Nachvollziehbare Schritte
Trip Advisor Hotel Reviews

Sanja Srdanovic, 18.02.2022

1. Kurze Darstellung des Problembereichs / Aufriss des Themas

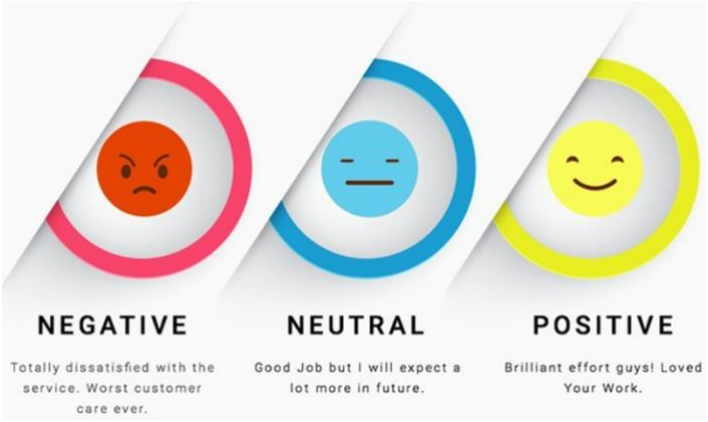
1.1 Inhaltlich

Kern der Untersuchung: Creating an Ensemble (consisting of the following machine learning classifiers: Decision Tree, Random Forest and SVC) for the *Trip Advisor Hotel Reviews* from *Kaggel*
Grobziele der Arbeit: Inspect the dataset and do the text pre-processing (NLP) to prepare the reviews for the analysis; create an ensemble consisting of a Random Forest, Decision Tree and SVC, which is trained with the hotel reviews at Trip Advisor.

1.2 Begründung des Themas

Darstellung der Relevanz des Themas?

Sentiment analysis is aimed at determining the general emotional state of a text and can focus on the polarity of a text (positive, negative). If the precision of the mood is important, the categories can be further elaborated (very positive, positive, neutral, negative, and very negative).
Companies or brands want to know what people are talking about their products or services. They also want to identify their customers' expectations, so that they can tailor products and services to their customers' needs. And they can do that with the help of Sentiment analysis, which is becoming more important every day.



Quelle:https://www.kdnuggets.com/2018/03/5-things-sentiment-analysis-classification.html

Darstellung eines persönlichen Erkenntnisinteresses.

This dataset was particularly interesting to me because it deals with the analysis of reviews, i.e., textual data and it involves some linguistic expertise. I would like to learn more about NLP and text pre-processing, and how to apply it in Data Science, and therefore I chose this dataset. Sentiment analysis has been more popular since companies want to get feedback about their products and services, or about their advertisements, and the analysis of customers' comments and reviews could help them improve the quality of their services and brands. I will create an ensemble of different learning classifiers to investigate which of them would best predict and classify the ratings.

2. Nachvollziehbare Schritte

2.1 Der Stand der Forschung / Auswertung der vorhandenen Literatur / Tutorials ...

There are various approaches to Sentiment Analysis. For my project, I created an ensemble consisting of a Decision Tree, Random Forest, and Support Vector Classifier (SVC).

Lösungswege strukturieren!

- Loading and inspecting the data
- Label encoding (3 categories – positive, neutral, and negative)
- Down-sampling
- Text pre-processing
- Tokenization and Padding
- Splitting the data into train and test
- Defining learning classifiers
- Voting Classifier (hard, soft) to see which model performs the best
- Visualisation of the models' performance (confusion matrix)

2.2 Fragestellung

Can we predict the ratings using different models with good accuracy and can the ensemble improve the predictions?

2.3 Methode

Loading the packages and libraries

First, various modules are imported: Pandas and NumPy for working with dataframes and arrays, Seaborn and Matplotlib for plotting the count plot and confusion matrix, SpaCy, RegEx and NLTK for word processing and TensorFlow and Keras for tokenization and padding.

```
# import packages and libraries

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
# NLP
import re
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')
stopwords = nltk.corpus.stopwords.words("english")
import spacy
nlp = spacy.load('en_core_web_sm')

# Scikit-Learn:
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import VotingClassifier

# Keras and Tensorflow
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Loading and inspecting the dataset

```
# load dataset from csv
df = pd.read_csv("/Users/sanjasrdanovic/Desktop/Data Science/Machine Learning/Projekt ML/tripadvisor_hotel_reviews.csv")

# data info
print(df.head())
print(df.info()) # 20491 entries 2 columns

print(f"Theres {df.shape[0]} reviews in this dataset.")
```

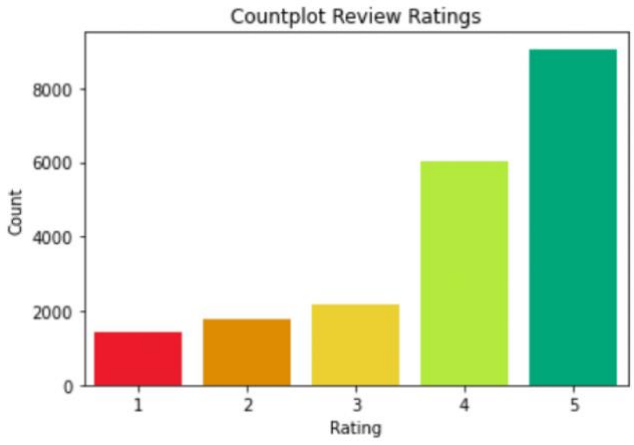
I loaded the dataset 'tripadvisor_hotel_reviews.csv' from CSV. By looking at the *data.head()* or *data.info()*, it can be observed that there are 20491 entries and 2 columns, namely *Review* (textual data) and *Rating* (from 1 to 5).

First visualisations

First, I wanted to have a look at visualisations to get some ideas about the distribution of ratings, so I plotted a count plot with Seaborn. I chose a colour palette so that positive reviews are green (5 dark green, 4 light green), neutral is yellow and negative reviews are orange (2) and red (1).

```
# Visualisation - count of ratings
palette = ["#f31c1c", "#ed9517", "#ffe23b", "#c7ff46", "#17b882"]
sns.set_palette(palette)
sns.countplot(x='Rating', data = df)
plt.xlabel('Rating')
plt.ylabel('Count')
plt.title('Countplot Review Ratings')
plt.show()
plt.savefig("Countplot Review Ratings.png") # save figure
```

By creating a count plot, we can observe that there are more positive ratings than negative ones. There are most 5* ratings and the least 1* ratings.



Let's inspect the data with *data.describe()*, and check if there are some missing values with *data.isnull().sum()* and *data.isna().sum()*.

```
# Data Inspection

print(data.isnull().sum())
print(data.isna().sum())
print(data.describe())
```

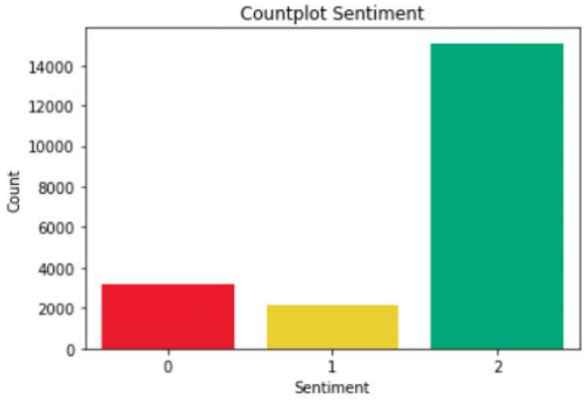
There are no zero, i.e., no missing values in the dataset.

Labelling ratings (3 categories – positive, neutral and negative)

For label encoding, I decided to treat ratings of 4 and 5 as positive, 3 as neutral, and 1 and 2 as negative. Therefore, I have 3 categories: 0 - negative, 1 - neutral and 2 - positive.

```
# Encoding

# define label encode
def label_encode(x):
    if x == 1 or x == 2:
        return 0
    if x == 3:
        return 1
    if x == 5 or x == 4:
        return 2

# define label to name
def label2name(x):
    if x == 0:
        return "Negative"
    if x == 1:
        return "Neutral"
    if x == 2:
        return "Positive"

# encode label and mapping label name, add columns to the data frame
df["label"] = df["Rating"].apply(lambda x: label_encode(x))
df["label_name"] = df["label"].apply(lambda x: label2name(x))
```

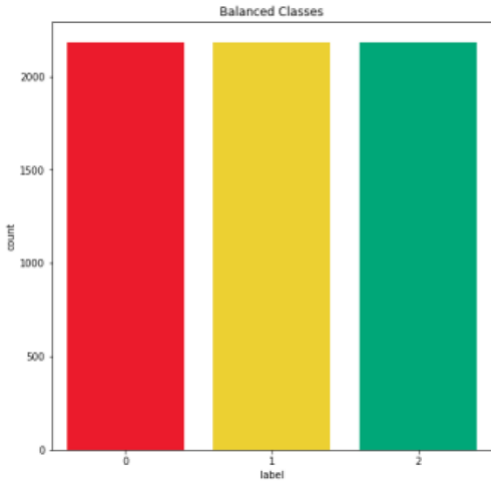
With *print(df['label'].value_counts())*, we get the overview of the distribution of ratings, namely there are: 15093 positive reviews (2), 2184 neutral reviews (1) and 3214 negative ones (0). See the graph below.



Down-sampling

The dataset is severely imbalanced, given that the majority of ratings are positive. The distribution variations (percentage of each class) of a dataset are also important, as the model will tend to overfit to class with a higher percentage because it will get high accuracy there. Therefore, I decided to overcome this issue with data resampling – down-sampling.

```
# Downsampling

# Shuffle the Dataset.
shuffled_df = df.sample(frac=1,random_state=4)
sample_size = 6552
class_0 = df[df['label'] == 0]
class_1 = df[df['label'] == 1]
class_2 = df[df['label'] == 2]

sample_class = int(sample_size /3)
df = pd.concat([class_0.sample(sample_class),
                class_1.sample(sample_class),
                class_2.sample(sample_class)] , axis =0)
print('class_0:', class_0.shape)
print('class_1:', class_1.shape)
print('class_2:', class_1.shape)

#plot the dataset after the undersampling
plt.figure(figsize=(8, 8))
sns.countplot('label', data=df)
plt.title('Balanced Classes')
plt.show()
```



Text pre-processing

In order to prepare the data for the analysis, I applied some standard procedures for pre-processing of the textual data: cleaning special characters and removing punctuation, cleaning digits, removing contractions, and removing stopwords. Additionally, I also did the lemmatization, to get word lemmas.

```
# PREPROCESSING

# Cleaning Special Characters and Removing Punctuations:
def clean_text(review):
    pattern = r'[^a-zA-z0-9\s]'
    review = re.sub(pattern, '', review)
    return review

# Cleaning digits
def clean_numbers(review):
    review = ''.join([i for i in review if not i.isdigit()])
    return review

# Removing Contractions
contraction_dict = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because",

def _get_contractions(contraction_dict):
    contraction_re = re.compile('(%s)' % '|'.join(contraction_dict.keys()))
    return contraction_dict, contraction_re

contractions, contractions_re = _get_contractions(contraction_dict)

def replace_contractions(review):
    def replace(match):
        return contractions[match.group(0)]
    return contractions_re.sub(replace, review)

df["Review"] = df["Review"].apply(clean_text).apply(clean_numbers).apply(replace_contractions)

# Remove stopwords
df["Review"] = [word for word in df["Review"] if word not in stopwords]

# Lemmatisierung
df["Review"] = df["Review"].apply(lambda row: " ".join([w.lemma_ for w in nlp(row)]))
```

Tokenization and Padding

Next step, we transform the text by doing **tokenization**. This class allows vectorizing a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf.

```
review = df["Review"].copy() # Use a copy of the clean reviews
token = Tokenizer() # Initialize the tokenizer
token.fit_on_texts(review) # Fit the tokenizer to the reviews
texts = token.texts_to_sequences(review) # Convert the reviews into sequences for keras to use
```

Pad Sequence is used in NLP because models expect that each sequence is of the same length (same number of words/tokens). This function transforms a list of sequences (integers) into a 2D Numpy array. *padding=" post":* add the zeros at the end of the sequence to make the samples the same size.

```
# Padding

# Print an example sequence to make sure everything is working
print("Into a Sequence: ")
print(texts[26])

from tensorflow.keras.preprocessing.sequence import pad_sequences
texts = pad_sequences(texts, padding='post') # Pad the sequences to make them similar lengths

# Print an example padded sequence to make sure everything is working
print("After Padding: ")
print(texts[26])
```

Next, I divided the data into the input-training (X) and output values-target (y), the input being the cleaned tokenized textual reviews, and the output is the sentiment label (0,1, 2).

```
X = texts # Input values.
y = df['label']   # Output values
```

Then, I split the data into train and test data: 80% for the training data that would be used for the learning phase (16392 entries), and the rest 20% for the test data for validation (4099 entries).

```
# split data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,  test_size=0.2, random_state=42)

print('X_train: ' + str(X_train.shape))
print('y_train: ' + str(y_train.shape))
print('X_test: ' + str(X_test.shape))
print('y_test: ' + str(y_test.shape))

print(type(X_test))
```

Defining learning classifiers

Given that we are dealing with the classification problem, the machine learning classifiers I used for the project are Decision Tree (dt_clf), Random Forest (rnd_clf) and Support Vector Classification (svm_clf).

Decision Tree is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree as a predictive model to go from observations about an item to conclusions about the item's target value. Tree models where the target variable can take a discrete set of values are called classification trees.

Random Forest builds decision trees on different samples and takes their majority vote for classification.

Support Vector Classifier is capable of performing binary and multi-class classification on a dataset. The objective is to fit the data, returning a "best fit" hyperplane that divides or categorizes the data.

```
# Learning Classifiers

dt_clf = DecisionTreeClassifier(random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", random_state=42,probability=True)
```

Voting Classifier

The goal of ensemble methods is to combine the predictions of multiple base estimates (estimators) generated with a particular learning algorithm to improve generalizability/robustness over a single estimator. The idea behind the **Voting Classifier** is to combine conceptually different machine learning classifiers and use majority voting (**hard voting**) or the average predicted probabilities (**soft voting**) to predict the class labels. Such a classifier can be useful for a set of equally well-performing models to compensate for their individual weaknesses.

```
# hard voting

voting = VotingClassifier(
            estimators=[('dt', dt_clf),
                        ('rf', rnd_clf),
                        ('svc', svm_clf)],
            voting='hard',
            )

voting.fit(X_train,y_train)

for clf, label in zip([dt_clf, rnd_clf, svm_clf, voting], ['Decision Tree', 'Random Forest', 'SVC', 'Ensemble']):
    scores = cross_val_score(clf, X_train, y_train, scoring='accuracy', cv=5, error_score="raise")
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
    # will compute the individual accuracy of each model
```

```
# soft voting

voting = VotingClassifier(
             estimators=[('dt', dt_clf),
                         ('rf', rnd_clf),
                         ('svc', svm_clf)],
             voting='soft',
             flatten_transform=False)

dt_clf = dt_clf.fit(X_train, y_train)
rnd_clf = rnd_clf.fit(X_train, y_train)
svm_clf = svm_clf .fit(X_train, y_train)

ensemble= voting.fit(X_train, y_train)

for clf in (dt_clf, rnd_clf, svm_clf, voting):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

After executing hard voting and soft voting, we can see that they provide similar outputs. Namely, the accuracy of the Decision Tree is the worst – 50.41%, then comes SVC – 54.61%, and the best was Random Forest with 68.5 %. The accuracy of *Hard voting* is 62% because it uses majority voting, and *Soft voting* is 52.33% because it calculates the argmax of the sum of the predicted probabilities.

Hard voting

```
Accuracy: 0.50 (+/- 0.01) [Decision Tree]
Accuracy: 0.65 (+/- 0.03) [Random Forest]
Accuracy: 0.53 (+/- 0.01) [SVC]
Accuracy: 0.62 (+/- 0.02) [Ensemble]
```

Soft voting

```
DecisionTreeClassifier 0.5041952707856598
RandomForestClassifier 0.6849733028222731
SVC 0.5461479786422578
VotingClassifier 0.5232646834477498
```

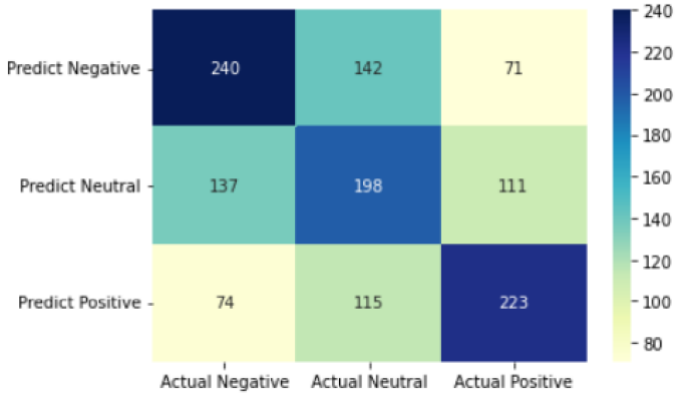
We can also check the predictions and accuracy of each classifier with the *Confusion Matrix* from *Scikit-Learn*, where we compare actual data and data predicted by the model.

Confusion matrix and Classification report voting

Decision Tree

```
# Classification reports and confusion matrix

# Discision Tree

# compare y_test and y_pred
y_pred = dt_clf.predict(X_test)
y_pred_dt = np.around(y_pred)

# Confusion Matrix

cm_dt = confusion_matrix(y_test, y_pred_dt)
print(cm_dt)

# Classification Report

classrep_dt = metrics.classification_report(y_test,y_pred_dt)
print(classrep_dt)
```

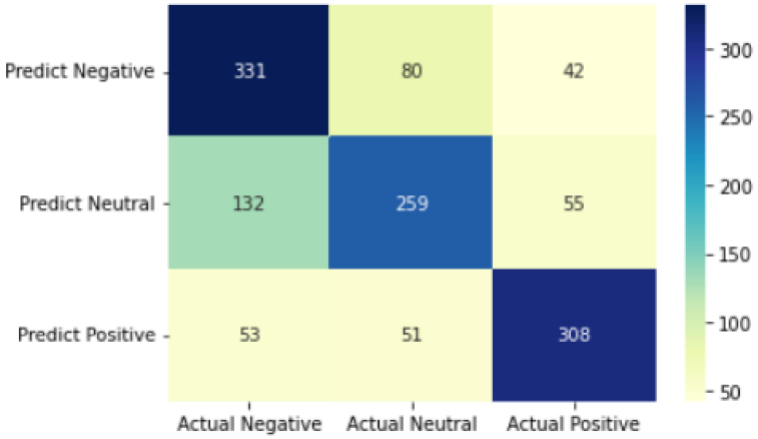
```
[[240 142  71]
 [137 198 111]
 [ 74 115 223]]
              precision    recall  f1-score   support

           0       0.53      0.53      0.53       453
           1       0.44      0.44      0.44       446
           2       0.55      0.54      0.55       412

    accuracy                           0.50      1311
   macro avg       0.51      0.51      0.51      1311
weighted avg       0.50      0.50      0.50      1311
```



Random Forest

```
# Random Forest

# compare y_test and y_pred
y_pred = rnd_clf.predict(X_test)
y_pred_rnd = np.around(y_pred)

# Confusion Matrix

cm_rnd = confusion_matrix(y_test, y_pred_rnd)
print(cm_rnd)

# Classification Report

classrep_rnd = metrics.classification_report(y_test,y_pred_rnd)
print(classrep_rnd)
```

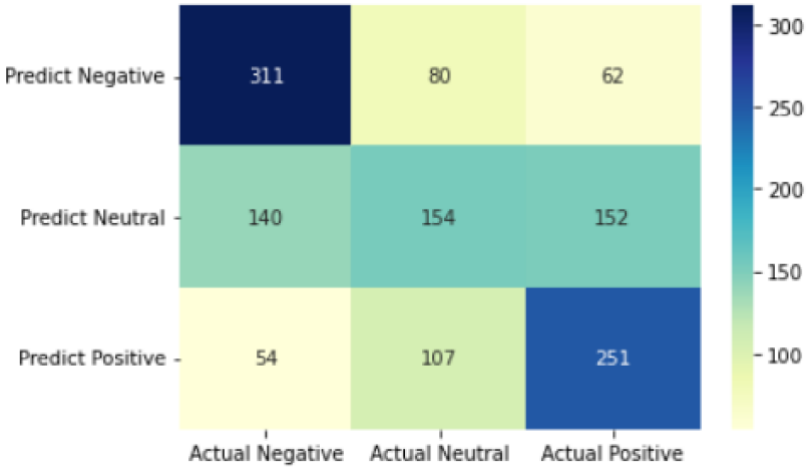
```
[[331  80  42]
 [132 259  55]
 [ 53  51 308]]
              precision    recall  f1-score   support

           0       0.64      0.73      0.68       453
           1       0.66      0.58      0.62       446
           2       0.76      0.75      0.75       412

    accuracy                           0.68      1311
   macro avg       0.69      0.69      0.69      1311
weighted avg       0.69      0.68      0.68      1311
```



SVC

```
# SVC

# compare y_test and y_pred
y_pred = svm_clf.predict(X_test)
y_pred_svm = np.around(y_pred)

# Confusion Matrix

cm_svm = confusion_matrix(y_test, y_pred_svm)
print(cm_svm)

# Classification Report

classrep_svm = metrics.classification_report(y_test,y_pred_svm)
print(classrep_svm)
```

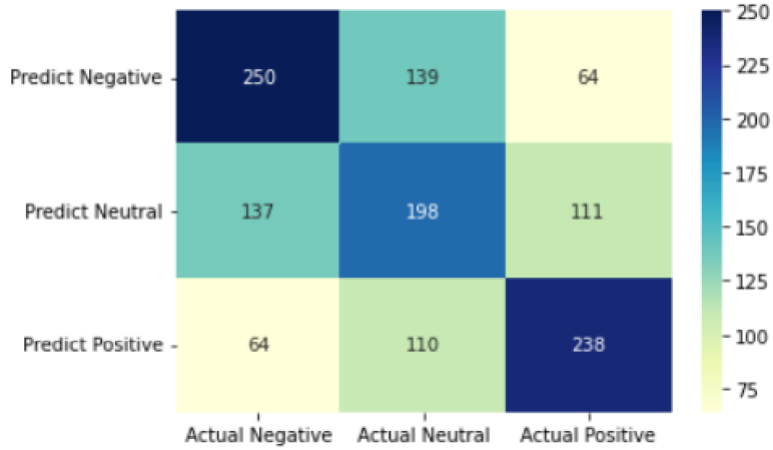
```
[[311  80  62]
 [140 154 152]
 [ 54 107 251]]
              precision    recall  f1-score   support

           0       0.62      0.69      0.65       453
           1       0.45      0.35      0.39       446
           2       0.54      0.61      0.57       412

    accuracy                           0.55      1311
   macro avg       0.54      0.55      0.54      1311
weighted avg       0.54      0.55      0.54      1311
```



Voting

```
# compare y_test and y_pred
y_pred=voting.predict(X_test)
y_pred = np.around(y_pred)
#%%

# Confusion Matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
#%%
# Classification Report

modelrep = metrics.classification_report(y_test,y_pred)
print(modelrep)
```

```
[[250 139  64]
 [137 198 111]
 [ 64 110 238]]
              precision    recall  f1-score   support

           0       0.55      0.55      0.55       453
           1       0.44      0.44      0.44       446
           2       0.58      0.58      0.58       412

    accuracy                           0.52      1311
   macro avg       0.52      0.52      0.52      1311
weighted avg       0.52      0.52      0.52      1311
```



I created two new reviews to check whether they will be correctly classified, one positive and the other negative. And they were correctly classified.

```
# Let's predict on few reviews
neg_review = ['Rooms were old and not very clean. Nothing like in pictures. Staff difficult to reach.
              Food bad. Loud room parties. Pick another hotel. This is definitely on a low level']

# Let's tokenize it and do the pad_sequence to make it in right format acceptable by model
neg_review_token = token.texts_to_sequences(neg_review)

# padding
neg_review_padded = pad_sequences(neg_review_token,maxlen=1911,padding='post')
review_predict = (voting.predict(neg_review_padded)>0.5).astype('int32')

# 1 is Positive review and 0 is negative review
if review_predict[0] == 0:
    print("It's a negative review")
elif review_predict[0] == 1:
    print("It's a neutral review")
else:
    print("It's a positive review")

# Let's try another one.This time we will take a positive review
pos_review = ["It was beautiful, and the staff was very friendly. The rooms are clean and modern. \
              Very impressed with this hotel!"]

# Tokenization
pos_review = token.texts_to_sequences(pos_review)

# padding
pos_review = pad_sequences(pos_review,maxlen=1911,padding='post')

# prediction
review_predict = (voting.predict(pos_review)>0.5).astype('int')

# 1 is Positive review and 0 is negative review
if review_predict[0] == 0:
    print("It's a negative review")

else:
    print("It's a positive review")
```

2.5 Ausblick

Using a number of machine learning classifiers – decision tree, random forest and support vector classifier and combining them in an ensemble, it was possible to get predictions for the classification of ratings with good accuracy of 68.5%, but unfortunately, the ensemble did not improve the predictions.
However, the accuracy could be even better. On the first try, I performed the analysis on the whole dataset and the accuracy was higher (around 75%), but there were major issues as the data were only classified as positive for Random Forest and SVC. This is probably because the dataset is unbalanced – most of the ratings are positive. Then, I read that I could overcome this issue by doing resampling and I did it with down-sampling.
To improve the accuracy score, I wanted to try implementing class-weights for each learning classifier instead of doing down-sampling - which might be the reason for lower accuracy (as some important data points might be lost), but due to the time limitation, I could not do it because the compiling takes a lot of time when it is performed on the whole dataset.
What could be also done in further projects to improve the quality and accuracy of the classifiers is to use a different combination of other machine learning classifiers or neural networks in the ensemble. Additionally, conducting an in-depth analysis of the ratings obtained by a sentiment analysis predicted from the textual data and comparing them with actual ratings would be a great next step or exploring other NLP methods to get better performance of the classification.

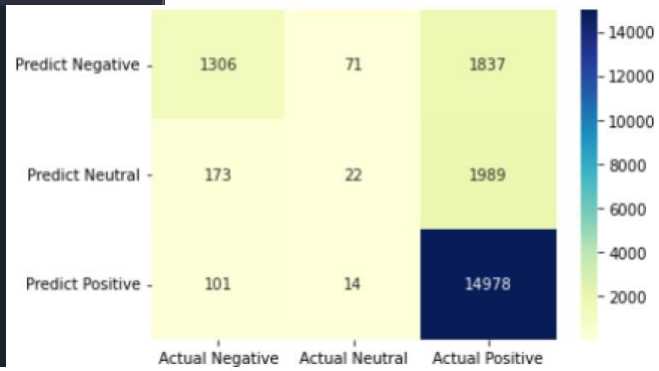
The procedure described here, however, is a good starting point for practising how to create an ensemble of different machine learning classifiers in order to see which of them performs the best. It could help us predict the ratings of hotels based on the customers' reviews quite well, and a comparable procedure could be applied to other datasets with a similar topic.

****** For fun **** NLP VADER Sentiment Analysis**

In the dataset, we already had the ratings that customers wrote, and I did the analysis based on these ratings. For fun, I wanted to check the Sentiment Analysis with VADER (following Tutorial 8) based on the text itself, i.e., how the positive, neutral, and negative reviews from the data set will be recognized. (The analysis was performed on the whole data set)

```
# VADER Sentiment Analysis

print(df['Rating'].value_counts())
# 5    9054
# 4    6039
# 3    2184
# 2    1793
# 1    1421
# df.dropna(inplace=True)
print(df['label'].value_counts())
# 2    15093
# 0     3214
# 1     2184

df.loc[0]['Review'] # first review

sid = SentimentIntensityAnalyzer()
print(sid.polarity_scores(df.loc[0]['Review']))

# {'neg': 0.072, 'neu': 0.643, 'pos': 0.285, 'compound': 0.9747} -> Positive

df['scores'] = df['Review'].apply(lambda review: sid.polarity_scores(review))
df.head()

df['compound']  = df['scores'].apply(lambda score_dict: score_dict['compound'])
df.head()

df['comp_score'] = df['compound'].apply(lambda c: 'Positive' if c >=0.05
                                        else ( 'Negative' if c<= - 0.05
                                              else 'Neutral'))

print(df.head())

print(accuracy_score(df['label_name'],df['comp_score']))

print(classification_report(df['label_name'],df['comp_score']))

print(confusion_matrix(df['label_name'],df['comp_score']))
```

```
0.7957639939485628
              precision    recall  f1-score   support

    Negative       0.83      0.41      0.54      3214
     Neutral       0.21      0.01      0.02      2184
    Positive       0.80      0.99      0.88     15093

    accuracy                           0.80     20491
   macro avg       0.61      0.47      0.48     20491
weighted avg       0.74      0.80      0.74     20491

[[ 1306    71  1837]
 [  173    22  1989]
 [  101    14 14978]]
```



From the confusion matrix and classification report, we can observe that accuracy is 80%, which is much better than the accuracy from the classifiers in the ensemble.