University of St. Gallen

School of Management, Economic, Law,

Social Sciences and International Affairs (HSG)

Programming with Advanced Computer Languages

Dr. Mario Silic

# Sentiment Analysis of Restaurant Reviews

## A Case Study

---

Fabrice Gürmann 14-607-873

Niki Naghavipour 13-750-401

Isabel Oechslin 15-608-631

St. Gallen, 20th December 2019

## Table of content

**1 INTRODUCTION ..... 3**

**2 PROJECT-SETUP ..... 3**

2.1 Determination of project environment ..... 3
2.2 Determination of data set ..... 3

**3 CODE OUTLINE ..... 4**

**4 CODE ..... 6**

**5 CONCLUSION ..... 12**

**6 SOURCES ..... 13**

# 1 Introduction

A large share of customers opts for or against a restaurant based on past reviews. Thus, it is crucial for companies to know what is written about them online. Because of increasing data availability companies struggle to gain strategic insights and track the companies' performance (Weisskopf & Masset, 2018). Therefore, the goal of this project is to help restaurants gain real-time indications of customer satisfaction. This will allow companies to make data-based decisions to improve their performance. In the context of our project, we will analyze the sentiments of individual restaurant reviews with the help of text-mining.

# 2 Project-setup

## 2.1 Determination of project environment

The entire project, from data acquisition to graphics creation, was written in the integrated Colaboratory development environment. Specifically, Colaboratory was chosen to create and share live code, visualizations and code documentation within the team without the need to install Python and Machine Learning tools locally on the desktop. Furthermore, the entire project was written in Python 3 in order to use the required libraries from Python.

## 2.2 Determination of data set

To create a sentiment analysis and key messages based on customer reviews, we needed a restaurant record. Above all, open-source platforms such as GitHub and Kaggle offered themselves here. In the end, anonymous restaurant reviews were used to provide consistent customer reviews. The raw data used contains 1000 lines and two columns ("Review" and "Liked"). Each line corresponds to a customer review ("Review"), whereby the rating ("Liked") differentiates between negative (0) and positive (1). (Kaggle, 2018)

## 3 Code outline

- **Line 1 – 23 Installation of required libraries:** All necessary libraries are installed and imported such as the panda data reader which is a necessity for the program to run.
- **Line 25 – 40 Data import and saving to a panda data frame:** The data is imported from the local .tsv file and saved to a data frame using the panda library. Furthermore, we check how many entries we have.
- **Line 41 – 67 Building a sentiment label**: The tagging of the data ("labeling") consists essentially of being able to train the model on the basis of tasks and their solutions to solve similar and unseen tasks (Martins, 2018). In the context of our project, the task for the model was to find the sentiment of a review. Therefore, we created a new column in the data frame that is called sentiment to represent a positive and negative experience in the restaurant. We created the column based on the “liked” column where 0 indicates negative and 1 indicates positive.
- **Line 68 – 81 Label Transformation:** To train machine learning models with the labels, the labels had to be converted into a numeric form (Mayer, 2019). Therefore, we used the label encoder to create two sentiments id (0 and 1).
- **Line 82 – 113 Helper function for Text normalisation:** As mentioned above, textual data must be represented in a numerical form for the creation of the model. Therefore, we need to tokenise data, meaning transforming the reviews text to vectors. To limit tokenization to the most important words, text normalization is applied. The normalization of the data is intended to erase unimportant information that affects the accuracy of the model. Consequently, within the framework of our project, the review data had to be normalized and then transformed into a numerical form. For reasons of efficiency, normalization was included as a helper function of tokenization. Specifically, we removed all the stopwords and applied stemming.
- **Line 119 – 161 Text normalisation and Model creation:** The tokenizer transforms the data into streams of token objects, with each token object identifying its own word or punctuation within a sentence (Porsteinsson, 2019). While there are several ways to make the tokenization of the data, we chose the "bag of words" method to measure the frequency of each token. In a next step, the tokens were converted into numbers using a vectorizer (vectorizer). As a tokenizer the CountVectorizer of Sklearn was used. (D'Souza, 2018) In a final step, the significance of a token was weighted using the TF-IDF methodology. TF-IDF stands for term frequency-inverse document frequency. The TF-IDF weight is a statistical measure of how important a word is for a document in a collection or corpus. The meaning increases in proportion to the number of occurrences of a word in the document but is balanced by the frequency of the word in the corpus. For the TF-IDF transformation also the Sklearn Library was used (D'Souza, 2018). Furthermore, we split the data between test and training data in order to make sure that we test the model with data that the

model has not yet seen to make sure we rule out overfitting. In the last step, we create a model with the help of the Multinomial Naive Bayes as a check to see whether the text normalisation and test-training split have worked successfully.

- **Line 162 – 251 Model training with other classifiers:** Given the variety of classification algorithms available to create and train a model, a model evaluation was done within the work (Sidana, 2019). To be able to evaluate the models objectively, therefore, a helper function was written which standardizes the procedure per model and uses the same vectors as an argument for each model. The choice of classification algorithms to be evaluated was based on the Naive Bayes algorithm, the logistic regression, the random forest classifier, and the linear SVC.
- **Line 252 – 273 Model evaluation:** In order to determine the classification algorithm to be optimized, the accuracy value ("accuracy value") was used as the decisive criterion. In this regard, the classification algorithm having the highest accuracy value was followed up. In our case, the naive bayes was the algorithm that created the most accurate model, therefore we decided to visualize the results of the naive bayes algorithm only.
- **Line 274 – 348 Model visualisation**: In order to get a better understanding of our results we illustrated the test results graphically in a confusion matrix. This allowed us to have a better overview of the recall and precision value.

## 4 Code

```
# -*- coding: utf-8 -*-
"""Untitled1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1iF5tNNoR2lhqVU4bNgQL1viJa8n6mv0Q
"""

# install mglearn:
!pip install mglearn
# import libraries required:
import os
import glob
from pprint import pprint

import sklearn
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

from tqdm.auto import tqdm

import io

"""# Data import and Data preparation"""

# upload Restaurant_Reviews.tsv from your local drive

from google.colab import files
uploaded = files.upload()

# Saving the dataset in a Panda Dataframe

df = pd.read_csv("Restaurant_Reviews.tsv", sep='\t')

df.head(100)

# show row length to see how much data we have
len(df)

"""# Building a sentiment classifier"""

#New column created with help of a mapping from the values in the "Liked" column
df['Sentiment'] = df.Liked.map({
 0: 'Negativ',
 1: 'Positive'
    })
df.head(10)

# drop the null values in the review column to have only clean data. check the new leng
th
df.Review.isna().sum()
#keep the ones that are not not null
df = df[~df.Review.isna()]
len(df)

# show number of entries per sentiment to make sure we have a balanced dataset (roughly
 50%)
import matplotlib.pyplot as plt; plt.style.use('seaborn')

fig = plt.figure(figsize=(8, 6))

df\
    .groupby('Sentiment')['Review']\
```

```
    .count()\
    .plot(kind='bar', ylim=0)

plt.show()

"""# Label transformation & Text normalisation"""

# Use LabelEncoder  to convert text labels into values/id's
from sklearn.preprocessing import LabelEncoder


lbl_enc = LabelEncoder()
lbl_enc.fit(sorted(df['Sentiment'].unique()))

# obtaining a numeric representation of an array
df['sentiment_id'] = lbl_enc.transform(df['Sentiment'])

df.head(10)

# Here we introduce the stemmatization function, which we use afterwards within the count vectorizer
# We have to describe the function separately because Sklearn does not offer a native stemming function
""" Helper function for Text normalisation"""


import nltk
from nltk.stem.porter import PorterStemmer
from textblob import TextBlob
import re
!pip install textblob
nltk.download('punkt')
SENT_DETECTOR = nltk.data.load('tokenizers/punkt/english.pickle')


porter_stemmer = PorterStemmer()

# Use TextBlob function in order to stem the data of the review rext
def textblob_tokenizer(str_input):
            blob = TextBlob(str_input.lower())
            tokens = blob.words
            words = [token.stem() for token in tokens]
            return words

        # Use NLTK's PorterStemmer
        def stemming_tokenizer(str_input):
            words = re.sub(r"[^A-Za-z0-9\-]", " ", str_input).lower().split()
            words = [porter_stemmer.stem(word) for word in words]
            return words

            stemming_tokenizer("I went fishing to get fishes")
            textblob_tokenizer("I went fishing to get fishes")
          """# Text normalization and model creation"""

        ## NNOTE: In this code section we transform the review data into text data. Labels have already been encoded
        from sklearn.model_selection import cross_val_score, StratifiedShuffleSplit, StratifiedKFold
        from sklearn.model_selection import train_test_split
        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer
        from sklearn.naive_bayes import MultinomialNB


        # Split data
        train_index, test_index = next(StratifiedShuffleSplit(n_splits=1, test_size=0.2).split(df['Review'], df['sentiment_id']))
```

```
124.
125.        # Get data
126.        train_df = df.iloc[train_index]
127.        test_df = df.iloc[test_index]
128.
129.        # Get data
130.        X_train = train_df['Review'].to_numpy()
131.        X_test = test_df['Review']
132.        y_train_labels = train_df['sentiment_id']
133.        y_test_labels = test_df['sentiment_id']
134.        print(type(X_train))
135.
136.        # Convert labels to numbers NOTE: Here we do not convert to labels because we ha
    ve done so above!!!
137.        y_train = y_train_labels.to_numpy()
138.        y_test = y_test_labels
139.        print(type(y_train))
140.
141.
142.        # Use a CountVectorizer for bag-of-words
143.        count_vect = CountVectorizer(min_df=5, ngram_range=(1, 2), stop_words='english',
     tokenizer=textblob_tokenizer)
144.        X_train_counts = count_vect.fit_transform(X_train)
145.        X_test_counts = count_vect.fit_transform(X_test)
146.
147.        # Use Tfidf to transform the counts into tfidf weights
148.        tfidf_transformer = TfidfTransformer(sublinear_tf=True, norm='l2')
149.        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
150.        X_test_tfidf = tfidf_transformer.fit_transform(X_test_counts)
151.
152.        # TO GET FEATURES (get_feature_names()), YOU HAVE TO USE TfidfVectorizer
153.        # This is the same as the two above
154.        tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', ngram_range=(1,
    2), stop_words='english')
155.        X_train_tfidf = tfidf.fit_transform(X_train)
156.        X_test_tfidf = tfidf.transform(X_test)
157.
158.        # To test whether the transforamtion of the data worked, we insert the created v
    ariables as inputs to the multinomialNB to check if it can create a model
159.        clf = MultinomialNB()
160.        clf.fit(X_train_counts, y_train)
161.
162.        """# Model training with other classifiers"""
163.
164.        # A reusable function that we can loop through for every single classifier
165.        # import Classifiers
166.        from sklearn.naive_bayes import MultinomialNB
167.        from sklearn.linear_model import LogisticRegression
168.        from sklearn.ensemble import RandomForestClassifier
169.        from sklearn.svm import LinearSVC
170.
171.        # Import training utilities
172.        from sklearn.feature_selection import chi2
173.
174.        # Import metrics
175.        from sklearn.metrics import accuracy_score, precision_recall_fscore_support, cla
    ssification_report, confusion_matrix
176.
177.        #Here we create a standardised function that can be used for each classifier to
    create a model. this allows comparability between the effectiveness of the classifiers

178.        def train(model, features, labels, num_cv):
179.
180.            results = []
181.
182.            kfold = StratifiedShuffleSplit(n_splits=num_cv, test_size=0.2)
183.
```

```
184.            for train, test in kfold.split(features, labels):
185.
186.                clf = sklearn.clone(model)
187.
188.                X_train = features[train]
189.                X_test = features[test]
190.                y_train = labels[train]
191.                y_test = labels[test]
192.
193.                clf.fit(X_train, y_train)
194.
195.                y_pred = clf.predict(X_test)
196.
197.                a = accuracy_score(y_test, y_pred)
198.                p, r, f, _ = precision_recall_fscore_support(y_test, y_pred, average='ma
    cro')
199.                report = classification_report(y_test, y_pred)
200.
201.                results.append({
202.                    'accuracy': a,
203.                    'precision': p,
204.                    'recall': r,
205.                    'fscore': f,
206.                    'report': report
207.                })
208.
209.            return results
210.
211.        #In this code part we loop through the function above with different classifiers
    !!
212.        #NOTE: This code has dependencies with the text representation, since we take th
    e vectorized data from there
213.
214.        # Define the different classifier that we want to usemodels
215.        models = [
216.            MultinomialNB(),
217.            LogisticRegression(random_state=0, solver='liblinear', multi_class='auto'),

218.            RandomForestClassifier(n_estimators=200, max_depth=3),
219.            LinearSVC()
220.        ]
221.        # trainings set is split 5 times and the results are appended to the back of the
     entries list
222.
223.        num_cv = 5
224.
225.        entries = []
226.
227.        # Go over each Classifier
228.        for model in models:
229.            model_name = model.__class__.__name__
230.
231.        # NOTE: the train x_train_tfid and y_train are variables we created earlier! We
    shall not overwrite them. review data is not vectorised and the model can not analyze i
    t!!!!!!
232.
233.            results = train(model, X_train_tfidf, y_train, num_cv)
234.        #Create areport for the test results and store it to a list.
235.
236.            for fold_idx, metric in enumerate(results):
237.                entries.append({
238.                    'model_name': model_name,
239.                    'fold_idx': fold_idx,
240.                    'accuracy': metric['accuracy'],
241.                    'precision': metric['precision'],
242.                    'recall': metric['recall'],
243.                    'fscore': metric['fscore'],
```

```
                    'report': metric['report']
                })

        # convert list to a panda dataframe
        cv_df = pd.DataFrame(entries)

        cv_df

        """# Model evaluation"""

        # reusable function to illustrate data with a certain type of box plot
        import matplotlib.pyplot as plt; plt.style.use('seaborn')
        import seaborn as sns

        def plot_metric(df, metric):
            fig = plt.figure(figsize=(6, 8))

            sns.boxplot(x='model_name', y=metric, data=cv_df[['model_name', metric]])
            sns.stripplot(x='model_name', y=metric, data=cv_df[['model_name', metric]],

                          size=8, jitter=True, edgecolor="gray", linewidth=2)
            plt.xticks(rotation=30, ha='right')
            plt.show()

        #create a plot for the accuracy values
        plot_metric(cv_df, 'accuracy')

        #overarching model evaulation. we see that MultinomialNB is the most efficient o
    ne. hence we train the entire training data with SVC to see
        # if we can increase the scores..
        cv_df.groupby('model_name')[['accuracy', 'precision', 'recall', 'fscore']].mean(
    )

        """# Model visualisation"""

        # We use the MultinomialNB and train on the WHOLE training set, without k-
    fold.
        model = MultinomialNB()

        model.fit(X_train_tfidf, y_train)

        y_pred = model.predict(X_test_tfidf)

        #we want to graphically illustrate our results in more detail. true positives, f
    alse negatives
        from sklearn.utils.multiclass import unique_labels
        # First, we need a function to draw the confustion matrix
        def plot_confusion_matrix(y_true, y_pred, classes,
                                  normalize=False,
                                  title=None,
                                  cmap=plt.cm.Reds, ax=None):
            if not title:
                if normalize:
                    title = 'Normalized confusion matrix'
                else:
                    title = 'Confusion matrix, without normalization'

            classes = [classes[i] for i in unique_labels(y_true, y_pred)]

            # Compute confusion matrix
            cm = confusion_matrix(y_true, y_pred)

            if normalize:
                cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

            if ax is None:
                ax = plt.gca()
```

```
306.
307.           im = ax.imshow(cm, cmap=cmap)
308.
309.           cb = ax.figure.colorbar(im, ax=ax)
310.
311.           tick_marks = np.arange(len(classes))
312.
313.           ax.set_xticks(tick_marks)
314.           ax.set_yticks(tick_marks)
315.
316.           ax.set_xticklabels(classes, rotation=0, ha='right', rotation_mode='anchor')
317.           ax.set_yticklabels(classes, rotation=0)
318.
319.           ax.set_xlabel('Predicted class')
320.           ax.set_ylabel('True class')
321.
322.           # Loop over data dimensions and create text annotations.
323.           fmt = '.2f' if normalize else 'd'
324.           thresh = cm.max() / 2
325.           for i in range(cm.shape[0]):
326.               for j in range(cm.shape[1]):
327.                   ax.text(j, i, format(cm[i, j], fmt),
328.                           ha="center", va="center",
329.                           color="white" if cm[i, j] > thresh else "black")
330.
331.           #fig.tight_layout()
332.           plt.grid(False)
333.
334.           return ax
335.
336.       fig, ax = plt.subplots(figsize=(12, 12))
337.
338.       plot_confusion_matrix(y_test, y_pred, classes=lbl_enc.classes_, normalize=True)
339.
340.       ax.set_title('Confusion Matrix (Normalized Values)')
341.       plt.show()
342.
343.       fig, ax = plt.subplots(figsize=(12, 12))
344.
345.       plot_confusion_matrix(y_test, y_pred, classes=lbl_enc.classes_, normalize=True)
346.
347.       ax.set_title('Confusion Matrix (Normalized Values)')
348.       plt.show()
```

## 5 Conclusion

| model_name | accuracy | precision | recall | fscore |
|---|---|---|---|---|
| **LinearSVC** | 0.71000 | 0.713416 | 0.71000 | 0.708801 |
| **LogisticRegression** | 0.73625 | 0.744178 | 0.73625 | 0.734107 |
| **MultinomialNB** | 0.74500 | 0.747444 | 0.74500 | 0.744389 |
| **RandomForestClassifier** | 0.71000 | 0.755534 | 0.71000 | 0.695975 |

*Figure 1: Accuracy of the four different models (own figure)*

The table indicates that the MultionomialNB provides the highest accuracy rate of 74.5%. Therefore, we used the MultionomialNB to train the entire data set to see if we could increase the scores. Afterwards, we created a confusion matrix to visualize our model.

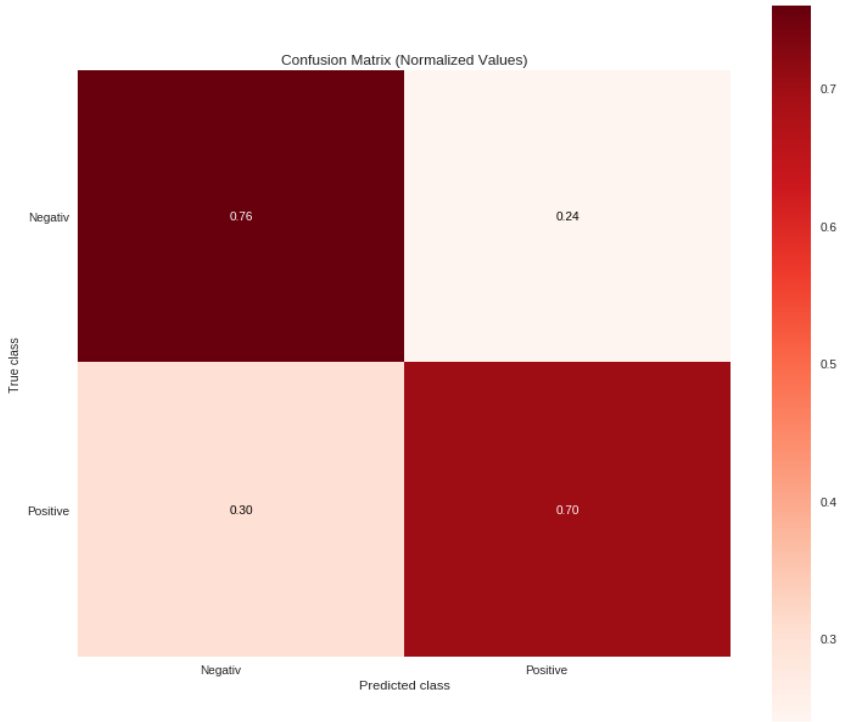


*Figure 2: Confusion Matrix for the MultinomialNB (own figure)*

The matrix shows that the model identified 70% of the positive ratings correctly and 76% of the negative ratings. Therefore, this model can be used by restaurants to gain a real-time overview of their current online performance with respect to customer reviews and enable them to take accurate actions to boost their performance.

# 6 Sources

D'Souza, J. (2018, June 21). An Introduction to Bag-of-Words in NLP [Article in a Forum]. Retrieved December 11, 2019 from https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428

Kaggle. (2019, July 26). Restaurant Customer Reviews [Dataset]. Retrieved December 11, 2019 from https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews

Martins, P. (2018, February). Do You Know What a Data Labeler Does? [Article in a Forum]. Retrieved December 11, 2019 from https://towardsdatascience.com/do-you-know-what-does-a-data-labeler-do-98561cb0029

Mayer, S. (2019). Introduction to Computer Systems and Networks_VL1 [Lecture]. St.Gallen, Schweiz

Porsteinsson, V. (2019, October 22). tokenizer. Retrieved December 11, 2019 from https://pypi.org/project/tokenizer/

Sidana, M. (2019, July 5). Types of classification algorithms in Machine Learning [Article in a forum]. Abgerufen 27. November 2019, von https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14

Weisskopf, J.-P., Masset, P. (2018, August 30). Restaurant Reviews: Their Use and Their Impact on Business. Retrieved December 11, 2019 from https://www.hospitalitynet.org/opinion/4089762.html