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 (https://martenpostma.github.io) and Isa Maks (https://research.vu.nl/en/persons/e-maks). Adaptations were made by Filip Ilievski (http://ilievski.nl).

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 (http://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html)](http://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html).
- 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 (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}
```

Answers:

Sentence 1: The world love is a very positive word while apples is neutral so this makes the sentence really positive.

Sentence 2: Love is a strong positive word but it is negated by the "don't" which makes this count as highly negative. This combined with the neurtrality of apples, the sentence becomes negative.

Sentence 3: This is similar to the first sentence, but the smiley emoticon is positive which makes the compound more positive than the first sentence.

Sentence 4: This sentence is generally a statement which is neutral but the "ruins" is negative which makes the compound, and hence sentence, negative.

Sentence 5: Similar to sentence 4, the sentence is more of a statement but sinc the "not" negates the negativity of "ruins" making the description more positive. However, the "considered" is more factual, inclreasing the neutrality compared to sentence 4, which "certainly" is an added positive, which is why the compound is more than the compound of sentence 4.

Sentence 6: The full statement is an observation which makes it neutral, however, the word "lies" is considered as he told a lie which is negative. Meaning the compound of this sentence is incorrect as it should only be neutral as there's nothing negative in the sentence.

Sentence 7: Overall the sentence is a statement which makes it mostly neutral, however the word "like" is considered as a positive word, even though the sentence would probably be considered as a negative by a human as it means the house is not special.

[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 eng
    aged and stand up. Each of us can make a difference, and all of us ought to try. S
    o 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 [50]:

```
import json
```

In [51]:

```
my_tweets = json.load(open('my_tweets.json'))
```

In [52]:

```
for id_, tweet_info in my_tweets.items():
    print(id_, tweet_info)
    break
```

```
1 {'sentiment_label': 'positive', 'text_of_tweet': 'Very happy to score my f
irst goal with the blues. Come on!! Next round @ChelseaFC @EmiratesFACup',
'tweet_url': 'https://twitter.com/saulniguez/status/1499135951003697156'}
```

[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 [53]:

```
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': 0.0}) == 'ne
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.01}) == 'p
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': -0.01}) == '
```

In [59]:

```
import nltk
from nltk.sentiment import vader
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import spacy
import pathlib
import sklearn
import numpy
import nltk
from nltk.corpus import stopwords
from collections import Counter
from sklearn.metrics import classification_report
vader_model = SentimentIntensityAnalyzer()

nlp = spacy.load('en_core_web_sm') # 'en_core_web_sm'
cwd = pathlib.Path.cwd()
tweets = cwd.joinpath('my_tweets.json')
#print('path:', tweets)
#print('this will print True if the folder exists:', tweets.exists())

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 = run_vader(the_tweet, lemmatize=True) # run vader
    vader_label = vader_output_to_label(vader_output)# convert vader output to category

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(tweet_info['sentiment_label'])

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

```
                                                  precision    recall  f1-s
core   support

                                                       0.00      0.00
0.00         1
                                          negative       0.77      0.50
0.61        20
                                           neutral       0.40      0.40
0.40        10
                                          positive       0.48      0.72
0.58        18
she’s just a little guy who’s just here to vibe       0.00      0.00
0.00         1

                                          accuracy
0.54        50
                                         macro avg       0.33      0.32
```

```
0.32        50
                                   weighted avg       0.56       0.54
0.53        50
```

```
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\metrics\_classifi
cation.py:1245: UndefinedMetricWarning: Precision and F-score are ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\metrics\_classifi
cation.py:1245: UndefinedMetricWarning: Precision and F-score are ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\metrics\_classifi
cation.py:1245: UndefinedMetricWarning: Precision and F-score are ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divisi
on` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Answers

a. The precision for the tweets analyzed using Vader seems to be overall not high. Indeed, the weighted average barely goes beyond 50% settling at 56%. Out of all the categories, negatives seem to perform better than neutral and positive. Whereas negative see a precision of 77%, neutral and positive do not reach the 50%, being 40% and 48% respectively. For what concerns recall, instead, the general weighted recall is also not high, reaching 54%. In the lead of highest recall is positive with 72%, followed by negative and neutral, 50% and 40% respectively. To sum up the two, we analyze the f1 score, which we can therefore consider as the most describing metric. Overall, the positive category seem to have a higher f1 score, namely 58%, neutral have 40% and negative 61%. The weighted average in this case is 53%. The results have a low f1, which is confirmed by the low accuracy of 54%. Considering the imbalance between the scores of the same categories across precision and recall, the most relevant metric to consider is the f1, which suggest that the category positives is the most relevant since it is the one with the highest score. Its high recall must mean that there are probably more positively categorized instances than actual positive instances. It might be due to the tweets set chosen.

In [60]:

```
errors = []
scores = vader_model.polarity_scores(the_tweet)
if tweet_info['sentiment_label'] != vader_output_to_label(scores):
    errors.append((id_, vader_output_to_label(scores)))

all_list = []
index = 1
for e in errors:
    id_, sentiment_labeling = e
    print("misclassified", index, my_tweets[id_]['text_of_tweet'])
    print("vader: ", sentiment_labeling)
    print("real class: ", my_tweets[id_]['sentiment_label'])
    index += 1
```

```
misclassified 1 NAH BRUH THIS ONE OF THE CRAZIEST THINGS IGE EVER SEEN BRON
IS THE GOAT
vader:  negative
real class:  positive
```

Answers b. The only instance we got incorrectly classified was the following:

misclassified 1 NAH BRUH THIS ONE OF THE CRAZIEST THINGS IGE EVER SEEN BRON IS THE GOAT vader: negative real class: positive

The reason why it is misclassified is that the statement is sarcastic, therefore using words in a non-strict semantic manner. Indeed, literally "nah" is considered as negative in the lexicon, with a score of -0.4 and "craziest" is also considered as negative, scoring -0.2. Therefore, the statement is interpreted as overall negative.

[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 [26]:

```
#imports
import nltk
#nltk.download('vader_lexicon', quiet=False) #only need to run it once
from nltk.sentiment import vader
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import spacy
import pathlib
import sklearn
import numpy
import nltk
from nltk.corpus import stopwords
from collections import Counter
from sklearn.metrics import classification_report
```

In [32]:

```
#Get the tweets
cwd = pathlib.Path.cwd()
airline_tweets_folder = cwd.joinpath('airlinetweets')
print('path:', airline_tweets_folder)
print('this will print True if the folder exists:',
      airline_tweets_folder.exists())

airline_tweets_train = load_files("C:/Users/zahra_6hcxkfv/ba-text-mining/lab_sessions/lab3/
```

```
path: c:\Users\zahra_6hcxkfv\ba-text-mining\lab_sessions\lab3\airlinetweets
this will print True if the folder exists: True
```

In [33]:

```
#Vader
nlp = spacy.load('en_core_web_sm')

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 string) (by loo
    :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 [34]:

```
# output to label
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': 0.0}) == 'ne
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.01}) == 'p
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': -0.01}) == '
```

In [35]:

```
# Run VADER (as it is) on the set of airline tweets
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet)
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [36]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

```
              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
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet,lemmatize=True)
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only adjectives
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, parts_of_speech_to_consider={'ADJ'})
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only adjectives and after having lemmatized t
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, lemmatize=True, parts_of_speech_to_consider={'ADJ'}
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only nouns
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, parts_of_speech_to_consider={'NOUN'})
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only nouns and after having lemmatized the te
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, lemmatize=True, parts_of_speech_to_consider={'NOUN'
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only verbs
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, parts_of_speech_to_consider={'VERB'})
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

In [ ]:

```
# Run VADER on the set of airline tweets with only verbs and after having lemmatized the te
vader_model = SentimentIntensityAnalyzer() #reset model

tweets = []
all_vader_output = []
gold = []

for i in range(len(airline_tweets_train.data)):
    the_tweet = str(airline_tweets_train.data[i])
    vader_output = run_vader(the_tweet, lemmatize=True, parts_of_speech_to_consider={'VERB'
    vader_label = vader_output_to_label(vader_output)

    tweets.append(the_tweet)
    all_vader_output.append(vader_label)
    gold.append(airline_tweets_train.target_names[airline_tweets_train.target[i]])
```

In [ ]:

```
# analyize
report = classification_report(gold,all_vader_output)
print(report)
```

Comparison: Firstly, we will compare how well the models perform when with and without lemmatizing the text. Since the data is balanced considering the 3 classes have similar instances, namely 1750, 1515, and 1490 for negative, neutral, and positive respectively, we will focus on the macro average instead of the weighted average, however, the micro average, or the accuracy, will be used for an overall comparison of performance, regardless of class.

For the basic model, the precision has a macro average of 0.65, while the recall is 0.63, leaving the f1_score at 0.62, and the accuracy, or micro average at 0.63. While, for the basic model with lemmatizing the text, the precision is 0.65, recall is 0.63, f1_score of 0.62, and accuracy of 0.62. This means the difference between the models is only evident in the accuracy score, however since the difference is only 0.01, we will consider that lemmatizing the text has no effect on the basic model.

For the model concerning the adjectives, the model without lemmatizing the text has a precision of 0.65, recall of 0.52, and f1_score of 0.48, which the accuracy is 0.50. While the same model but with lemmatized text, the precision is 0.65, recall is 0.52, f1_score is 0.48, and accuracy is 0.50. Similar to the previous situation, the comparison between lemmatized and none-lemmatized shows that lemmatizing the text has no effect on the model.

For the model concerning the nouns, the model without lemmatizing the text has a precision of 0.54, recall of 0.43, and f1_score of 0.38, which the accuracy is 0.42. While the same model but with lemmatized text, the precision is 0.53, recall is 0.43, f1_score is 0.39, and accuracy is 0.42. Similar to the previous situations, the comparison between lemmatized and none-lemmatized shows that lemmatizing the text has no effect on the model.

Finally, For the model concerning the verbs, the model without lemmatizing the text has a precision of 0.58, recall of 0.48, and f1_score of 0.45, which the accuracy is 0.47. While the same model but with lemmatized text, the precision is 0.56, recall is 0.47, f1_score is 0.45, and accuracy is 0.47. Similar to the previous situations, the comparison between lemmatized and none-lemmatized shows that lemmatizing the text has no effect on the model.

Therefore, we can conclude that lemmatizing the text only has no effect on the model's performance, especially in the accuracy as it's always the same in both models of lemmatizing and none-lemmatizing.

Secondly, we compare the performances between the models, in which the comparison will be divided upon comparing the models with lemmatized text and models without. In the case of none-lemmatized text, the model with the most precision is the model that filters on the adjectives part-of-speech, and the basic model with the same 0.65 precision. However, the nouns and verbs models have significantly lower precision with a difference of 0.70 and 0.11 difference. The verbs model is the higher of the two with again a 0.02 difference between it and the nouns model. However, when considering the recall of the models, the basic model performs significantly better than the other three, with a 0.11 difference between the adjectives model, 0.20 with the nouns model, and 0.15 with the verbs model. However, when we check the f1_scores of the models, we can conclude that the basic model performs much better than the other models, which is supported by the accuracy comparison.

Now looking at the models with lemmatized text, we can see the same pattern of the basic model performing better than the others overall, with the adjectives model being second, followed by the verbs model then the nouns model.

A reason for this observation could be that since tweets are normally short, only focusing on parts of the sentence does not allow the model to perform well, and the reason for the nouns model performing the worst is because the nouns usually do not give a good indication of the label of a sentence or tweet, but rather give context. Similarly, verbs do not give a good indication of the nature of the tweet, unless it's an extreme verb like crashed would be negative or smiled would be positive but most verbs would be neutral like walked, said, etc. Similarly, with lemmatizing text, it doesn't have much of an effect on the label of the text as for example crash and crashed are both negative, and hence returning the word to its base/dictionary form doesn't have much of an effect on how positive, negative, or neutral it is.

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 [28]:

```
import pathlib
import sklearn
import numpy
import nltk
from nltk.corpus import stopwords
from collections import Counter
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

In [30]:

```
cwd = pathlib.Path.cwd()
airline_tweets_folder = cwd.joinpath('airlinetweets')
airline_tweets_train = load_files("C:/Users/zahra_6hcxkfv/ba-text-mining/lab_sessions/lab3/
```

In [31]:

```
# initialize airline object, and then turn airline tweets train data into a vector

airline_vec = CountVectorizer(min_df=2, # If a token appears fewer times than this, across
                              tokenizer=nltk.word_tokenize, # we use the nltk tokenizer
                              stop_words=stopwords.words('english')) # stopwords are removed

airline_counts = airline_vec.fit_transform(airline_tweets_train.data)
# Convert raw frequency counts into TF-IDF values
tfidf_transformer = TfidfTransformer()
airline_tfidf = tfidf_transformer.fit_transform(airline_counts)
```

```
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\feature_extractio
n\text.py:388: UserWarning: Your stop_words may be inconsistent with your pr
eprocessing. Tokenizing the stop words generated tokens ["'d", "'ll", "'re",
"'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  warnings.warn('Your stop_words may be inconsistent with '
```

In [37]:

```
# airline tweets 80/20 split, tf-idf, min_df=2
docs_train, docs_test, y_train, y_test = train_test_split(
    airline_tfidf, # the tf-idf model
    airline_tweets_train.target, # the category values for each tweet
    test_size = 0.20 # we use 80% for training and 20% for development
    )
```

In [38]:

```
clf = MultinomialNB().fit(docs_train, y_train)
y_pred = clf.predict(docs_test)
```

In [39]:

```
report = classification_report(y_test,y_pred,digits = 3)

print("Results for airline tweets with 80/20 split, tf-idf, min_df=2")
print(report)
```

```
Results for airline tweets with 80/20 split, tf-idf, min_df=2
              precision    recall  f1-score   support

           0      0.810     0.905     0.854       357
           1      0.809     0.699     0.750       296
           2      0.845     0.839     0.842       298

    accuracy                          0.820       951
   macro avg      0.821     0.814     0.815       951
weighted avg      0.820     0.820     0.818       951
```

In [40]:

```
# airline tweets 80/20 split, bag-of-words
docs_train1, docs_test1, y_train1, y_test1 = train_test_split(
    airline_counts, # the bag-of-words model
    airline_tweets_train.target, # the category values for each tweet
    test_size = 0.20 # we use 80% for training and 20% for development
    )
```

In [41]:

```
clf1 = MultinomialNB().fit(docs_train1, y_train1)
y_pred1 = clf1.predict(docs_test1)

report1 = classification_report(y_test1,y_pred1,digits = 3)

print("Results for airline tweets with 80/20 split, bag-of-words")
print(report1)
```

```
Results for airline tweets with 80/20 split, bag-of-words
              precision    recall  f1-score   support

           0      0.863     0.919     0.890       357
           1      0.870     0.754     0.808       285
           2      0.852     0.893     0.872       309

    accuracy                          0.861       951
   macro avg      0.862     0.855     0.857       951
weighted avg      0.862     0.861     0.860       951
```

In [42]:

```
# airline tweets 80/20 split, tf-idf, min_df=5
airline_vec2 = CountVectorizer(min_df=5, # If a token appears fewer times than this, across
                               tokenizer=nltk.word_tokenize, # we use the nltk tokenizer
                               stop_words=stopwords.words('english')) # stopwords are removed
airline_counts2 = airline_vec2.fit_transform(airline_tweets_train.data)
tfidf_transformer = TfidfTransformer()
airline_tfidf2 = tfidf_transformer.fit_transform(airline_counts2)

docs_train2, docs_test2, y_train2, y_test2 = train_test_split(
    airline_tfidf2, # the tf-idf model
    airline_tweets_train.target, # the category values for each tweet
    test_size = 0.20 # we use 80% for training and 20% for development
    )
```

```
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\feature_extractio
n\text.py:388: UserWarning: Your stop_words may be inconsistent with your pr
eprocessing. Tokenizing the stop words generated tokens ["'d", "'ll", "'re",
"'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  warnings.warn('Your stop_words may be inconsistent with '
```

In [44]:

```
clf2 = MultinomialNB().fit(docs_train2, y_train2)
y_pred2 = clf2.predict(docs_test2)

report2 = classification_report(y_test2,y_pred2,digits = 3)

print("Results for airline tweets with 80/20 split, tf-idf, min_df=5")
print(report2)
```

```
Results for airline tweets with 80/20 split, tf-idf, min_df=5
              precision    recall  f1-score   support

           0      0.828     0.897     0.861       349
           1      0.803     0.747     0.774       273
           2      0.850     0.824     0.836       329

    accuracy                          0.829       951
   macro avg      0.827     0.823     0.824       951
weighted avg      0.828     0.829     0.828       951
```

In [45]:

```
# airline tweets 80/20 split, tf-idf, min_df=10
airline_vec3 = CountVectorizer(min_df=10, # If a token appears fewer times than this, acros
                               tokenizer=nltk.word_tokenize, # we use the nltk tokenizer
                               stop_words=stopwords.words('english')) # stopwords are removed
airline_counts3 = airline_vec3.fit_transform(airline_tweets_train.data)
tfidf_transformer = TfidfTransformer()
airline_tfidf3 = tfidf_transformer.fit_transform(airline_counts3)

docs_train3, docs_test3, y_train3, y_test3 = train_test_split(
    airline_tfidf3, # the tf-idf model
    airline_tweets_train.target, # the category values for each tweet
    test_size = 0.20 # we use 80% for training and 20% for development
    )
```

```
C:\Users\zahra_6hcxkfv\anaconda3\lib\site-packages\sklearn\feature_extractio
n\text.py:388: UserWarning: Your stop_words may be inconsistent with your pr
eprocessing. Tokenizing the stop words generated tokens ["'d", "'ll", "'re",
"'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  warnings.warn('Your stop_words may be inconsistent with '
```

In [46]:

```
clf3 = MultinomialNB().fit(docs_train3, y_train3)
y_pred3 = clf3.predict(docs_test3)

report3 = classification_report(y_test3,y_pred3,digits = 3)

print("Results for airline tweets with 80/20 split, tf-idf, min_df=10")
print(report3)
```

```
Results for airline tweets with 80/20 split, tf-idf, min_df=10
              precision    recall  f1-score   support

           0      0.802     0.868     0.833       340
           1      0.787     0.710     0.747       307
           2      0.804     0.809     0.807       304

    accuracy                          0.798       951
   macro avg      0.798     0.796     0.795       951
weighted avg      0.798     0.798     0.797       951
```

Q5a Classification reports gathered below

In [47]:

```
print("Results for airline tweets with 80/20 split, tf-idf, min_df=2")
print(report)

print("Results for airline tweets with 80/20 split, bag-of-words")
print(report1)
```

```
Results for airline tweets with 80/20 split, tf-idf, min_df=2
              precision    recall  f1-score   support

           0      0.810     0.905     0.854       357
           1      0.809     0.699     0.750       296
           2      0.845     0.839     0.842       298

    accuracy                          0.820       951
   macro avg      0.821     0.814     0.815       951
weighted avg      0.820     0.820     0.818       951

Results for airline tweets with 80/20 split, bag-of-words
              precision    recall  f1-score   support

           0      0.863     0.919     0.890       357
           1      0.870     0.754     0.808       285
           2      0.852     0.893     0.872       309

    accuracy                          0.861       951
   macro avg      0.862     0.855     0.857       951
weighted avg      0.862     0.861     0.860       951
```

In [48]:

```
print("Results for airline tweets with 80/20 split, tf-idf, min_df=5")
print(report2)

print("Results for airline tweets with 80/20 split, tf-idf, min_df=10")
print(report3)
```

```
Results for airline tweets with 80/20 split, tf-idf, min_df=5
              precision    recall  f1-score   support

           0      0.828     0.897     0.861       349
           1      0.803     0.747     0.774       273
           2      0.850     0.824     0.836       329

    accuracy                          0.829       951
   macro avg      0.827     0.823     0.824       951
weighted avg      0.828     0.829     0.828       951

Results for airline tweets with 80/20 split, tf-idf, min_df=10
              precision    recall  f1-score   support

           0      0.802     0.868     0.833       340
           1      0.787     0.710     0.747       307
           2      0.804     0.809     0.807       304

    accuracy                          0.798       951
   macro avg      0.798     0.796     0.795       951
weighted avg      0.798     0.798     0.797       951
```

Q5b The bag-of-words approach seems to get the best results, as indicated by the f1-score. It has higher f1-scores than all three of the tf-idf results for each category (so positive/neutral/negative). The accuracy score is also higher with the bag-of-words approach.

The frequency threshold does have a slight influence on the categories separately, as indicated by the f1-scores. However, it doesn't seem to have too much of an influence on the bigger picture, as all three of the tf-idf strategies have nearly the same accuracy (0.816/0.817/0.816).

[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 [49]:

```
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),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.feature_count_[1], feature_names),reverse=True)[:n]
    topn_class3 = sorted(zip(classifier.feature_count_[2], feature_names),reverse=True)[:n]
    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, clf)
```

```
Important words in negative documents
0 157.88531551543522 united
0 113.873151027286 .
0 98.12839667842987 ``
0 96.98103486178553 @
0 55.646084640149795 flight
0 50.57437534912116 ?
0 45.08738792587808 #
0 40.9325096519484 !
0 40.838513332793184 n't
0 32.74447996440087 ''
0 23.833706773832752 service
0 23.210892142295194 's
0 21.920904111130767 virginamerica
0 21.64121036839044 delayed
0 21.513437607220986 bag
0 20.864141433090914 customer
0 19.976657169737457 cancelled
0 19.562853880284525 plane
```

Answers:

1. For the negative documents, I expected words in their negated form, showed by the output "n't" and more punctuation. For the neutral documents, I expected words that would be used more frequently, so the name of the airlines was a good fit to this. Lastly, for the positive documents, I expected words of appreciation or just positive words in general, and the output contained words related to "thank you" and also punctuation. Punctuations in both positive and negative were to be expected as they are normally used when conveying emotions, as we don't need them in a neutral setting, they also don't appear as much.
2. There are no words that I didn't expect, given that all of the outputs relate to their categories. Perhaps, I didn't expect punctuation, such as question marks, to have such a high scoring in neutral documents. However, this can be liked to just asking a direct question to the airline.
3. Even though punctuation can be very expressive, it is easier to understand in context. For example, exclamation marks have a high scoring in both postive and negative settings, so it is hard to analyze why they are there with just this information. Therefore, I would remove punctuation in order to improve the

readability and understandability of the model. I would also consider removing "@" symbols as they are just used to tag the airline in a tweet and the name of the airlines themselves as they will always appear, don't matter the context.

[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 [ ]: