

|  |
| --- |
| 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**

**Durch das Hinzufügen von Überschriften (Format > Absatzstile) erscheinen diese im Inhaltsverzeichnis.**

# Introduction

A large share of customers opt 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 take data-based decisions to improve their performance. In the context of our project, we will analyse the sentiments of individual restaurant reviews with the help of text-mining.

# Project-setup

## 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.

## 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)

# 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 - 45: Dataimport and saving to a panda dataframe:** The data is imported from the local .tsv file and saved to a dataframe using the panda library. Furthermore we check how many entries we have.
* **Line 46 - 73: 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 dataframe 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 73 - 88: 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 89 - 118: 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 - 167: 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 splitted the data between test and trainings 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 help of the Multinomial Naive Bayes as a check to see whether the text normalisation and test-training split have worked successfully.

* **Line 168 - 255: 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 forrest classifier and the linear SVC.

* **Line 256 -276: 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 visualise the results of the naive bayes algorithm only.
* **Line 277 - 366: 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 on the recall and precision value.

# Code

1. # -\*- coding: utf-8 -\*-
2. """PythonSentimentRestaurant.ipynb
4. Automatically generated by Colaboratory.
6. Original file is located at
7. https://colab.research.google.com/drive/1hdHMx75onBerghnUiK9oBshDUwvOJu38
8. """
10. # install mglearn:
11. !pip install mglearn
12. # import libraries required:
13. **import** os
14. **import** glob
15. **from** pprint **import** pprint
17. **import** sklearn
18. **import** numpy as np # linear algebra
19. **import** pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)
21. **from** tqdm.auto **import** tqdm
23. **import** io
25. """# Data import and Data preparation"""
27. # upload Restaurant\_Reviews.tsv from your local drive
29. **from** google.colab **import** files
30. uploaded = files.upload()
32. # refactoring to do: insert header
33. # Saving the datasets in Panda Dataframe


37. df = pd.read\_csv("Restaurant\_Reviews.tsv", sep='\t')
39. tsv\_read.head(100)
41. df.head(100)
43. # show row length
44. len(df)
46. """# Building a sentiment classifier"""
48. #New column
49. df['Sentiment'] = df.Liked.map({
50. 0: 'Negativ',
51. 1: 'Positive'
52. })
53. df.head(10)
55. # dropping all the entries that do not contain text. Then we check how many lines have been dropped aka how many entries did not have text.
56. df.Review.isna().sum()
57. #keep the ones that are not not null
58. df = df[~df.Review.isna()]
59. len(df)
61. # show number per sentiment
62. **import** matplotlib.pyplot as plt; plt.style.use('seaborn')
64. fig = plt.figure(figsize=(8, 6))
66. df\
67. .groupby('Sentiment')['Review']\
68. .count()\
69. .plot(kind='bar', ylim=0)
71. plt.show()
73. """# Label transformation """
75. # Labelling the Data
76. **from** sklearn.preprocessing **import** LabelEncoder

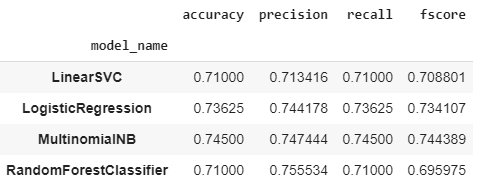
79. lbl\_enc = LabelEncoder()
80. lbl\_enc.fit(sorted(df['Sentiment'].unique()))
82. # obtaining a numeric representation of an array
83. df['sentiment\_id'] = lbl\_enc.transform(df['Sentiment'])
85. df.head(10)
87. # Here we introduce the stemmatization function, which we use afterwards within the count vectorizer
88. # We have to describe the function separately because Sklearn does not offer a native stemming function
89. """ Helper function for Text normalisation"""
91. **import** nltk
92. **from** nltk.stem.porter **import** PorterStemmer
93. **from** textblob **import** TextBlob
94. **import** re
95. !pip install textblob
96. nltk.download('punkt')
97. SENT\_DETECTOR = nltk.data.load('tokenizers/punkt/english.pickle')

100. porter\_stemmer = PorterStemmer()
102. # Use TextBlob
103. **def** textblob\_tokenizer(str\_input):
104. blob = TextBlob(str\_input.lower())
105. tokens = blob.words
106. words = [token.stem() **for** token **in** tokens]
107. **return** words
109. # Use NLTK's PorterStemmer
110. **def** stemming\_tokenizer(str\_input):
111. words = re.sub(r"[^A-Za-z0-9\-]", " ", str\_input).lower().split()
112. words = [porter\_stemmer.stem(word) **for** word **in** words]
113. **return** words
115. stemming\_tokenizer("I went fishing to get fishes")
116. textblob\_tokenizer("I went fishing to get fishes")
118. """#Text normalisation"""
120. ## NOTE: In this code section we transform the review data into text data. Labels have already been encoded
121. **from** sklearn.model\_selection **import** cross\_val\_score, StratifiedShuffleSplit, StratifiedKFold
122. **from** sklearn.model\_selection **import** train\_test\_split
123. **from** sklearn.feature\_extraction.text **import** CountVectorizer, TfidfTransformer, TfidfVectorizer
124. **from** sklearn.naive\_bayes **import** MultinomialNB

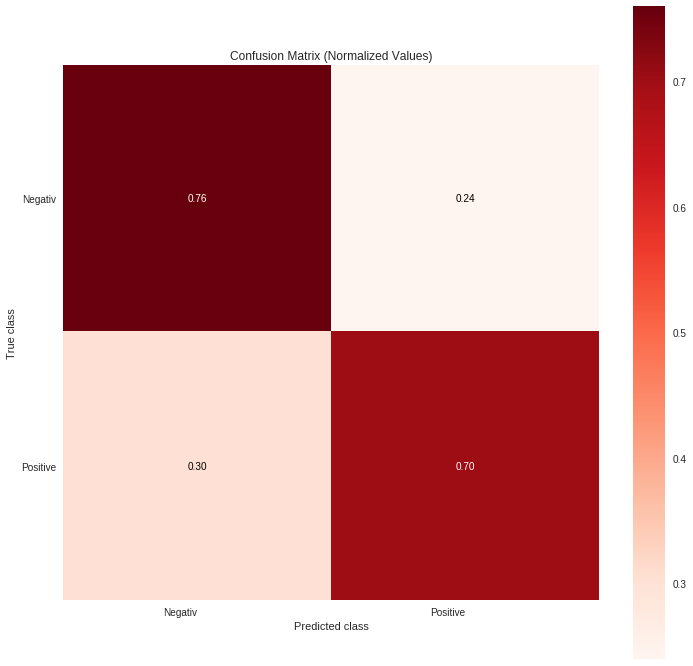
127. # Split data
128. train\_index, test\_index = next(StratifiedShuffleSplit(n\_splits=1, test\_size=0.2).split(df['Review'], df['sentiment\_id']))
130. # Get data
131. train\_df = df.iloc[train\_index]
132. test\_df = df.iloc[test\_index]
134. # Get data
135. X\_train = train\_df['Review'].to\_numpy()
136. X\_test = test\_df['Review']
137. y\_train\_labels = train\_df['sentiment\_id']
138. y\_test\_labels = test\_df['sentiment\_id']
139. **print**(type(X\_train))
141. # Convert labels to numbers NOTE: Here we do not convert to labels because we have done so above.
142. y\_train = y\_train\_labels.to\_numpy()
143. y\_test = y\_test\_labels
144. **print**(type(y\_train))

147. # Use a CountVectorizer for bag-of-words
148. count\_vect = CountVectorizer(min\_df=5, ngram\_range=(1, 2), stop\_words='english', tokenizer=textblob\_tokenizer)
149. X\_train\_counts = count\_vect.fit\_transform(X\_train)
150. X\_test\_counts = count\_vect.fit\_transform(X\_test)
152. # Use Tfidf to transform the counts into tfidf weights
153. tfidf\_transformer = TfidfTransformer(sublinear\_tf=True, norm='l2')
154. X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)
155. X\_test\_tfidf = tfidf\_transformer.fit\_transform(X\_test\_counts)
157. # To get features (get\_feature\_names()), you have to use TfidfVectorizer
158. # This is the same as the two above
159. tfidf = TfidfVectorizer(sublinear\_tf=True, min\_df=5, norm='l2', ngram\_range=(1, 2), stop\_words='english')
160. X\_train\_tfidf = tfidf.fit\_transform(X\_train)
161. X\_test\_tfidf = tfidf.transform(X\_test)
163. # To test whether the transforamtion of the data worked, you use: Train Naïve Bayes
164. clf = MultinomialNB()
165. clf.fit(X\_train\_counts, y\_train)
167. """# Model training with other classifiers"""
169. # A reusable function that we can loop through for every single classifier
170. # import Classifiers
171. **from** sklearn.naive\_bayes **import** MultinomialNB
172. **from** sklearn.linear\_model **import** LogisticRegression
173. **from** sklearn.ensemble **import** RandomForestClassifier
174. **from** sklearn.svm **import** LinearSVC
176. # Import training utilities
177. **from** sklearn.feature\_selection **import** chi2
179. # Import metrics
180. **from** sklearn.metrics **import** accuracy\_score, precision\_recall\_fscore\_support, classification\_report, confusion\_matrix
182. **def** train(model, features, labels, num\_cv):
184. results = []
186. kfold = StratifiedShuffleSplit(n\_splits=num\_cv, test\_size=0.2)
188. **for** train, test **in** kfold.split(features, labels):
190. clf = sklearn.clone(model)
192. X\_train = features[train]
193. X\_test = features[test]
194. y\_train = labels[train]
195. y\_test = labels[test]
197. clf.fit(X\_train, y\_train)
199. y\_pred = clf.predict(X\_test)
201. a = accuracy\_score(y\_test, y\_pred)
202. p, r, f, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred, average='macro')
203. report = classification\_report(y\_test, y\_pred)
205. results.append({
206. 'accuracy': a,
207. 'precision': p,
208. 'recall': r,
209. 'fscore': f,
210. 'report': report
211. })
213. **return** results
215. #In this code part we loop through the function above with different classifiers!!
216. #NOTE: This code has dependencies with the text representation!!
218. # Define different classifier
219. models = [
220. MultinomialNB(),
221. LogisticRegression(random\_state=0, solver='liblinear', multi\_class='auto'),
222. RandomForestClassifier(n\_estimators=200, max\_depth=3),
223. LinearSVC()
224. ]
226. num\_cv = 5
228. entries = []
230. # Go over each Classifier
231. **for** model **in** models:
232. model\_name = model.\_\_class\_\_.\_\_name\_\_
234. # NOTE: data is taken from the naive bayes model. We take the data from line 154
235. # This is important! Otherwise the review data is not vectorised and the model can not analyze it!!!!!!
236. # trainings set is split 5 times and the results are appended to the back of the entries list
237. results = train(model, X\_train\_tfidf, y\_train, num\_cv)
239. **for** fold\_idx, metric **in** enumerate(results):
240. entries.append({
241. 'model\_name': model\_name,
242. 'fold\_idx': fold\_idx,
243. 'accuracy': metric['accuracy'],
244. 'precision': metric['precision'],
245. 'recall': metric['recall'],
246. 'fscore': metric['fscore'],
247. 'report': metric['report']
248. })
250. # DataFrame to store results
251. cv\_df = pd.DataFrame(entries)
253. cv\_df
255. """# Model evaluation"""
257. # reusable function to illustrate data with a certain type of box plot
258. **import** matplotlib.pyplot as plt; plt.style.use('seaborn')
259. **import** seaborn as sns
261. **def** plot\_metric(df, metric):
262. fig = plt.figure(figsize=(6, 8))
264. sns.boxplot(x='model\_name', y=metric, data=cv\_df[['model\_name', metric]])
265. sns.stripplot(x='model\_name', y=metric, data=cv\_df[['model\_name', metric]],
266. size=8, jitter=True, edgecolor="gray", linewidth=2)
267. plt.xticks(rotation=30, ha='right')
268. plt.show()
270. #Example for a plot of accuracy. We could do this with every column
271. plot\_metric(cv\_df, 'accuracy')
273. # Overarching model evaluation. We see that MultinomialNB is the most efficient one. Hence, we train the entire training data with MultinomialNB to see if we can increase the scores..
274. cv\_df.groupby('model\_name')[['accuracy', 'precision', 'recall', 'fscore']].mean()
276. """# Model Visualization"""
278. # We use the MultinomialNB and train on the WHOLE training set, without k-fold.
279. model = MultinomialNB()
281. model.fit(X\_train\_tfidf, y\_train)
283. y\_pred = model.predict(X\_test\_tfidf)
285. # We want to graphically illustrate our results very nicely since we are true designers. This time we want to know exactly how many true positives, false negatives etc we have
287. # First, we need a function to draw the confusion matrix
288. **def** plot\_confusion\_matrix(y\_true, y\_pred, classes,
289. normalize=False,
290. title=None,
291. cmap=plt.cm.Reds, ax=None):
292. **if** **not** title:
293. **if** normalize:
294. title = 'Normalized confusion matrix'
295. **else**:
296. title = 'Confusion matrix, without normalization'
298. classes = [classes[i] **for** i **in** unique\_labels(y\_true, y\_pred)]
300. # Compute confusion matrix
301. cm = confusion\_matrix(y\_true, y\_pred)
303. **if** normalize:
304. cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
306. **if** ax **is** None:
307. ax = plt.gca()
309. im = ax.imshow(cm, cmap=cmap)
311. cb = ax.figure.colorbar(im, ax=ax)
313. tick\_marks = np.arange(len(classes))
315. ax.set\_xticks(tick\_marks)
316. ax.set\_yticks(tick\_marks)
318. ax.set\_xticklabels(classes, rotation=0, ha='right', rotation\_mode='anchor')
319. ax.set\_yticklabels(classes, rotation=0)
321. ax.set\_xlabel('Predicted class')
322. ax.set\_ylabel('True class')
324. # Loop over data dimensions and create text annotations.
325. fmt = '.2f' **if** normalize **else** 'd'
326. thresh = cm.max() / 2
327. **for** i **in** range(cm.shape[0]):
328. **for** j **in** range(cm.shape[1]):
329. ax.text(j, i, format(cm[i, j], fmt),
330. ha="center", va="center",
331. color="white" **if** cm[i, j] > thresh **else** "black")
333. #fig.tight\_layout()
334. plt.grid(False)
336. **return** ax
338. fig, ax = plt.subplots(figsize=(12, 12))
340. plot\_confusion\_matrix(y\_test, y\_pred, classes=lbl\_enc.classes\_, normalize=True)
342. ax.set\_title('Confusion Matrix (Normalized Values)')
343. plt.show()

# Conclusion



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.



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 in respect to customer reviews and enable them to take accurate actions to boost their performance.

# 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