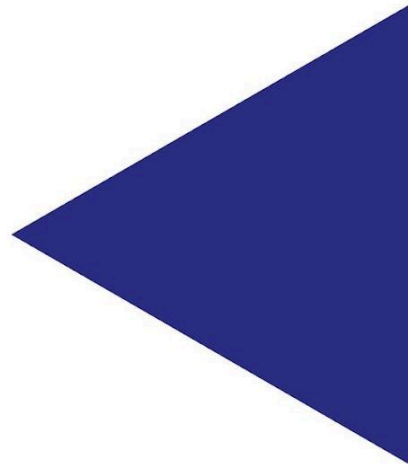
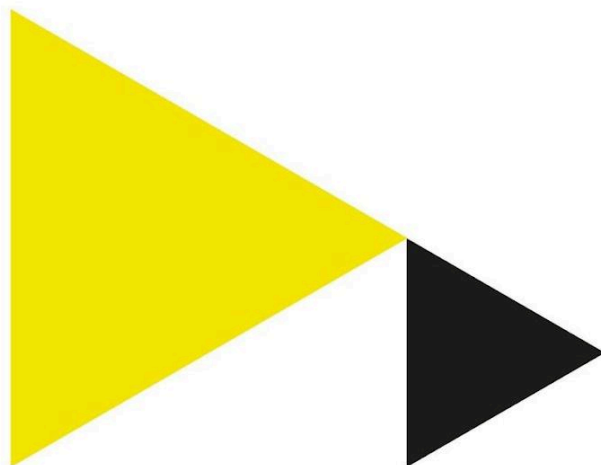


# Big Data Scientist and Engineer Assignment I

Sentiment Analysis on hotel reviews using Machine Learning  
algorithms

Big Data / HBO-ICT / FDMCI  
2021 – 2022



# Big Data Scientist and Engineer Assignment I

Sentiment Analysis on hotel reviews using Machine Learning algorithms

**Author**

Hugo Villanueva, #500808745

**department**

Big Data / HBO-ICT / FDMCI

**Date**

30-Oct-21

**Project type**

Assignment

**Version**

1.0

© 2020 Copyright Amsterdam University of Applied Sciences

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or made public in any form or by any means, electronic, mechanical, printouts, copies, or in any way whatsoever without the prior written consent of the Amsterdam University of Applied Sciences.

# Abstract

Machine Learning is defined by the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data. This kind of approach is getting more common as industries have to handle larger amounts of data from their clients, and the traditional analysis tools are no longer viable.

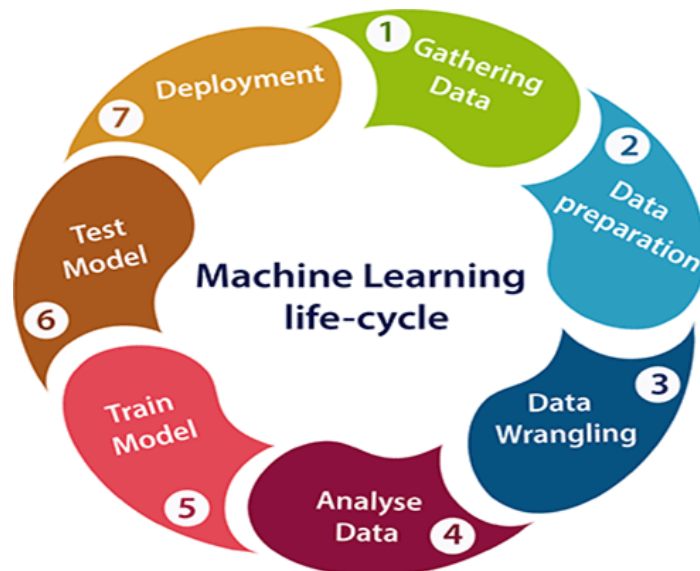
In order to receive feedbacks from the clients, companies usually observe their reviews and customer service communications, but if the load is too big, humans are no longer capable of processing all the reviews. That is why Sentiment Analysis is applied to automatically detect what emotions carry a given text. In this report, we will cover a Sentiment Analysis to detect whether a review carries a positive or negative sentiment, using hotel reviews as dataset, from different hotels and websites. In order to get a wider vision of the solutions, 3 different classifiers will be used, and their performance will be compared.

# Table of contents

<b>Introduction</b>	
<b>1. Data Discovery</b>	
1.1 Data Sources	7
1.1.1 Kaggle Dataset	7
1.1.2 Handwritten reviews	7
1.1.3 Webscrapped reviews	7
1.2 Column values	10
<b>2. Data Preparation</b>	
2.1 Insights before cleaning	11
2.2 Data Cleaning	11
2.2.1 Removing empty reviews	12
2.2.2 Removing punctuation	12
2.2.3 Normalizing the words to lowercase	12
2.2.4 Removing stopwords with the help of WordCloud	13
2.3 Insights after cleaning	14
2.3.1 Positive/Negative review ratio by website	14
2.3.2 Positive/Negative reviews by score range	15
<b>3. Model building</b>	
3.1 Pre-processing	17
3.1.1 Tokenizing	17
3.1.2 Stemming	17
3.1.3 Vectorizing	18
3.1.4 Dataset split	19
3.2 Model 1: Logistic Regression	20
3.2.1 Logistic Regression background	20
3.2.2 Model performance	20
3.2.3 Hyperparameter Tuning	22
3.2.4 Model saving	23
3.2.5 Conclusions	24
3.3 Model 2: K-nearest neighbor	24
3.3.1 KNN background	24
3.3.2 KNN performance	24
3.3.3 KNN Tuning	25
3.3.4 Conclusions	26
3.4 Model 3: Support Vector Machine	27
3.4.1 SVM background	27
3.4.2 SVM performance	28
3.4.3 SVM Tuning	28
3.4.4 Conclusions	28
3.5 Model comparison and final conclusions	29
<b>Bibliography/references</b>	<b>30</b>

# Introduction

The goal of this project is to be able to predict, given just the text, whether a hotel review will carry a positive or negative sentiment. On this process, we will go through all stages of the Machine Learning life-cycle:



But not all 7 stages require the same amount of work. That is why this paper is divided into three main sections: Data collection (stage 1), Data preparation (stages 2-4) and Model building (stages 5-7).

The data collection will be done through a previously arranged dataset, webscraping from various websites and handwritten reviews. All this data will be combined into one single Pandas DataFrame and no distinction will be made when it comes to process it. This DataFrame will be stored in a MySQL database using MySQLWorkbench as the IDE.

For the Data preparation, the data will be cleaned and wrangled through SQL stored procedures, and then the data will be fetched in order to draw various plots giving insights from the data.

As for the model building, 3 different classifiers from the scikit-learn library will be used to train and test the model, and after getting the performance results, these classifiers will have their parameters optimized to maximize accuracy on the models. To conclude, the three models will be compared based on efficiency and accuracy.

# 1. Data Discovery

The first step in building our model will be gathering all the raw data that we are going to use for the project. We will fetch hotel reviews from different hotels and websites, with some additional data included in the review itself that will be used in the future, but not for the first model.

## 1.1 Data Sources

Different sources have been used to collect the data, and since the original format is different on each source, every one of them has gone through a different collecting and processing method.

As indicated in the manual, for webscrapped and handwritten reviews, a different hotel must be chosen by the student. In this project, the selected hotel is Caesars Palace, a luxury hotel and casino in Las Vegas, United States, that is widely visited and thus widely reviewed. Since the model is going to be built with reviews written in english, it is interesting to pick a hotel with relevance in an english speaking country.

### 1.1.1 Kaggle Dataset

Given by the university, this dataset from Kaggle (<https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe>) contains over 515,000 hotel reviews from different hotels in Europe in english. Much data is included for each entry, but since this assignment is focused on building a prediction model for the sentiment analysis, only a couple of values will be selected, as shown in the column values subchapter.

In this .csv file, each row contains a positive review as well as a negative review. The function defined for loading the csv into a Dataframe explodes each row into two, taking positive and negative reviews separately and keeping the score. Each one is also labeled with the corresponding sentiment "POSITIVE" or "NEGATIVE" in the "label" column. In this function some preliminary cleaning is also made by removing the reviews with "Nothing", "No Positive" and "No Negative".

### 1.1.2 Handwritten reviews

As indicated in the manual, 10 handwritten reviews must be included in the dataset. For this task, a Dataframe can be created only with the values in the code by defining a dictionary with columns as key and values with lists with each row. The Pandas library contains a method called `pd.DataFrame` that converts this into a tabular dataframe.

The text used in the review has been inspired by reviews found in Google, but it is not webscrapped from there. 5 positive and 5 negative reviews have been typed into the code.

### 1.1.3 Webscrapped reviews

Since the Caesars Palace is frequently visited, many reviews are available across many different sites. However, not all websites are webscraping-friendly, since they would rather sell their API to the data scientists. It is common that websites use JavaScript to hide deeper their reviews thus making it not accessible trough simple BeautifulSoup commands.

Nonetheless, after having visited Expedia, Trip.com, Google Hotels, Trivago, Travelocity, Orbitz, Wotif, Yelp, Hotels.nl, Kayak and Trivago; the last four have been proven to be valid for webscrapping. We will be using Hotels.nl and Yelp, since the other two are too similar to the previous and therefore the functions are less interesting.

For the Yelp webscrapping function, we can fetch many reviews from the same hotel by changing the page number on the URL, thus making Yelp the main source of webscrapped reviews. Yelp works on a 5-star rating system, where the numeric value itself is not contained within the HTML. However, each rating is labelled with a different attribute so it is possible to sort them out. After getting the numeric value, it is stored multiplied by two in order to maintain consistency with the rest of the database.

To illustrate, this is how a Yelp review looks like, as well as the URL:

yelp.com/biz/caesars-palace-las-vegas-10?sort\_by=date\_desc



**Leslie Z.**  
Santa Clarita, CA  
 0  6



9/22/2021

Place was bomb nice and cool. Would go back cuz that food was no joke. Also that service was real good. The room was cute as well. It was luxurious and expensive and I like it. Definitely recommend since I spent a good time. Ok

The criteria for labelling the sentiment from the reviews as positive is decided with a score over 4 stars or more, and with 2 stars or less it is considered negative.

The following code illustrates how the score is scraped by looking for a specific attribute in the HTML, as well as the review fetching. The rest of the attributes are added in the same iteration:

```

for url in yelp_urls:

    page = requests.get(url)
    soup = BeautifulSoup(page.content, 'html.parser')
    entries = soup.find_all("li", class_="margin-b5__373c0__3ho0z
border-color--default__373c0__1WK1L")

    for e in entries:
        # Score fetching
        scr = e.find(class_="margin-t1__373c0__1zX1r
margin-b1-5__373c0__jjw8Y border-color--default__373c0__1WK1L")
        if scr.find(attrs={"aria-label": "5 star rating"}):
            s=10.0
            label="POSITIVE"
        elif scr.find(attrs={"aria-label": "4 star rating"}):
            s=8.0
            label="POSITIVE"
        elif scr.find(attrs={"aria-label": "2 star rating"}):
            s=4.0
            label="NEGATIVE"
        elif scr.find(attrs={"aria-label": "1 star rating"}):
            s=2.0
            label="NEGATIVE"
        else:
            s=0
            label=0
        score_list.append(s)
        label_list.append(label)
        hotel_list.append("Caesars Palace")
        site_list.append("yelp")

    # Review fetching
    rev = e.find("span", class_="raw__373c0__tQAx6")
    review_list.append(rev.text)

```


For the Hotels.nl webscrapping function, modifiable arguments have been defined so that it can be reused in different ways in the future. The amount of reviews collected can be changed, as well as the url where it comes from. The score system on Hotels.nl works from 1 to 10, and the decimal point is separated by a comma, but it has been corrected to store it as a float without cohesion errors in the database. Again, to illustrate:

 nl.hotels.com/ho124363

**8,0** Zeer goed

## Nice Hotel

It's a nice place! Easy digital contactless check in. Rooms were spacious but beds were a little hard.

 **Michelle, Reis van 1 nacht**, 25 sep. 2021

Gecontroleerde Hotels.com-gastenbeoordeling



The criteria for labelling the sentiment from the reviews as positive is similar as the one took for the Yelp reviews. A score over 5,0 means positive sentiment, and a score under 5,0 means negative sentiment.

## 1.2 Column values

The dataset built contains the following columns:

- **score:** A float value, with one decimal digit, which is submitted by the user when posting the review. The range is from 0 to 10, both included. Some websites use a 5-stars rating system, but their value has been normalized to range 10 score within the fetching function, as we will see in the webscrapping functions.
- **Hotel\_Name:** String value. Contains the name of the hotel whose review is posted
- **review:** String value. Contains the text from the review. The longest review contains up to 888 words (4948 characters), but there is also short reviews, with fewer than 10 words.
- **label:** String value. Can be "POSITIVE" or "NEGATIVE". Reflects whether the client posted a review with a positive or negative sentiment. This column is the value that we want to predict given the review text, thus building a classifier model.
- **site:** String value. Contains the name of the webpage where the review comes from.

This is the head of how the dataframe will look like, once all the reviews have been stored into a variable:

	score	Hotel_Name	review	label	site
0	2.9	Hotel Arena	I am so angry that i made this post available...	NEGATIVE	Booking
1	7.5	Hotel Arena	No real complaints the hotel was great great ...	POSITIVE	Booking
2	7.1	Hotel Arena	Location was good and staff were ok It is cut...	POSITIVE	Booking
3	3.8	Hotel Arena	My room was dirty and I was afraid to walk ba...	NEGATIVE	Booking
4	6.7	Hotel Arena	Amazing location and building Romantic setting	POSITIVE	Booking

Once all the data is collected into their respective Dataframe, they can all be combined into one (using the Pandas function `pd.concat` ) and load it into the MySQL database.

# 2. Data Preparation

## 2.1 Insights before cleaning

Once the data is loaded into the MySQL database, we can fetch it through the use of Stored Procedures. Our first glance will be done by fetching all the reviews and using the Pandas `.describe()` method, which gives us the following data:

	index	score
count	803587.000000	803587.000000
mean	257547.936907	8.314578
std	148903.356003	1.608868
min	0.000000	1.000000
25%	128610.000000	7.500000
50%	257300.000000	8.800000
75%	386178.500000	9.600000
max	515737.000000	10.000000

Since the index column does not give any meaningful data, we can ignore it. The score column is not of much interest either, since it will not be used that much on this assignment.

However, this screenshot allows us to see how many rows does the dataset contains (803,587 rows), what the mean score is (8.314578) and how it is distributed.

But this procedure will not be essential to the project, since the most important one that we will be using will serve the purpose of fetching review samples, but not randomly, because it will give different results each time, and not from any section, because then they would not be mixed.

Instead, the procedure defined for this task, called “get\_reviews\_by\_module” fetches every nth review of the dataset, given n as a parameter, ensuring that it gets data from every section of the database.

```
12  DELIMITER //
```

13 • **CREATE PROCEDURE** get\_reviews\_by\_module(**IN** module **INT**)

14 **BEGIN**

15 **select** \* **from** hotel\_reviews **where** hotel\_reviews.index **mod** module = 0;

16 **END //**

17 **DELIMITER ;**

Henceforth, if we want a sample of 100 reviews, and we know that the dataset is roughly 800,000 rows long, we will specify a module of 8,000 ( $800,000/8,000 = 100$ ).

## 2.2 Data Cleaning

But before working with samples, it is important to ensure we clean the whole database by analyzing how the data is presented. For this task, we will be using different methods.

### 2.2.1 Removing empty reviews

By using the following line of code, we can sort the dataframe by the length of the review, thus easing the task of easing empty reviews and one-word reviews that do not give any information about the sentiment and need to be cleaned.

```
# See what reviews are too empty
df_sorted=df.sort_values(by="review", key=lambda x: x.str.len())
```

After analyzing this data, it has been concluded that most reviews with less than 12 characters do not give useful information, so they will be deleted from the dataset. For this task, a stored procedure called "remove\_empty\_reviews" has been created.

```
63 DELIMITER //
64 • CREATE PROCEDURE remove_empty_reviews()
65 BEGIN
66 DELETE FROM hotel_reviews where length(review) < 12;
67 END //
68 DELIMITER ;
```

### 2.2.2 Removing punctuation

By using regular expressions, we can indicate to the stored procedure that we only want the text to have alphanumeric characters, which, in other words, removes the punctuation. With the regular expression `[^0-9a-zA-Z]`, we indicate that we select every character that is NOT a number, a lowercase letter from a to z or an uppercase letter from A to Z. They will be replaced with an empty string.

```
70 DELIMITER //
71 • CREATE PROCEDURE remove_punctuation()
72 BEGIN
73 UPDATE hotel_reviews SET review = REGEXP_REPLACE(review, '[^0-9a-zA-Z ]', '');
74 END //
75 DELIMITER ;
```

### 2.2.3 Normalizing the words to lowercase

Words with different uppercase letters are counted as separate words, but they carry the same sentiment, so in order to avoid this issue, all words will be lowercase:

```
77 DELIMITER //
78 • CREATE PROCEDURE to_lowercase()
79 BEGIN
80 UPDATE hotel_reviews SET review = LOWER(review);
81 END //
82 DELIMITER ;
```

### 2.2.4 Removing stopwords with the help of WordCloud

It is also possible to remove useless information within the review texts, and it is possible to spot by the use of WordCloud:



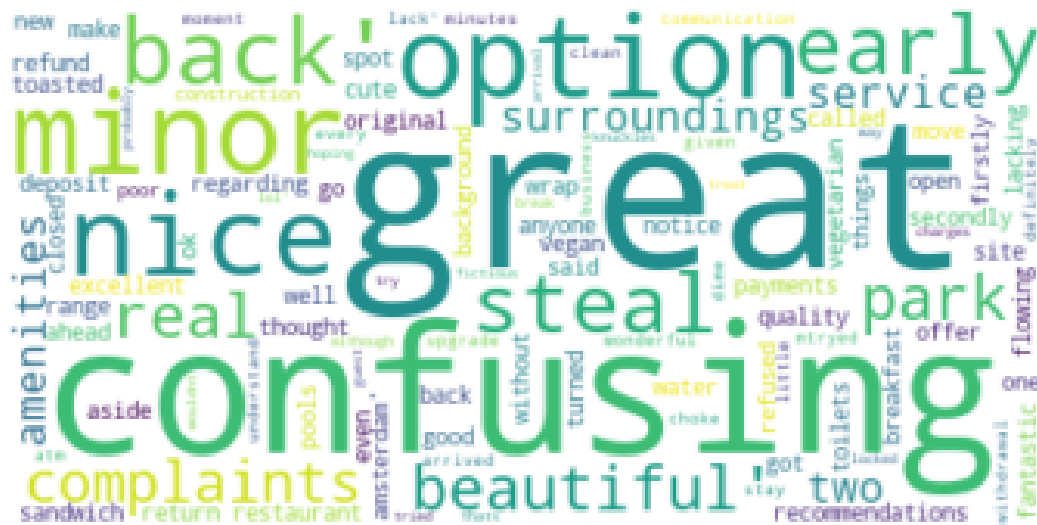
Many big words on the wordcloud are not relevant to the analysis we want to perform, so they will be added to the list of stopwords. A default set of stopwords is provided by the WordCloud library, which includes the gramatical stopwords. Here, the context stopwords will be added to this set, using the information given by this wordcloud.

```
# Default stopwords set
stopwords=set(STOPWORDS)
# Adding stopwords from the wordcloud
context_stopwords=["hotel","room","rooms","staff","upon","bit","payment","will",
"location","checkout","outside","check","food"]
stopwords.update(context_stopwords)
```

By cleaning the dataset with this set of stopwords:

```
df['review'] = df['review'].apply(lambda x: ' '.join([word for word in x.split()
if word not in (stopwords)]))
```

We now get the following wordcloud:



Whose big words carry more sentiment (great, nice, confusing, beautiful...) so the dataset is now better fit to train the model.

## 2.3 Insights after cleaning

Now that we have a more accurate version of what will be used in the model building, let us perform some visual Exploratory Data Analysis by using plots.

### 2.3.1 Positive/Negative review ratio by website

This store procedure allows us to view different pie charts for each website in the dataset, and informs about the ratio between positive and negative reviews on each webpage.

```

35 DELIMITER //
36 • CREATE PROCEDURE label_ratio_by_website()
37 BEGIN
38     SELECT table1.site,negat as NEGATIVE,posit as POSITIVE FROM
39     (SELECT site,count(label) AS negat FROM hotel_reviews WHERE label="NEGATIVE" GROUP BY site) AS table1
40 JOIN
41     (SELECT site,count(label) AS posit FROM hotel_reviews WHERE label="POSITIVE" GROUP BY site) AS table2
42 ON table1.site=table2.site;
43 END //
44 DELIMITER ;

```

If we transform the resulting DataFrame to a transposed version of itself, the plotting is quite straightforward, but for that we have to perform a couple of transformations:

	site	NEGATIVE	POSITIVE
0	Booking	350646	452827
1	Google	5	5
2	yelp	74	24
3	hotels.nl	2	4

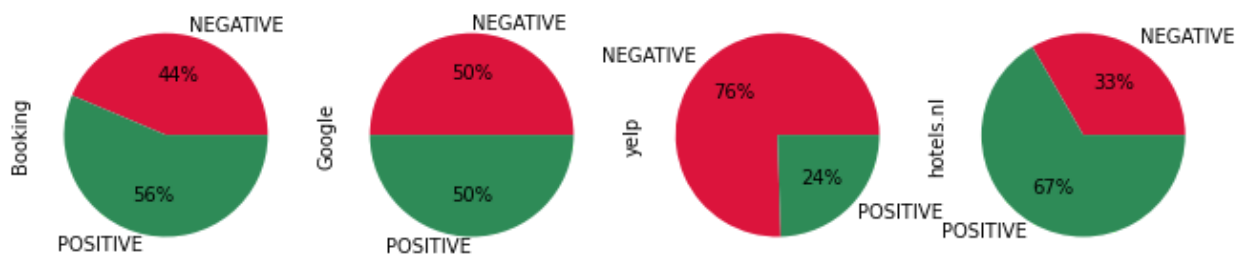
	site	Booking	Google	yelp	hotels.nl
NEGATIVE		350646	5	74	2
POSITIVE		452827	5	24	4

```
# Transpose, set sites as columns and remove the duplicate sites row
df_pie=df_pie.transpose().set_axis(df_pie.site, axis='columns').iloc[1:,:]
```

And now the pie chart can be programmed with just one line of code:

```
plot = df_pie.plot(kind="pie", legend=False, autopct='%1.0f%%', subplots=True, rot=1, figsize=(11, 3), title="Positive/Negative review ratio per review site", colors = ['#DC143C', '#2E8B57'])
```

Positive/Negative review ratio per review site



This plot could tell us about future troubles that we might have when the model has to train for a certain sentiment, since it is ideal to have equal amount of reviews of each sentiment. However, both numbers are big enough on their own so they won't give much trouble, also, the majority of reviews are from Booking, which is evenly distributed.

### 2.3.2 Positive/Negative reviews by score range

The following stored procedure groups the reviews by score range and label, and informs about how many reviews in each group:

```

34 DELIMITER //
35 • CREATE PROCEDURE score_ranges()
36 BEGIN
37 SELECT (CASE WHEN score >= 0 AND score < 3 THEN '1-2'
38           WHEN score >= 3 AND score < 6 THEN '3-5'
39           WHEN score >= 6 AND score < 9 THEN '6-8'
40           WHEN score >= 9 AND score <=10 THEN '9-10'
41         END)
42 AS scorerange, label, count(*) AS count
43 FROM hotel_reviews GROUP BY label, scorerange ORDER BY min(score),label;
44 END //
45 DELIMITER ;

```

However, this table format is not the most appropriate to plot the way we want to, so it will be rearranged to this format:



	scorerange	label	count
0	0-2	NEGATIVE	3353
1	0-2	POSITIVE	1696
2	3-5	POSITIVE	31104
3	3-5	NEGATIVE	46403
4	6-8	POSITIVE	186808
5	6-8	NEGATIVE	178162
6	9-10	POSITIVE	233252
7	9-10	NEGATIVE	122809

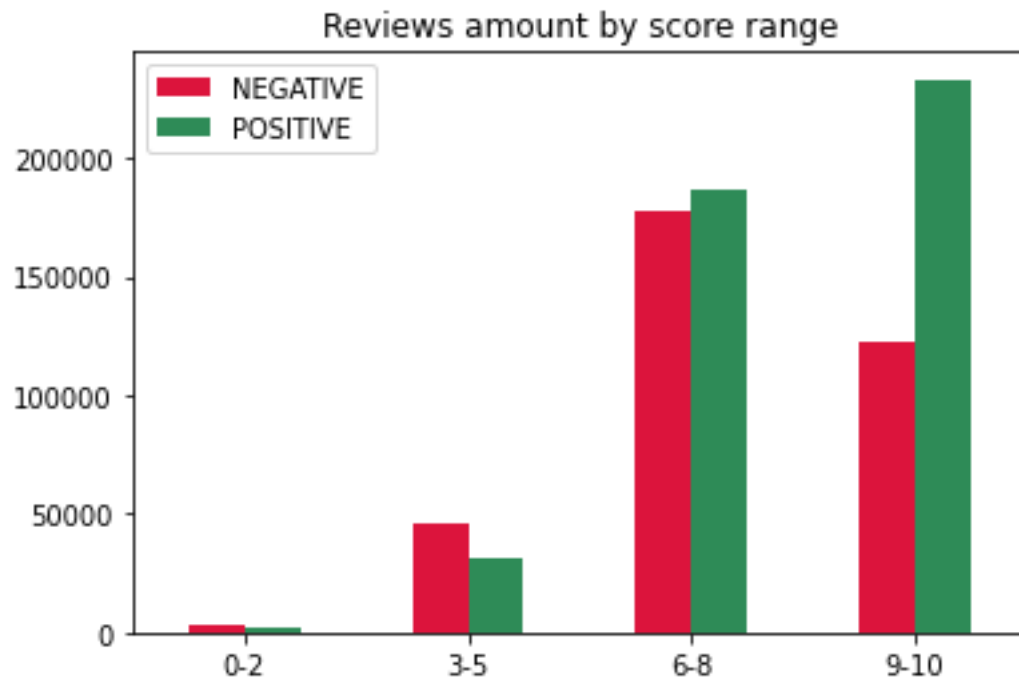
	NEGATIVE	POSITIVE
0-2	3353	1696
3-5	46403	31104
6-8	178162	186808
9-10	122809	233252

And the plotting can be done by simply calling the function `.plot`. The rest of parameters are aesthetics:

```

rev_score_range.plot(kind="bar",title="Reviews amount by score
range",rot=0,color={"NEGATIVE": "#DC143C", "POSITIVE": "#2E8B57"})

```



As expected, higher scores have more positive reviews, though the distribution is not totally predictable. In the 6-8 score range, many users have given negative review, even though the score is higher than 5. Even more surprising is the fact that up to 12,5k reviewers have given negative reviews despite having a score between 9 and 10, even after having cleaned the empty negative reviews.



## 3. Model building

Once we have understood the data and it is ready for the model, it is time to actually design it. All the models that will be used in this project will require vectorized text columns, that is, texts represented as numeric arrays, that are easier to feed to the machine algorithm.

### 3.1 Pre-processing

As expected, the POSITIVE/NEGATIVE column would not last until the end. Numeric values are always preferred, so first thing, we are transforming our target column into numeric values: 1 for "POSITIVE", 0 for "NEGATIVE":

```
df["label"] = df.apply(lambda df: 1 if df["label"] == "POSITIVE" else 0, axis=1)
```

#### 3.1.1 Tokenizing

The first step in pre-processing will be tokenizing the reviews. A token, by definition, is "an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing." (Manning et al., 2008)

```
df['review'] = df.apply(lambda x: word_tokenize(x['review']), axis=1)
```

#### 3.1.2 Stemming

The point of tokenizing the words is getting them ready to perform Stemming. Stemming is the process of reducing a word back to its simplest form, without any suffixes or prefixes. It must not be confused with Lemmatization, which also tries to derive words to its simplest form, but these forms must be actual words in the dictionary. In this case, we are using Stemming because we do not need grammatically accurate reviews, but functional ones, and the algorithm does not care if the word is correct or not. Also, Stemming is usually faster than Lemmatization.

```
porter = PorterStemmer()
df['review'] = df['review'].apply(lambda x: [porter.stem(y) for y in x])
```

Having tokenized and stemmed the reviews, the dataset now looks like this:

df[["review", "label"]].sample(5) ...		
	review	label
2438	[style, lobbi, classi]	1
23311	[bed, comfi, tea, coff, alway, plu]	1
62489	[includ, breakfast, offer, made, stay, pleasan...]	0
37621	[friendli, good, hyde, concert]	1
12458	[best, near, china, town]	1

### 3.1.3 Vectorizing

Vectorizing, as stated before, is the process of converting our reviews into a matrix containing numbering values. Each column of the matrix represents a word, and the rows represent the texts analyzed. Each time a word appears in a text, the cell corresponding that word will increase its count by 1. Let us see it with an dummy example:

Review 1: <b>"It was a nice hotel"</b>									
Review 2: <b>"The hotel was not nice at all"</b>									
Total words (also known as vocabulary): <b>[It, was, a, nice, hotel, the, not, at, all]</b> (9 words)									
	<b>It</b>	<b>was</b>	<b>a</b>	<b>nice</b>	<b>hotel</b>	<b>The</b>	<b>not</b>	<b>at</b>	<b>all</b>
Review 1	1	1	1	1	1	0	0	0	0
Review 2	0	1	0	1	1	1	1	1	1

As one can expect, the matrix will get huge when processing such a high number of reviews, and it will contain mostly zeroes. However, opposite to humans, this is the easiest way for machines to understand text.

This specific method of approaching Natural Language Processing (NLP) is called Bag of Words (BOW). This is what the function CountVectorizer uses, and it is widely used in text mining. However, here we are using TfidfVectorizer, a more sophisticated approach, that instead of directly counting a word when it appears in a document, it takes into account the proportions amongst words versus every document. We are gonna look into detail:

TFIDF is short for term frequency – inverse document frequency. That is, we have two different concepts to work on.

- Term frequency is a measure that tells us how frequently a word appears in each document.
- Inverse document frequency, on the other hand, tells us in how many document appears a single word. This ensures that very common words across all documents (such as "hotel", for instance) are less valuable, as they are usually not that relevant to the sentiment.

Both TF and IDF are multiplied for each word, and so forth we get the weighted matrix, which is usually more precise than the former.

```
vect=TfidfVectorizer(ngram_range=(1,2))
# Other params of Tfidf: max_df=[0.0,1.0] ; min_df=[0.0,1.0] ; max_features=int
X = vect.fit_transform(df.review)
y = df['label']
```

The TfidfVectorizer can also receive a number of parameters that can be configured so the model receives even more relevant words. We can tune the maximum and minimum document frequency (from 0 to 1), but after testing all models it has not shown any significant difference.

The max\_features parameter can be used to reduce the range of vocabulary used. Indicating 25,000, for instance, would only take into consideration the 25,000 most common words (highest DF) across the corpus. Tuning this parameter has not shown any enhancement either.

One way to see the size of our vocabulary is with the following line of code:

```
print("Vocabulary size: {}".format(len(vect.vocabulary_)))
```

Which prints a vocabulary size of 396,564 different tokens.

One parameter that has shown improvement on the accuracy performance, however, is the ngram\_range. This parameter allows the user to indicate how many words each token contains. In this case, we are using a range of 1 to 2 ngrams, which means that a sentence like:

**“The hotel was not nice”**

Will be vectorized as:

**1-grams:** [The, hotel, was, not, nice,

**2-grams:** the hotel, hotel was, was not, not nice]

Note that this can be very useful to detect negation, since otherwise, the token “not nice”, which is clearly a negative sentiment, would be lost and probably “nice” would be analyzed as a positive feature

#### 3.1.4 Dataset split

Now that we have X, our vectorized feature column, and y, the numeric target column, we could already give it to the ML algorithm and get the results. However, if we want to evaluate how well a model performs, it is common practice to split the data into two sets: Train and Test.

The reasoning behind is to train the algorithm with a certain subset of the data (X\_train, y\_train), and after the training is complete, we are going to run predictions with the data in X\_test. Then, we are going to check how accurate these predictions are. That is, comparing the predicted results with the actual results in y\_test.

As shown in the parameters, 30% of our dataset will be used for testing, the rest, for training. Stratifying the sample allows it to get even amounts of positive and negative reviews. The random state value is the seed generator for the split, so it can be reproduced in the future.

```
# Dataset split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
random_state=1,stratify=y)
```

Now, we are going to build the models, using these variables.

## 3.2 Model 1: Logistic Regression

### 3.2.1 Logistic Regression background

Logistic regression works in a similar way as linear regression. However, it does not predict continuous values, but instead it labels each prediction as 0 or 1, with an associated likelihood for each prediction.

By locating the data points on one side (positive) or each other (negative), a sigmoid function (S-shaped) is defined, where certain x-values will have either value 1 or 0, except for one point in the middle, which is where the algorithm will have to select randomly, since it has no inclination to one side or another.



Fig. SEQ Fig. 1\* ARABIC 1. Logistic Regression visualization

Some reviews are easier to classify than others. As shown in the example, reviews with words like “beautiful” or “perfect” are very frequent across positive reviews, hence a review with this words is likely to be catalogued as one of them, thus positioning clearly on one side or another.

However, other words, like “staff” are present in both positive and negative reviews, so they are not of much use to the algorithm. If the review contains many words without clear sentiment, the algorithm will locate the point closer to the middle, and might not be an accurate result.

### 3.2.2 Model performance

Running the model is quite straightforward, thanks to the sklearn libraries. We will run it with the parameter C=5 (we will get to parameters in the next section)

```
log_reg = LogisticRegression(C=5).fit(X_train, y_train)
y_predict = log_reg.predict(X_test)
```

For the evaluation of the model, we will use three metrics.

The first measure is accuracy. It measures how many data points were predicted correctly. Since the accuracy might vary based on the train/test split, the Crossfold Validation method makes 5 different random splits, and measures the accuracy on each one. Then we average the accuracy amongst them all.

```
# Crossfold validation with 5 folds (accuracy)
cv_results = cross_val_score(log_reg, X_train, y_train, cv=5)
print(np.mean(cv_results))

cv_results = cross_val_score(log_reg, X_train, y_train, cv=5) ...
... 0.9171969454829567
```

Then we are going to take a look at the Confusion matrix. It is one of the key concepts in evaluating the models, but by itself it is not quite common. Many data can be derived from it though, so it is important to understand it. It displays different combinations of predicted and actual values, as positive or negative sentiment.

In this example, 40% of the actual positive reviews were predicted correctly (True Positive), whereas 51% of the actual negative reviews were predicted correctly (True Negative). 4% of the positive reviews were labeled as negative (False Negative) and 3% of the negative reviews were labeled as positive (False Positive).

From these four numbers, by using simple adding and division, we can obtain:

- Accuracy  $(TP + TN / TP + TN + FP + FN)$ : Explained before.
- Precision  $(TP / TP + FP)$ : From all the reviews predicted as positive, how many are actually positive.
- Recall  $(TP / TP + FN)$ : From all positive reviews, how many were correctly predicted.
- ...

```
# Confusion matrix
confusion_matrix(y_test, y_predict)/len(y_test)

... array([[0.40051705, 0.0370886 ],
          [0.04454609, 0.51784826]])
```

We can deduct that our model has a harder time labelling positive sentiment, but it is a good score overall. In this particular context, falsely predicting positive or negative values is not as relevant as it would be in incorrectly predicting disease results, for instance.

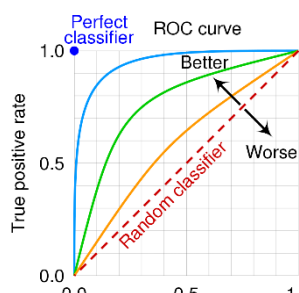


Fig. SEQ Fig. 1 ARABIC 2. ROC curve

Last but not least, let us apply the AUC score. First, we need to understand what the ROC curve is. A Receiver Operating Characteristic curve is a graph that is plotted with two parameters: True Positive Rate and False Positive Rate.

Both parameters can be calculated with the Confusion Matrix. Once the ROC curve is plotted, it is easy to see that we want the curve as close to the point (0,1) as possible, that would mean that all Positives are correct and none are FP.

The AUC, or Area Under Curve, is simply the integral function that measures the area under the curve, as the name says. It is preferred to be as close to 1 as possible, as well.

```
# Area under ROC curve
roc_auc_score(y_test, y_predict)

... 0.9180193070431786
```

The AUC in this model is 0.91, which is quite good, even suspicious, as the model might be overfitted, but we can test that later by running predictions on new data.

### 3.2.3 Logistic Regression Hyperparameter Tuning

Since it is the most efficient model, we will be using this model for demonstrating Hyperparameter Tuning, a technique that runs the model multiple times, using different parameters, and it is used to find the optimal ones to get the desired accuracy on the model.

First, we have to define the parameters that we want to tune in a dictionary, which will be called `search_space`. The parameters that are going to be tuned in Logistic Regression are “penalty” and “C”. C is called “the regulator”. By having a lower C, the likelihood of having an overfitting model is reduced by penalizing feature values that are too far away from the average (e.g. reviews with a sentiment too pronounced). This value can be calculated with different approaches, such as the L1 penalty, L2 penalty or Elastic net, which is a combination of both.

Then, the `GridSearch` function trains and evaluates the specified models with the indicated set of parameters, and runs it multiple times, storing the desired scoring methods, in this case, accuracy and `roc_auc`. It can also use Crossfold validation (`cv=2`).

```
search_space = {
    "C" : [10,5,1,0.1],
    "penalty" : ["l1","l2","elasticnet"],
    "solver" : ["newton-cg", "lbfgs", "liblinear", "sag", "saga"]
}

from sklearn.model_selection import GridSearchCV
# make a GridSearchCV object
GS = GridSearchCV(estimator = log_reg,
                  param_grid = search_space,
                  scoring = ["accuracy","roc_auc"],
                  refit = "accuracy",
                  cv = 2,
                  verbose = 2)

GS.fit(X_train, y_train)
```

The best parameters and score can be fetched directly used predefined methods

```
print(GS.best_estimator_) # get the complete details of the best model
print(GS.best_params_) # get the best hyperparameter values that we searched for
print(GS.best_score_) # score according to the metric we passed in refit
```

However, it is more interesting to build a `DataFrame` with the results ranked, because from there, we can make choices based on accuracy and efficiency. Sometimes, the second most accurate model is almost as good as the first and it is quite faster than the former, and we would not be able to make that choice if we restrict to the best score.

```
df_gridsearch = pd.DataFrame(GS.cv_results_)
df_gridsearch[["mean_fit_time","params","mean_test_accuracy"]].sort_values("mean_test_accuracy",ascending=False).head()
```

	mean_fit_time	params	mean_test_accuracy
12	4.386832	{'C': 5, 'penalty': 'l2', 'solver': 'newton-cg'}	0.914697
14	0.968749	{'C': 5, 'penalty': 'l2', 'solver': 'liblinear'}	0.914697
13	8.629278	{'C': 5, 'penalty': 'l2', 'solver': 'lbfgs'}	0.914683
4	9.127859	{'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}	0.913461
3	5.847206	{'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}	0.913418

In this case, it is more logical to take the second parameter selection, as the accuracy is almost the same, and the required time is significantly faster.

This method of hyperparameter tuning will be used for further models, but will not be explained in depth again, only for the choice of parameters.

### 3.2.4 Model saving and predicting

The model can be stored and loaded using the library Pickle:

```
# Save model
filename = 'log_reg_model.sav'
pickle.dump(log_reg, open(filename, 'wb'))
# load the model from disk
loaded_log_reg = pickle.load(open(filename, 'rb'))
```

This model can be loaded at any time without having to wait for the training time. We can also load it and make a quick prediction right away with a new review:

```
one_pos = ["Excellent service from all. Clean hotel. Room was phenomenal. Shower  
and beds. They rock. Stay here cannot beat it. Highly recommend. Lots of food  
choices. Carry out delivery available. Staff very helpful. Even have tallvehicle  
parking."]

print("Pos                                prediction:                                {}".format(loaded_log_reg.predict(vect.transform(one_pos))))
```

```
Pos prediction: [1]
```

As we can see, the model correctly predicted a positive sentiment review.

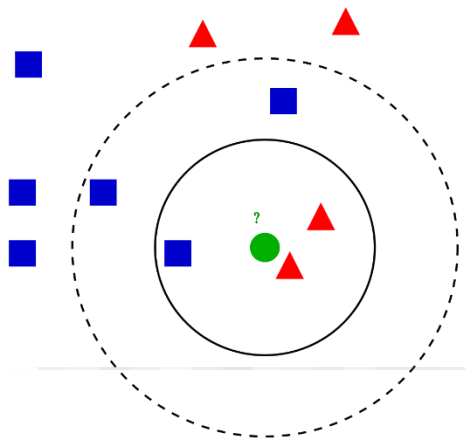
### 3.2.5 Conclusions

One reason to choose this model could be its simplicity, it does the job and gives a good result, also using a reasonable amount of computing time. However, it is not very flexible for other kind of problems and cannot be tweaked as much as other models. It could perform very badly on more sophisticated problems, but for our dataset, it gets the job done.

## 3.3 Model 2: K-nearest neighbor

### 3.3.1 KNN background

K-Nearest Neighbour is one of the first classifier that is usually explained to students, due to its simplicity.



The principle behind this algorithm is that given classified clusters of points (red and blue), in order to decide to which group does a new point belong to, the nearest neighbors take a majority vote. The K in K-nearest neighbors indicates how many data points are taking the vote. In the image, if K=3 (inner circle), we have two votes for red and one vote for blue. Hence, the new point would be classified as red.

However, if K=5, the majority vote would make the new dot be classified as blue. That is when the KNN becomes very sensitive to the K parameter.

But K is not the only parameter in the algorithm. The way the distance is measured can also be modified (Euclidean/Manhattan), as well as making the votes from closer points to have more weight.

### 3.3.2 KNN performance

```
knn = KNeighborsClassifier(n_neighbors=7,p=2)
knn.fit(X_train, y_train)

# First evaluation
knn_score = knn.score(X_test, y_test)
```

The KNearestNeighbor library from sklearn provides a function `knn.score()` which measures the score of the KNN algorithm:

```
knn.score(X_test, y_test)
✓ 1m 2.2s
0.6160882967087601
```

It gives us a score of 61.6% accuracy, which is not very good, but given the short time it took (it is usually quite bigger for other algorithms), it is another choice consider (but not necessarily a good one)

The confusion matrix sheds some light on why this low accuracy:



```
y_predict = knn.predict(X_test) ...
array([[0.09654967, 0.34105598],
       [0.01504756, 0.54734679]])
```

Apparently, our KNN algorithm has quite trouble when predicting positive reviews. This might be due to the higher amount of negative data points, hence the majority vote is usually negative if the points are mixed across the possible values.

Finally, we are introducing a new measuring tool: The F1-score. This function is the harmonic mean of precision and recall, in other words,  $2 * ((\text{precision} * \text{recall}) / (\text{precision} + \text{recall}))$ .

```
f1_score(y_test, y_predict) ...
0.7545462852965367
```

In this case it is significantly bigger than the accuracy, but still nothing to be proud of. With some fine tuning we might be able to increase it a bit.

### 3.3.3 KNN Tuning

For this algorithm, whose main parameter is K, we are also going to look for other parameters, such as leaf\_size, that affects performance by selecting the amount of neighbour candidates, or p, that indicates Manhattan or Euclidean distance, respectively

```
search_space = {
    "n_neighbors" : [3,5,7,9],
    "leaf_size" : [10,30,50],
    "p" : [1,2]
}
```

With 24 combinations on a cv=2 Crossfold, it takes roughly 48 minutes to compute the DataFrame with parameter tuning:

	mean_score_time	params	mean_test_accuracy
9	30.732409	{'leaf_size': 30, 'n_neighbors': 3, 'p': 2}	0.620007
1	30.180858	{'leaf_size': 10, 'n_neighbors': 3, 'p': 2}	0.620007
17	30.429766	{'leaf_size': 50, 'n_neighbors': 3, 'p': 2}	0.620007
0	45.869926	{'leaf_size': 10, 'n_neighbors': 3, 'p': 1}	0.618941
16	49.385979	{'leaf_size': 50, 'n_neighbors': 3, 'p': 1}	0.618941

We will select the second row of parameters. We can now set these parameters, run the model and save it with Pickle.

### 3.3.4 Conclusions

Despite having a short computing time in our case, KNN is not usually an efficient choice, since it has to calculate the distance to every point in the dataset. It works well with a reduced amount of features though, so it's a good choice for our project. However, given the low accuracy, it is not a good model choice at all, having many others available, but can be used to easily understand how the predictions are calculated, and how they change by tuning the parameters.

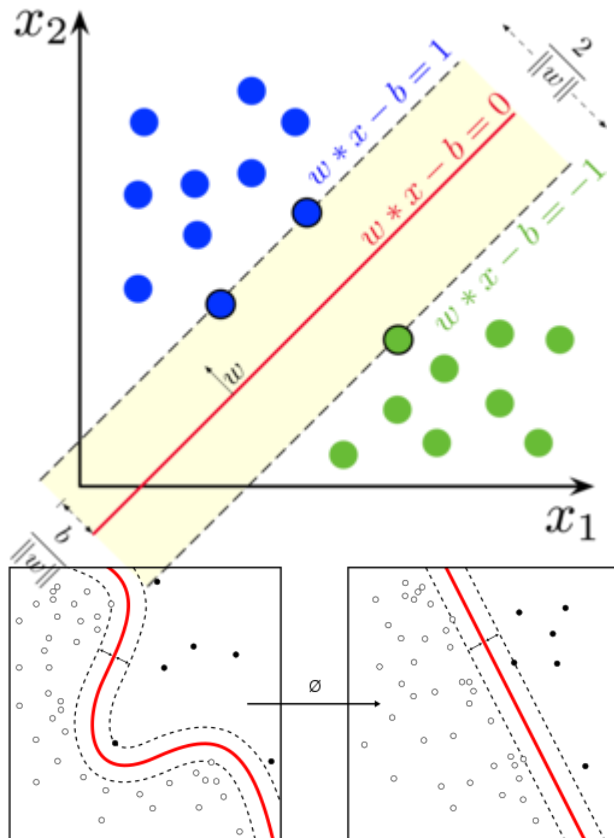
A possible way to improve the accuracy of this classifier could be changing the way the data is prepared. Since we are using n-grams, it might be interesting to exclude the removal of the word "not", since tokens

like “not good” or “not come” are strong negative indicators that may improve significantly the model's accuracy.

## 3.4 Model 3: Support Vector Machine

### 3.4.1 SVM background

SVM is an algorithm that can be used for classification or regression. It is based around the concept of finding a hyperplane that separates the data into groups (or regions in space), where depending on where the point is located, it belongs to one group or another. What divides the space is called a hyperplane.



As seen on the graph, there are several possible hyperplanes. However, the algorithm always provides the one with the maximum margin. That is, the hyperplane that is further away from the least extreme observations in both classes.

The SVM algorithm from sklearn also possesses the regularization parameter, with default value 1, that we will tune later, in order to ensure the proper fit of the model.

Additionally, depending on the data, sometimes a straight hyperplane is not good enough to split the data, hence we are facing a nonlinear classification. For this case, the “kernel” feature is modified, where instead of a linear one, a polynomial, sigmoid or hyperbolic kernel can be used. When they are represented in a 2D dimension, they appear as curvilinear functions, but in reality, they use a higher dimension than the data points. Hence the word “hyperplane” makes even more sense.

### 3.4.2 SVM performance

```
svmach = svm.SVC(C=1.0, kernel='linear')
svmach.fit(X_train, y_train)
y_pred = svmach.predict(X_test)

# Confusion matrix
confusion_matrix(y_test, y_predict)/len(y_test)

# Area under ROC curve
roc_auc_score(y_test, y_predict)
```

Let us take a look at how the model performs:

<code>confusion_matrix(y_test, y_pred)/len(y_test)</code>	<code>roc_auc_score(y_test, y_pred) ...</code>
<code>array([[0.40389778, 0.03370787],        [0.04573929, 0.51665507]])</code>	<code>0.9208212451394131</code>

Before doing any parameter tuning, we already have the best results so far, at the cost of more computing power.

### 3.4.3 SVM Tuning

As stated in the model background, we can edit these two parameters in order to find the most adequate combination. However, since the model takes longer to train, the search will require the computer working overnight.

```
search_space = {
    "C" : [100,10,1,0.1,0.01],
    "kernel" : ["linear","poly","sigmoid"]
}
```

	mean_fit_time	mean_score_time	params	mean_test_accuracy
3	209.015429	47.271024	{'C': 1, 'kernel': 'linear'}	0.915535
5	192.906426	41.711081	{'C': 1, 'kernel': 'sigmoid'}	0.907750
6	119.720330	69.124551	{'C': 0.1, 'kernel': 'linear'}	0.903233
8	102.564268	74.384993	{'C': 0.1, 'kernel': 'sigmoid'}	0.902239
0	374.942678	55.689677	{'C': 10, 'kernel': 'linear'}	0.898943

It seems like the default settings were the right choice in this case, so we will leave them as they are and the current evaluation is accurate.

### 3.4.4 Conclusions

SVM is a very powerful model and fitted for our current task. The required time is quite higher compared to the other three, but since there is enough separation between the classes, the splitting is accurate to give a high accuracy.

## 3.5 Model comparison and final conclusions

Model	Fitting time for train set	Predicting time for test set	Average accuracy
Logistic Regression	4.4s	0.3s	0.9147
K-Nearest Neighbor	0.6s	1min 3.7s	0.62

SVM	14min 14.8s	1min 42.7s	0.9155
-----	-------------	------------	--------

Having analyzed the pros and cons of each model, and with this table displaying the main results of each model, if a client is fully interested in accuracy, SVM is the way to go. However, since Logistic Regression is a simple, efficient model that usually does not give that much accuracy, it is a wise choice to go with it, because the way that the data is pre-processed allows Logistic Regression to be a powerful tool in this problem. KNN, on the other hand, is nowhere near being an appropriate algorithm for our dataset, at least in the way it has been prepared.

Nonetheless, there are many classifier models that have not been even tested for this dataset, such as Naïve Bayes or Random Forest. They are considered very powerful algorithms, and they usually take more computing time, but feel free to experiment with them as well. They tend to perform better on sentiment analysis, like this case, so it might be worth the try, but for now, these models get the job done.

Let us do one last set of predictions, applying the same reviews to all models:

```
pos_test=["It was a pleasant stay at the hotel. Covid rules were enforced, which
reassured us of our safety. The (bath)room is overall clean and the staff were
friendly. Kudos to the gentleman at the restaurant for bringing the croissants to
our room after a mistake when placing an order"]
neg_test = ["This is the dirtiest hotel we have ever been to. Everything was
dirty. Bedsheets and Towels even after the fourth change, every new one had dirty
spots. Dust everywhere, Stains on the sink, cups, glasses ... We left one day
earlier because my boyfriend did not want to spend his birthday in this messy
hotel. We felt very uncomfortable even though it is a hotel with four stars."]
```

```
Pos prediction by log_reg: [1]
Pos prediction by knn: [1]
Pos prediction by svm: [1]
Neg prediction by log_reg: [0]
Neg prediction by knn: [1]
Neg prediction by svm: [0]
```

The positive review was correctly guessed by the three models, but the negative one was incorrectly guessed by the KNN algorithm, which also showed the lowest accuracy score.

This satisfies the results found, so now we can go into any hotel review set and get their sentiment with a margin of error that we also know.

# Bibliography

## Model theory:

Manning, C. D., Raghavan, P., & Schütze, H. (2018). *Introduction to information retrieval*. Cambridge University Press. <https://nlp.stanford.edu/IR-book/>

Shalev-Shwartz, S., & Ben-David, S. (2019). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

## Documentation and usage:

*Scikit-Learn library documentation*. scikit-learn. (n.d.). Retrieved October 25, 2021, from <https://scikit-learn.org/stable/>.

Misheva, V. (n.d.). *Sentiment Analysis in Python*. DataCamp. Retrieved October 25, 2021, from <https://app.datacamp.com/learn/courses/sentiment-analysis-in-python>.

## External figures:

Fig. 1: <https://thenounproject.com/term/binary-logistic-regression/2424489/>

Fig. 2: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic#/media/File:Roc\\_curve.svg](https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roc_curve.svg)

Fig. 3: [https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm#/media/File:KnnClassification.svg](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg)

Fig. 4: [https://en.wikipedia.org/wiki/Support-vector\\_machine#/media/File:SVM\\_margin.png](https://en.wikipedia.org/wiki/Support-vector_machine#/media/File:SVM_margin.png)

Fig. 5: [https://en.wikipedia.org/wiki/File:Kernel\\_Machine.svg](https://en.wikipedia.org/wiki/File:Kernel_Machine.svg)

All images are used under the appropriate CC license