Basic introduction to Ski-Kit Learn: Analyzing Twitter tweets

A. Connecting to Twitter API and getting Twitter credentials:

1. Create a Twitter account on https://twitter.com.
2. Go to https://developer.twitter.com/en/apps and log in with your Twitter account.
3. Click “Create an App” and fill in details of the application (See Minerva for a more detailed explanation).
4. The application’s tokens and keys are available in the “Keys and Access Tokens” tab.

To gather Tweets, we will use the Tweepy library. To install Tweepy, open the command prompt as an administrator and run the following pip command: **pip install tweepy**.

```
!pip install tweepy
```

```
Requirement already satisfied: tweepy in /usr/local/lib/python3.6/dist-packages (3.6.0)
Requirement already satisfied: PySocks>=1.5.7 in /usr/local/lib/python3.6/dist-packages (from tweepy) (1.7.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tweepy) (1.12.0)
Requirement already satisfied: requests>=2.11.1 in /usr/local/lib/python3.6/dist-packages (from tweepy) (2.21.0)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from tweepy) (1.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests>=2.11.1->tweepy) (201
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.11.1->tweepy) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests>=2.11.1->tweepy) (
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.11.1->tweepy) (
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from requests-oauthlib>=0.7.0->tweepy)
```

```
import tweepy

# Consumer keys and access tokens, used for OAuth
consumer_key = 'removed keys for security reasons'
consumer_secret = 'removed keys for security reasons'
access_token = 'removed keys for security reasons'
access_token_secret = 'removed keys for security reasons'
```

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# Calling the api
api = tweepy.API(auth)
```

Now we can interact with the API. For this lab, we will deal with gathering and analyzing tweets from certain Twitter accounts. Statuses posted by a specified user can be collected with a GET statuses/user_timeline request. This boils down to a Tweepy user_timeline method. Have a look at the syntax and possibilities. Let's take a look at the latest 5 Tweets by @Bart_DeWever (a Belgian politician):

```
handle = '@Bart_DeWever'
number_of_tweets = 5
tweets = api.user_timeline(screen_name=handle, count=number_of_tweets)

print(type(tweets))
print(type(tweets[0]))
```

```
<class 'tweepy.models.ResultSet'>
<class 'tweepy.models.Status'>
```

Note that tweets is a set of Tweepy **'Status' objects**. Take a look here to see all the attributes and subattributes of the object. For example, we can get the text and author's name of the latest tweet by using the following commands:

```
print(tweets[0].text)
print(tweets[1].author.name)
```

```
In Antwerpen nam onze politie de eerste tegen woekerprijs aangeboden mondmaskers in beslag en bracht die naar een z… https://t.c
Bart De Wever
```

Task: Find the author's location. How many retweets did his latest post have? Check his Twitter page to verify your answer.

```
location = tweets[0].author.location
```

```
location = tweets[0].author.location
retweets = tweets[0].retweet_count

print('The author\'s location is '+location)
print('His latest post has '+repr(retweets)+' retweets')
```

```
The author's location is Antwerpen
His latest post has 218 retweets
```

Retweets can be excluded by adding the parameter '**include_rts = False**' to the user_timeline command. Note however that the count parameter still counts retweets, and hence usually len(tweets) < number_of_tweets.

Task: Out of his latest 10 posts, print all his original Tweets (= a Tweet that is not a retweet).

```
number_of_tweets = 10
tweets = api.user_timeline(screen_name=handle, count=number_of_tweets, include_rts=False)
for i, tweet in enumerate(tweets):
  print(f'{i}\t: {tweet.text}')
```

```
0       : In Antwerpen nam onze politie de eerste tegen woekerprijs aangeboden mondmaskers in beslag en bracht die naar een z… h
1       : Veel steun aan wie dezer dagen werkt in de zorg of distributie, in winkels of supermarkten, en ook aan hulpdiensten… h
2       : Dit zijn de belangrijkste standpunten die wij verdedigen aan de onderhandelingstafel: 👇
https://t.co/jl1jzH6tQv
3       : We nemen akte van het PS-dictaat. Tegen de wil van de Vlaamse kiezer in moet er voor hen een zo links mogelijke reg… h
4       : In de ranking van @FT schuift Antwerpen naar de 2de plaats in de lijst van de beste grote steden om te investeren.… ht
5       : Het unitarisme dat @GLBouchez voorstelt werpt ons terug naar de 19de eeuw. Iedereen heeft recht op een eigen opinie… h
6       : Nooit mag terechte onvrede over het Europees migratiebeleid zich omzetten in wrok naar mensen. In de Vlaamse natie… ht
7       : Met een dierbare herinnering aan ons lang gesprek, anderhalf jaar geleden, bied ik mijn deelneming aan voor het ove… h
```

B. SciKit Learn and Machine Learning in practice: an introduction

The **problem** is as follows: We want to analyze the latest tweets by the leaders of the Belgian political parties. Based on their chosen language, the topics they discuss, the hashtags etc., we would like to see if we can predict which Tweet belongs to who. **Text classification** is a common

Machine Learning task, and we expect that our problem should have a somewhat decent solution. Indeed, we can imagine a right-wing politician and a left-wing/green politican to tweet differently about ongoing events or political topics.

First, install the following packages: **pip install sklearn**, **pip install nltk** and **pip install numpy**

1. Bag-of-words Model

In our model, each datapoint consists of the text of a Tweet and an associated label (= the Author's name). However, in order to run some generic black-box classifier, we need to convert this data to numeric values. The **labels** can easily be converted by just assigning an integer value to each Author. The text can be converted to numeric **features** by using the well-known Bag-of-Words model. Basically, we keep track of all the unique words in all the Tweets, and for each unique word, we count how many times it occurs in each Tweet.

In scikit-learn, this is done by a **vectorizer**, see Section 4.2.3 here. Let's apply this to our corpus of Tweets.

```
!pip install sklearn nltk numpy
```

```
Requirement already satisfied: sklearn in /usr/local/lib/python3.6/dist-packages (0.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.6/dist-packages (3.2.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (1.18.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from sklearn) (0.22.2.post1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from nltk) (1.12.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sklearn) (0.14.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sklearn) (1.4.1)
```

```
# Importing packages.
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import re
import numpy as np
import nltk

# Twitter handles of the politicians.
handles = ['@Bart_DeWever', '@conner_rousseau', '@RuttenGwendolyn', '@MeyremAlmaci', '@tomvangrieken']
number_of_tweets = 200
corpus = [] # a list of the text in all the tweets from the handles.
labels = [] # a list with the numeric label of each tweet (1='@Bart_DeWever', 2='@vbeke', etc.)
```

```
labels = [] # a list with the numeric label of each tweet (1= @BartDeWever , 2= @wbeke , etc.)
teller = 1

# Getting the Tweets from the handles, excluding Retweets and getting full_text from the tweets.
for name in handles:
    tweets = api.user_timeline(screen_name=name, count=number_of_tweets, include_rts=False, tweet_mode="extended")
    corpus = corpus + [t.full_text for t in tweets] # getting the full text tweets.
    labels = labels + [teller for i in range(0,len(tweets))]
    teller +=1

print(corpus[0:5]) # Printing the first tweets in the corpus.
```

```
['In Antwerpen nam onze politie de eerste tegen woekerprijs aangeboden mondmaskers in beslag en bracht die naar een ziekenhuis.
```

```
# Now we convert this corpus to a numeric matrix X:
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

print(vectorizer.get_feature_names()) #__Checking the words that are vectorized, and thus part of the model.
print(X.shape) #__ Checking the dimension of the matrix.
```

```
['000', '03', '04', '0nl9mfedfi', '10', '100', '1000', '1000den', '1099ehhglt', '10n', '11november', '12', '125', '13', '135', '
(595, 4300)
```

2. Applying generic Machine Learning classifiers

Given our datapoints (the rows in X) and the labels, we are ready to run some basic classifiers. First, in order to evaluate our models properly, we need to split our data into **training data** and **test data**.

TASK: use the 'train_test_split' function to split the data automatically. Make sure this is done in a **random** manner.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, labels, shuffle=True, random_state=2020)
```

Now can train our models on the train data, and evaluate them on the test data. We will start by applying a Decision Tree Classifier and a Naive Bayes Classifier. To check **accuracy**, we simply count the number of times the classifier correctly predicted the label on the test data.

```
from sklearn import tree
from sklearn.naive_bayes import MultinomialNB

# Decision Tree Classifier
model_tree = tree.DecisionTreeClassifier(random_state=1) # Setting random_state = 1, to remove randomness in fitting.
model_tree = model_tree.fit(X_train, y_train)
y_predict = model_tree.predict(X_test)
accuracy_tree = np.sum(y_predict == y_test)
print('Decision Tree accuracy:'+'\t'+repr(accuracy_tree))

# Naive Bayes Classifier:
model_bayes = MultinomialNB()
model_bayes = model_bayes.fit(X_train, y_train)
y_predict = model_bayes.predict(X_test)
accuracy_bayes = np.sum(y_predict == y_test)
print('Naive Bayes accuracy:'+'\t'+repr(accuracy_bayes))
```

```
Decision Tree accuracy: 63
Naive Bayes accuracy:   77
```

Now that we have a model, we will try to improve it. The first thing we can try, is to **redefine our corpus**. There are two problems with our current corpus:

a. There are many tweets with **urls** in them, which is clearly not informative in the classification.

b. We have not removed the **stopwords** from the corpus (= very common words without much meaning).

TASK: Make a new corpus, where the urls + Dutch stopwords are removed. Test the difference in performance of the classifiers. Explain why the performance, at first sight, might not have improved.

```
# Download the Dutch stop words from the NLTK repository.
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
from nltk.corpus import stopwords

## Removing the links from the tweets (starting with https:// until a space)
# HINT: iterate over the corpus, and use re.sub function to replace 'https' pieces with an empty string ''.
corpus_no_url = [re.sub('https://\S* ', '', tweet) for tweet in corpus]

## Removing stopwords from Dutch language
# HINT: Split the words in a tweet, only keep words that are not in stopWords, then join the seperate words into a string.
stopWords = set(stopwords.words('dutch'))
corpus_no_stops_no_url = [' '.join([token for token in tweet.split(' ') if not token in stopWords]) for tweet in corpus_no_url]

print(corpus_no_stops_no_url[0:5])
```

```
['In Antwerpen nam onze politie eerste woekerprijs aangeboden mondmaskers beslag bracht ziekenhuis. \n👉 Ziet zoiets? \n📞 Meld
```

```
# Performance testing with the new corpus
# Now we convert this corpus to a numeric matrix X:
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus_no_stops_no_url)

X_train, X_test, y_train, y_test = train_test_split(X, labels, shuffle=True, random_state=2020)

# Decision Tree Classifier
model_tree = tree.DecisionTreeClassifier(random_state=1) # Setting random_state = 1, to remove randomness in fitting.
model_tree = model_tree.fit(X_train, y_train)
y_predict = model_tree.predict(X_test)
accuracy_tree = np.sum(y_predict == y_test)
print('Decision Tree accuracy:'+'\t'+repr(accuracy_tree))

# Naive Bayes Classifier:
model_bayes = MultinomialNB()
```

```
model_bayes = model_bayes.fit(X_train, y_train)
y_predict = model_bayes.predict(X_test)
accuracy_bayes = np.sum(y_predict == y_test)
print('Naive Bayes accuracy:'+'\t'+repr(accuracy_bayes))
```

```
Decision Tree accuracy: 71
Naive Bayes accuracy:   86
```

Give a couple of reasons why the accuracy might not be higher than with the original corpus:

The accuracy of both the Decision Tree and the Naive Bayes classifier went up. For the Decision Tree classifier, the accuracy increased from 63 to 71. For the Naive Bayes classifier, the accuracy increased from 77 to 86.

By removing the URLs and stopwords, a lot of clutter is removed from the dataset. These did not contribute towards a correct classification and even 'distracted' the classifier from important information. Now the classifiers can focus on words or word groups that really matter.