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}
```

Answers

- Sentence 1: Contains no negative words with 'neg' label resulting in a 0.0 negative score. The rest of the sentence is either neutral or positive, resulting in a positive compound score.

- Sentence 2: 'don't' adds a negative connotation to the overall sentence, because of it 63% of the sentence is seen as negative. However, the sentence in itself does not imply negative feelings, it just expresses one's opinion which is more likely to be considered neutral by a human expert.

- Sentence 3: Higher positive compound score than the one of sentence 1. This is most likely due to emoticon ':-)' which is considered positive in the VADER lexicon with a 1.3 score.

- Sentence 4: 'ruins' in this sentence does not necessarily has a negative context, it rather accesses the constructional state of the houses without any explicit emotions. VADER does not take into account the context of the sentence, but the combined sentiment score of individual words, where ruins has a score of -1.9 in the VADER lexicon.

- Sentence 5: The sentence from the previous bullet point now includes a negative clause 'not' in front of the 'ruins' resulting in a reversed sentiment score of 1.9. Because of this, the sentence now has a positive compound score.

- Sentence 6: Contains no positive words with 'pos' label which results in a 0.0 positive score. The sentence only has neutral or negative scores, which results in a negative compound score. However, this sentence does not necessarily imply a negative connotation. The negative connotation is likely to be due to the word 'lies', which has various meaning to it. Does not take the context into account.

- Sentence 7: The sentence has a negative score of 0.0 which means that it contains no negative words. The sentence has mostly a neutral score, being twice the score of the positive score. This also results in a positive compound score. The scores seem to fit the sentence fairly well since the sentence is indeed quite neutral. The slight positive score may have occured from the word "like" in the sentence.

[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 [12]:

```python
import json
```

In [13]:

```python
my_tweets = json.load(open('my_tweets.json',encoding="utf8"))
```

In [14]:

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

```
1 {'sentiment_label': 'negative', 'text_of_tweet': 'Whoever started this is ridiculous. Putin is an authorit
arian criminal putting his own citizens at war with Ukraine. End Putin Regime and stand with Ukraine', 'twee
t_url': 'https://twitter.com/winnie_LaLisa/status/1499348443667615746?s=20&t=q-bEiPGrxb2aQf5F7hn9Sw'}
```

[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 [2]:

```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [6]:

```python
vader_model = SentimentIntensityAnalyzer()
```

In [7]:

```python
import spacy
nlp = spacy.load("en_core_web_sm") # 'en_core_web_sm'
```

In [8]:

```python
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 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, input_to_vader
```

In [9]:

```
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}) == 'neutral'
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.01}) == 'positive'
assert vader_output_to_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': -0.01}) == 'negative'
```

In [15]:

```
from sklearn.metrics import classification_report

vader_input = []
all_vader_output = []
verbose_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, input_to_vader = run_vader(the_tweet) # run vader
    vader_label = vader_output_to_label(vader_output) # convert vader output to category

    all_vader_output.append(vader_label)
    verbose_output.append(vader_output)
    vader_input.append(input_to_vader)
    gold.append(tweet_info['sentiment_label'])

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

```
              precision    recall  f1-score   support

    negative       0.75      0.67      0.71        18
     neutral       0.50      0.38      0.43        13
    positive       0.58      0.74      0.65        19

    accuracy                           0.62        50
   macro avg       0.61      0.60      0.60        50
weighted avg       0.62      0.62      0.61        50
```

In [17]:

```
n_counter = 0
p_counter = 0
neutral_counter = 0

zip_list = list(zip(all_vader_output, gold))

for idx, item in enumerate(zip_list):
    if item[0] != item[1]:
        print('INPUT:\n{}.\nOUTPUT:\n{}.\nTARGET/PREDICTED: {}/{}\n'.format(vader_input[idx], verbose_output
        if item[1] == 'neutral':
            neutral_counter += 1
        elif item[1] == 'positive':
            p_counter += 1
        else:
            n_counter += 1

print('MISLABELLED COUNT - negative: {}, positive: {}, neutral: {}'.format(n_counter, p_counter, neutral_cou
```

```
INPUT:
['Russian', 'aircraft', 'bombed', 'a', 'residential', 'neighborhood', 'in', 'Chernihiv', '.', 'There', 'wer
e', 'no', 'military', 'installations', 'nearby', '.', 'Only', 'civilians', 'who', 'had', 'gone', 'out', 't
o', 'buy', 'groceries', '.', 'The', 'whole', 'world', 'should', 'know', 'about', 'this', 'war', 'crime',
'.'].
OUTPUT:
{'neg': 0.269, 'neu': 0.731, 'pos': 0.0, 'compound': -0.884}.
TARGET/PREDICTED: neutral/negative

INPUT:
['When', 'the', 'united', 'state', 'of', 'America', 'did', 'this', 'in', 'afghanistan', ',', 'libya', ',',
'iraq', 'did', 'anybody', 'condem', 'it', 'where', 'there', 'no', 'civilians', 'in', 'those', 'buildings',
'they', 'bombed', '?', '?', '?', 'Why', 'is', 'Ukraine', 'so', 'special', '?', '?', '?', '?', '?', '?', '?',
'?', '?', '?', '?'].
OUTPUT:
{'neg': 0.06, 'neu': 0.741, 'pos': 0.199, 'compound': 0.7228}.
TARGET/PREDICTED: negative/positive

INPUT:
['Russia', 'give', 'them', 'the', 'test', 'of', 'their', 'own', 'medicine'].
OUTPUT:
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}.
TARGET/PREDICTED: negative/neutral

INPUT:
['do', 'n’t', 'know', 'who', 'made', 'this', ',', 'but', 'I', '’m', 'not', 'sure', 'I', '’ve', 'loved', 'a',
'piece', 'of', 'art', 'more', ',', 'in', 'a', 'very', 'long', 'time', '.', '#', 'Ukriane', '#', 'Lego'].
OUTPUT:
{'neg': 0.25, 'neu': 0.75, 'pos': 0.0, 'compound': -0.7692}.
TARGET/PREDICTED: positive/negative

INPUT:
['When', '@JM_Scindia', 'was', 'lying', 'to', 'Indian', 'students', ',', 'the', 'mayor', 'of', 'Romania', 'i
nterrupted', 'him', 'and', 'asked', 'him', 'to', 'speak', 'the', 'truth', '.', 'That', 'his', 'government',
'has', 'made', 'all', 'the', 'arrangements', 'for', 'the', 'children', 'to', 'live', 'and', 'eat', '.'].
OUTPUT:
{'neg': 0.14, 'neu': 0.802, 'pos': 0.058, 'compound': -0.5106}.
TARGET/PREDICTED: neutral/negative

INPUT:
['Hello', 'and', 'good', 'morning', '.', 'From', 'today', 'all', 'tweets', 'will', 'be', 'in', 'English',
'.'].
OUTPUT:
{'neg': 0.0, 'neu': 0.791, 'pos': 0.209, 'compound': 0.4404}.
TARGET/PREDICTED: neutral/positive

INPUT:
['I', 'love', 'how', '#', 'CancelCulture', 'is', 'so', 'engrained', 'in', 'our', 'society', 'that', 'we', 't
ry', 'to', 'use', 'it', 'against', 'warlords', 'invading', 'countries', '.', 'Like', 'he', '#', 'gives2fuck
s', '.', 'I', '’m', 'sure', 'those', 'Ukrainians', 'are', 'grateful', 'you', '’re', 'not', 'drinking', 'Russ
ian', 'vodka', '.'].
OUTPUT:
{'neg': 0.0, 'neu': 0.714, 'pos': 0.286, 'compound': 0.9001}.
TARGET/PREDICTED: negative/positive

INPUT:
['Who', 'knew', 'it', 'would', 'be', 'this', 'easy', 'for', '#', 'Singapore', 'and', '#', 'SouthKorea', 't
o', 'enact', 'sanctions', 'against', 'Russia', '?', 'All', 'my', '#', 'Myanmar', 'folks', '(', 'and', 'all',
'from', "'", 'uncivilized', "'", 'part', 'of', 'the', 'world', ')', ',', 'all', 'we', 'need', 'now', 'are',
'bleaching', 'creams', ',', 'blonde', 'hair', 'dyes', ',', 'and', 'a', 'bit', 'of', 'contact', 'lens', '.',
'You', "'re", 'with', 'me', '?'].
OUTPUT:
{'neg': 0.0, 'neu': 0.925, 'pos': 0.075, 'compound': 0.5768}.
TARGET/PREDICTED: negative/positive

INPUT:
['small', 'account', 'matters', '.', 'BTS', 'fans', 'drop', 'your', 'handles', 'to', 'gain', '200', 'mutual
s'].
OUTPUT:
{'neg': 0.135, 'neu': 0.577, 'pos': 0.288, 'compound': 0.34}.
TARGET/PREDICTED: neutral/positive

INPUT:
['This', 'crocheted', 'baby', 'looks', 'like', 'Caroline', 'Manzo'].
OUTPUT:
{'neg': 0.0, 'neu': 0.706, 'pos': 0.294, 'compound': 0.3612}.
TARGET/PREDICTED: neutral/positive

INPUT:
['Government', 'tax', 'credits', 'for', 'electric', 'car', 'purchases', 'contribute', 'substantially', 'to',
'Tesla', "'s", 'growth', '.'].
OUTPUT:
{'neg': 0.0, 'neu': 0.683, 'pos': 0.317, 'compound': 0.6249}.
TARGET/PREDICTED: neutral/positive

INPUT:
['Hypocrisy', '#', 'Ukraine', '#', 'RussianUkrainianWar', ' ', '#', 'russia'].
OUTPUT:
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}.
TARGET/PREDICTED: negative/neutral

INPUT:
['This', 'is', 'so', 'true', '!', '#', 'RussianUkrainianWar'].
OUTPUT:
{'neg': 0.0, 'neu': 0.506, 'pos': 0.494, 'compound': 0.6005}.
TARGET/PREDICTED: neutral/positive

INPUT:
['This', 'is', 'the', 'true', 'face', 'of', 'America', 'and', 'Biden', 'is', 'blaming', 'Putin', '.', 'Ever
y', 'country', 'has', 'a', 'right', 'to', 'think', 'about', 'out', 'their', 'interest', '.', ' ', '#', 'ista
ndwithrussia', '#', 'IStandWithPutin', '#', 'BidenIsALaughingstock', '#', 'RussianUkrainianWar'].
OUTPUT:
{'neg': 0.1, 'neu': 0.719, 'pos': 0.181, 'compound': 0.3818}.
TARGET/PREDICTED: neutral/positive

INPUT:
['Ahhh', '!', '!', '!', '!', '!', 'After', 'so', 'many', 'days', 'finally', 'it', 'came', 'in', 'my', 'tr*nd
ing', 'list', '..', 'FOCUS', 'ON', 'FATEJO'].
OUTPUT:
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}.
TARGET/PREDICTED: positive/neutral

INPUT:
['crying', 'over', 'these', 'replies', 'even', 'if', 'it', "'s", 'not', 'for', 'me', 'but', 'still', '...'].
OUTPUT:
{'neg': 0.136, 'neu': 0.864, 'pos': 0.0, 'compound': -0.2617}.
TARGET/PREDICTED: positive/negative

INPUT:
['the', 'way', 'u', 'feel', 'about', 'a', 'situation', 'is', "n't", 'dramatic', ',', 'do', "n't", 'let', 'ot
hers', 'invalidate', 'it', '.'].
OUTPUT:
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}.
TARGET/PREDICTED: positive/neutral

INPUT:
['do', 'Not', 'ask', 'me', 'about', 'anything'].
OUTPUT:
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}.
TARGET/PREDICTED: positive/neutral

INPUT:
['i', 'ca', 'n’t', 'live', 'like', 'this'].
```

```
OUTPUT:
{'neg': 0.0, 'neu': 0.615, 'pos': 0.385, 'compound': 0.3612}.
TARGET/PREDICTED: negative/positive

MISLABELLED COUNT - negative: 6, positive: 5, neutral: 8
```

Answer

Part a:

Precision: Precision is the number of True Positives divided by the number of True Positives and False Positives. In other words, precision shows how useful the output of the model actually is. The highest precision out of the three values was achieved for the negative tweets with score 0.75, followed up by the positive tweets with score 0.58. The lowest score of 0.50 was achieved for the neutral tweets.
Recall: Recall is the number of True Positives divided by the number of True Positives and False Negatives. In other words, recall shows how complete the output is. The highest recall was acheived for the positive tweets with 0.74 score, meaning that the positive tweets are the most complete out of the three. This is follwed by the neagtive tweets with recall score of 0.67. The lowest score of 0.38 is once again attributed to the neutral tweets, meaning that less than a half of overall neutral tweets was identified correctly.
F1-score: F1-score is the harmonic mean or weighted average of precision and recall, taking into account both false positives and false negativesinto account. A high f1-score would tell us that there is high precision and recall. The f1-score is highly relevant since it is used to counter class imbalance, which is present in our dataset since there is an unequal amount of positive, negative and neutral tweets. In this case, 'negative' and 'positive' have relatively high f1-scores (0,73,0.68 respectively) as compared to 'neutral'.
Support: Support is the count of positive, negative, and neutral tweets in the data. We can see that there is some class imbalance, since the number of each type of tweet is unequal

Part b:

Based on the output above, there are 8 neutral tweets that are incorrectly classified. This is due to the fact that a tweet is only classfied as 'neutral' if the text does not contain any positive or negative words.

Moreover, there are only 4 misclassified positive tweets. 3 of them were incorrectly classified as neutral. This is caused by the lack of use 'positive' words in the sentence even though the overall sentiment of the sentence is rather positive. The 4th tweet was incorrectly classified as negative because it contained negative words such as 'crying' even though the tweet itself was positive.

Finally, there were 6 misclassified negative tweets. Out of these 4 were incorrectly missclassified as postive and 2 as neutral. Most of the ones classified as positive contain positive words such as "love" and "grateful", which results in VADER incorrectly classifying them as positve. For the 2 misclassified as neutral, it is because there are no strong negative words in the sentences even though the sentences are negative overall.

[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 [11]:

```python
import pathlib
from sklearn.datasets import load_files

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())

# loading all files as training data.
airline_tweets_train = load_files(str(airline_tweets_folder))

data = airline_tweets_train.data
labels = airline_tweets_train.target
```

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

In [12]:

```python
def vader_output_to_int_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 0
    elif compound == 0.0:
        return 1
    elif compound > 0.0:
        return 2

assert vader_output_to_int_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.0}) == 1
assert vader_output_to_int_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.01}) == 2
assert vader_output_to_int_label( {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': -0.01}) == 0
```

In [13]:

```python
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet)) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.80      0.51      0.63      1750
           1       0.60      0.51      0.55      1515
           2       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 [14]:

```python
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), lemmatize = True) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.79      0.52      0.63      1750
           1       0.60      0.49      0.54      1515
           2       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 [15]:

```python
# Adjectives
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), parts_of_speech_to_consider={'ADJ'}) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.87      0.22      0.35      1750
           1       0.41      0.89      0.56      1515
           2       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 [16]:

```python
#Adjectives + Lemmatisation
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), lemmatize=True, parts_of_speech_to_consider={'ADJ'}) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.87      0.22      0.35      1750
           1       0.41      0.89      0.56      1515
           2       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 [17]:

```python
#Nouns
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), parts_of_speech_to_consider={'NOUN'}) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.71      0.14      0.23      1750
           1       0.36      0.82      0.50      1515
           2       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 [18]:

```python
#Nouns + Lemmatisation
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), lemmatize=True, parts_of_speech_to_consider={'NOUN'}) # run vade
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support
```

```
           0       0.70      0.15      0.25      1750
           1       0.36      0.81      0.50      1515
           2       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 [19]:

```
#Verbs
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), parts_of_speech_to_consider={'VERB'}) # run vader
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.78      0.28      0.41      1750
           1       0.38      0.81      0.52      1515
           2       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 [20]:

```
#Verbs + Lemmatisation
all_vader_output_air = []

for tweet in data:
    vader_output, _ = run_vader(str(tweet), lemmatize=True, parts_of_speech_to_consider={'VERB'}) # run vade
    vader_label = vader_output_to_int_label(vader_output) # convert vader output to category

    all_vader_output_air.append(vader_label)

print(classification_report(labels, all_vader_output_air))
```

```
              precision    recall  f1-score   support

           0       0.75      0.29      0.42      1750
           1       0.38      0.79      0.51      1515
           2       0.56      0.35      0.43      1490

    accuracy                           0.47      4755
   macro avg       0.56      0.48      0.45      4755
weighted avg       0.57      0.47      0.45      4755
```

Answer Based on the classification report, it looks like there is negligible difference between the classification reports with and without lemmatisation (majority of the report values are similar), hence leading to the conclusion that lemmatizing verbs, nouns and adjectives does not impact performance of the models.
For this task it seems that taking into account all parts of speech gives the best results, where only focusing on one part of speech seems to be disadvantageous and results in a decrease in accuracy. However, between choosing only part of speech, adjectives seem to have the best accuracy although not by much.

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 [1]:

```
import nltk
from nltk.corpus import stopwords
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
```

In [3]:

```
def get_bag_of_words(input, threshold):
    airline_vec = CountVectorizer(min_df = threshold, # If a token appears fewer times than this, across all
                                  tokenizer = nltk.word_tokenize, # we use the nltk tokenizer
                                  stop_words = stopwords.words('english')) # stopwords are removed

    airline_counts = airline_vec.fit_transform(input.data)
    return airline_counts, airline_vec
```

In [4]:

```
def get_tfidf(input, threshold):
    airline_counts, _ = get_bag_of_words(input, threshold)
    tfidf_transformer = TfidfTransformer()
    airline_tfidf = tfidf_transformer.fit_transform(airline_counts)
    return airline_tfidf
```

In [16]:

```
bag_2, vectorizer_2 = get_bag_of_words(airline_tweets_train, 2)
bag_5, _ = get_bag_of_words(airline_tweets_train, 5)
bag_10, _  = get_bag_of_words(airline_tweets_train, 10)

tfidf_2 = get_tfidf(airline_tweets_train, 2)
tfidf_5 = get_tfidf(airline_tweets_train, 5)
tfidf_10 = get_tfidf(airline_tweets_train, 10)
```

```
C:\Users\lmps\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\feature_extraction\text.py:40
1: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words gener
ated tokens ["'d", "'ll", "'re", "'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  % sorted(inconsistent)
C:\Users\lmps\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\feature_extraction\text.py:40
1: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words gener
ated tokens ["'d", "'ll", "'re", "'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  % sorted(inconsistent)
C:\Users\lmps\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\feature_extraction\text.py:40
1: UserWarning: Your stop_words may be inconsistent with your preprocessing. Tokenizing the stop words gener
ated tokens ["'d", "'ll", "'re", "'s", "'ve", 'could', 'might', 'must', "n't", 'need', 'sha', 'wo', 'would']
not in stop_words.
  % sorted(inconsistent)
```

In [25]:

```
def get_model(representation, verbose = False):
    docs_train, docs_test, y_train, y_test = train_test_split(
        representation, # the representation model
        airline_tweets_train.target, # the category values for each tweet
        test_size = 0.20 # we use 80% for training and 20% for development
        )

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

    if verbose == True:
        print(classification_report(y_test, y_pred))

    return clf, y_pred
```

In [26]:

```
clf, _ = get_model(bag_2, True)
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       356
           1       0.83      0.70      0.76       280
           2       0.84      0.85      0.85       315

    accuracy                           0.83       951
   macro avg       0.83      0.82      0.83       951
weighted avg       0.83      0.83      0.83       951
```

In [27]:

```
_ = get_model(bag_5, True)
```

```
              precision    recall  f1-score   support

           0       0.84      0.93      0.88       347
           1       0.83      0.71      0.76       299
           2       0.82      0.84      0.83       305

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

In [28]:

```python
_ = get_model(bag_10, True)
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       347
           1       0.80      0.73      0.76       303
           2       0.87      0.82      0.84       301

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

In [29]:

```python
_ = get_model(tfidf_2, True)
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87       352
           1       0.86      0.72      0.78       315
           2       0.82      0.86      0.84       284

    accuracy                           0.83       951
   macro avg       0.84      0.83      0.83       951
weighted avg       0.84      0.83      0.83       951
```

In [30]:

```python
_ = get_model(tfidf_5, True)
```

```
              precision    recall  f1-score   support

           0       0.82      0.93      0.87       341
           1       0.88      0.72      0.79       324
           2       0.82      0.87      0.85       286

    accuracy                           0.84       951
   macro avg       0.84      0.84      0.84       951
weighted avg       0.84      0.84      0.84       951
```

In [31]:

```python
_ = get_model(tfidf_10, True)
```

```
              precision    recall  f1-score   support

           0       0.85      0.87      0.86       345
           1       0.78      0.77      0.78       296
           2       0.84      0.83      0.83       310

    accuracy                           0.83       951
   macro avg       0.82      0.82      0.82       951
weighted avg       0.83      0.83      0.83       951
```

Answer The model with tdidf with min_df 5 is the best model out of the generated models based on its macro average f1-score, where it obtained 0.84. However, it only performs marginally better than the other models. We chose to use the macro average as a benchmark as the performance amongst the classes is quite balanced. It seems to perform similarly for all cases.

The frequency threshold seems to affect the scores, but rather minimally, since the the model tfidf with min_df 5 performs the best out of the other models, but generally, the precision, recall, f1-score and macro average etc. have little variation across the other tfidf models.

[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 [32]:

```python
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(vectorizer_2, clf)
```

```
Important words in negative documents
0 1491.0 @
0 1382.0 united
0 1232.0 .
0 405.0 ``
0 405.0 ?
0 394.0 flight
0 380.0 !
0 294.0 #
0 212.0 n't
0 159.0 ''
0 127.0 's
0 115.0 :
0 102.0 virginamerica
0 99.0 get
0 98.0 service
0 93.0 cancelled
0 88.0 time
0 87.0 bag
0 85.0 delayed
0 85.0 customer
0 84.0 plane
0 79.0 ...
0 78.0 -
0 75.0 'm
0 74.0 http
0 74.0 ;
0 72.0 hours
0 68.0 &
0 66.0 gate
0 65.0 hour
0 63.0 still
0 61.0 late
0 60.0 help
0 60.0 airline
0 58.0 amp
0 58.0 2
0 57.0 would
0 54.0 worst
0 51.0 one
0 49.0 flights
0 48.0 flightled
0 47.0 never
0 47.0 like
0 46.0 ca
0 45.0 waiting
0 45.0 delay
0 45.0 've
0 44.0 $
0 43.0 back
0 43.0 3
0 42.0 (
0 40.0 us
0 40.0 lost
0 39.0 luggage
0 39.0 ever
0 39.0 check
0 39.0 )
0 38.0 due
0 37.0 really
0 37.0 day
0 36.0 wait
0 36.0 seat
0 36.0 people
0 36.0 fly
0 36.0 another
0 34.0 u
0 33.0 trying
0 33.0 problems
0 33.0 hold
0 32.0 ticket
0 32.0 got
0 32.0 days
0 31.0 thanks
0 31.0 going
0 31.0 bags
0 31.0 baggage
0 30.0 last
0 30.0 even
0 30.0 airport
0 30.0 4
-----------------------------------------
Important words in neutral documents
1 1426.0 @
1 536.0 ?
1 503.0 .
1 320.0 jetblue
1 276.0 :
1 261.0 united
1 259.0 #
1 258.0 southwestair
1 247.0 ``
1 233.0 flight
1 204.0 americanair
1 178.0 http
1 176.0 !
1 164.0 usairways
1 136.0 's
1 86.0 get
1 77.0 -
1 75.0 virginamerica
1 72.0 ''
1 71.0 flights
1 63.0 please
1 60.0 )
1 58.0 help
1 53.0 need
1 52.0 n't
1 51.0 ...
1 51.0 (
1 47.0 ;
1 45.0 dm
1 44.0 tomorrow
1 43.0 us
1 40.0 flying
1 40.0 &
1 37.0 would
1 36.0 way
1 36.0 thanks
1 35.0 'm
1 34.0 know
1 34.0 cancelled
1 32.0 ”
1 32.0 hi
1 32.0 fleet
1 32.0 fleek
1 31.0 “
1 31.0 one
1 31.0 amp
1 30.0 change
1 29.0 like
1 28.0 new
1 28.0 fly
1 28.0 could
1 27.0 number
1 26.0 time
1 26.0 go
1 25.0 airport
1 24.0 travel
1 23.0 sent
1 23.0 see
1 23.0 check
1 22.0 guys
1 21.0 trying
1 21.0 today
1 21.0 ticket
1 21.0 make
1 21.0 going
1 21.0 follow
1 21.0 back
1 20.0 use
1 20.0 tickets
1 20.0 first
1 20.0 chance
1 20.0 2
1 19.0 want
1 19.0 trip
1 19.0 start
1 19.0 rt
1 19.0 add
1 18.0 think
1 18.0 still
1 18.0 seat
-----------------------------------------
Important words in positive documents
2 1303.0 @
2 1008.0 !
2 744.0 .
2 318.0 #
2 306.0 southwestair
2 286.0 thanks
2 285.0 jetblue
2 243.0 thank
2 241.0 united
2 241.0 ``
2 175.0 flight
2 170.0 americanair
2 167.0 :
2 135.0 usairways
2 129.0 great
2 88.0 service
2 83.0 virginamerica
2 81.0 )
2 78.0 love
2 71.0 http
2 65.0 much
2 63.0 's
2 62.0 customer
2 61.0 best
2 58.0 guys
2 58.0 ;
2 54.0 awesome
2 49.0 airline
2 48.0 -
2 46.0 got
2 46.0 good
2 43.0 amazing
2 42.0 &
2 40.0 n't
2 39.0 us
2 39.0 today
2 39.0 time
2 36.0 help
2 36.0 get
2 35.0 fly
2 35.0 amp
2 34.0 made
2 33.0 gate
2 33.0 flying
2 32.0 crew
2 31.0 home
2 30.0 ...
2 27.0 new
2 27.0 back
2 26.0 see
2 26.0 appreciate
2 25.0 nice
2 25.0 ?
2 25.0 're
2 24.0 tonight
2 24.0 day
2 23.0 work
2 23.0 u
2 23.0 response
2 23.0 ''
2 22.0 would
2 22.0 well
2 22.0 team
2 22.0 like
2 21.0 yes
2 21.0 plane
2 21.0 know
2 21.0 flights
2 21.0 ever
2 21.0 always
2 20.0 staff
2 20.0 first
2 20.0 class
2 20.0 'll
2 19.0 please
2 19.0 'm
2 18.0 thx
2 18.0 southwest
2 18.0 make
2 18.0 helpful
```

```
C:\Users\Gebruiker\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_
feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use g
et_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Answer

For the negatively labeled documents, features like *cancelled*, *delayed* and , are expected words related to reviews with a negative sentiment (or experience). Regarding the positively labeled document, positively associated words like *great*,*awesome* and *amazing* can also be seen as part of the important features. Lastly, for the neutral labeled documents, words like *help* and *would*, which are often associated with questions rather than direct positive or negative feedback, are expected to be part of the important words regarding that class.

In general, interpunction, nouns related to organizations and numbers, are not the type of words that one would expect to be quite decisive in class labeling. Leaving those type of words out of the models, would in our opinion improve the models since these type of words often do not have strong sentimental meaning/association. On the other had, words that do have a strong sentimental meaning/association would need to remain in order to have those type of words be more decisive for the labeling

[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 [ ]: