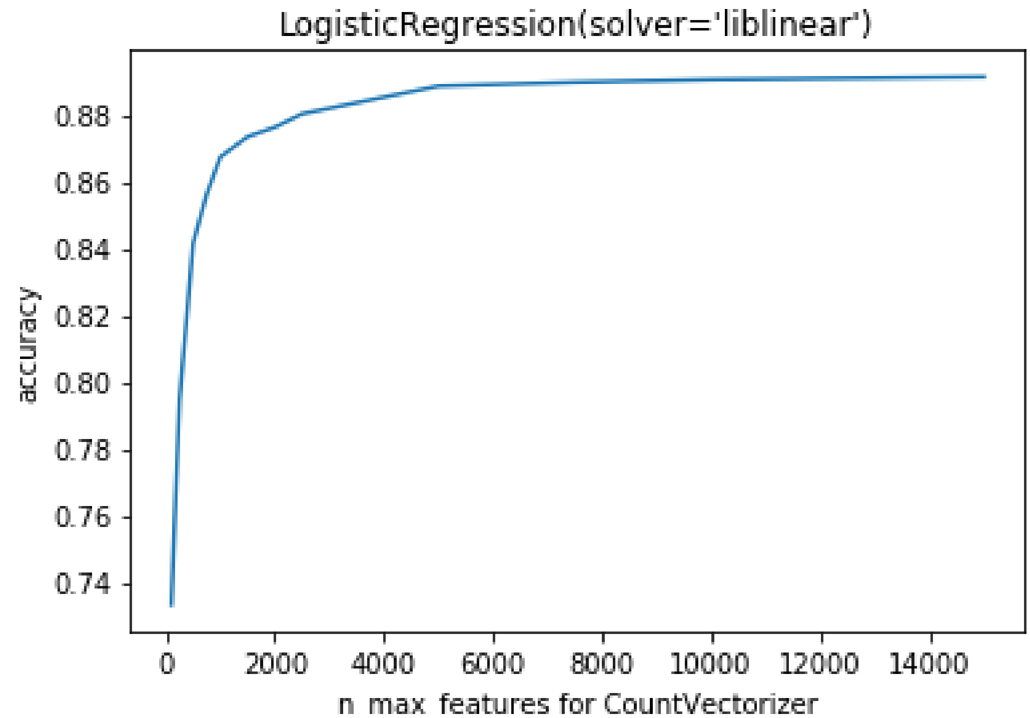
[23]:
```
# Approach 6: use customized TextBlob

def try_customized_textblob():

    train_df['sentiment_posneg'] = train_df['sentiment'].apply(
        lambda x: "pos" if x == 1 else "neg"
    )

    textblob_train_data_rawreview = [
        tuple(row)
        for row in train_df.loc[:, ['review', 'sentiment_posneg']].values
    ]

    textblob_train_data_clnreview = [
        tuple(row)
        for row in train_df.loc[:, ['cleaned_review', 'sentiment_posneg']].
 ↪values
    ]

    # textblob_nb_clf_rawreview =␣
 ↪NaiveBayesClassifier(textblob_train_data_rawreview[:1000])  # 38% MEM
    # del textblob_nb_clf_rawreview
```

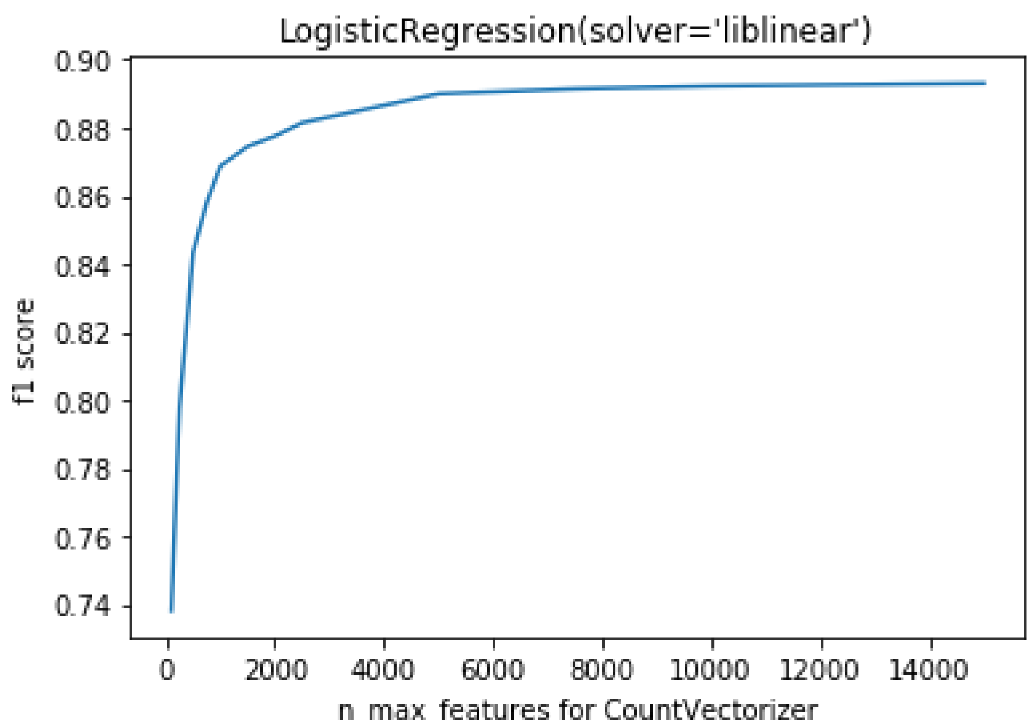
```
    # textblob_nltk_clf_rawreview =␣
↪NLTKClassifier(textblob_train_data_rawreview[:1000])  # 34% MEM
    # textblob_dtree_clf_rawreview =␣
↪DecisionTreeClassifier(textblob_train_data_rawreview[:1000])  # 56% MEM

    # train_df['textblob_nb_raw'] = train_df['review'].apply(
    #     lambda x: 1 if textblob_nb_clf_rawreview.classify(x) == "pos" else 0
    # )

    # train_df['textblob_nb_raw'] = train_df['review'].apply(
    #     lambda x: 1 if textblob_nb_clf_rawreview.classify(x) == "pos" else 0
    # )

    # textblob_nb_clf_clnreview =␣
↪NaiveBayesClassifier(textblob_train_data_clnreview[:1000])  # 71% MEM
    # textblob_nltk_clf_clnreview =␣
↪NLTKClassifier(textblob_train_data_clnreview[:1000])  # 75% MEM
    # textblob_dtree_clf_clnreview =␣
↪DecisionTreeClassifier(textblob_train_data_clnreview[:1000])  # 85% MEM

    # after that: use .prob_classify OR .classify

# Conclusion: because memory usage is too high for only 1000 rows (out of␣
↪25000) - skip this approach
```

[24]:
```
# Approach 7: try to clean data differently: with/without lowering/lemmatizing/
↪stopwords_removal/

n_max_features = [100, 250, 500, 750, 1000, 1500, 2000, 2500, 5000, 7500,␣
↪10000, 15000]
accuracy_values = []
f1_score_values = []

train_df['cleaned_review'] = train_df['review'].apply(
    lambda x: clean_review(
        x,
        to_lower=True, lemmatize=False, remove_numbers=False,␣
↪remove_stopwords=False  # best combination
    )
)

for n_max_features_value in n_max_features:
    X_train_full = vectorize_df_col(train_df, 'cleaned_review',␣
↪perform_tfidf=True, cv_max_features=n_max_features_value)
    y_train_full = train_df['sentiment']
    model = LogisticRegression(solver='liblinear')
```

```
    y_true, y_pred = apply_model(X_train_full, y_train_full, model,␣
↪display_stats=False, return_validat=True)
    accuracy_values.append(accuracy_score(y_true, y_pred))
    f1_score_values.append(f1_score(y_true, y_pred))
    print("dbg: solved for {0} param".format(n_max_features_value))

plt.plot(n_max_features, accuracy_values)
plt.title("LogisticRegression(solver='liblinear')")
plt.xlabel("n_max_features for CountVectorizer")
plt.ylabel("accuracy")
plt.show()

plt.plot(n_max_features, f1_score_values)
plt.title("LogisticRegression(solver='liblinear')")
plt.xlabel("n_max_features for CountVectorizer")
plt.ylabel("f1 score")
plt.show()

display(max(accuracy_values), max(f1_score_values))
```

```
dbg: solved for 100 param
dbg: solved for 250 param
dbg: solved for 500 param
dbg: solved for 750 param
dbg: solved for 1000 param
dbg: solved for 1500 param
dbg: solved for 2000 param
dbg: solved for 2500 param
dbg: solved for 5000 param
dbg: solved for 7500 param
dbg: solved for 10000 param
dbg: solved for 15000 param
```


LogisticRegression(solver='liblinear')
accuracy
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
2000
4000
6000
8000
10000
12000
14000
n_max_features for CountVectorizer


LogisticRegression(solver='liblinear')
f1 score
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
0
2000
4000
6000
8000
10000
12000
14000
n_max_features for CountVectorizer

0.8914285714285715

0.8930661864025213

[25]:
```
# Approach 7: try out tfidf vectorizer

train_df = load_tsv_data("data/labeledTrainData.tsv")

test_df = load_tsv_data("data/testData.tsv")

tfidf_vectorizer = TfidfVectorizer(
    ngram_range=(1, 3),
    use_idf=1,
    smooth_idf=1,
    sublinear_tf=1,
    stop_words='english'
)

train_df['cleaned_review'] = train_df['review'].apply(
    lambda x: clean_review(
        x,
        to_lower=True, lemmatize=False, remove_numbers=True,␣
 ↪remove_stopwords=False
    )
)

test_df['cleaned_review'] = test_df['review'].apply(
    lambda x: clean_review(
        x,
        to_lower=True, lemmatize=False, remove_numbers=True,␣
 ↪remove_stopwords=False
    )
)

train_vectorized_reviews = tfidf_vectorizer.
 ↪fit_transform(train_df['cleaned_review'])
test_vectorized_reviews = tfidf_vectorizer.transform(test_df['cleaned_review'])

clf = MultinomialNB()
clf.fit(train_vectorized_reviews, train_df['sentiment'])
pred = clf.predict(test_vectorized_reviews)

display(pred)

df = pd.DataFrame({"id": test_df['id'],"sentiment": pred})
```

```
df.to_csv('submission.csv', index=False, header=True)
```

```
array([1, 0, 1, ..., 1, 1, 0])
```

[27]:
```
# Apply transformations to test set and create a prediction

# test_df['cleaned_review'] = test_df['review'].apply(
#     lambda x: clean_review(
#         x,
#         to_lower=True, lemmatize=False, remove_numbers=True,␣
 ↪remove_stopwords=False
#     )
# )

# model = LogisticRegression(solver='liblinear')
# model.fit(
#     vectorize_df_col(train_df, 'cleaned_review', perform_tfidf=True,␣
 ↪cv_max_features=10000),
#     train_df['sentiment']
# )

# y_pred = model.predict(
#     vectorize_df_col(test_df, 'review', perform_tfidf=True,␣
 ↪cv_max_features=10000)
# )

# # Submit predictions

# output = pd.DataFrame(
#     {'id': test_df['id'], 'sentiment': y_pred}
# )

# output.to_csv('submission.csv', index=False, quoting=3)
```

[28]:
```
# src: https://www.kaggle.com/varun08/sentiment-analysis-using-word2vec

# NOTE: performance of word2vec is much better when applying to big datasets.
    # In this example, because we are considering only 25,000 training␣
 ↪examples, the
    # performance is similiar to the BOW approach
```

[48]:
```
# Create list of lists for word2vec

tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def split_clean_review(raw_text, tokenizer, to_lower, lemmatize,␣
 ↪remove_numbers, remove_stopwords):
```

```
    raw_sentences = tokenizer.tokenize(raw_text.strip())
    cleaned_sentences = list()
    for raw_sentence in raw_sentences:
        if len(raw_sentence) > 0:
            cleaned_sentences.append(
                clean_review(
                    raw_sentence, return_tokens=True,
                    to_lower=to_lower, lemmatize=lemmatize,␣
 ↪remove_numbers=remove_numbers, remove_stopwords=remove_stopwords
                )
            )
    return cleaned_sentences
```

[49]:
```
# Create list of lists for word2vec: list of sentences

sentences = list()
for review in train_df['review']:
    sentences += split_clean_review(
        review, tokenizer,
        True, False, True, False
    )
```

```
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/bs4/__init__.py:294: UserWarning: "b'.'" looks like a filename, not
markup. You should probably open this file and pass the filehandle into
Beautiful Soup.
  ' Beautiful Soup.' % markup)
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/bs4/__init__.py:357: UserWarning: "http://www.happierabroad.com"" looks
like a URL. Beautiful Soup is not an HTTP client. You should probably use an
HTTP client like requests to get the document behind the URL, and feed that
document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
```

[50]:
```
display(len(sentences[0]))

display(sentences[0])
```

```
36
```

```
['',
 'with',
 'all',
 'this',
 'stuff',
 'going',
```

```
'down',
'at',
'the',
'moment',
'with',
'mj',
'i',
've',
'started',
'listening',
'to',
'his',
'music',
'',
'watching',
'the',
'odd',
'documentary',
'here',
'and',
'there',
'',
'watched',
'the',
'wiz',
'and',
'watched',
'moonwalker',
'again',
'']
```

[51]:

```
# Creating the model and setting values for the various parameters
num_features = 300  # Word vector dimensionality
min_word_count = 40 # Minimum word count
num_workers = 4     # Number of parallel threads
context = 10        # Context window size
downsampling = 1e-3 # (0.001) Downsample setting for frequent words

# Initializing the train model
from gensim.models import word2vec
print("Training model....")
model = word2vec.Word2Vec(sentences,\
                          workers=num_workers,\
                          size=num_features,\
                          min_count=min_word_count,\
                          window=context,
                          sample=downsampling)
```

```python
# To make the model memory efficient
model.init_sims(replace=True)

# Saving the model for later use. Can be loaded using Word2Vec.load()
model_name = "300features_40minwords_10context"
model.save(model_name)
```

```
Training model...
```

[70]:
```python
def featureVecMethod(words, model, num_features):
    """Average all word vectors in a paragraph"""
    featureVec = np.zeros(num_features, dtype="float32")
    nwords = 0
    index2word_set = set(model.wv.index2word)  # set() for speed purposes
    for word in  words:
        if word in index2word_set:
            nwords = nwords + 1
            featureVec = np.add(featureVec,model[word])
    featureVec = np.divide(featureVec, nwords)
    return featureVec

def getAvgFeatureVecs(reviews, model, num_features):
    """Calculating the average feature vector"""
    reviewFeatureVecs = np.zeros((len(reviews),num_features),dtype="float32")
    for idx, review in enumerate(reviews):
        if idx%1000 == 0:
            print(idx)
        reviewFeatureVecs[idx] = featureVecMethod(review, model, num_features)
    return reviewFeatureVecs
```

[71]:
```python
# Get average feature vector for training set

clean_train_reviews = []
for review in train_df['review']:
    clean_train_reviews.append(
        clean_review(review, True, False, True, True)
    )

trainDataVecs = getAvgFeatureVecs(clean_train_reviews, model, num_features)
```

```
0

/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  if __name__ == '__main__':
```

[72]:
```python
# Get average feature vector for test set

clean_test_reviews = []
for review in test_df['review']:
    clean_test_reviews.append(
        clean_review(review, True, False, True, True)
    )

testDataVecs = getAvgFeatureVecs(clean_test_reviews, model, num_features)
```

```
0

/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/ipykernel_launcher.py:9: DeprecationWarning: Call to deprecated
`__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__()
instead).
  if __name__ == '__main__':
```

[73]:
```python
model_rndforest = RandomForestClassifier(n_estimators=100)

model_rndforest = model_rndforest.fit(trainDataVecs, train_df['sentiment'])

y_pred = model_rndforest.predict(testDataVecs)
```

[75]:
```python
# Submit predictions

output = pd.DataFrame(
    {'id': test_df['id'], 'sentiment': y_pred}
)

output.to_csv('submission.csv', index=False)
```

[78]:
```python
display(test_df.shape)

display(y_pred.shape)

display(train_df.head(5))
display(test_df.head(5))
```

```
(25000, 3)


(25000,)


         id  sentiment                                             review  \
0  "5814_8"          1  "With all this stuff going down at the moment ...
1  "2381_9"          1  "\"The Classic War of the Worlds\" by Timothy ...
```

```
2  "7759_3"          0  "The film starts with a manager (Nicholas Bell...
3  "3630_4"          0  "It must be assumed that those who praised thi...
4  "9495_8"          1  "Superbly trashy and wondrously unpretentious ...

                                     cleaned_review
0   with all this stuff going down at the moment ...
1     the classic war of the worlds   by timothy ...
2   the film starts with a manager  nicholas bell...
3   it must be assumed that those who praised thi...
4   superbly trashy and wondrously unpretentious ...


           id                                             review  \
0  "12311_10"  "Naturally in a film who's main themes are of ...
1    "8348_2"  "This movie is a disaster within a disaster fi...
2    "5828_4"  "All in all, this is a movie for kids. We saw ...
3    "7186_2"  "Afraid of the Dark left me with the impressio...
4   "12128_7"  "A very accurate depiction of small time mob l...

                                     cleaned_review
0   naturally in a film who s main themes are of ...
1   this movie is a disaster within a disaster fi...
2   all in all  this is a movie for kids  we saw ...
3   afraid of the dark left me with the impressio...
4   a very accurate depiction of small time mob l...
```

[82]:
```python
# Try out LinearSVC

stop_words = ['in', 'of', 'at', 'a', 'the']

ngram_vectorizer = CountVectorizer(binary=True, ngram_range=(1, 3),␣
 ↪stop_words=stop_words)

ngram_vectorizer.fit(train_df['cleaned_review'])

X = ngram_vectorizer.transform(train_df['cleaned_review'])

X_test = ngram_vectorizer.transform(test_df['cleaned_review'])
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-82-f5e3321a8758> in <module>
```

```
     12
     13 X_train, X_val, y_train, y_val = train_test_split(
---> 14     X, target, train_size = 0.75
     15 )
     16


    NameError: name 'target' is not defined
```

[83]:
```
X_train, X_val, y_train, y_val = train_test_split(
    X, train_df['sentiment'], train_size = 0.75
)

for c in [0.001, 0.005, 0.01, 0.05, 0.1]:
    svm = LinearSVC(C=c)
    svm.fit(X_train, y_train)
    print("Accuracy for C={0}: {1}".format(c, accuracy_score(y_val, svm.
 ↪predict(X_val))))

# Accuracy for C=0.001: 0.88544
# Accuracy for C=0.005: 0.89088
# Accuracy for C=0.01: 0.88992
# Accuracy for C=0.05: 0.8896
# Accuracy for C=0.1: 0.88944
```

```
Accuracy for C=0.001: 0.88544
Accuracy for C=0.005: 0.89088
Accuracy for C=0.01: 0.88992
Accuracy for C=0.05: 0.8896
Accuracy for C=0.1: 0.88944
```

[84]:
```
# src: https://www.kaggle.com/drscarlat/
 ↪imdb-sentiment-analysis-keras-and-tensorflow

# Import keras and tensorflow libraries

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils, to_categorical

from keras.datasets import imdb
```

```
Using TensorFlow backend.
```

[93]:
```
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.
 ↪load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
```

[99]:
```
# Vectorize inputs.
# Encoding the integer sequences into a binary matrix - one hot encoder␣
 ↪basically
# From integers representing words, at various lengths - to a normalized one␣
 ↪hot encoded tensor (matrix) of 10k columns

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

[106]:
```
X_train = vectorize_sequences(train_data)
X_test = vectorize_sequences(test_data)

print("x_train ", X_train.shape, X_train.dtype)
print("x_test ", X_test.shape, X_train.dtype)
```

```
x_train  (25000, 10000) float64
x_test  (25000, 10000) float64
```

[107]:
```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

print("y_train", y_train.shape, y_train.dtype)
print("y_test ", y_test.shape, y_train.dtype)
```

```
y_train (25000,) float32
y_test  (25000,) float32
```

[110]:
```
X_val = X_train[:10000]
partial_X_train = X_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
print("x_val ", X_val.shape)
print("partial_x_train ", partial_X_train.shape)
print("y_val ", y_val.shape)
print("partial_y_train ", partial_y_train.shape)
```

```
x_val  (10000, 10000)
partial_x_train  (15000, 10000)
y_val  (10000,)
partial_y_train  (15000,)
```

[111]:
```
# NN model

model = models.Sequential()
model.add(layers.Dense(
    16, kernel_regularizer=regularizers.l1(0.001), activation='relu',␣
 ↪input_shape=(10000,))
)
model.add(layers.Dropout(0.5))
model.add(layers.Dense(
    16, kernel_regularizer=regularizers.l1(0.001),activation='relu')
)
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0729 16:13:04.391217 140410629236544 deprecation_wrapper.py:119] From
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.

W0729 16:13:04.424128 140410629236544 deprecation_wrapper.py:119] From
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

W0729 16:13:04.427786 140410629236544 deprecation_wrapper.py:119] From
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

W0729 16:13:04.444843 140410629236544 deprecation_wrapper.py:119] From
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:133: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.
```

```
W0729 16:13:04.451516 140410629236544 deprecation.py:506] From
/home/max/.conda/envs/studyingenv/lib/python3.7/site-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

[113]:

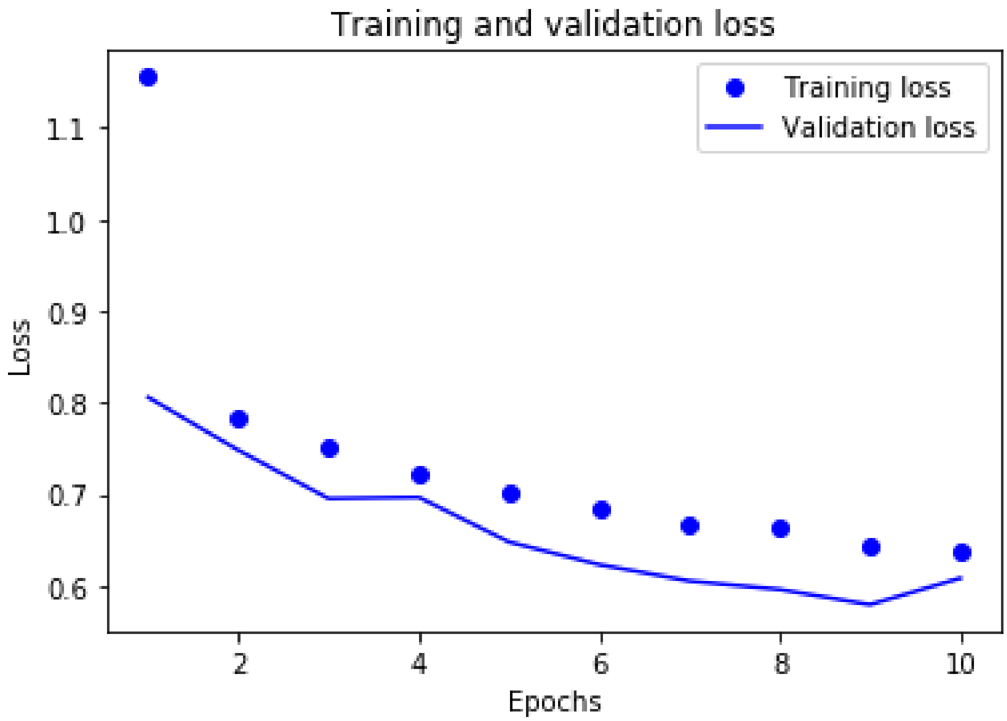
```python
NumEpochs = 10
BatchSize = 512

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

history = model.fit(
    partial_X_train, partial_y_train,
    epochs=NumEpochs, batch_size=BatchSize, validation_data=(X_val, y_val)
)

results = model.evaluate(X_test, y_test)

print("Test Loss and Accuracy")
print("results ", results)

history_dict = history.history
display(history_dict.keys())
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/10
15000/15000 [==============================] - 4s 247us/step - loss: 1.1558 -
acc: 0.6286 - val_loss: 0.8060 - val_acc: 0.8060
Epoch 2/10
15000/15000 [==============================] - 2s 101us/step - loss: 0.7839 -
acc: 0.7042 - val_loss: 0.7480 - val_acc: 0.7492
Epoch 3/10
15000/15000 [==============================] - 2s 105us/step - loss: 0.7509 -
acc: 0.7437 - val_loss: 0.6954 - val_acc: 0.8469
Epoch 4/10
15000/15000 [==============================] - 1s 84us/step - loss: 0.7230 -
acc: 0.7745 - val_loss: 0.6967 - val_acc: 0.7913
Epoch 5/10
15000/15000 [==============================] - 1s 85us/step - loss: 0.7024 -
acc: 0.7918 - val_loss: 0.6479 - val_acc: 0.8456
Epoch 6/10
15000/15000 [==============================] - 1s 88us/step - loss: 0.6854 -
acc: 0.8030 - val_loss: 0.6231 - val_acc: 0.8512
Epoch 7/10
```

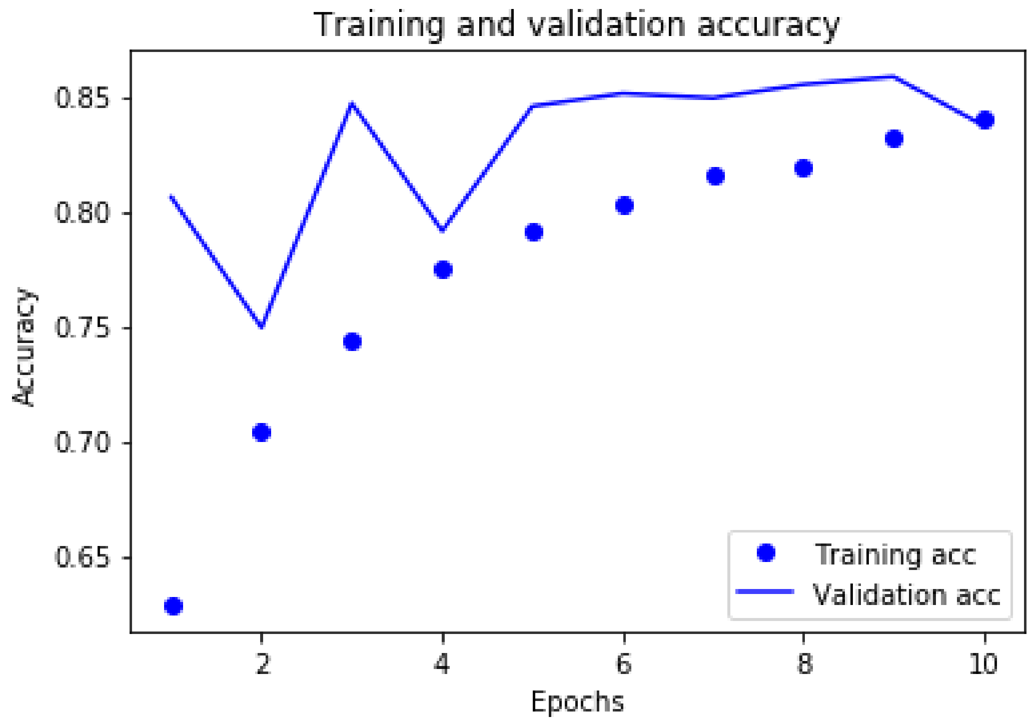
```
15000/15000 [==============================] - 1s 86us/step - loss: 0.6682 -
acc: 0.8155 - val_loss: 0.6054 - val_acc: 0.8492
Epoch 8/10
15000/15000 [==============================] - 1s 85us/step - loss: 0.6632 -
acc: 0.8195 - val_loss: 0.5961 - val_acc: 0.8551
Epoch 9/10
15000/15000 [==============================] - 1s 86us/step - loss: 0.6435 -
acc: 0.8323 - val_loss: 0.5796 - val_acc: 0.8585
Epoch 10/10
15000/15000 [==============================] - 1s 85us/step - loss: 0.6380 -
acc: 0.8396 - val_loss: 0.6085 - val_acc: 0.8371
25000/25000 [==============================] - 2s 76us/step
Test Loss and Accuracy
results  [0.6119729307556152, 0.83376]

dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

[114]:

```
# Loss curve

plt.clf()
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, (len(history_dict['loss']) + 1))
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[115]:
```
# Validation accuracy curve

plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
epochs = range(1, (len(history_dict['acc']) + 1))
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[117]:
```
model.predict(X_test)
```

[117]:
```
array([[0.26617092],
       [0.95781994],
       [0.5343111 ],
       ...,
       [0.19702148],
       [0.18490258],
       [0.26771948]], dtype=float32)
```