Lab3 - Assignment Sentiment

This notebook describes the LAB-2 assignment of the Text Mining course. It is about sentiment analysis.

The aims of the assignment are:

- Learn how to run a rule-based sentiment analysis module (VADER)
- Learn how to run a machine learning sentiment analysis module (Scikit-Learn/ Naive Bayes)
- Learn how to run scikit-learn metrics for the quantitative evaluation
- Learn how to perform and interpret a quantitative evaluation of the outcomes of the tools (in terms of Precision, Recall, and F_1)
- Learn how to evaluate the results qualitatively (by examining the data)
- Get insight into differences between the two applied methods
- Get insight into the effects of using linguistic preprocessing
- Be able to describe differences between the two methods in terms of their results
- Get insight into issues when applying these methods across different domains

In this assignment, you are going to create your own gold standard set from 50 tweets. You will the VADER and scikit-learn classifiers to these tweets and evaluate the results by using evaluation metrics and inspecting the data.

We recommend you go through the notebooks in the following order:

- **Read the assignment (see below)**
- **Lab3.2-Sentiment-analysis-with-VADER.ipynb**
- **Lab3.3-Sentiment-analysis.with-scikit-learn.ipynb**
- **Answer the questions of the assignment (see below) using the provided notebooks and submit**

In this assignment you are asked to perform both quantitative evaluations and error analyses:

- a quantitative evaluation concerns the scores (Precision, Recall, and F_1) provided by scikit's classification_report. It includes the scores per category, as well as micro and macro averages. Discuss whether the scores are balanced or not between the different categories (positive, negative, neutral) and between precision and recall. Discuss the shortcomings (if any) of the classifier based on these scores
- an error analysis regarding the misclassifications of the classifier. It involves going through the texts and trying to understand what has gone wrong. It servers to get insight in what could be done to improve the performance of the classifier. Do you observe patterns in misclassifications? Discuss why these errors are made and propose ways to solve them.

Credits

The notebooks in this block have been originally created by Marten Postma and Isa Maks. Adaptations were made by Filip Ilievski.

Part I: VADER assignments

Preparation (nothing to submit):

To be able to answer the VADER questions you need to know how the tool works.

- Read more about the VADER tool in this blog.
- VADER provides 4 scores (positive, negative, neutral, compound). Be sure to understand what they mean and how they are calculated.
- VADER uses rules to handle linguistic phenomena such as negation and intensification. Be sure to understand which rules are used, how they work, and why they are important.
- VADER makes use of a sentiment lexicon. Have a look at the lexicon. Be sure to understand which information can be found there (lemma?, wordform?, part-of-speech?, polarity value?, word meaning?) What do all scores mean? https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vader_lexicon.txt)

[3.5 points] Question1:

Regard the following sentences and their output as given by VADER. Regard sentences 1 to 7, and explain the outcome **for each sentence**. Take into account both the rules applied by VADER and the lexicon that is used. You will find that some of the results are reasonable, but others are not. Explain what is going wrong or not when correct and incorrect results are produced.

```
INPUT SENTENCE 1 I love apples
VADER OUTPUT {'neg': 0.0, 'neu': 0.192, 'pos': 0.808,
'compound': 0.6369}

INPUT SENTENCE 2 I don't love apples
VADER OUTPUT {'neg': 0.627, 'neu': 0.373, 'pos': 0.0,
'compound': -0.5216}

INPUT SENTENCE 3 I love apples :-)
VADER OUTPUT {'neg': 0.0, 'neu': 0.133, 'pos': 0.867,
'compound': 0.7579}

INPUT SENTENCE 4 These houses are ruins
VADER OUTPUT {'neg': 0.492, 'neu': 0.508, 'pos': 0.0,
'compound': -0.4404}

INPUT SENTENCE 5 These houses are certainly not considered ruins
VADER OUTPUT {'neg': 0.0, 'neu': 0.51, 'pos': 0.49, 'compound':
0.5867}

INPUT SENTENCE 6 He lies in the chair in the garden
VADER OUTPUT {'neg': 0.286, 'neu': 0.714, 'pos': 0.0,
'compound': -0.4215}

INPUT SENTENCE 7 This house is like any house
```

```
VADER OUTPUT {'neg': 0.0, 'neu': 0.667, 'pos': 0.333,
'compound': 0.3612}
```

1) Love is a positive word according to the lexicon 2) Negation followed by positive word -> Therefore negative. 3) Smileys like ":-)" are also in the lexicon as positive words, so 3) is more positive than 1) 4) Ruin is a negative verb but as a noun it's neutral. So this sentence is incorrectly scored. 5) Negation followed by "ruins" -> Positive. But this sentence could be considered negative (e.g. if it's a review of a landmark that's supposedly old ruins) 6) Lie has multiple meanings. Lying as in deceiving is negative, lying on a chair is neutral. This sentence is incorrectly scored. 7) The verb "like" is positive, but this sentence doesn't use "like" as a verb

[Points: 2.5] Exercise 2: Collecting 50 tweets for evaluation

Collect 50 tweets. Try to find tweets that are interesting for sentiment analysis, e.g., very positive, neutral, and negative tweets. These could be your own tweets (typed in) or collected from the Twitter stream.

We will store the tweets in the file **my_tweets.json** (use a text editor to edit). For each tweet, you should insert:

- sentiment analysis label: negative | neutral | positive (this you determine yourself, this is not done by a computer)
- the text of the tweet
- the Tweet-URL

from:

```
    "1": {
        "sentiment_label": "",
        "text_of_tweet": "",
        "tweet_url": "",
```

to:

```
"1": {
        "sentiment_label": "positive",
        "text_of_tweet": "All across America people chose to get
involved, get engaged and stand up. Each of us can make a
difference, and all of us ought to try. So go keep changing the
world in 2018.",
        "tweet_url" :
"https://twitter.com/BarackObama/status/946775615893655552",
    },
```

You can load your tweets with human annotation in the following way.

In [6]:
```
import json
```

In [7]:
```
my_tweets = json.load(open('my_tweets.json'))
```

In [8]:
```
for id_, tweet_info in my_tweets.items():
```

```
    print(id_, tweet_info)
    break
```

```
0 {'sentiment_label': -1, 'text_of_tweet': 'A few months ago #bitcoin #btc $bt
c #Rekt \n\n385 THOUSAND retail accounts\n\nWhat person in their right mind wo
uld think #btc is a good investment now?\n\nA few #Whales control #btcusd they
will try to suck you in then phukk you again #FoolMeOnce etc', 'tweet_url':
''}
```

[5 points] Question 3:

Run VADER on your own tweets (see function **run_vader** from notebook **Lab2-Sentiment-analysis-using-VADER.ipynb**). You can use the code snippet below this explanation as a starting point.

- [2.5 points] a. Perform a quantitative evaluation. Explain the different scores, and explain which scores are most relevant and why.
- [2.5 points] b. Perform an error analysis: select 10 positive, 10 negative and 10 neutral tweets that are not correctly classified and try to understand why. Refer to the VADER-rules and the VADER-lexicon. Of course, if there are less than 10 errors for a category, you only have to check those. For example, if there are only 5 errors for positive tweets, you just describe those.

In [9]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.metrics import classification_report
```

In [10]:
```
vader_model = SentimentIntensityAnalyzer()
```

In [11]:
```
def vader_output_to_label(vader_output):
    """
    map vader output e.g.,
    {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.4215}
    to one of the following values:
    a) positive float -> 'positive'
    b) 0.0 -> 'neutral'
    c) negative float -> 'negative'

    :param dict vader_output: output dict from vader

    :rtype: str
    :return: 'negative' | 'neutral' | 'positive'
    """
    compound = vader_output['compound']

    if compound < 0:
        return 'negative'
    elif compound == 0.0:
        return 'neutral'
    elif compound > 0.0:
        return 'positive'

assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound'
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound'
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound'
```

In [12]:
```
tweets = []
all_vader_output = []
gold = []

# settings (to change for different experiments)
to_lemmatize = True
pos = set()

for id_, tweet_info in my_tweets.items():
    the_tweet = tweet_info['text_of_tweet']
    vader_output = vader_model.polarity_scores(the_tweet)
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)

    sentiment_label_int = tweet_info['sentiment_label']
    if sentiment_label_int == 1:
        sentiment_label_str = 'positive'
    elif sentiment_label_int == 0:
        sentiment_label_str = 'neutral'
    else:
        sentiment_label_str = 'negative'

    gold.append(sentiment_label_str)

# use scikit-learn's classification report
print(classification_report(gold, all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.67      0.62      0.65        16
     neutral       0.10      0.07      0.08        15
    positive       0.52      0.67      0.58        21

    accuracy                           0.48        52
   macro avg       0.43      0.45      0.44        52
weighted avg       0.44      0.48      0.46        52
```

[4 points] Question 4:

Run VADER on the set of airline tweets with the following settings:

- Run VADER (as it is) on the set of airline tweets
- Run VADER on the set of airline tweets after having lemmatized the text
- Run VADER on the set of airline tweets with only adjectives
- Run VADER on the set of airline tweets with only adjectives and after having lemmatized the text
- Run VADER on the set of airline tweets with only nouns
- Run VADER on the set of airline tweets with only nouns and after having lemmatized the text
- Run VADER on the set of airline tweets with only verbs
- Run VADER on the set of airline tweets with only verbs and after having lemmatized the text

- [1 point] a. Generate for all separate experiments the classification report, i.e., Precision, Recall, and F_1 scores per category as well as micro and macro averages. **Use a different code cell (or multiple code cells) for each experiment.**

- [3 points] b. Compare the scores and explain what they tell you.
- ▪ Does lemmatisation help? Explain why or why not.
- ▪ Are all parts of speech equally important for sentiment analysis? Explain why or why not.

In [13]:
```
import spacy
import pathlib
from sklearn.datasets import load_files
```

In [15]:
```
nlp = spacy.load('en_core_web_sm') # 'en_core_web_sm'
```

In [16]:
```
cwd = pathlib.Path.cwd()

airline_tweets = load_files(str(cwd.joinpath('airlinetweets')))
```

In [17]:
```
target_names = airline_tweets.target_names
airline_tweets_target_as_labels = list(map(lambda x: target_names[x], airline_

print(target_names)
print(airline_tweets.target[:10])
print(airline_tweets_target_as_labels[:10])
```

```
['negative', 'neutral', 'positive']
[1 1 2 1 0 0 1 1 0 1]
['neutral', 'neutral', 'positive', 'neutral', 'negative', 'negative', 'neutra
l', 'neutral', 'negative', 'neutral']
```

In [18]:
```
# Your code here
def run_vader(textual_unit,
              lemmatize=False,
              parts_of_speech_to_consider=None,
              verbose=0):
    """
    Run VADER on a sentence from spacy

    :param str textual unit: a textual unit, e.g., sentence, sentences (one s
    (by looping over doc.sents)
    :param bool lemmatize: If True, provide lemmas to VADER instead of words
    :param set parts_of_speech_to_consider:
    -None or empty set: all parts of speech are provided
    -non-empty set: only these parts of speech are considered.
    :param int verbose: if set to 1, information is printed
    about input and output

    :rtype: dict
    :return: vader output dict
    """
    doc = nlp(textual_unit)

    input_to_vader = []

    for sent in doc.sents:
        for token in sent:

            to_add = token.text

            if lemmatize:
                to_add = token.lemma_
```

```
                if to_add == '-PRON-':
                    to_add = token.text

            if parts_of_speech_to_consider:
                if token.pos_ in parts_of_speech_to_consider:
                    input_to_vader.append(to_add)
            else:
                input_to_vader.append(to_add)

    scores = vader_model.polarity_scores(' '.join(input_to_vader))

    if verbose >= 1:
        print()
        print('INPUT SENTENCE', sent)
        print('INPUT TO VADER', input_to_vader)
        print('VADER OUTPUT', scores)

    return scores
```

In [ ]:
```
# * Run VADER (as it is) on the set of airline tweets

outputs_1 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=False)
    outputs_1.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_1))
```

```
              precision    recall  f1-score   support

    negative       0.80      0.51      0.63      1750
     neutral       0.60      0.51      0.55      1515
    positive       0.56      0.88      0.68      1490

    accuracy                           0.63      4755
   macro avg       0.65      0.63      0.62      4755
weighted avg       0.66      0.63      0.62      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets after having lemmatized the text

outputs_2 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=True)
    outputs_2.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_2))
```

```
              precision    recall  f1-score   support

    negative       0.79      0.52      0.63      1750
     neutral       0.60      0.49      0.54      1515
    positive       0.55      0.87      0.68      1490

    accuracy                           0.62      4755
   macro avg       0.65      0.63      0.61      4755
weighted avg       0.65      0.62      0.62      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets with only adjectives

outputs_3 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=False, parts_of_speech_to_consider=
    outputs_3.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_3))
```

```
              precision    recall  f1-score   support

    negative       0.87      0.22      0.35      1750
     neutral       0.41      0.89      0.56      1515
    positive       0.67      0.44      0.53      1490

    accuracy                           0.50      4755
   macro avg       0.65      0.52      0.48      4755
weighted avg       0.66      0.50      0.47      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets with only adjectives and after havin

outputs_4 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=True, parts_of_speech_to_consider=
    outputs_4.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_4))
```

```
              precision    recall  f1-score   support

    negative       0.87      0.22      0.35      1750
     neutral       0.41      0.89      0.56      1515
    positive       0.67      0.44      0.53      1490

    accuracy                           0.50      4755
   macro avg       0.65      0.52      0.48      4755
weighted avg       0.66      0.50      0.47      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets with only nouns

outputs_5 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=False, parts_of_speech_to_consider=
    outputs_5.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_5))
```

```
              precision    recall  f1-score   support

    negative       0.71      0.14      0.23      1750
     neutral       0.36      0.82      0.50      1515
    positive       0.54      0.35      0.43      1490

    accuracy                           0.42      4755
```

```
   macro avg       0.54      0.44      0.39      4755
weighted avg       0.55      0.42      0.38      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets with only nouns and after having lem

outputs_6 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=True, parts_of_speech_to_consider=
    outputs_6.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_6))
```

```
              precision    recall  f1-score   support

    negative       0.70      0.15      0.25      1750
     neutral       0.36      0.81      0.50      1515
    positive       0.53      0.34      0.42      1490

    accuracy                           0.42      4755
   macro avg       0.53      0.43      0.39      4755
weighted avg       0.54      0.42      0.38      4755
```

In [ ]:
```
# Run VADER on the set of airline tweets with only verbs

outputs_7 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=False, parts_of_speech_to_consider
    outputs_7.append(vader_output_to_label(output))
```

In [ ]:
```
print(classification_report(airline_tweets_target_as_labels, outputs_7))
```

```
              precision    recall  f1-score   support

    negative       0.78      0.28      0.41      1750
     neutral       0.38      0.81      0.52      1515
    positive       0.57      0.34      0.43      1490

    accuracy                           0.47      4755
   macro avg       0.58      0.48      0.45      4755
weighted avg       0.59      0.47      0.45      4755
```

In [34]:
```
# Run VADER on the set of airline tweets with only verbs and after having lem

outputs_8 = []

for doc in airline_tweets.data:
    output = run_vader(str(doc), lemmatize=True, parts_of_speech_to_consider=
    outputs_8.append(vader_output_to_label(output))
```

Question Answers

- Lemmatization has almost no effect on any of the metrics. Every metric is within 0.03 of each other when comparing Lemmatized vs lon-lemmatized

- No, different parts of speech have different degrees of importance for sentiment analysis it seems like. Adjective only: about 50% accuracy, verb only: about 47% accuracy, and noun only: about 42% accuracy. This makes sense because adjectives describe things so it's natural that they're the most important. Verbs also carry a lot of semantic meaning e.g. (hate, love, insults, disappoint, etc.) Nouns are more neutral.

Part II: scikit-learn assignments

[4 points] Question 5

Train the scikit-learn classifier (Naive Bayes) using the airline tweets.

- Train the model on the airline tweets with 80% training and 20% test set and default settings (TF-IDF representation, min_df=2)
- Train with different settings:
  - with respect to vectorizing: TF-IDF ('airline_tfidf') vs. Bag of words representation ('airline_count')
  - with respect to the frequency threshold (min_df). Carry out experiments with increasing values for document frequency (min_df = 2; min_df = 5; min_df =10)
- [1 point] a. Generate a classification_report for all experiments
- [3 points] b. Look at the results of the experiments with the different settings and try to explain why they differ:
  - which category performs best, is this the case for any setting?
  - does the frequency threshold affect the scores? Why or why not according to you?

In [19]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
```

In [20]:
```
def train_and_test(data):
    X_train, X_test, y_train, y_test = train_test_split(
        data,
        airline_tweets_target_as_labels,
        test_size=0.2,
        random_state=7
    )

    clf = MultinomialNB().fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print(classification_report(y_test, y_pred))
```

In [22]:
```
# Vectorize Data
airline_vec_2 = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize, stop_

# Bag of words (2)
airline_counts_2 = airline_vec_2.fit_transform(airline_tweets.data)

# TD-IDF (2)
tfidf_transformer = TfidfTransformer()
airline_tdidf_2 = tfidf_transformer.fit_transform(airline_counts_2)

# df = 5
```

```
airline_vec_5 = CountVectorizer(min_df=5, tokenizer=nltk.word_tokenize, stop_v

# Bag of words (5)
airline_counts_5 = airline_vec_5.fit_transform(airline_tweets.data)

# TD-IDF (5)
tfidf_transformer = TfidfTransformer()
airline_tdidf_5 = tfidf_transformer.fit_transform(airline_counts_5)

# df = 10
airline_vec_10 = CountVectorizer(min_df=10, tokenizer=nltk.word_tokenize, sto

# Bag of words (10)
airline_counts_10 = airline_vec_10.fit_transform(airline_tweets.data)

# TD-IDF (10)
tfidf_transformer = TfidfTransformer()
airline_tdidf_10 = tfidf_transformer.fit_transform(airline_counts_10)
```

In [27]:

```
train_and_test(airline_counts_2)
train_and_test(airline_counts_5)
train_and_test(airline_counts_10)
```

```
              precision    recall  f1-score   support

    negative       0.83      0.91      0.87       348
     neutral       0.83      0.73      0.78       319
    positive       0.83      0.85      0.84       284

    accuracy                           0.83       951
   macro avg       0.83      0.83      0.83       951
weighted avg       0.83      0.83      0.83       951

              precision    recall  f1-score   support

    negative       0.83      0.92      0.88       348
     neutral       0.82      0.75      0.78       319
    positive       0.85      0.82      0.83       284

    accuracy                           0.83       951
   macro avg       0.83      0.83      0.83       951
weighted avg       0.83      0.83      0.83       951

              precision    recall  f1-score   support

    negative       0.82      0.91      0.87       348
     neutral       0.82      0.76      0.79       319
    positive       0.85      0.81      0.83       284

    accuracy                           0.83       951
   macro avg       0.83      0.83      0.83       951
weighted avg       0.83      0.83      0.83       951
```

In [28]:

```
train_and_test(airline_tdidf_2)
train_and_test(airline_tdidf_5)
train_and_test(airline_tdidf_10)
```

```
              precision    recall  f1-score   support

    negative       0.81      0.91      0.85       348
     neutral       0.84      0.69      0.76       319
    positive       0.81      0.85      0.83       284

    accuracy                           0.82       951
```

```
   macro avg       0.82      0.81      0.81       951
weighted avg       0.82      0.82      0.81       951

              precision    recall  f1-score   support

    negative       0.82      0.90      0.86       348
     neutral       0.81      0.73      0.77       319
    positive       0.84      0.84      0.84       284

    accuracy                           0.82       951
   macro avg       0.82      0.82      0.82       951
weighted avg       0.82      0.82      0.82       951

              precision    recall  f1-score   support

    negative       0.83      0.89      0.86       348
     neutral       0.80      0.75      0.77       319
    positive       0.83      0.81      0.82       284

    accuracy                           0.82       951
   macro avg       0.82      0.82      0.82       951
weighted avg       0.82      0.82      0.82       951
```

Answer

- There is hardly any difference, but a small pattern is: From 2 -> 5 a few metrics increase by 1-2% and from 5->10 a few metrics decrease from 1-2% (more increase from 2->5 than decrease, and more decrease from 5->10 than increase). So it seems like df=5 is slightly optimal.

[4 points] Question 6: Inspecting the best scoring features

- Train the scikit-learn classifier (Naive Bayes) model with the following settings (airline tweets 80% training and 20% test; Bag of words representation ('airline_count'), min_df=2)
- [1 point] a. Generate the list of best scoring features per class (see function **important_features_per_class** below) [1 point]
- [3 points] b. Look at the lists and consider the following issues:
  - [1 point] Which features did you expect for each separate class and why?
  - [1 point] Which features did you not expect and why ?
  - [1 point] The list contains all kinds of words such as names of airlines, punctuation, numbers and content words (e.g., 'delay' and 'bad'). Which words would you remove or keep when trying to improve the model and why?

In [30]:
```
clf = MultinomialNB().fit(airline_tdidf_5, airline_tweets_target_as_labels)
```

In [33]:
```
def important_features_per_class(vectorizer,classifier,n=80):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names()
    topn_class1 = sorted(zip(classifier.feature_count_[0], feature_names),rev
    topn_class2 = sorted(zip(classifier.feature_count_[1], feature_names),rev
    topn_class3 = sorted(zip(classifier.feature_count_[2], feature_names),rev
    print("Important words in negative documents")
    for coef, feat in topn_class1:
        print(class_labels[0], coef, feat)
    print("-----------------------------------------")
    print("Important words in neutral documents")
```

```
    for coef, feat in topn_class2:
        print(class_labels[1], coef, feat)
    print("----------------------------------------")
    print("Important words in positive documents")
    for coef, feat in topn_class3:
        print(class_labels[2], coef, feat)

# example of how to call from notebook:
important_features_per_class(airline_vec_2, clf, n=10)
```

```
Important words in negative documents
negative 239.56755603845951 fees
negative 170.7799542209877 -
negative 147.12414701657335 180
negative 140.99046790427093 19
negative 81.73940260550893 beautifully
negative 73.80977474891945 18
negative 66.69298751227853 #
negative 59.75193782649458 !
negative 57.36639480248299 connected
negative 37.9776066452263 ''
----------------------------------------
Important words in neutral documents
neutral 161.4561155359084 180
neutral 126.53269806022911 18
neutral 95.34723671110271 center
neutral 87.96658020564564 emailing
neutral 85.60196178795331 -
neutral 84.27091155437739 19
neutral 76.54808981458166 1700
neutral 75.94201131364919 2day
neutral 62.49380196746564 finding
neutral 58.730519150370334 fees
----------------------------------------
Important words in positive documents
positive 237.45170641266313 !
positive 149.7318967231002 180
positive 124.26268687742932 -
positive 112.30393690022524 expensive
positive 108.8663677564461 expecting
positive 95.62077994407386 center
positive 94.18069464318873 emailing
positive 89.07561531794971 19
positive 73.34089142895422 #
positive 70.60185307632261 2day
```

[Optional! (will not be graded)] Question 7

Train the model on airline tweets and test it on your own set of tweets

- Train the model with the following settings (airline tweets 80% training and 20% test; Bag of words representation ('airline_count'), min_df=2)
- Apply the model on your own set of tweets and generate the classification report
- [1 point] a. Carry out a quantitative analysis.
- [1 point] b. Carry out an error analysis on 10 correctly and 10 incorrectly classified tweets and discuss them
- [2 points] c. Compare the results (cf. classification report) with the results obtained by VADER on the same tweets and discuss the differences.

[Optional! (will not be graded)] Question 8: trying to improve the model

- [2 points] a. Think of some ways to improve the scikit-learn Naive Bayes model by playing with the settings or applying linguistic preprocessing (e.g., by filtering on part-of-speech, or removing punctuation). Do not change the classifier but continue using the Naive Bayes classifier. Explain what the effects might be of these other settings
- [1 point] b. Apply the model with at least one new setting (train on the airline tweets using 80% training, 20% test) and generate the scores
- [1 point] c. Discuss whether the model achieved what you expected.

End of this notebook

In [ ]: