Group8_lab3

February 23, 2025

1 Lab3 - Assignment Sentiment

This notebook describes the LAB-3 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 F1) * 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 F1) 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.

1.1 Credits

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

1.2 Part I: VADER assignments

1.2.1 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)

1.2.2 [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}
```

1.3 Sentence 1: "I love apples"

Explanation:
In this sentence, VADER identifies the word "love" as strongly positive. The remaining words ("I" and "apples") are not associated with any sentiment in the lexicon. So, the overall sentiment is driven almost completely by "love," resulting in a high positive score. The classification of the sentence is correct.

1.4 Sentence 2: "I don't love apples"

Explanation:
The presence of the negation "don't" inverts the sentiment of the word "love." VADER's negation rule switches the polarity of a positive term to negative, leading to an overall negative sentiment for the sentence. This shows how even a single negator can significantly change the sentiment outcome. The classification of the sentence is correct.

1.5 Sentence 3: "I love apples :-)"

Explanation:
The positive term "love" is again present, and the additional emoticon ":-)" provides an extra boost. VADER is designed to recognize common emoticons and assign them sentiment values. The combination results in an even higher overall positive sentiment than the sentence without the emoticon. The classification of the sentence is correct.

1.6 Sentence 4: "These houses are ruins"

Explanation:
The key term in this sentence is "ruins," which in VADER's lexicon carries a negative connotation. As a result, the sentence is interpreted as negative. However, the context is unknown, "ruins" could describe old structures as well but VADER does not perform context disambiguation, so it relies solely on the lexicon value. The negative classification may not be very reasonable.

1.7 Sentence 5: "These houses are certainly not considered ruins"

Explanation:
In this sentence, the negative word "ruins" is modified by the negation "not," which causes VADER to invert its sentiment. Additionally, the modifier "certainly" slightly improves the sentiment intensity. The inversion of a negative term leads VADER to interpret the sentence as leaning towards a positive sentiment, even though one might expect it to be more neutral. The positive classification may not be very reasonable.

1.8 Sentence 6: "He lies in the chair in the garden"

Explanation:
The word "lies" is ambiguous. Although it can mean "reclines" in a neutral sense, VADER's lexicon associates "lies" with dishonesty, a negative trait. Without the ability to disambiguate between the meanings, VADER assigns a negative sentiment to the sentence, even though the intended meaning is simply descriptive (and, thus, one would be expect the sentiment to be neutral). The negative classification is incorrect.

1.9 Sentence 7: "This house is like any house"

Explanation:
In this case, the word "like" is interpreted by VADER as a positive signal. However, the sentence is meant to express a neutral comparison, that the house is unremarkable. Because VADER registers "like" with a positive bias, it results in a slight positive sentiment, which does not fully capture the neutral intent of the sentence. This positive classification is incorrect.

1.10 Final Outcome:

VADER's approach relies on a fixed lexicon and a set of heuristic rules. While it effectively captures clear positive and negative signals (handles negation and emoticons), its inability to disambiguate word meanings or interpret subtle contextual cues can lead to sentiment scores that do not always match the intended sentiment of the sentence.

1.10.1 [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. If you have trouble accessing Twitter, try to find an existing dataset (on websites like kaggle or huggingface).

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 sta
        "tweet_url" : "https://twitter.com/BarackObama/status/946775615893655552",
    },
```

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

[1]:
```
import json
############################ imports below have been added manually for solving␣
 ↪the questions in this NB
import nltk
from sklearn.datasets import load_files
from nltk.tokenize import word_tokenize
```

[2]:
```
my_tweets = json.load(open('my_tweets.json'))
```

[3]:
```
for id_, tweet_info in list(my_tweets.items())[::-1]:
    print(id_, tweet_info)
    break
```

```
50 {'sentiment_label': 'neutral', 'text_of_tweet': 'Scientists discovered a new
type of organism in the depths of the ocean.', 'tweet_url': 'manually created'}
```

1.10.2 [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.

[4]:
```
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'
```

[5]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import spacy

nlp = spacy.load("en_core_web_sm")
vader_model = SentimentIntensityAnalyzer()
```

Reuse `run_vader` from ***Lab3.2*** *notebook.*

[6]:
```
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', textual_unit) # change to textual_unit so the␣
 ↪whole tweet is displayed
```

```
        print('INPUT TO VADER', input_to_vader)
        print('VADER OUTPUT', scores)

    return scores
```

[7]:
```
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=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

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


from sklearn.metrics import classification_report, precision_score,␣
 ↪recall_score, f1_score

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

```
              precision    recall  f1-score   support

    negative       1.00      0.75      0.86        16
     neutral       0.75      0.75      0.75        16
    positive       0.82      1.00      0.90        18

    accuracy                           0.84        50
   macro avg       0.86      0.83      0.84        50
weighted avg       0.85      0.84      0.84        50
```

(A) Quantitative Evaluation **!! For balanced multi-class classification, the micro averages are equal to the accuracy because all of them are computed using the total count of correct predictions divided by the total number of samples.**

VADER analyzes the polarity of words and produces 4 different sentiment scores for each text input:

- Positive (`pos`): The proportion of words that express a positive sentiment
- Negative (`neg`): The proportion of words that express a negative sentiment
- Neutral (`neu`): The proportion of words that are neutral / lack clear sentiment
- Compound (`compound`): An aggregated sentiment score; ranges from -1 (extremely negative) to +1 (extremely positive)

After producing those 4 scores, the final sentiment label of the tweet is assigned based on the `compound` score, as detailed in the `vader_output_to_label` function. Therefore, the `compound` score is the most relevant score for the final label classification, which is done according to the following scheme:

- **`compound` $> 0 \rightarrow$** *positive* label
- **`compound` $< 0 \rightarrow$** *negative* label
- **`compound` $= 0 \rightarrow$** *neutral* label

In order to evaluate the performence of the VADER classifier, we used the scikit-learn's classification report. We present and explain the insights it provides: 1. **Precision** - *how many tweets did VADER classify correctly out of all classifications for a certain category*:

- Negative (`1.00`) - VADER achieves perfect precision on negative examples (correctly predicts them 100% of the time)
- Neutral (`0.75`) - some tweets misclassified as neutral were actually positive or negative
- Positive (`0.82`) - overall high precision on positive tweets, however, *18%* of those classified as positive are being misclassified

2. **Recall** - *how many tweets did VADER classify correctly out of all the tweets in a category (as determined by the gold label)*:

- Negative (`0.75`) - some negative tweets were misclassified as neutral or positive
- Neutral (`0.75`) - VADER manages to identify *75%* of actual neutral tweets
- Positive (`1.00`) - all actual positive tweets were detected correctly

3. **F1-score** - *harmonic mean of precision and recall*:

- Negative (`0.86`) - strong performance in detecting negative tweets
- Neutral (`0.75`) - decent performance in classifying neutral tweets. Across the three categories, VADER has the lowest score for neutral tweets
- Positive (`0.90`) - highly effective at identifying positive tweets

4. **Accuracy** - *overall percentage of correct classifications* (equal to **Micro Average Precision, Recall and F1-score**):

- `0.84` - VADER correctly classified *84%* of all tweets

5. **Macro average** & **Weighted average**

- Macro average - the average of precision, recall, and F1-score across all classes, treating each class equally (no weights)
- Weighted average - the average of precision, recall, and F1-score, weighted by the number of samples in each class
- Macro and weighted averages both showed nearly identical scores, suggesting that there are no severe class imbalance issues, and the model has a good stable performance across all classes. The micro average (accuracy) also produces very similar values, which further verifies this observation.

For evaluating the overall performance, F1-scores and accuracy are most relevant as they give a good general idea of how well the classifier works. Accuracy alone can be a misleading metric, as it does not give insight on the types of errors the model makes and is sensitive to class imbalance - when one category is much more frequent that the rest (this is not the case here), therefore, combining it with F1-score, we can get a good overview of the performance.

Error Analysis:

[8]:
```
misclassified_counts = {"positive": 0, "negative": 0, "neutral": 0}
misclassified_tweets = {"positive": [], "negative": [], "neutral": []}

for i in range(len(tweets)):
    if gold[i] != all_vader_output[i]:  #classification is incorrect
        true_label = gold[i]
        predicted_label = all_vader_output[i]

        misclassified_counts[true_label] += 1

        misclassified_tweets[true_label].append((tweets[i], predicted_label))

# Summary of misclassified words
print("\nMisclassification Summary")
print("-" * 50)
for sentiment, count in misclassified_counts.items():
    print(f"{sentiment.capitalize()} misclassified: {count}")
print("*" * 50)
for sentiment, errors in misclassified_tweets.items():
    print(f"\nMisclassified {sentiment.capitalize()} Tweets")
    print("~" * 50)

    for tweet, predicted in errors: # print the tweets
        print(f"\nTweet: {tweet}")
        print(f"Expected: {sentiment}; Predicted: {predicted}")

        print("\nVADER with verbose:") # see what vader assigns
        run_vader(tweet, lemmatize=True, verbose=1)
        print("*" * 50)
```

```
Misclassification Summary
--------------------------------------------------
Positive misclassified: 0
Negative misclassified: 4
Neutral misclassified: 4
**************************************************

Misclassified Positive Tweets
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Misclassified Negative Tweets
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Tweet: The traffic today was unbearable. Took me whole two hours to get home.
Expected: negative; Predicted: neutral

VADER with verbose:

INPUT SENTENCE The traffic today was unbearable. Took me whole two hours to get
home.
INPUT TO VADER ['the', 'traffic', 'today', 'be', 'unbearable', '.', 'take', 'I',
'whole', 'two', 'hour', 'to', 'get', 'home', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
**************************************************

Tweet: The food at the restaurant was cold and tasteless. Never going back.
Expected: negative; Predicted: neutral

VADER with verbose:

INPUT SENTENCE The food at the restaurant was cold and tasteless. Never going
back.
INPUT TO VADER ['the', 'food', 'at', 'the', 'restaurant', 'be', 'cold', 'and',
'tasteless', '.', 'never', 'go', 'back', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
**************************************************

Tweet: The service at the cafe was slow and the coffee was cold.
Expected: negative; Predicted: neutral

VADER with verbose:

INPUT SENTENCE The service at the cafe was slow and the coffee was cold.
INPUT TO VADER ['the', 'service', 'at', 'the', 'cafe', 'be', 'slow', 'and',
'the', 'coffee', 'be', 'cold', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
**************************************************

Tweet: The restaurant was overpriced and the food was mediocre.
Expected: negative; Predicted: neutral

VADER with verbose:

INPUT SENTENCE The restaurant was overpriced and the food was mediocre.
INPUT TO VADER ['the', 'restaurant', 'be', 'overprice', 'and', 'the', 'food',
'be', 'mediocre', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
**************************************************
```

```
Misclassified Neutral Tweets
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Tweet: Reading about the latest tech innovations. There have been a lot of new
developments.
Expected: neutral; Predicted: positive

VADER with verbose:

INPUT SENTENCE Reading about the latest tech innovations. There have been a lot
of new developments.
INPUT TO VADER ['read', 'about', 'the', 'late', 'tech', 'innovation', '.',
'there', 'have', 'be', 'a', 'lot', 'of', 'new', 'development', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 0.822, 'pos': 0.178, 'compound': 0.3818}
**************************************************

Tweet: Listening to a podcast about space exploration. So much new
information…
Expected: neutral; Predicted: positive

VADER with verbose:

INPUT SENTENCE Listening to a podcast about space exploration. So much new
information…
INPUT TO VADER ['listen', 'to', 'a', 'podcast', 'about', 'space', 'exploration',
'.', 'so', 'much', 'new', 'information', '…']
VADER OUTPUT {'neg': 0.0, 'neu': 0.84, 'pos': 0.16, 'compound': 0.2263}
**************************************************

Tweet: Would you like to watch the sunset at the park with me tonight?
Expected: neutral; Predicted: positive

VADER with verbose:

INPUT SENTENCE Would you like to watch the sunset at the park with me tonight?
INPUT TO VADER ['would', 'you', 'like', 'to', 'watch', 'the', 'sunset', 'at',
'the', 'park', 'with', 'I', 'tonight', '?']
VADER OUTPUT {'neg': 0.0, 'neu': 0.815, 'pos': 0.185, 'compound': 0.3612}
**************************************************

Tweet: Listening to a podcast about productivity. It provides many useful tips.
Expected: neutral; Predicted: positive

VADER with verbose:

INPUT SENTENCE Listening to a podcast about productivity. It provides many
useful tips.
```

```
INPUT TO VADER ['listen', 'to', 'a', 'podcast', 'about', 'productivity', '.',
'it', 'provide', 'many', 'useful', 'tip', '.']
VADER OUTPUT {'neg': 0.0, 'neu': 0.756, 'pos': 0.244, 'compound': 0.4404}
**************************************************
```

(B) Error Analysis In total, 0 positive tweets were misclassified, 4 negative tweets were misclassified as neutral, and 4 neutral tweets were misclassified as positive.

1. Misclassified Positive Tweets

- All true-positive tweets are correctly identified.

2. Misclassified Negative Tweets

- All 4 true-negative tweets were misclassified as neutral. In all of them, we see that the `neu` score is 1.0, meaning that the classifier is absolutely sure the tweets convey a neutral tone. Therefore, its lexicon fails to associate the negative words in the sentence with a negative sentiment. For example, the tweet `"The traffic today was unbearable. Took me whole two hours to get home."` clearly carries a negative sentiment, because of the word `unbearable`, which seems to be missing from VADER's lexicon, possibly accounting for the fact that it classified the tweet as `neutral`. Similarly, the words `cold` and `tasteless` from the second tweet `"The food at the restaurant was cold and tasteless. Never going back."` are also absent from VADER's lexicon, which leads to a misclassification as neutral despite the negative sentiment. The words `slow` and `cold` (again) indicating a negative sentiment in the third tweet `"The service at the cafe was slow and the coffee was cold."` are missing from the lexicon as well, contributing to the tweet's neutral classification. Lastly, the terms `overpriced` and `mediocre` from the fourth tweet `"The restaurant was overpriced and the food was mediocre."` are not found in VADER's lexicon, which likely resulted in the neutral classification of the tweet, even though it has a negative tone.

3. Misclassified Neutral Tweets

- As shown in the performance analysis, the classifier has the lowest performance score when it comes to neutral tweets. All 4 of the misclassified neutral tweets were classified as positive, suggesting that VADER tends to assign a sentiment even when the intended tone is neutral. A common issue observed is that VADER overweights slightly positive words, such as `innovation`, `exploration`, `useful`, and `like`. These words do not necessarily indicate strong positive emotion, but VADER assigns them a positive sentiment score. For example, in `"Reading about the latest tech innovations. There have been a lot of new developments."`, the word `innovations` is factual rather than an expression of enthusiasm, yet it leads to a positive classification. The same is true for the word `exploration` in `"Listening to a podcast about space exploration. So much new information..."`, which also causes a misclassification. Moreover, VADER lacks context awareness, causing misinterpretations of the tone. In `"Would you like to watch the sunset at the park with me tonight?"`, the phrase `would you like` is not an expression of excitement but rather a neutral request, yet VADER assigns it a positive connotation. Similarly, in `"Listening to a podcast about productivity. It provides many useful tips."`, the word `useful` describes a fact rather than carrying a positive sentiment, yet VADER considers the tweet to be positive.
- In addition to the points made above, another possible reason for the observed misclassifica-

tions is that the threshold for classifying a tweet as positive is too lenient. Currently, any `compound` score greater than `0.0` results in a positive classification, even if the score is barely a positive number. This means that tweets with weakly positive words, even if the intended tone is neutral, are pushed towards the positive category. Raising the threshold slightly could help reduce these errors and better differentiate between truly neutral and positive tweets.

1.10.3 [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 F1 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.

[9]:
```
# path to folder
from pathlib import Path

cur_dir = Path().resolve()
path_to_folder = Path.joinpath(cur_dir, r"airlinetweets")
#path_to_folder = Path.joinpath(cur_dir, r"airlinetweets/airlinetweets")

print(path_to_folder)
```

```
/Users/joanapetkova/Documents/VU AI/Text
Mining/TextMiningGroup8/Lab3/airlinetweets
```

[10]:
```
# load the data
import os
```

```
data = []
categories = ["positive", "negative", "neutral"]
for c in categories:
    category_path = os.path.join(path_to_folder, c)

    for txt_file in os.listdir(category_path):
        txt_file_path = os.path.join(category_path, txt_file)

        with open(txt_file_path, "r", encoding="utf-8") as file:
            tweet = file.read().strip()
            data.append((tweet, c))

# checking random example
print(data[5])
```

```
('@JetBlue great flight on a brand new jet. Great seating. Beautiful plane. Big
fan of this airline.', 'positive')
```

1.10.4 Experiment 1: Basic VADER

[11]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

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

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              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      0.64      0.62      4755
weighted avg       0.66      0.63      0.62      4755
```

1.10.5 Experiment 2: VADER with Lemmatized Text

[12]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

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

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.79      0.52      0.63      1750
     neutral       0.60      0.49      0.54      1515
    positive       0.56      0.88      0.68      1490

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

1.10.6 Experiment 3: VADER with Only Adjectives

[13]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

# settings (to change for different experiments)
```

```
to_lemmatize = False
pos = {"ADJ", "JJ", "JJR", "JJS"}  # include universal as well as Penn Treebank␣
 ↪tags for adjective

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.86      0.20      0.33      1750
     neutral       0.40      0.89      0.55      1515
    positive       0.67      0.44      0.53      1490

    accuracy                           0.50      4755
   macro avg       0.64      0.51      0.47      4755
weighted avg       0.65      0.50      0.46      4755
```

1.10.7 Experiment 4: VADER with Lemmatized Text of Only Adjectives

[14]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

# settings (to change for different experiments)
to_lemmatize = True
pos = {"ADJ", "JJ", "JJR", "JJS"}  # include universal as well as Penn Treebank␣
 ↪tags for adjective

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores
```

```
    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.86      0.20      0.33      1750
     neutral       0.40      0.89      0.55      1515
    positive       0.67      0.44      0.53      1490

    accuracy                           0.50      4755
   macro avg       0.64      0.51      0.47      4755
weighted avg       0.65      0.50      0.46      4755
```

1.10.8 Experiment 5: VADER with Only Nouns

[15]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

# settings (to change for different experiments)
to_lemmatize = False
pos = {"NOUN", "NN", "NNS"}  # include universal as well as Penn Treebank tags␣
 ↪for nouns (proper nouns not included)

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.73      0.14      0.23      1750
     neutral       0.36      0.82      0.50      1515
    positive       0.53      0.35      0.42      1490
```

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

1.10.9 Experiment 6: VADER with Lemmatized Text of Only Nouns

[16]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

# settings (to change for different experiments)
to_lemmatize = True
pos = {"NOUN", "NN", "NNS"}  # include universal as well as Penn Treebank tags␣
 ↪for nouns (proper nouns not included)

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.71      0.15      0.25      1750
     neutral       0.36      0.81      0.50      1515
    positive       0.52      0.34      0.41      1490

    accuracy                           0.42      4755
   macro avg       0.53      0.44      0.39      4755
weighted avg       0.54      0.42      0.38      4755
```

1.10.10 Experiment 7: VADER with Only Verbs

[17]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []
```

```
# settings (to change for different experiments)
to_lemmatize = False
pos = {"VERB", "MD", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ"}  # include␣
 ↪universal as well as Penn Treebank tags for verbs

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores

    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.79      0.29      0.42      1750
     neutral       0.38      0.81      0.52      1515
    positive       0.57      0.35      0.43      1490

    accuracy                           0.47      4755
   macro avg       0.58      0.48      0.46      4755
weighted avg       0.59      0.47      0.46      4755
```

1.10.11 Experiment 8: VADER with Lemmatized Text of Only Verbs

[18]:
```
airline_tweets = []
airline_all_vader_output = []
airline_gold = []

# settings (to change for different experiments)
to_lemmatize = True
pos = {"VERB", "MD", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ"}  # include␣
 ↪universal as well as Penn Treebank tags for verbs

# perform sentiment analysis with VADER on each tweet
for tweet, label in data:
    vader_output = run_vader(tweet, lemmatize=to_lemmatize,␣
 ↪parts_of_speech_to_consider=pos) # use run_vader function
    vader_label = vader_output_to_label(vader_output) # use the predefined␣
 ↪function above to get the labels based on scores
```

```
    airline_tweets.append(tweet)
    airline_all_vader_output.append(vader_label)
    airline_gold.append(label)

# use scikit-learn's classification report
print(classification_report(airline_gold, airline_all_vader_output))
```

```
              precision    recall  f1-score   support

    negative       0.75      0.29      0.42      1750
     neutral       0.38      0.78      0.51      1515
    positive       0.57      0.36      0.44      1490

    accuracy                           0.47      4755
   macro avg       0.57      0.48      0.46      4755
weighted avg       0.58      0.47      0.46      4755
```

1.10.12 Comparison

Before discussing the results, we want to note that for balanced multi-class classification, the micro averages are equal to the accuracy because all of them are computed using the total count of correct predictions divided by the total number of samples.

Looking at the results, we can observe that the basic VADER model from Experiment 1, which applies no lemmatization and considers all parts of speech, seems to perform the best, achieving the highest results in terms of accuracy (42%) and macro averages of precision (65%), recall (64%) and F1-score (62%). The VADER model from Experiment 2, which applies lemmatization and also considers all parts of speech, follows with only a slight decrease of 1-2% in the accuracy and macro average metrics.

When restricting input to specific parts of speech as in Experiments 3 to 8, a drop in the performance of VADER in terms of all average metrics is observed compared to the versions that consider all parts of speech. Adjectives alone (Experiments 3 and 4) yielded the best results compared to using only nouns or only verbs, with accuracy of 50%, macro average precision of 64%, macro average recall of 51%, and macro average F1-score of 47%. There was no difference in averages between versions with and without lemmatization that use only adjectives. Verbs alone (Experiments 7 and 8) were second best, with accuracy of 47%, macro average precision of 57-58%, macro average recall of 48%, and macro average F1-score of 46%. There was a slight difference of 1% in the average precision between the versions that use only verbs with and without lemmatization (no lemmatization lead to the higher macro average value of 58%). Finally, using only nouns (Experiments 5 and 6) led to the worst results, with accuracy of 42%, macro average precision of 53-54%, macro average recall of 44%, and macro average F1-score of 39%. Again, a difference of 1% was observed in the macro average precision between the versions that use only nouns with and without lemmatization (no lemmatization again showed the slightly higher macro average of 54%). Thus, the results indicate that adjectives contribute the most sentiment to sentiment classification followed by verbs, which aligns with the fact that sentiment is often conveyed through descriptive words (e.g. “pleasant”, “exploit”).

All versions of VADER do not exhibit significant differences between the accuracy and the macro averages of recall and F1-score (the maximum difference is 3%). However, most of them (except for the versions from Experiments 1 and 2, which look at all parts of speech) exhibit a difference of around 10% in micro average precision (accuracy) and macro average precision, with the micro average (accuracy) being lower. Since the proportions of the different sentiment categories are relatively balanced within the dataset, this difference is most likely due to VADER struggling more with certain sentiment categories, particularly neutral sentiment, which has lower precision than positive and negative sentiment in all experiments with POS filtering. Since micro averages consider the number of examples from each category, poor performance on one category (e.g., neutral) pulls down micro average precision (accuracy) more than macro average precision, which treats all categories equally.

Finally, the trend between sentiment categories (negative, neutral, positive) overall remains consistent across different experiments, though the exact values fluctuate depending on whether lemmatization or part-of-speech filtering is applied. In most experiments, the negative sentiment has high precision but low recall, meaning that when VADER predicts a tweet as negative, it is usually correct, but it often fails to detect all negative tweets. The precision of the neutral sentiment is lower than the recall for experiments that apply POS filtering, while the opposite is observed for those that do not apply it. This means that in the cases where certain parts of speech are filtered, VADER frequently classifies tweets as neutral when many of those tweets should have been classified as positive or negative instead. Likely due to the filtering, there is not sufficient information about the sentiment of those tweets, so VADER defaults to neutral classification. In contrast to neutral sentiment, the precision of the positive sentiment is higher than the recall for experiments that apply POS filtering, while the opposite is observed for those that do not apply it. In the cases without filtering, VADER captures many positive tweets correctly, but in the process, it also misclassifies some neutral or negative tweets as positive.

The F1-score for all sentiment categories also remains relatively stable across experiments, being considerably lower for versions that filter parts of speech compared to those that do not. In the versions without filtering, the F1 score for positive sentiment is highest, followed by the F1-score for negative sentiment, and then the F1-score for neutral sentiment. On the other hand, when POS filtering is applied, the F1 score for neutral sentiment is highest, followed by the F1-score for positive sentiment, and then the F1-score for negative sentiment. Based on these observations, it seems that VADER is slightly better at detecting positivity than negativity.

Does lemmatization help? Based on the fact that almost no difference in results is seen between experiments with and without lemmatization that use the same parts of speech (Experiments 1 vs. 2, 3 vs. 4, 5 vs. 6, and 7 vs. 8,) as described in detail above, we can conclude that lemmatization does not significantly improve sentiment classification. The accuracy and macro average scores remain nearly identical with at most a 2% difference, usually in favor of the non-lemmatized text. This suggests that sentiment classification with VADER is not highly dependent on whether words are in their base form. One possible reason is that VADER's sentiment lexicon already accounts for many common word variations (e.g. "regret", "regrets", "regretful", "regretting"), so reducing words to their lemma does not add much value.

Are all parts of speech equally important for sentiment analysis? No, all parts of speech are not equally important for sentiment analysis. The results show that using only adjectives leads to the best performance, followed by using only verbs, and finally using only nouns yields the

worst classification, as discussed in detail above. This makes sense because adjectives are more directly associated with sentiment (e.g., “amazing”, “annoying”, “disappointing”), and some verbs also express certain emotions (e.g., “appreciate”, “struggle”, “reject”), whereas nouns often carry only a small amount of direct emotional weight.

Overall, VADER exhibits the best performance when the full text of a tweet is used. Removing any part of speech reduces the classification scores. This suggests that while adjectives and verbs seem to be the most important, a combination of all parts of speech contributes to a better sentiment analysis.

1.11 Part II: scikit-learn assignments

1.11.1 [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?

[19]:
```
# imports
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
```

[20]:
```
# data split
airline_tweets = load_files(
    "airlinetweets",
    categories=["neutral", "positive", "negative"],
    encoding="utf-8",
    shuffle=True,
    random_state=0)

tweets = airline_tweets.data
labels = airline_tweets.target
label_names = airline_tweets.target_names
labels = [label_names[label] for label in labels]
```

```
X_train, X_test, y_train, y_test = train_test_split(tweets, labels,␣
 ↪train_size=0.8, random_state=0, shuffle=False)
```

[21]:
```
# training with default settings (TF-IDF representation, min_df=2)
vectorizer = TfidfVectorizer(min_df=2)
X_train_vectorizer = vectorizer.fit_transform(X_train)
X_test_vectorizer = vectorizer.transform(X_test)

# using Multinomial NB because it is good for txt classification
model = MultinomialNB()
model.fit(X_train_vectorizer, y_train)
y_pred = model.predict(X_test_vectorizer)

print("Basic settings (TF-IDF representation, min_df=2)")
print(classification_report(y_test, y_pred))
```

```
Basic settings (TF-IDF representation, min_df=2)
              precision    recall  f1-score   support

    negative       0.79      0.91      0.84       336
     neutral       0.85      0.65      0.74       303
    positive       0.84      0.88      0.86       312

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

[22]:
```
# DIFFERENT SETTINGS 1
# with respect to vectorizing: TF-IDF ('airline_tfidf') vs. Bag of words␣
 ↪representation ('airline_count')

# Bag of words representation ('airline_count')
# for BoW use CountVectorizer
vectorizer_count = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize)
X_train_counter = vectorizer_count.fit_transform(X_train)
X_test_counter = vectorizer_count.transform(X_test)

model_count = MultinomialNB()
model_count.fit(X_train_counter, y_train)
y_pred_count = model_count.predict(X_test_counter)

print("Bag of words representation ('airline_count')")
print(classification_report(y_test, y_pred_count))
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/feature_extraction/text.py:517: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
```

```
  warnings.warn(

Bag of words representation ('airline_count')
              precision    recall  f1-score   support

    negative       0.82      0.90      0.86       336
     neutral       0.86      0.70      0.77       303
    positive       0.83      0.88      0.85       312

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

1.11.2 TF-IDF vs BoW

To compare TF-IDF and BoW results, we analyze the output of the classification_report. First of all, the overall accuracy is slightly better with BoW than with TF-IDF (83% and 82%, respectively). Furthermore, if we compare by category, the results are very similar for the neutral and positive categories (TF-IDF has slightly better precision for positive and BoW for neutral, but the difference is very small).

Neutral category:

- TF-IDF: Precision 0.85, Recall 0.65, F1-score 0.74;
- BoW: Precision 0.86, Recall 0.70, F1-score 0.77

–> BoW has slighly better precision, recall, and F1-score but the difference is not big

Positive category:

- TF-IDF: Precision 0.84, Recall 0.88, F1-score 0.86;
- BoW: Precision 0.83, Recall 0.88, F1-score 0.85

–> TF-IDF has slighly better precision and F1-score but the difference is very small

Negative category:

- TF-IDF: Precision 0.79, Recall 0.91, F1-score 0.84;
- BoW: Precision 0.82, Recall 0.90, F1-score 0.86

–> for the negative category, BoW performs better- it results in higher precision (0.82 vs. 0.79) while maintaing very similar recall (0.91 for TF-IDF and 0.90 for BoW). From this, we can conclude that BoW produces fewer false positives. The overall higher accuracy of BoW can be explained by its better performance on negative tweets.

These differences are due to the specificity of both text vectorization techniques:

1) BoW counts how many times each word appears in the tweet (ignoring grammar, order, meaning). It is a simpler approach and can be effective for capturing common words. However, it does not consider the importance of words, meaning that even frequent but unimportant words (such as “the”) can receive high weight.
2) TF-IDF is similar to BoW, but it adjusts for word importance. Instead of just taking raw word counts, it gives weight to words that are more important in a document. It assigns

weights based on term frequency (TF) and inverse document frequency (IDF), where IDF penalizes common words, reducing their influence in classification.

In the case of our tweets, BoW slightly outperformed TF-IDF because tweets contain strong, frequent words that aid classification. Examples of frequent negative words include “bad”, “worst”, “delayed”, “cancelled”, “late”, “horrible”, and “never”.

[23]:
```
# DIFFERENT SETTINGS 2
# with respect to the frequency threshold (min_df)

# values of document frequency (min_df = 2; min_df = 5; min_df =10)
min_df_values = [2, 5, 10]

# train a model with respect to each frequency threshold
for min_df in min_df_values:
    print(f"\nTraining with TF-IDF, min_df={min_df}")

    vectorizer = TfidfVectorizer(min_df=min_df)
    X_train_vectorizer = vectorizer.fit_transform(X_train)
    X_test_vectorizer = vectorizer.transform(X_test)

    model = MultinomialNB()
    model.fit(X_train_vectorizer, y_train)
    y_pred = model.predict(X_test_vectorizer)

    print("TF-IDF classification report:")
    print(classification_report(y_test, y_pred))

    print(f"\nTraining with BoW, min_df={min_df}")

    vectorizer_bow = CountVectorizer(min_df=min_df, tokenizer=nltk.
 ↪word_tokenize)
    X_train_bow = vectorizer_bow.fit_transform(X_train)
    X_test_bow = vectorizer_bow.transform(X_test)

    model_bow = MultinomialNB()
    model_bow.fit(X_train_bow, y_train)
    y_pred_bow = model_bow.predict(X_test_bow)

    print("BoW classification report:")
    print(classification_report(y_test, y_pred_bow))
```

```
Training with TF-IDF, min_df=2
TF-IDF classification report:
              precision    recall  f1-score   support

    negative       0.79      0.91      0.84       336
```

```
     neutral       0.85      0.65      0.74       303
    positive       0.84      0.88      0.86       312

    accuracy                           0.82       951
   macro avg       0.82      0.82      0.81       951
weighted avg       0.82      0.82      0.82       951


Training with BoW, min_df=2

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/feature_extraction/text.py:517: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(

BoW classification report:
              precision    recall  f1-score   support

    negative       0.82      0.90      0.86       336
     neutral       0.86      0.70      0.77       303
    positive       0.83      0.88      0.85       312

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


Training with TF-IDF, min_df=5
TF-IDF classification report:
              precision    recall  f1-score   support

    negative       0.80      0.88      0.84       336
     neutral       0.81      0.70      0.75       303
    positive       0.83      0.86      0.84       312

    accuracy                           0.81       951
   macro avg       0.82      0.81      0.81       951
weighted avg       0.82      0.81      0.81       951


Training with BoW, min_df=5

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/feature_extraction/text.py:517: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(

BoW classification report:
              precision    recall  f1-score   support
```

```
    negative       0.82      0.89      0.85       336
     neutral       0.85      0.73      0.79       303
    positive       0.84      0.88      0.86       312

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


Training with TF-IDF, min_df=10
TF-IDF classification report:
              precision    recall  f1-score   support

    negative       0.79      0.87      0.83       336
     neutral       0.78      0.69      0.73       303
    positive       0.82      0.82      0.82       312

    accuracy                           0.80       951
   macro avg       0.80      0.79      0.79       951
weighted avg       0.80      0.80      0.80       951


Training with BoW, min_df=10

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/feature_extraction/text.py:517: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(

BoW classification report:
              precision    recall  f1-score   support

    negative       0.81      0.88      0.85       336
     neutral       0.83      0.73      0.78       303
    positive       0.85      0.88      0.86       312

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

1.11.3 Frequency threshold experiment results

Classification report results

1) TF-IDF vs. BoW (min_df=2)

As mentioned in the result analysis in the previous section, the accuracy for BoW is slightly better at 83% compared to TF-IDF's 82%.

Per category:

- Negative tweets: BoW outperforms TF-IDF with higher precision (0.82 vs 0.79) –> BoW has fewer false positives
- Neutral tweets: TF-IDF has slightly lower precision (0.85 vs 0.86) and lower recall (0.65 vs 0.70) but the difference is also small (TF-IDF model is missing more relevant instances)
- Positive tweets: TF-IDF has slightly better precision (0.84 vs 0.83) but the recall is the same (0.88)

As pointed out in the previous comparison, BoW slightly outperforms TF-IDF due to better performance on negative tweets (see the details above).

2) TF-IDF vs. BoW (min_df=5)

The overall accuracy slightly changed. TF-IDF has 81%, while BoW scores 83% –> it is still slightly better for BoW.

Per category:

- Negative tweets: BoW outperforms TF-IDF with higher precision (0.82 vs 0.80) and with higher recall (0.89 vs 0.88)
- Neutral tweets: Results are very close. BoW performs slightly better with precision (0.85 vs 0.81) and also the recall is higher (0.73 vs 0.70)
- Positive tweets: Results are very close. BoW has slightly better recall (0.88 vs 0.86), the precision is slightly better in BoW too (0.84 vs 0.83)

In this case, both TF-IDF and BoW resulted in similar accuracy as before. BoW is again better at handling negative tweets but also this time better at classifying neutral tweets. The performance is closer for the positive categories. Increasing min_df from 2 to 5 does not significantly impact the overall accuracy but slightly improves BoW's performance in neutral tweets.

3) TF-IDF vs. BoW (min_df=10)

BoW achieves 83% accuracy, while TF-IDF results in 80% accuracy. Compared to previous experiments, the accuracy of BoW remains the same, while the accuracy of TF-IDF has decreased by 1%.

Per category:

- Negative tweets: BoW has higher precision (0.81 vs. 0.79) and higher recall (0.88 vs. 0.87)
- Neutral tweets: BoW has slightly better recall (0.73 vs. 0.69), but also has better precision (0.83 vs. 0.78)
- Positive tweets: TF-IDF has slightly lower recall (0.82 vs. 0.88) and slightly lower precision (0.82 vs. 0.85)

In this case, BoW achieves 83% accuracy, while TF-IDF drops to 80% accuracy. Compared to min_df=2 and min_df=5, BoW remains stable but TF-IDF's accuracy has decreased. BoW continues to outperform TF-IDF in negative tweets, maintaining higher precision and recall . For neutral tweets, BoW improves in precision, while TF-IDF's recall drops further, indicating that TF-IDF struggles more with capturing neutral tweets as min_df increases. In positive tweets, BoW achieves higher recall, while TF-IDF retains slightly better precision.

Impact of increasing min_df Increasing the min_df from 2 to 10 removes rare words, resulting in a smaller vocabulary size. BoW is more affected by this change because it relies on raw word counts, whereas TF-IDF adjusts more easily by assigning weights to important terms. As a result, the accuracy of both models remains similar despite the reduction in vocabulary size. In conclusion, eliminating rare words has minimal impact on classification performance, suggesting that the classification of these tweets depends more on frequently occurring words rather than rare ones. The sentiment of airline tweets is primarily influenced by common terms, and even with min_df=10, the model maintains consistent accuracy. This indicates that the key words driving sentiment classification are shared across many tweets. Even with min_df = 10, the model maintains a similar accuracy –> `The key words influencing sentiment classification are common across many tweets`.

1.11.4 Conclusion

In conclusion, we saw different text vectorization techniques and their impact on classification performance. The experiments compared TF-IDF technique and Bag of Words technique, which demonstrated that while both methods yield similar results, BoW slightly outperformed TF-IDF in negative tweet classification. This is likely because negative sentiment is often expressed by using strong, frequently occuring words. For this, BoW is effective and captures frequent words without weighting adjustments. Moreover, the impact of increasing the frequency treshold (min_df) was explored. Removing more rare words (increasing min_df from 2 to 5 and then to 10) had minimal impact on classification performance. Therefore, sentiment in airline tweets is mostly determined by frequently used words. Overall, depending on the dataset and task, it is important to choose the right text representation–TF-IDF is useful for taking into account word importance; BoW is strong for fields where sentiment words occur frequently.

1.11.5 [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?

[24]:
```
def important_features_per_class(vectorizer,classifier,n=80):
    class_labels = classifier.classes_
    feature_names =vectorizer.get_feature_names_out() # had to change this line␣
↪of code to _out(), because in newer version of scikit-learn it was removed
    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)

# x_train = [" ".join(word_tokenize(text)) for text in x_train]  # we can use␣
 ↪this if we want to remove punctuation and maybe improve our model
# x_test = [" ".join(word_tokenize(text)) for text in x_test]

# Reuse train-test split from previous question
vectorizer = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize) # remove␣
 ↪tokenizer argument if we want to remove punctuation, and uncomment the lines␣
 ↪above
train_counts = vectorizer.fit_transform(X_train)
test_counts = vectorizer.transform(X_test)


classifier = MultinomialNB()
classifier.fit(train_counts, y_train)


important_features_per_class(vectorizer, classifier)
```

```
Important words in negative documents
negative 1518.0 @
negative 1390.0 united
negative 1230.0 .
negative 779.0 to
negative 590.0 i
negative 527.0 the
negative 448.0 a
negative 426.0 ``
negative 405.0 flight
negative 371.0 ?
negative 368.0 !
negative 365.0 you
negative 356.0 and
negative 337.0 for
```

```
negative 327.0 my
negative 325.0 on
negative 324.0 #
negative 297.0 is
negative 280.0 in
negative 228.0 n't
negative 211.0 of
negative 208.0 your
negative 198.0 that
negative 190.0 it
negative 176.0 not
negative 173.0 me
negative 166.0 have
negative 164.0 was
negative 163.0 ''
negative 159.0 with
negative 151.0 no
negative 145.0 at
negative 126.0 's
negative 124.0 this
negative 112.0 service
negative 109.0 from
negative 109.0 be
negative 108.0 do
negative 107.0 virginamerica
negative 103.0 we
negative 100.0 cancelled
negative 99.0 an
negative 98.0 now
negative 98.0 get
negative 94.0 :
negative 91.0 why
negative 90.0 but
negative 89.0 delayed
negative 89.0 customer
negative 88.0 plane
negative 88.0 are
negative 85.0 time
negative 85.0 just
negative 84.0 they
negative 83.0 bag
negative 82.0 been
negative 77.0 ;
negative 77.0 'm
negative 75.0 …
negative 74.0 hours
negative 73.0 so
negative 73.0 -
```

```
negative 72.0 &
negative 67.0 what
negative 67.0 gate
negative 67.0 can
negative 64.0 will
negative 62.0 would
negative 62.0 still
negative 62.0 amp
negative 61.0 when
negative 61.0 late
negative 61.0 how
negative 60.0 our
negative 60.0 has
negative 60.0 did
negative 60.0 about
negative 59.0 http
negative 59.0 hour
negative 59.0 again
----------------------------------------
Important words in neutral documents
neutral 1397.0 @
neutral 587.0 to
neutral 525.0 ?
neutral 525.0 .
neutral 455.0 i
neutral 311.0 the
neutral 306.0 jetblue
neutral 291.0 :
neutral 275.0 southwestair
neutral 269.0 a
neutral 262.0 you
neutral 249.0 united
neutral 242.0 on
neutral 238.0 ``
neutral 235.0 #
neutral 228.0 flight
neutral 202.0 for
neutral 196.0 my
neutral 195.0 americanair
neutral 183.0 http
neutral 182.0 in
neutral 171.0 !
neutral 167.0 is
neutral 160.0 usairways
neutral 157.0 can
neutral 142.0 and
neutral 136.0 's
neutral 130.0 from
```

```
neutral 125.0 of
neutral 123.0 it
neutral 116.0 do
neutral 113.0 me
neutral 112.0 have
neutral 91.0 with
neutral 76.0 this
neutral 76.0 any
neutral 75.0 that
neutral 75.0 at
neutral 73.0 get
neutral 73.0 be
neutral 72.0 what
neutral 72.0 -
neutral 70.0 please
neutral 70.0 if
neutral 70.0 are
neutral 69.0 will
neutral 68.0 virginamerica
neutral 67.0 flights
neutral 64.0 )
neutral 62.0 we
neutral 62.0 help
neutral 60.0 our
neutral 59.0 need
neutral 58.0 there
neutral 58.0 ''
neutral 56.0 (
neutral 55.0 n't
neutral 55.0 just
neutral 54.0 your
neutral 51.0 ;
neutral 49.0 how
neutral 46.0 when
neutral 46.0 or
neutral 44.0 out
neutral 43.0 dm
neutral 43.0 &
neutral 42.0 us
neutral 42.0 so
neutral 42.0 not
neutral 41.0 tomorrow
neutral 40.0 an
neutral 40.0 …
neutral 38.0 "
neutral 38.0 now
neutral 38.0 know
neutral 38.0 fleet
```

```
neutral 38.0 fleek
neutral 37.0 would
neutral 36.0 "
neutral 36.0 thanks
----------------------------------------
Important words in positive documents
positive 1328.0 @
positive 1008.0 !
positive 748.0 .
positive 456.0 you
positive 426.0 to
positive 423.0 the
positive 343.0 for
positive 317.0 #
positive 306.0 southwestair
positive 295.0 i
positive 284.0 jetblue
positive 277.0 thanks
positive 259.0 united
positive 251.0 thank
positive 238.0 ``
positive 215.0 a
positive 195.0 and
positive 180.0 flight
positive 168.0 americanair
positive 167.0 :
positive 157.0 my
positive 156.0 on
positive 132.0 in
positive 130.0 great
positive 129.0 usairways
positive 113.0 it
positive 113.0 is
positive 109.0 your
positive 109.0 of
positive 107.0 me
positive 105.0 so
positive 98.0 with
positive 90.0 was
positive 90.0 )
positive 88.0 service
positive 88.0 at
positive 83.0 this
positive 82.0 virginamerica
positive 70.0 http
positive 67.0 just
positive 67.0 best
positive 67.0 are
```

```
positive 66.0 love
positive 63.0 much
positive 63.0 from
positive 63.0 ;
positive 62.0 that
positive 61.0 's
positive 60.0 guys
positive 59.0 customer
positive 58.0 have
positive 55.0 awesome
positive 52.0 -
positive 50.0 we
positive 50.0 airline
positive 49.0 good
positive 49.0 be
positive 47.0 time
positive 45.0 up
positive 45.0 out
positive 45.0 all
positive 44.0 amazing
positive 42.0 &
positive 40.0 us
positive 40.0 got
positive 38.0 today
positive 38.0 crew
positive 36.0 fly
positive 35.0 will
positive 35.0 they
positive 35.0 our
positive 35.0 not
positive 35.0 n't
positive 35.0 flying
positive 35.0 amp
positive 34.0 do
positive 34.0 …
positive 33.0 now
positive 33.0 made
positive 31.0 very
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/feature_extraction/text.py:517: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

1.11.6 Expected Features

Negative Class Expected Words: `"delayed"`, `"cancelled"`, `"late"`

Why? People usually express frustration on Twitter when their flight is delayed, canceled, or when

they experience poor service.

Neutral Class Expected Words: `"flight"`, `"gate"`

Why? Neutral tweets are often just factual, mentioning general travel related words without expressing strong emotions, this also makes it harder to predict neutral words.

Positive Class Expected Words: `"best"`, `"great"`, `"amazing"`

Why? Happy customers tend to express gratitude and enthusiasm when they receive good service.

1.11.7 Unexpected Features

Some unexpected features appeared in the output, such as punctuation and symbols like `"@"`, `"."`, `"!"`, `"?"`, and `"#"`, which are structural elements of tweets but do not carry sentiment. Additionally, common stopwords such as `"the"`, `"a"`, `"is"`, `"with"`, and `"that"` were surprisingly ranked high despite being function words that frequently appear in all tweets without having meaningful sentiment. Furthermore, numbers and generic words like `"hours"`and `"http"` were unexpected. While `"hour"` and `"hours"` may sometimes indicate delays, numerical values and URLs generally do not convey sentiment on their own.

1.11.8 Words That Should Be Kept Or Removed

To improve the classification model, we should remove words that don't add sentiment meaning and keep those that do.

Words to Remove that have No Sentiment Meaning

- **Punctuation**: `"."`, `"!"`, `"?"`, `"..."`, `"@"`, `"#"`
- **Stopwords**: `"the"`, `"to"`, `"a"`, `"is"`, `"in"`, `"with"`, `"that"`, `"be"`
- **Mentions & URLs**: `"@"`, `"http"`
- **Symbols**: `"&"`, `"-"`, `":"`, `"'"`

Words to Keep that have Strong Sentiment Meaning

- **Negative Sentiment:** `"cancelled"`, `"delayed"`
- **Neutral Words:** `"service"`,`"flight"`, `"gate"`
- **Positive Sentiment:** `"best"`, `"great"`,`"good"`, `"awesome"`, `"amazing"`

1.11.9 Model Improvement

To make the classification more accurate, we can apply the following techniques:

1. **Remove Stopwords**: Use `stop_words="english"` in `CountVectorizer` to filter out common words that don't add meaning.

2. **Exclude Punctuation**
3. **Apply Stemming or Lemmatization**: Convert words to their root form (e.g., `"delayed"` → `"delay"`, `"cancelled"` → `"cancel"`).

By applying these refinements, we can improve our Naive Bayes model's ability to classify tweets more accurately and meaningfully.

1.11.10 [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.

1.11.11 [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.

1.12 End of this notebook