Graduate Programme

Major: Advanced Analytics - Big Data

Roberto Sannazzaro
RS79240

Natural Language Processing and Text Classification based on the Yelp Dataset

Master's thesis written in the
Collegium of Economic Analysis

Under scientific supervision of
Dr. Jarosław Józef Olejniczak

Warsaw 2019

Table of Contents

1 Introduction

In the era of social media, where users can create, upload and interact with content; businesses of any size and types profit a lot from being online. Among many reasons for that, are marketing purposes. It is estimated that only in the U.S. 77.6% (Mansfield, 2018) of businesses are on social media. The power of social media does not only give to those businesses the possibility to promote themselves, but also the possibility for the consumers to be more informed and perform decisions based on data and feedbacks from other people on data and feedbacks based from previous consumers. One of the most common feedback systems used to evaluate are the five-star system and the review.

A Survey sponsored by Zendesk (Gesenhues, 2013) shows that 90% of buying decisions are influenced by reviews. This makes any feedback system a crucial part of every business online presence, as it builds trust between business and clients.

One of the fields in which feedbacks and reviews are crucial is the food and beverage industry, for which one of the largest platforms where to share reviews and opinions about places is Yelp. This portal represents a crowd-sourced review platform. As to December 31st, 2018 Yelp accounts a cumulative number of 177 millions of reviews (Yelp, 2018) for which 19% are restaurants.

According to an Harvard study (Luca, 2016) a one-star review on Yelp can lead to a 9% decrease in revenue and 57% of users affirm that they avoid restaurants that have negative reviews or ratings.

This means that for a business paying attention to their online persona is a crucial step in their marketing process, and reviews can be a very important source of data, for both the customer and the businesses.

Beside the overall statistics provided by Yelp to business managers, a deeper analysis using modern techniques can be made. Extraction of plain text reviews might offer a lot of important information that may help a business to improve. All that is possible by implementing a series of techniques called Natural Language Processing.

1.1 Scope

The goal of this study is to apply a series of natural language processing (NLP) techniques to plain text reviews in order to find actionable insights and predictions based on automating the labelling of positive and negative reviews. Moreover, a comparison between different classification techniques is made in order to understand which one is the most convenient for this study, the comparison is made following three criteria: the resources needed, the time to train a model and the precision based on the metrics defined in chapter 3.6.

All the results from those analyses are useful under a business context and can be used xfor a data-driven decision-making approach. It is estimated that for every negative review that one business receives (1star), it has to get four 5-stars reviews to keep an average of four and half star. (Anon., 2019)
Those data along with the previously mentioned Harvard study suggests that it is very important to understand the content of negative reviews in order to have a positive impact on the revenues.

The dataset used in this study is called Yelp Dataset, it contains around six hundred thousand labelled reviews from different kind of businesses in different U.S. locations. Each review is labelled with a one-to-five stars feedback which makes this collection of data an ideal starting point for the application of machine learning and deep learning classification techniques such as Logistic regression, Naive Bayes, Support Vector Machines, Decision Trees and Neural Networks.

2 Natural Language Processing

The natural language processing field have its roots back in 1948 when Alan Mathison Turing proposed a paper named Intelligent Machinery which gave existence to the Turing test (Turing, 1950).
The Turing test evaluates how good a machine can resemble human behavior; it consists of an evaluator (a human) which judges natural language conversations between another human and a machine. Both parts are speared from each other and confined in a single communication channel (such as a keyboard and a monitor), if the evaluator is not able to distinguish from the human and the machine the test is said to be passed.

Few years later, in 1954 during the cold war, the Georgetown – IBM experiment was conceived. The Georgetown experiment involved machine translation from Russian to English based on six rules. (Hutchins, 2004) This approach did not take account of any relations or sentence analysis to understand the sentence structure, but it followed a "hard coded" approach, where each word has equivalent rules and steps associated with it.

Till the 80's most of the natural language processing systems were only based on a set of strictly hard coded and handwritten rules, but, during the 80's two important factors came in place:

- The introduction of machine learning algorithms for language processing
- The increase in computational power (Moore's law)

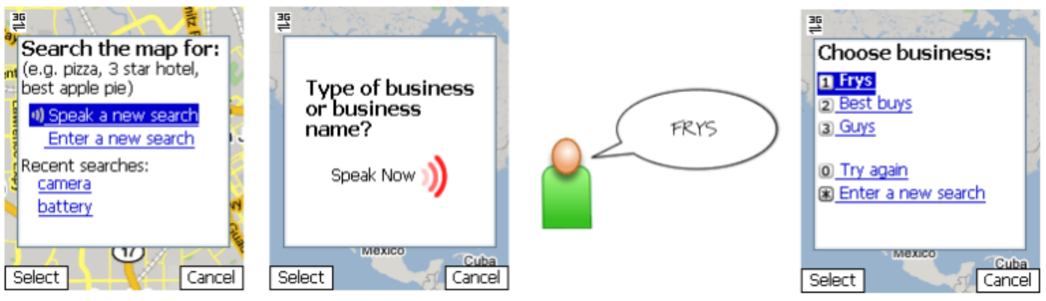
Those early models consisted mainly in the application of decision trees, producing complex system of "if-then" rules, but at the same time a different approach came in place: the statistical approach. The statistical approach, rather than defining rules by complex and written schemes introduced the weighting system, applying to each input feature a weight that describe the word importance in a specific context.

2.1 Natural language processing nowadays

One of the most notable and first achievement for the Natural language processing field in the 21st century is IBM Watson; a computer system originally designs to compete in the American quiz show *Jeopardy!* The system took part in the show in 2011, winning 1 Million dollars (Markoff, 2011). Currently IBM Watson expanded its abilities by being successfully used in different fields such as healthcare, HR and education (IBM, 2018).

Previously to IBM Watson, in 2008 Google introduced a system that can convert the user vocal input into a search query, it was applied in the Google Maps for Mobile app. (Schalkwyk, et al., 2010) It is important to state that this system involved natural language understanding only, the system did not provide any answer based on context, but only the transcription of the user input.

Figure 2.1 Google Maps for Mobile, with voice interface.



Source: (Schalkwyk, et al., 2010)

One of the first consumer application of a natural language processing system, able to answer to questions based on content and context, to query data to the internet and provide structured answers is Siri, developed by Apple in 2011 as a vocal assistant for the iOS operating system.

Nowadays, natural language processing offers an array of solutions for consumers as well, dealing with all the activities in which computers have to deal with human language: this varies from speech recognition, natural language understanding, and natural language generation.

Those techniques are applied in many different fields: Vocal assistants such as Amazon Alexa or Google Home give the possibility to the user to book a taxi, check the news or even order a pizza (Wong, 2017) just by stating it by voice. Up to December 24, 2018 52 millions Google Home devices (Kinsella, 2018) and 100 millions Alexa were sold across the globe (Bohn, 2019).

It is worth to mention that while a user asks for any third-party goods or services via a vocal assistant the choice of the provider of the final goods or services falls on the provider of the vocal assistant, empowering their bargain power toward suppliers and strengthening their position toward.

3 Research method and theoretical background

In order to deliver the scope of this study, different techniques are implemented, one of the first analyses necessary to understand the dataset is the exploratory analysis. The next step is to perform the classification of positive and negative reviews based on labelled data and the training of different classifiers.

3.1 Exploratory analyses

This phase focuses on giving some basic insights about the dataset, more precisely it will:

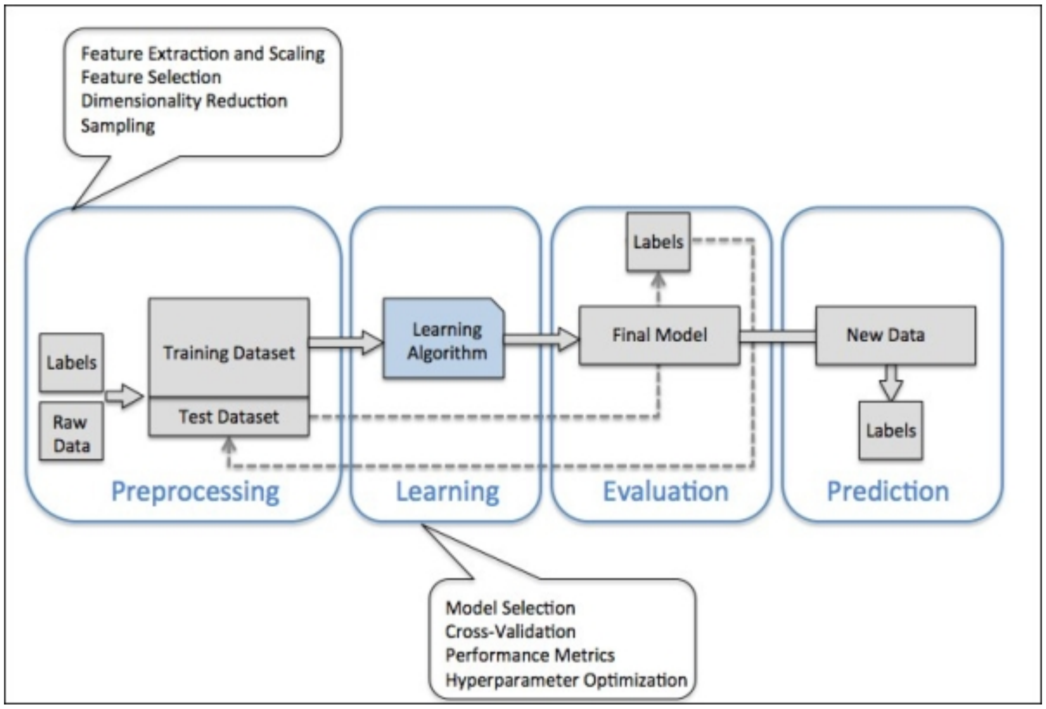
- Count and mean of reviews per place: this metric provide an insight on which are the most reviewed places.
- Check if the dataset is balanced: this can be achieved by splitting the total number of reviews in two groups based on the condition number of stars ≥ 3.
- The occurrence of positive reviews for a selected place: this metric gives a view on how a certain business keeps quality over time.
- The occurrence of positive reviews over time: this metric can give an insight on trend, seasonality and time series components in Yelp's reviews.
- Categories of reviewed businesses: this metric answers to the question "What categories of businesses are succeeding the most?"
- The categories of businesses which has the highest number of negative reviews: this metric gives a powerful tool to potential competitors as it exposes business categories which needs an improvement, revealing a gap in the market.

- Most frequent word in positive and negative reviewed businesses: this metric helps to understand which are the main keywords in positive and negative reviews.

3.2 Modelling with binary classification

Building a binary classifier for text classification is a delicate process that is made up of different phases and methodologies. According to Raschka, (Sebastian Raschka, 2017) a classifier can be built following this framework:

Figure 3.1 A roadmap for building machine learning systems



Source: (Sebastian Raschka, 2017, p. 11)

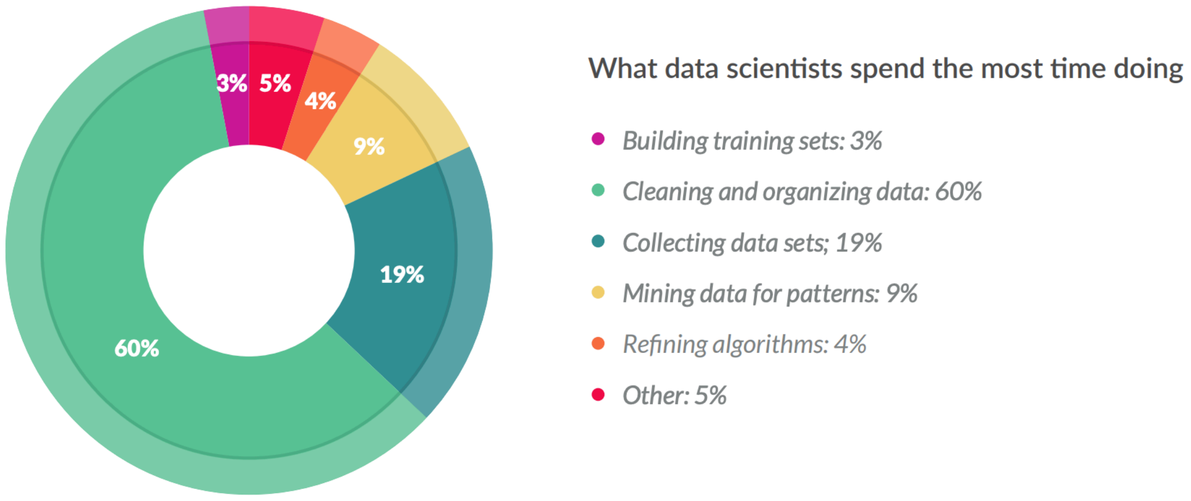
The workflow described in figure 3.1 is implemented in this study, although the last phase, the prediction, is out of the scope if this study. The models are evaluated on the test set which consists in the 20% of the dataset.

3.3 Preprocessing

The first phase, also called feature engineering, consists in a series of pre-processing techniques in order to transform raw data (plain text) in machine understandable format, this is a mandatory

phase because it is not possible to fit raw text to a machine learning algorithm, in addition there are different ways to do this operation, each of them leading to different results and each of them having their pros and cons.
In machine learning and data science this phase is one of the most important, as the way an algorithm is fitted makes the difference between a successful model and a model that is not able to capture anything from the data. Moreover, the preprocessing is the phase which data scientist spend most of their time (CrowdFlower, 2016).

Figure 3.2 Piechart from CrowdFlower's report



Source: (CrowdFlower, 2016)

In the NLP field this phase contains different techniques created *ad oc* for text pre-processing. The first step consists of reducing the *noise* contained inside the raw text, the term *noise,* should be understood as every character that does not provide any meaningful information such as HTML or XML tags, headers and characters not providing any information useful for the study.
The second step of this phase, according to KDnuggets (Mayo, 2018) consist of the tokenization, this step splits any long string of text into smaller parts, called tokens. A token corresponds to a single word contained into a document (corpus). Every single review is represented by a list of tokens. The comparison of text before and after this phase is shown in table 3.1.

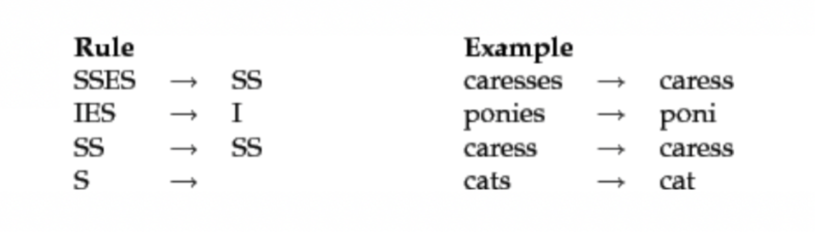
Table 3.1 Before and after the tokenization process

Before applying tokenization:	After tokenization is applied:
[‘In the midway of this our mortal life, I found me in a gloomy wood, astray Gone from the path direct: and e'en to tell It was no easy task, how savage wild That forest, how robust and rough its growth, Which to remember only, my dismay Renews, in bitterness not far from death.’]	['In', 'the', 'midway', 'of', 'this', 'our', 'mortal', 'life', ',', 'I', 'found', 'me', 'in', 'a', 'gloomy', 'wood', ',', 'astray', 'Gone', 'from', 'the', 'path', 'direct', ':', 'and', "e'en", 'to', 'tell', 'It', 'were', 'no', 'easy', 'task', ',', 'how', 'savage', 'wild', 'That', 'forest', ',', 'how', 'robust', 'and', 'rough', 'its', 'growth', ',', 'Which', 'to', 'remember', 'only', ',', 'my', 'dismay', 'Renews', ',', 'in', 'bitterness', 'not', 'far', 'from', 'death', '.']

Source: own elaboration

The third step in the standard text pre-processing workflow is the stemming.
In the natural language processing we use different forms of words such as “do”, “does”, “doing”, all that due to gramatical reasons. In order to be able to process the text smoothly it is neccesary to reduce this noises coused by inflection by transforming all the declined words into more basic forms.
One of the most popular (Jivani, 2011) stemming algorithm is the Porter Stemmer, it consists in a series of rules applied to each token, as shown in figure 3.3:

Figure 3.3 Stemming algorithm



Rule			**Example**		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

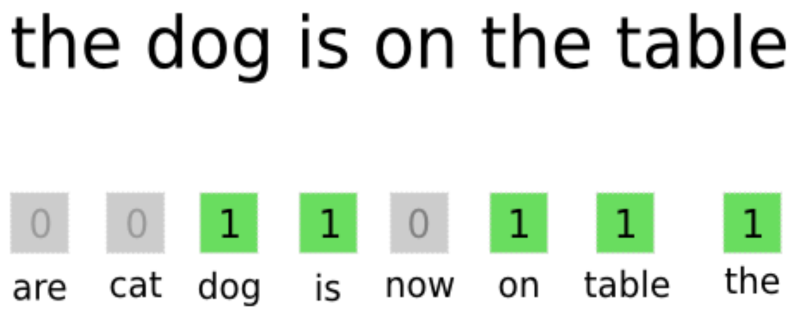
Source: (Porter, 1980, p. 135)

Following those pre-processing phases, it is finally possible to convert the text features matrix into a numerical feature, in other words: vectorization.
There are two main ways to convert string features into numerical features:

- Bag of words
- TF-IDF

The bag of words is the simplest model that exists, it is called bag of words because it does not take into account the order in which words appear, it only takes in account if a word is present or not in the corpus. This model builds the numerical features matrix by counting how many times a word appears in each review, in this way it is possible to compare how similar a document is to another one. This is a unigram approach, because only one word at a time is taken into account, so words such as *"New York"* or *"Soviet Union"* won't be taken into account as a single word.

Figure 3.4 Bag of words model



Source: (Kassabgi, 2017)

When using this approach, a very common problem might occurs: words such as *"the", "it", "to", "is"* which are words that occurs very often in a document get a really high score, this can lead to noise. A strategy for avoiding this problem is to remove the stop words from the corpus: a lot of NLP libraries that comes for different programming languages already provide a list of stop words which can be subtracted from the corpus.

The TF-IDF model, which stands for Term Frequency-Inverse Document Frequency. It is able to tell how important a word is in the corpus. This model contains two concepts: Term Frequency and Inverse Document Frequency.

Term Frequency is defined as how frequently a word appears in the corpus. As each sentence in the corpus might not be the same length, it may be possible that a word appears more often in a long sentence compared to shorter. Term frequency can be defined with the following formula:

[3.1]

$$TF = \frac{No\ of\ time\ word\ appears\ in\ the\ document}{Total\ No\ of\ words\ in\ the\ document}$$

Inverse Document frequency is the second concept behind the TF-IDF model. It is based on the principle that less frequent words are more meaningful and important. IDF is represented by equation 3.2:

[3.2]

$$IDF = \ log_{10} \frac{Number\ of\ documents}{Number\ of\ documents\ in\ which\ word\ appears}$$

The TF-IDF model consists in the product of the two components described above and it can be expressed as following:

[3.3]

$$w_{i\ ,j} = tf_{i,j} \log\left(\frac{N}{df_i}\right)$$

$$tf_{i,j} = number\ of\ occurrences\ of\ i\ in\ j$$

$$df_i = number\ of\ documents\ containing\ i$$

$$N = total\ number\ of\ document$$

3.4 Machine learning models

One of the most common algorithms to solve binary classification problems is the Logistic Regression. The logistic regression algorithm predicts the probability for an observation to fall into a category, it is a generalized linear model similar to the linear regression and it is very often used as baseline model.

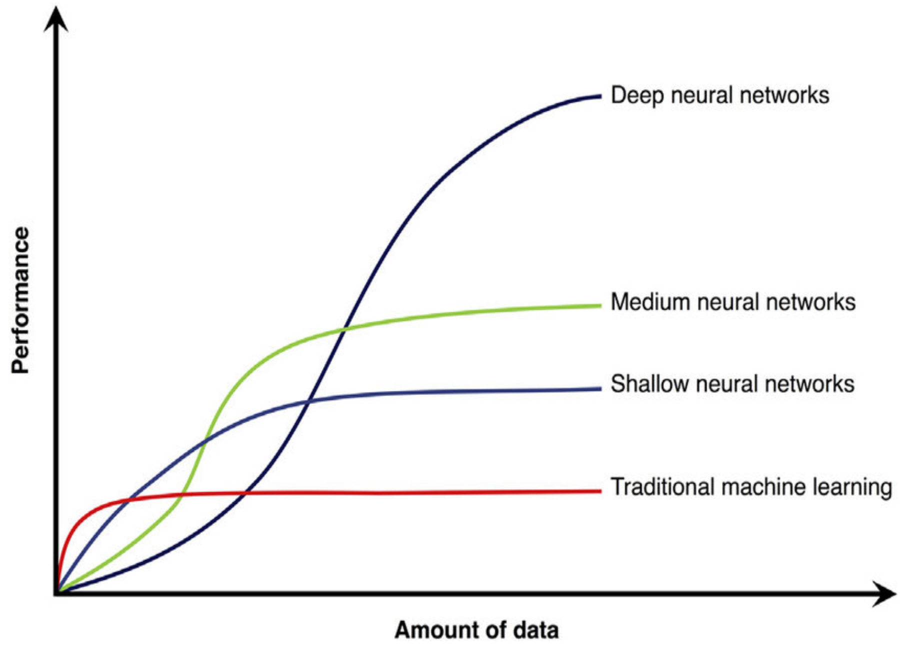
According to scikit-learn (Buitinck, et al., 2013), one of the biggest machine learning libraries available for Python, in order to predict a categorical outcome with text samples the Naïve Bayes is recommended, this classifier is based on the Bayes theorem and it is particularly suited for dataset with a large number of dimensions. Moreover, other common machine learning algorithms such as random forest or Support vector machine are applied.

The random forest classifier consists in an ensemble of decision trees and it is one of the most used algorithms in solving Kaggle's competion, together with Support Vector machines (Goldbloom, 2016).

3.5 Deep Learning models

While the statistical and machine learning techniques can give good results in an easy way to interpret, it is important to note how deep learning techniques, such as neural networks keep increasing performance when increasing the training set size. Moreover, machine learning algorithms requires very often hyperparameters tuning.

Figure 3.5 Performance of neural networks compared to traditional machine learning



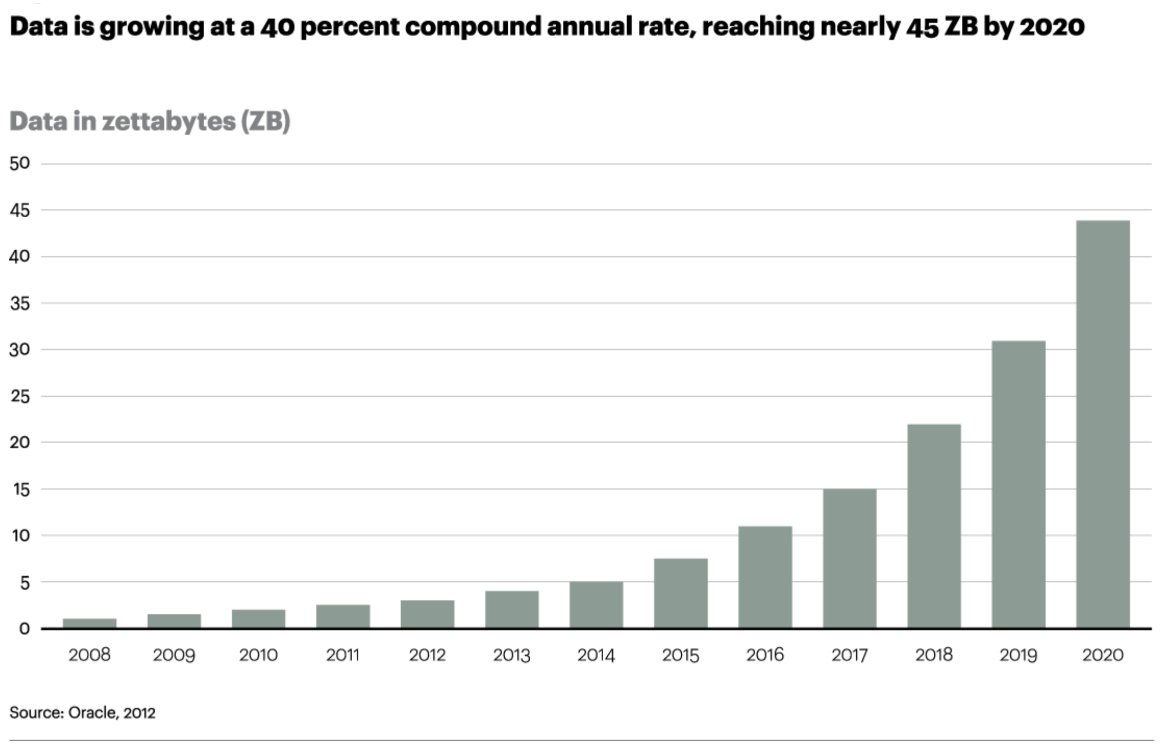
Source: (Tang, et al., 2018, p. 123)

In the last years those techniques became very popular because of the following factors:

- An increase of available data
- An increase in computational power
- The possibility to train neural networks on GPU reducing drastically training time

According to Oracle data is growing by 40% a year, reaching 45ZB by 2020:

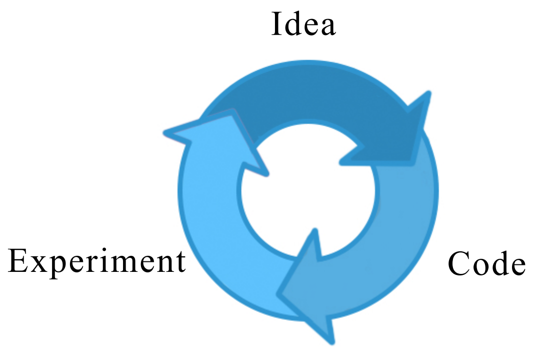
Figure 3.6 Growing rate of data



Source: (Subramanian & Palaniappan , 2015, p. 1)

Moreover, computational power, and the easy access to it contributes in the diffusion of those deep learning techniques along with the diffusion of an iterative method that characterize the deep learning (and machine learning) approach:

Figure 3.7 Machine Learning iterative cycle

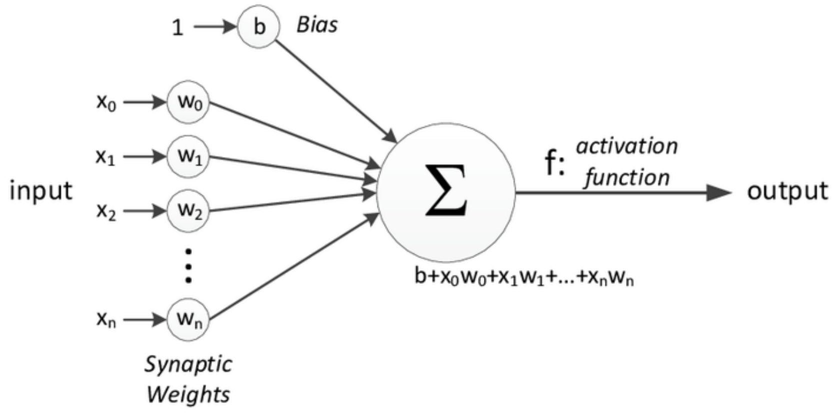


Source: own elaboration

In this study a combination of machine learning and deep learning techniques is used in order to compare results between each other. Once the text is pre-processed and vectorized it is possible to have a common ground for building, comparing and evaluating different models.

When talking about deep learning techniques it mainly referred to neural networks that contains more than one hidden layer, a neural network is a mathematical model that is inspired by biological neurons and nervous system. These models are used to recognize complex patterns and relationships that exists within a labelled data. A shallow neural network contains mainly three types of layers: input layer, hidden layer, output layer. The basic component of a neural network is the perceptron, the perceptron takes an input, processes it, passes it through an activation function and returns an output.

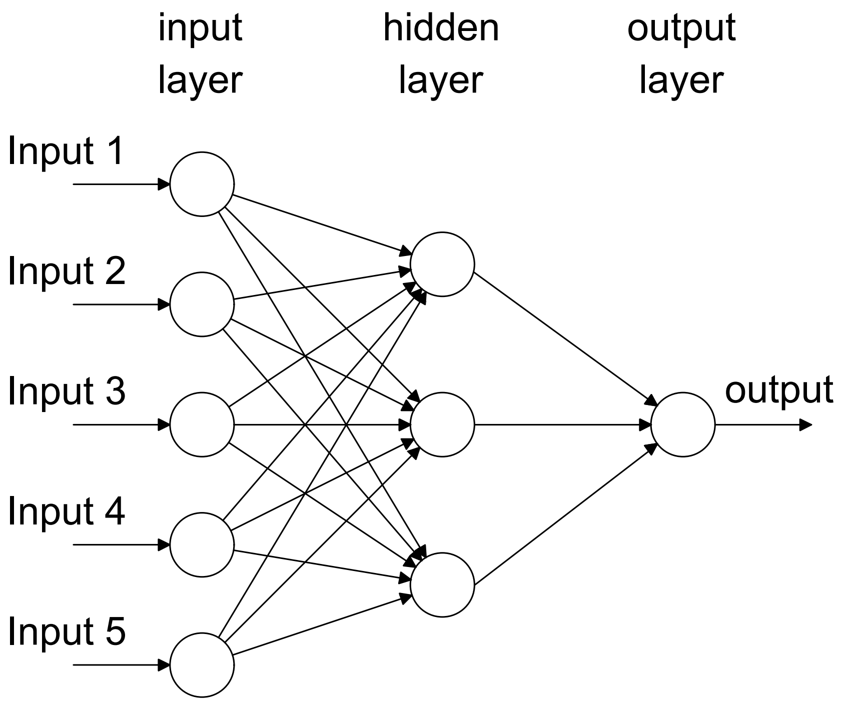
Figure 3.8 The perceptron model



Source: (Fountas, 2011, p. 12)

When combined together perceptron forms what is called a neural network, the main advantage of this configuration is that each perceptron in a neural network is an independent unit with its own weights $W_{j,i}$ and activation function.

Figure 3.9 A neural network



Source: own elaboration

For building a neural network model there are many libraries and material available, in this study one of the most popular deep learning framework is being used, TensorFlow (Google Brain, 2015) along with Keras: a high level API for TensorFlow.

TensorFlow is a deep learning open source framework developed at Google in 2015, it offers different APIs, for training the simplest possible neural networks to the state-of-the art implementations, it is used by many companies such as Airbus, Airbnb, Coca-Cola and many more. (TensorFlow, 2019)

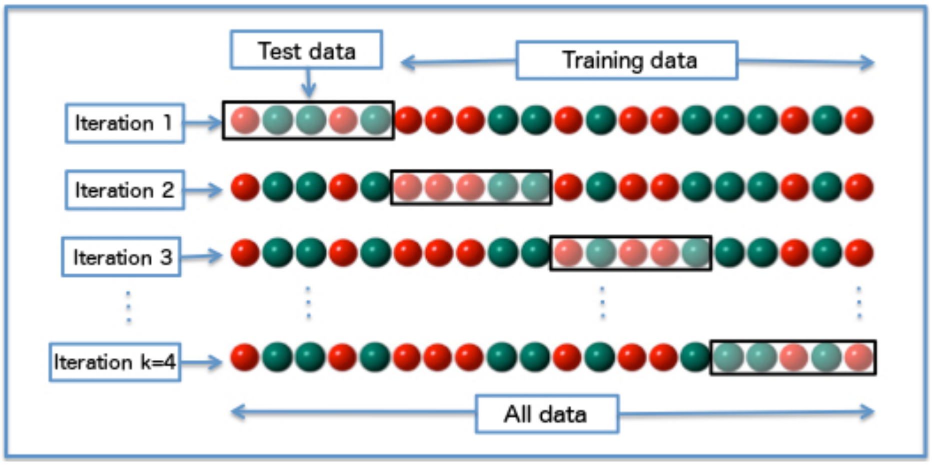
3.6 Model evaluation

While evaluating the performances in cross validation there are different techniques available. The most common one is the split of the dataset into training and test set. Along with this technique it is possible to further prevent the risks of overfitting and underfitting by applying a technique called

K-Fold cross validation. This method introduces the concept of validation dataset. K-Fold cross validation consist in one parameter *k* that refers to the number of groups that a given data sample is to be split into. The K-Fold cross validation technique works as follows:

- Take the first group as a holdout or test data set
- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set
- Retain the evaluation score and discard the model

Figure 3.10 A graphical representation of K-Fold cross validation



Source: (Flöck, 2016)

A slightly different version of K-Fold that results very good while dealing with class imbalances is the stratified K-Fold cross validation: it ensures that each K-Fold has the same proportion of observations with a given categorical variable, while doing this bias can be reduced. After training a model using those techniques it is then possible to validate the dataset with unseen data and get accuracy scores for the model.

Going further into model evaluation, common strategy to evaluate binary classifiers is the confusion matrix, this technique gives the possibility to know the proportion between the true positive (TP) prediction and the true negative (TN), while it lets observe type one and two errors.

Figure 3.11 Anatomy of a confusion matrix

		Actual Values	
		Positive	Negative
Predicted Values	Positive	**TP**	**FP**
	Negative	**FN**	**TN**

Source: own elaboration

After a confusion matrix is developed It is possible to estimate the accuracy of a classifier with the following formula:

[3.4]

$$accuracy(C) := \frac{TP + TN}{P + FN} F \in [0,1]$$

The closer the values of this equation are to 1 the better the classifier works and *vice versa.* It is however important to denote that this strategy would not work for highly imbalanced dataset, while predicting, for example, fraudulent transaction, where the classes are heavily imbalanced an accuracy score of 0.99 would not have any meaning. For those problems it is better to structure the classifier in a different way, for example ignoring and class and always predicting the probability that an observation appertains to a class.

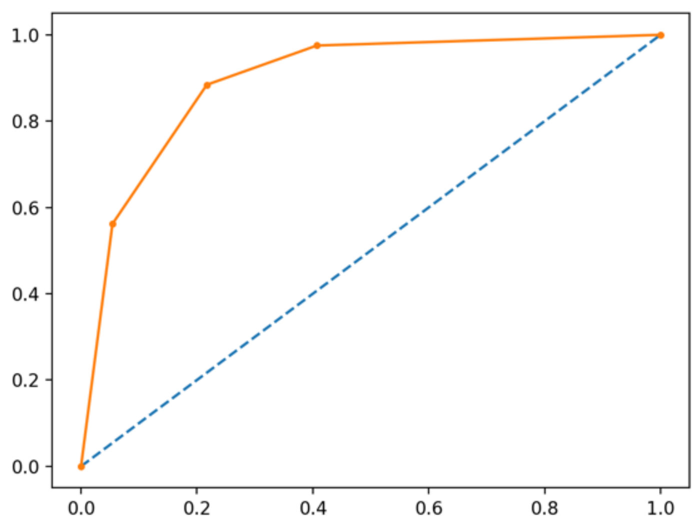
Another metric that is possible to obtain from the confusion matrix is the precision, or the total number of true elements that are actually true. It is denoted by the following formula:

[3.5]

$$precision(C) := \frac{TP}{TP + FP} \in [0,1]$$

The last tool that allows to measure performances is the AUC - ROC curve. The AUC - ROC curve is a measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represent degree or measure of separability. It tells how much a model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting classes. By analogy, higher the AUC, better the model is at distinguishing between classes. The following figure (3.12) represents the anatomy of the ROC-AUC curve:

Figure 3.12 An example of ROC-AUC curve



Source: (Brownlee, 2018)

The closest the curve in the ROC curve is to the top-left corner, the more accurate the classifier is. The line crossing the plot at 45 degrees represents and AUC of 0.50, meaning that a classifier with this ROC curve would be equal in classifying as much as tossing a coin. In this study the ROC-AUC curve together with test accuracy score are used as evaluation metrics.

4 Background analysis

This section provides information about the implementation of the study, it describes the methodologies applied to the data and the results obtained. The original dataset in JSON format is converted to CSV format.

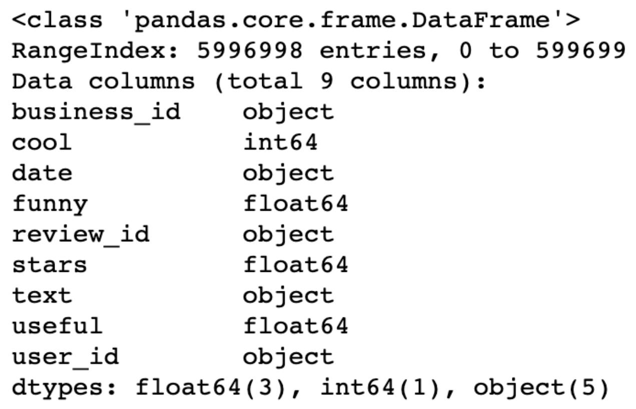
Throughout this exploratory analysis the full version of the dataset is implemented, it contains 5996998 reviews.

As the size of the dataset is quite large a normal laptop would not be able to perform many operations in a convenient amount of time, that is why for this study an instance of Jupyter Notebook on an Intel Core i7-4710HQ at 2.50 GHz with 16GB of RAM running Linux Ubuntu is used. While, for running deep learning models a Nvidia GeForce GTX 860 graphic card with 4GB of VRAM is used.

4.1 Data conversion

In order to begin the explorative analyses, the data needs to be converted in the CSV format, this because the JSON format consume a larger amount of memory compared to CSV. For this analysis the file containing all the reviews is used, it weights 4.15 GB and it is in the JSON format. Immediately after the data is converted in the right format is it possible to inspect the dataset:

Figure 4.1 An informative of the dataset yelp review



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5996998 entries, 0 to 599699
Data columns (total 9 columns):
business_id    object
cool           int64
date           object
funny          float64
review_id      object
stars          float64
text           object
useful         float64
user_id        object
dtypes: float64(3), int64(1), object(5)
```

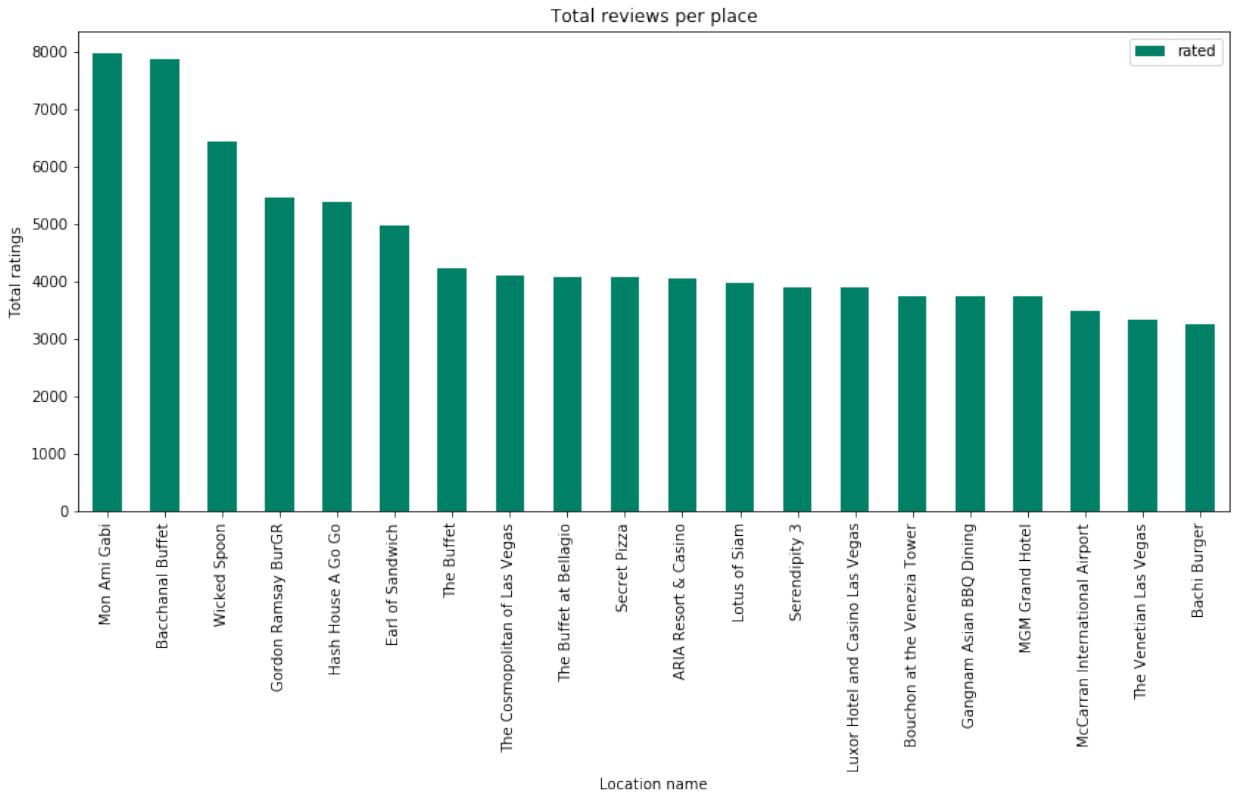
Source: own elaboration

Along with the columns needed in this study it is possible to see the presence of some columns su ch as *business_id*, *review_id*, *user_id*; those columns would become very helpful while joining to gether different tables in order to conduct, for example, user-based analyses or to find the name of the most reviewed places.

4.2 Exploratory analysis

The next metric and explorative insight developed in the previous chapter is to obtain some basic statistics regarding the count of reviews per place:

Figure 4.2 Total amount of reviews per place (first 20)

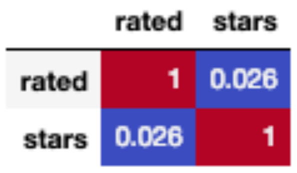


Source: own elaboration

Figure 4.2 shows the count of reviews per place, it can be noted that the locals that have the highest number of reviews matches with the most popular places (Mon Abi Gabi, a French restaurant chain very popular in the U.S. and Gordon Ramsay's restaurant, a worldwide famous British Chef).

Plotting each place by counting the number of reviews received offers the possibility to understand which the most popular places are, as a place that is reviewed many times is probably well known. Moreover, it is interesting to plot the average of stars received for the most popular places, if the plot suggests a correlation between popularity and average of stars received it might be a good starting point for building a model, however this thesis must be validated with some analytical tools, for example the correlation matrix:

Figure 4.3 Correlation matrix between number of reviews and average of stars

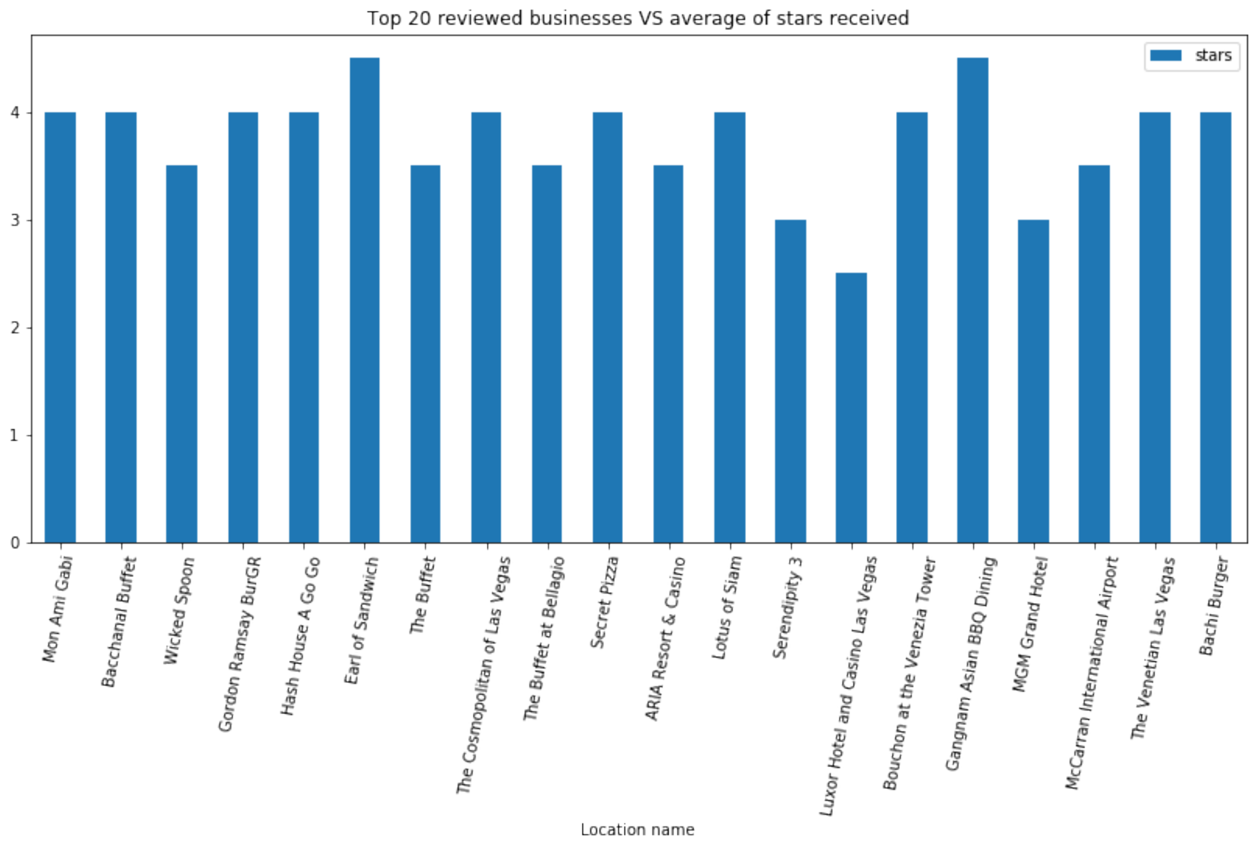


	rated	stars
rated	1	0.026
stars	0.026	1

Source: own elaboration

As the correlation matrix suggests there is no correlation between the two examined variables, in fact the bar plot suggests the same:

Figure 4.4 Top 20 reviewed places VS average of stars received



Source: (own elaboration)

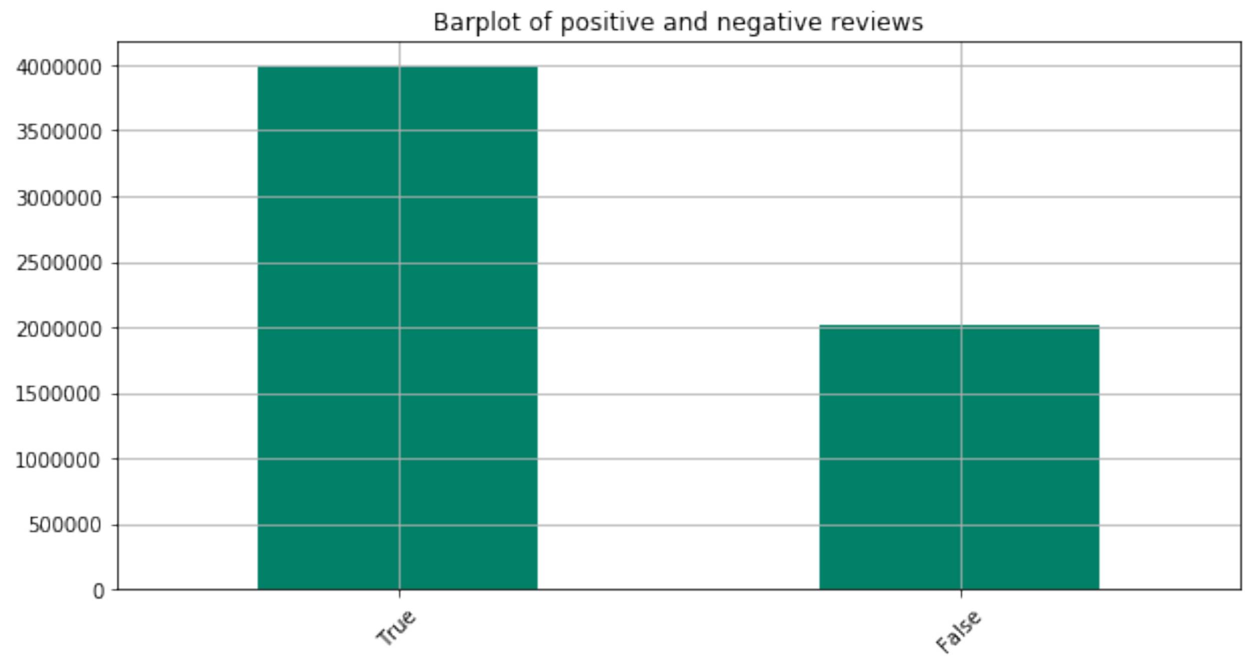
After those preliminary analyses it is possible to convert the discrete variable “stars” into a Boolean one, following the criterion:

[4.1]

$$number\ of\ stars \geq 3$$

This makes possible to continue the exploratory analysis, where a review containing at least three stars will be considered positive, and a review that contains less than three stars will be considere d negative. The balance of the dataset is being verified and the distribution between the positive a nd negative reviews can be expressed with the following graph:

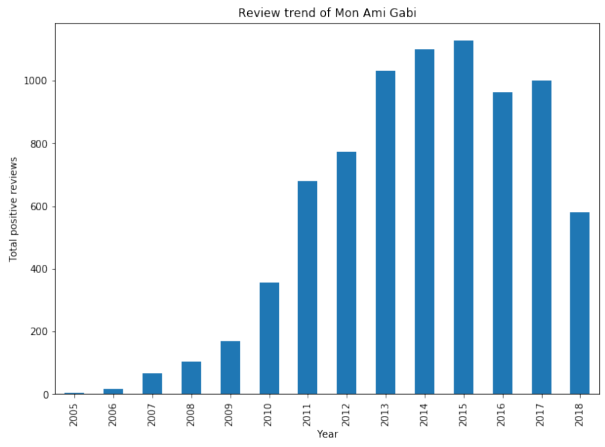
Figure 4.5 Bar plot of total amount of positive and negative reviews



Source: own elaboration

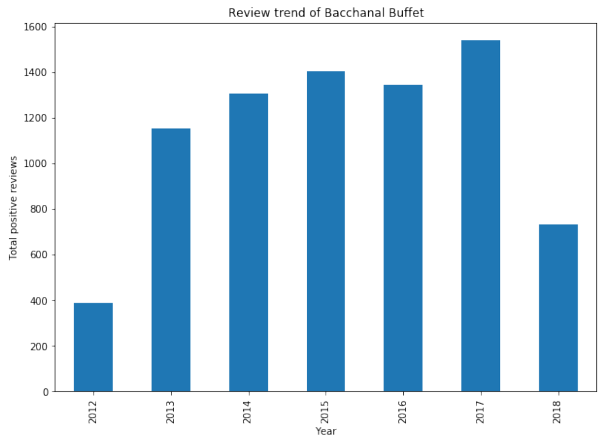
As it is possible to deduce from the plot there is a strong imbalance between the two classes, this i mbalance could lead to biased results during the creation of the model. In order to minimize the bi as a strategy like the stratified K-Fold sampling would become helpful. It is now possible to verif y the trend of positive reviews over time for a selected number of locations (first two):

Figure 4.6 Trend of positive reviews for Mon Ami Gabi



Source: own elaboration

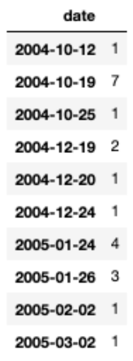
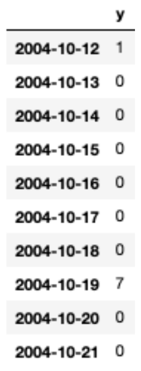
Figure 4.7 Trend of positive reviews for the Bacchanal Buffet



Source: own elaboration

To put this study into a temporal context, to eventually find some trends like seasonality it could be helpful to have a graphical visualization of the reviews over time. To achieve this, it is necessary to aggregate the dataset by date and number of reviews by date, however the dataset does not contain a regular and progressive collection of dates, but very often there are some gaps between dates, specially at the beginning of Yelp's existence (2004 - 2007). A good practice to overcome this is to re-index the dataset and create an index containing a progressive collection of dates, then for each observation the corresponding number of reviews will be attached:

Table 4.1 Before and after applying re-indexing

Before re-indexing	After re-indexing
	

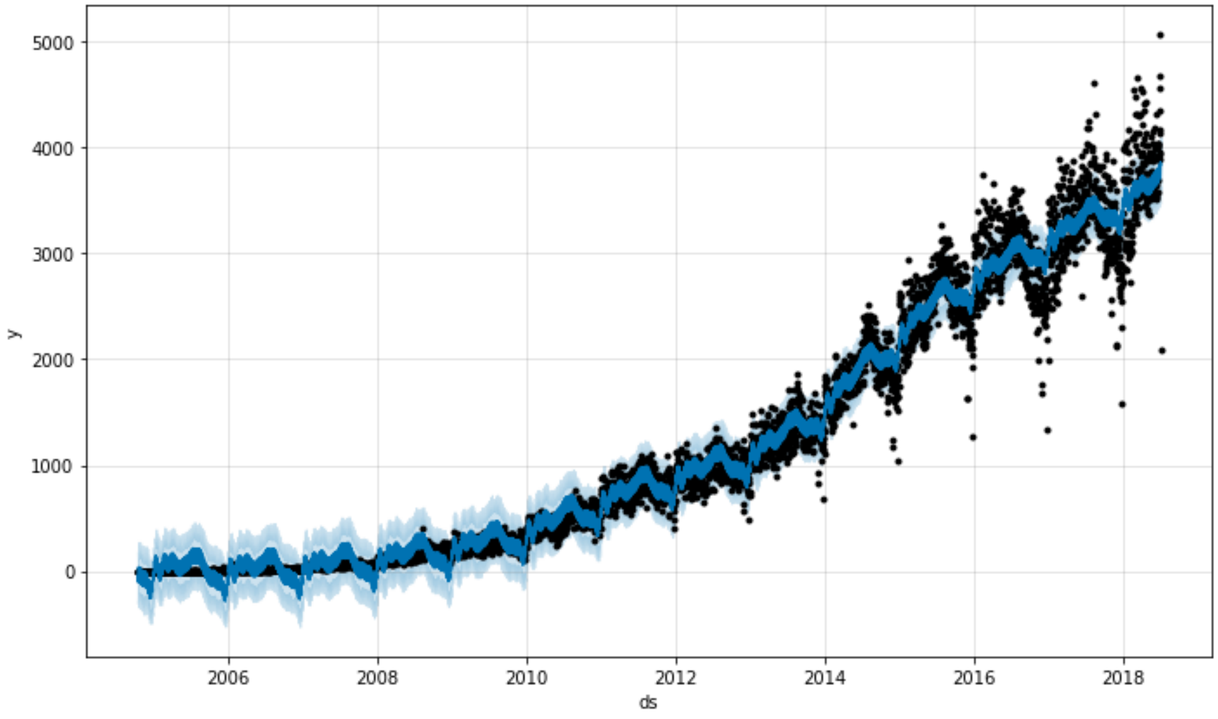
date	
2004-10-12	1
2004-10-19	7
2004-10-25	1
2004-12-19	2
2004-12-20	1
2004-12-24	1
2005-01-24	4
2005-01-26	3
2005-02-02	1
2005-03-02	1

	y
2004-10-12	1
2004-10-13	0
2004-10-14	0
2004-10-15	0
2004-10-16	0
2004-10-17	0
2004-10-18	0
2004-10-19	7
2004-10-20	0
2004-10-21	0

Source: own elaboration

A very powerful tool for time series decomposition, predictions and modelling is Prophet (Taylor & Letham, 2017). Prophet is a library for python developed by that consists in a very powerful and easy to use API:

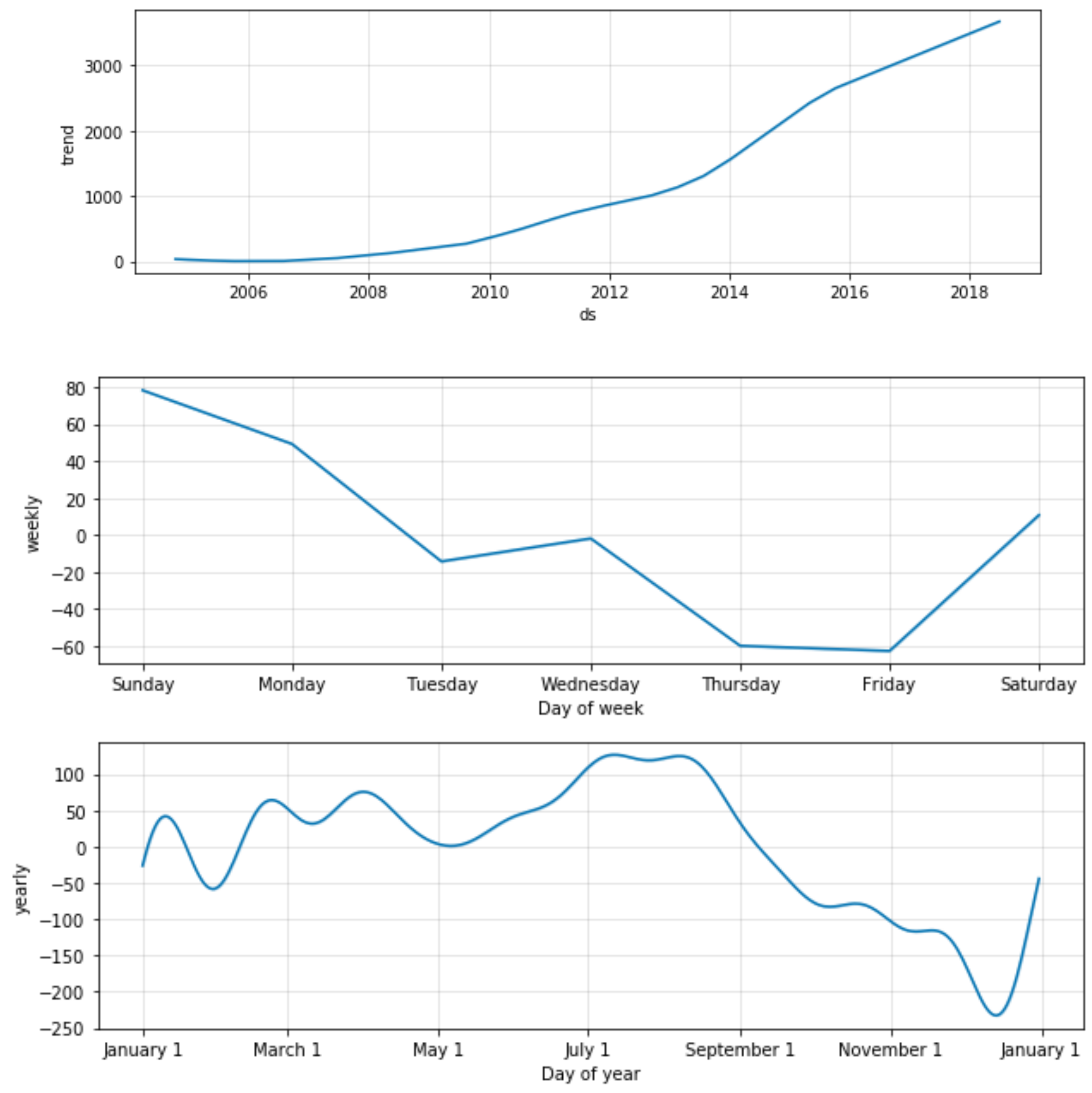
Figure 4.8 Plot of total number of reviews by date



Source: own elaboration

The plot represented in figure 4.8 shows an increasing trend in daily reviews along with a seasonality, in fact at the beginning of each year there is a decrease in total number of reviews, suggesting that users in this period tend to frequent less the businesses in the dataset, in addition to this there is an increase during warmer months. Figure 4.9 breaks down this plot into its components:

Figure 4.9 Decomposition of trend and seasonality for total number of reviews by day

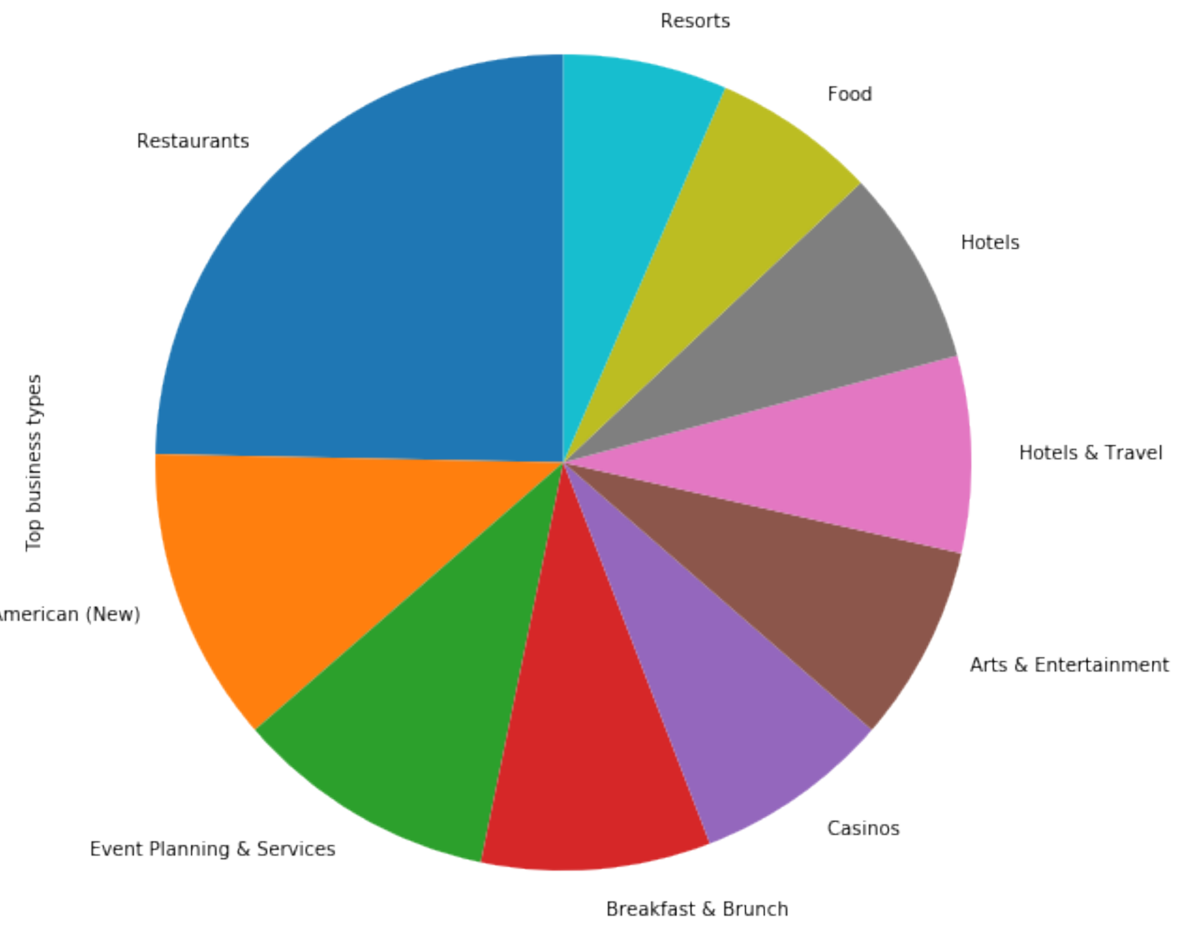


Source: own elaboration

The weekly seasonality increases in the weekend (the dataset contains mainly food and beverages businesses), suggesting that those businesses are experiencing and increase in customers during the weekend. In addition the increase of reviews during the warmer period of the year is confirmed and immediately after the Christmas period another increase is visible, matching the New Year's Eve celebrations.

The next point in this explorative analysis is to determine which business categories are the most reviewed and well which businesses are getting negative reviews, giving an important insight for competitors:

Figure 4.10 Top reviewed business categories

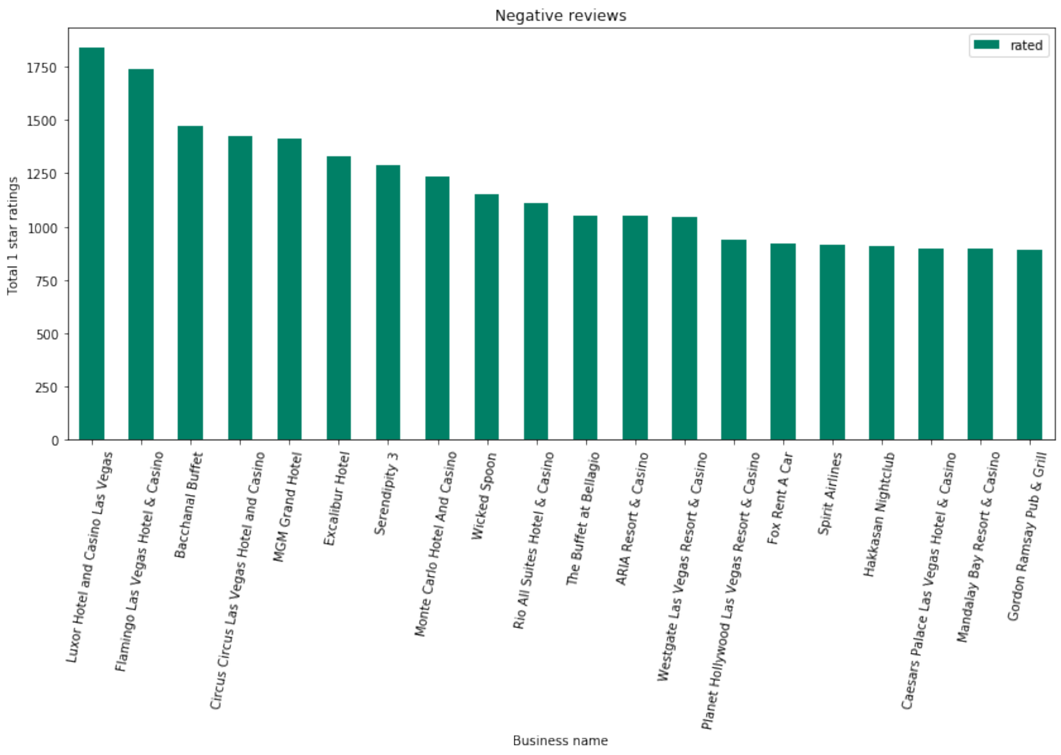


Source: own elaboration

From the pie chart it is evident that the majority of reviewed businesses are in the food and beverage market, however some noise is present: since the categories are user-generated some names such as “Food”, “Breakfast & Brunch” are probably still part of the food and beverage industry.

The next plot is the aggregation of one-star review per place, filtering the first twenty locations:

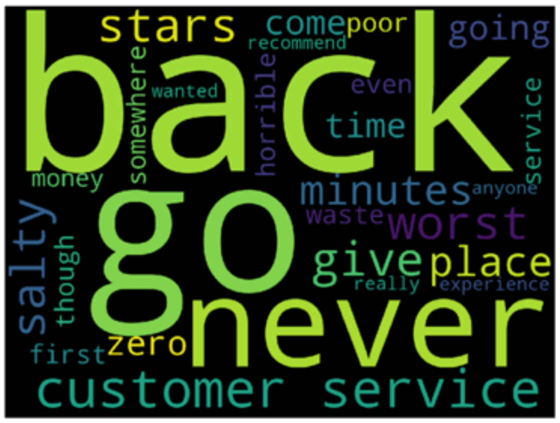
Figure 4.11 Total amount of one-star review by place (first 20)



Source: own elaboration

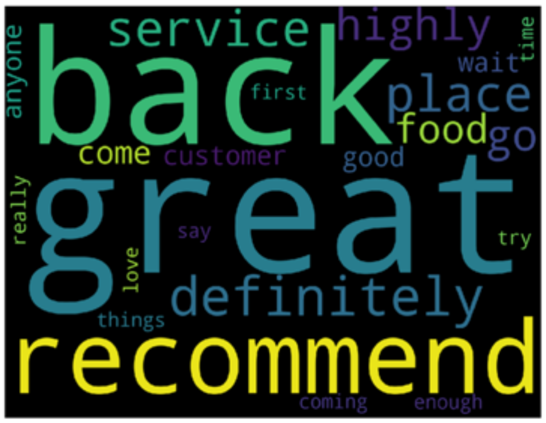
The outcome of Figure 4.11 gives a powerful insight into the worst places by number of stars, it is possible to use it both for a competitor deciding to operate a business nearby, or within the businesses, to understand pitfalls and weaknesses in the service and / or quality.

Figure 4.12 Negative review in word cloud



Source: own elaboration

Figure 4.13 Positive review in word cloud



Source: own elaboration

Having the best rated businesses and the lowest rated opens the possibility to conduct a deeper analysis on the review text, knowing which are the most common words in positive and negative reviews leads to understand in detail which are the most common problems. Moreover, applying this analysis to a single place would make possible for the management of the place to understand better its customers.

5 Modelling

The goal of this section is to implement and evaluate the performance of different models on binary classification, the dataset is being used to classify whether a review is positive or negative (based on the criterion number of stars ≥ 3) and the predictive variables used is the text variables, containing the corpus of a review written by Yelp's user. The first phase involves the pre-processing: tokenization, removal of stop words, stemming and feature conversion. For the stemming the most popular stemmer algorithm is implemented: The Porter stemmer, while for the feature conversion results from bag of words and TF-IDF algorithm will be compared.

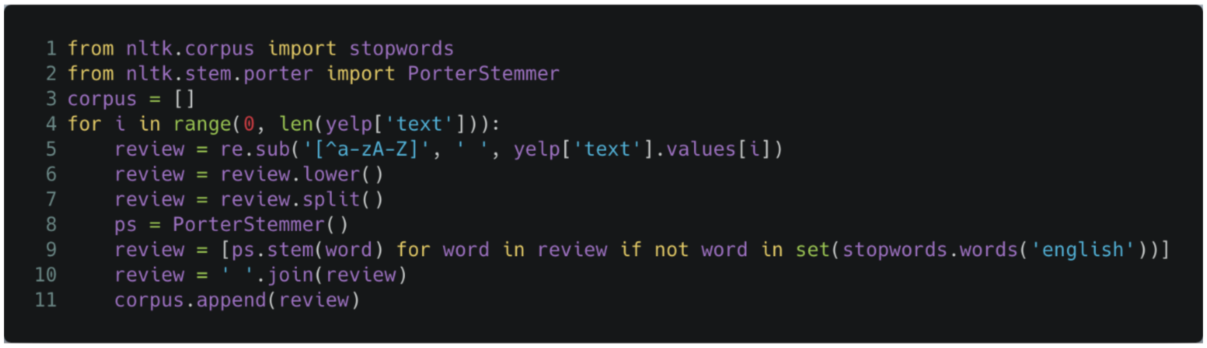
The list of stop words used is a list from *nltk* one of the most popular NLP libraries.

As the dataset is not balanced the training set is being fit with the aid of stratified K-Fold, this to keep balance in the dataset splits. Moreover, the dataset is being split 80 – 20 to evaluate it on unseen data.

The pre-processing phase implemented is kept the same for each in model, this for keeping a common ground while evaluating performances.

To keep consistency between models the dataset is being pre-processed and vectorized once, then the matrixes of features are being exported as a *.csv* file, the pre-processing applied consists in removing nonliteral characters through regular expressions, converting all the letters into lower and removing stop words:

Figure 5.1 Text pre-processing code snippet



```
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
corpus = []
for i in range(0, len(yelp['text'])):
    review = re.sub('[^a-zA-Z]', ' ', yelp['text'].values[i])
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]
    review = ' '.join(review)
    corpus.append(review)
```

Source: own elaboration

The stopwords lists comes built-in with the nltk library, hence there is not need to provide specific list, althought it can be possible.

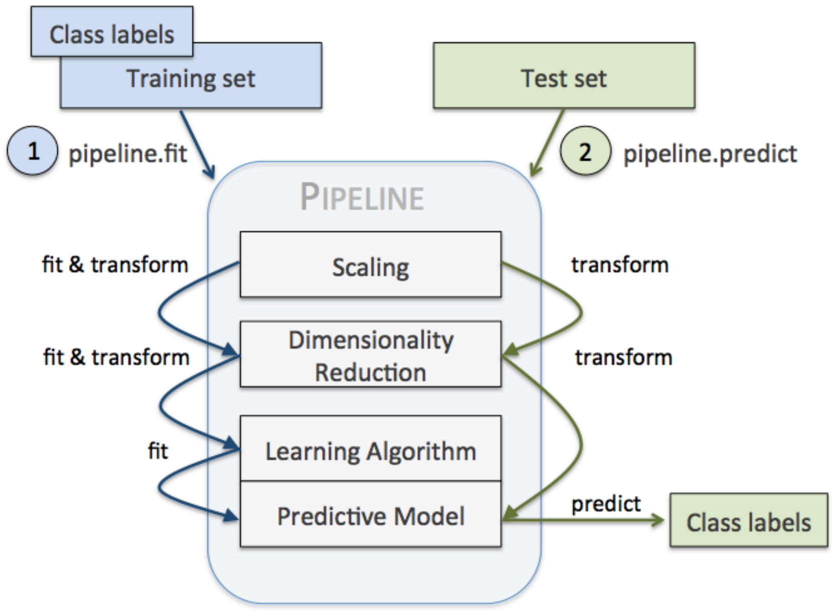
5.1 Machine learning models

The first models to be build is the logistic regression. For the Logistic Regression model a series of hyperparameter (according to the official documentation) such as C (inverse of regularization strength, this parameter applies a penalty in order to reduce overfitting), solver (the algorithm used to solve the optimization problem) and fit intercept (a Boolean parameter to specify if a constant is added in the decision function) are used in combination with the grid search, this in order to find the best hyperparameters combination. In addition, the code is written using a tool called *Pipeline*.

The purpose of Pipelines is to assemble together several steps that can be cross validated and / or grid searched while setting up different parameters, each step can be a `Transformer` or an `Estimator`. All the stages are run in order.

Pipelines comes very helpful when it is needed to keep consistency in the code, creating a smooth workflow and production-oriented approach.

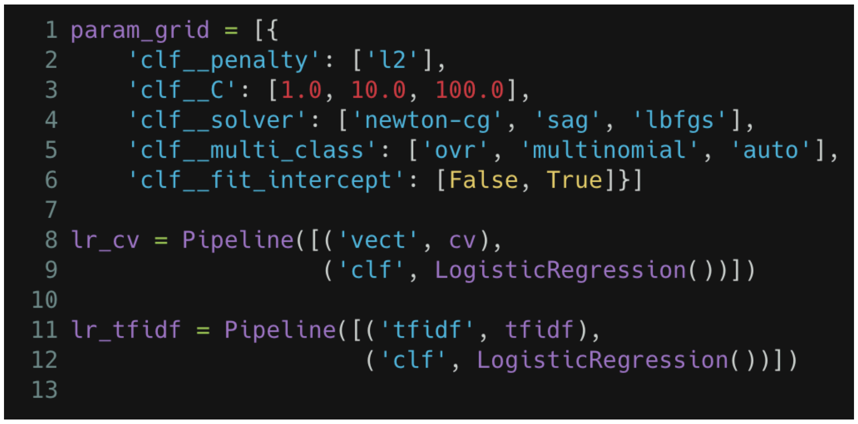
Figure 5.2 An example of a pipeline workflow



Source: (Sebastian Raschka, 2017, p.189)

The Logistic Regression model is fitted with 5 k-fold using the following hyperparameters values in order to find the best combination:

Figure 5.3 Hyperparameters used for grid search with logistic regression with countVectorizer and TfidfVectorizer



```
param_grid = [{
    'clf__penalty': ['l2'],
    'clf__C': [1.0, 10.0, 100.0],
    'clf__solver': ['newton-cg', 'sag', 'lbfgs'],
    'clf__multi_class': ['ovr', 'multinomial', 'auto'],
    'clf__fit_intercept': [False, True]}]

lr_cv = Pipeline([('vect', cv),
                  ('clf', LogisticRegression())])

lr_tfidf = Pipeline([('tfidf', tfidf),
                     ('clf', LogisticRegression())])

```

Source: own elaboration

However, while vectorizing the dataset the size of it increases exponentially, considering the original dataset of being 4.10 GB the count vectorized version of it takes 18.0 GB of memory.

Managing the memory with large dataset is a challenge in both the academical and the professional word. A possible solution to this problem, for implementing the logistic regression is to use a random portion of the whole dataset as a representative sample. Some other models, for example the Naïve Bayes or the Stochastic Gradient Descent classifier do not need to "see" the whole dataset at once, because of the way the algorithm works, at every iteration weights are being updated, so they implement a method called `partial_fit()`, which gives the possibility to read the dataset in batches and train the algorithm in few different steps, giving the possibility not to overflow the memory. However, in order to keep the same common ground for every model a representative sample pf the dataset has been taken.

The logistic regression model is trained on a random sample of 50000 observations, both using the count vectorizer and the TF-IDF vectorizer, the total training time is around 80 minutes per model on an Intel Core i7-4710HQ at 2.50 GHz with 16GB of RAM.

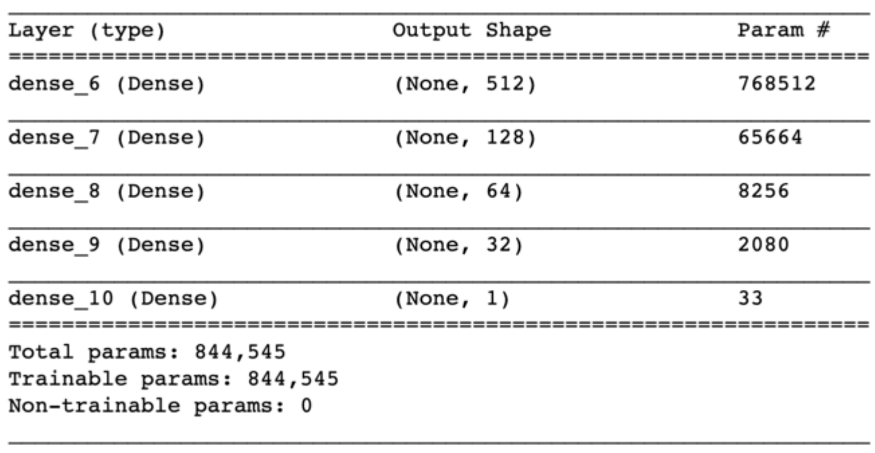
For the Stochastic Gradient Descent classifier, a similar approach is used: the algorithm is trained on the same sample of the dataset, using grid search for hyperparameters optimization.
The third machine learning model tested is the Naïve Bayes classifier. This classifier is been used frequently since the 1960s and it is commonly used as a baseline method for text categorization, spam filters and medical diagnosis (Rish, 2001).

The fourth and last model prepared is the Random Forest classifier, although its main applications are not related to text recognition, but rather other types of classification and regression it is a very used and multipurpose algorithm that it is very strong to prevent overfitting. Recalling the machine (and deep) learning iterative process (Figure 3.7) it is a good practice to try different algorithm, that is why the Random Forest algorithm is tested for this study.

5.2 Deep learning models

In this section deep learning models implemented in this study are discussed. The neural networks models trained in this study are developed using Keras, a very common deep learning library for python, which works on top of TensorFlow and gives a high-level API.
The model consists in two neural networks. The first neural network models are relatively a basic one, it contains three hidden layers and all the hyperparameters are set according to the Keras documentation.

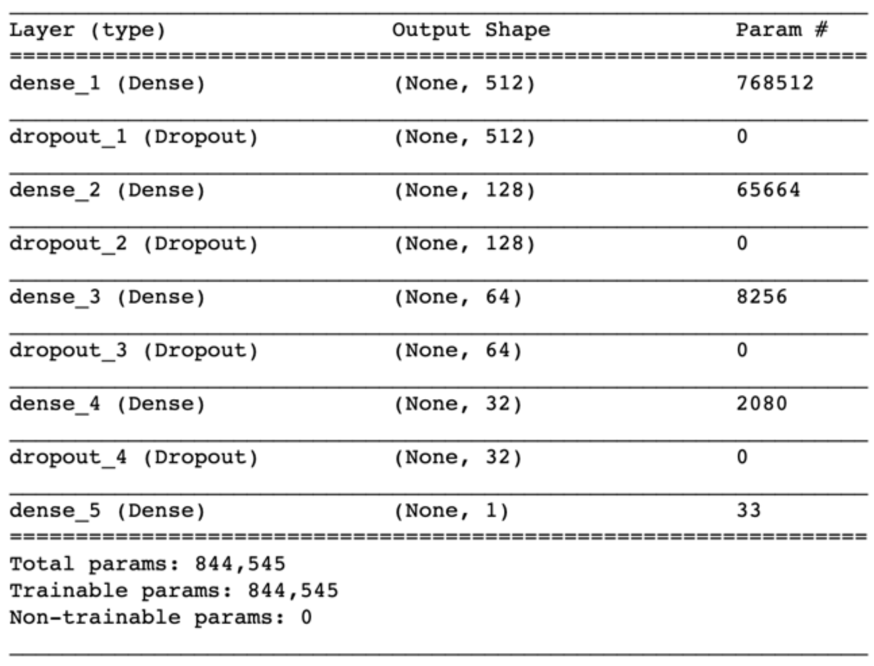
Table 5.1 Structure of the neural network model without dropout layers



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 512)               768512
_________________________________________________________________
dense_7 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_9 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 33
=================================================================
Total params: 844,545
Trainable params: 844,545
Non-trainable params: 0
_________________________________________________________________
```

Source: own elaboration

Table 5.2 Structure of the neural network model with dropout layers



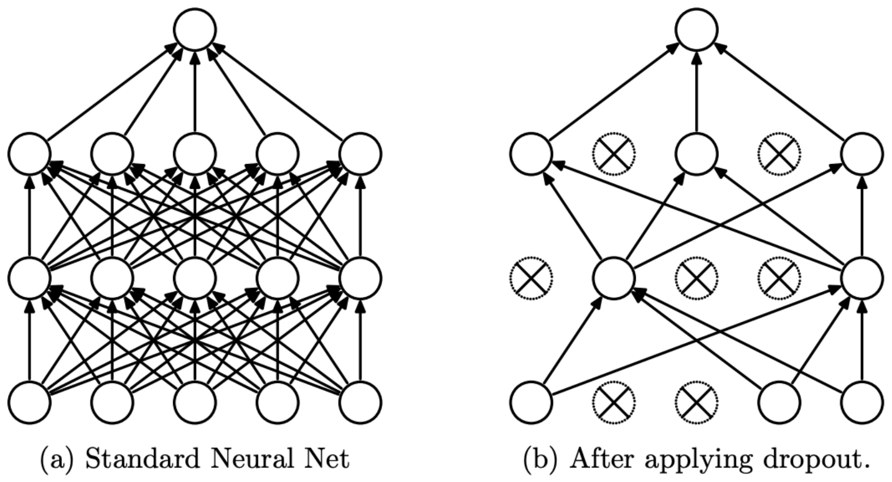
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               768512
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 33
=================================================================
Total params: 844,545
Trainable params: 844,545
Non-trainable params: 0
_________________________________________________________________
```

Source: own elaboration

The second neural network differs from the first one only by the implementation of dropout layers. Dropout is a common and modern deep learning technique to prevent overfitting, it was developed in 2014 at the University of Toronto (Srivastava, et al., 2014).

It works by dropping out of the training a certain number of neurons (usually between 30 to 50 %) at each layer for each epoch, in this way some noise is added to each hidden unit and it prevents units to "co-adapting too much".

Figure 5.4 Before (a) and after (b) applying dropout to a neural network



Source: (Srivastava, et al., 2014)

Dropout is not the only strategy to prevent overfitting, reducing the number of hidden layers can also help to reduce overfitting, as well as increasing the samples training set and / or reducing the number of epochs. (Chollet, 2018)

6 Model comparison and evaluation

This section's aim is to compare, evaluate and comment model performances, it is divided in machine learning models and deep learning models.

The metrics used in the evaluation sections are the metric previously mentioned: test set accuracy score and AUC area. In this section also the time factor, hence the time needed to train the model is taken into account, this because when training models using cloud solutions the time is one of the variables that influences costs.

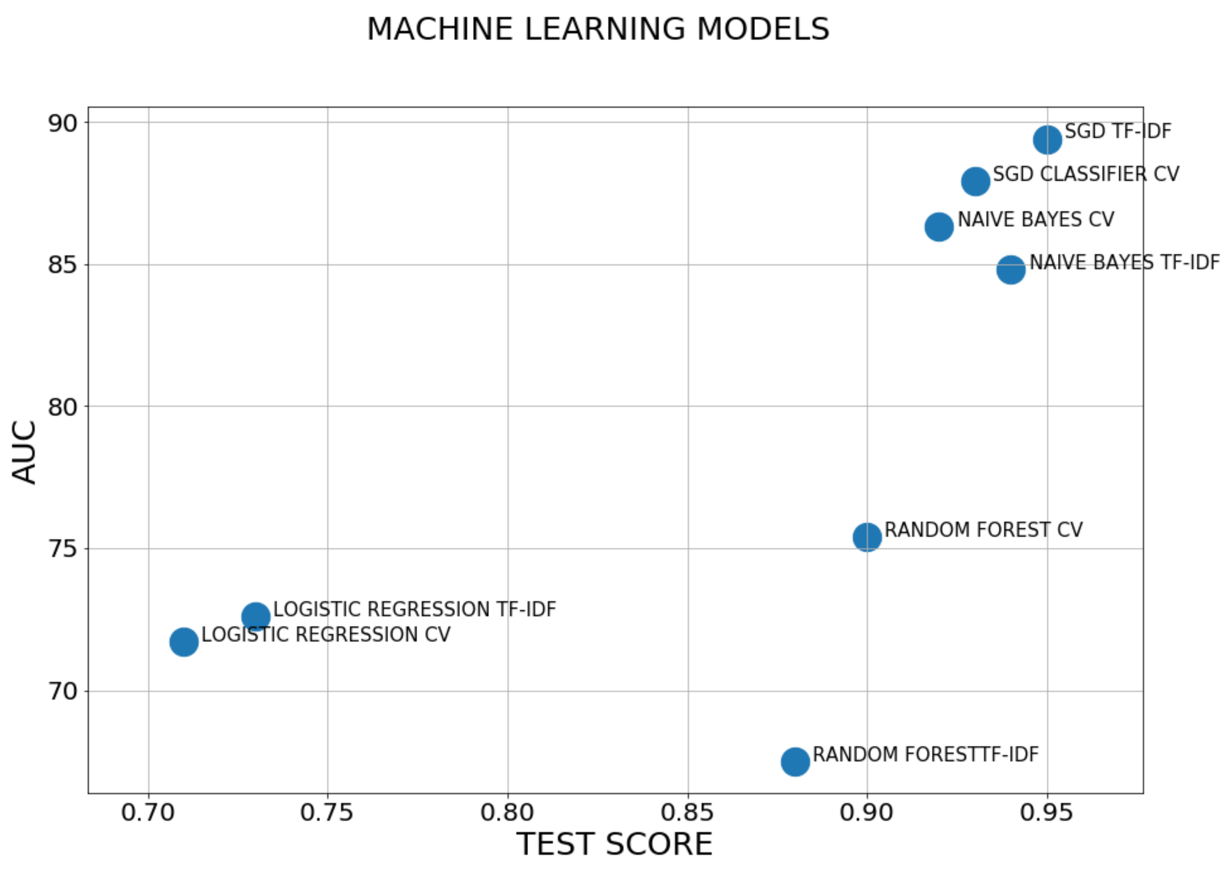
While, when evaluating deep learning models also the loss score will be taken into account, a training loss that rapidly falls to 0, while validation loss steady could be a strong sign of overfitting.

6.1 Machine learning models evaluation

This section contains plots, results, consideration and evaluation about the machine learning models trained in this study. It compares results of four different models: logistic regression, random forest classifier, Naïve Bayes and Stochastic Gradient Descent classifier.

Every model is trained with a sample size of 50000 observations and for each model there are two versions, one implementing the bag of words and the second one using TF-IDF vectorization. Both pre-processing techniques are based on a dictionary of the most frequent 1500 words. For each classifier a grid search was performed according to the available parameters of the classifier, and the training is done using the best hyperparameters combination.

Figure 6.1 Graphical comparison of machine learning models results



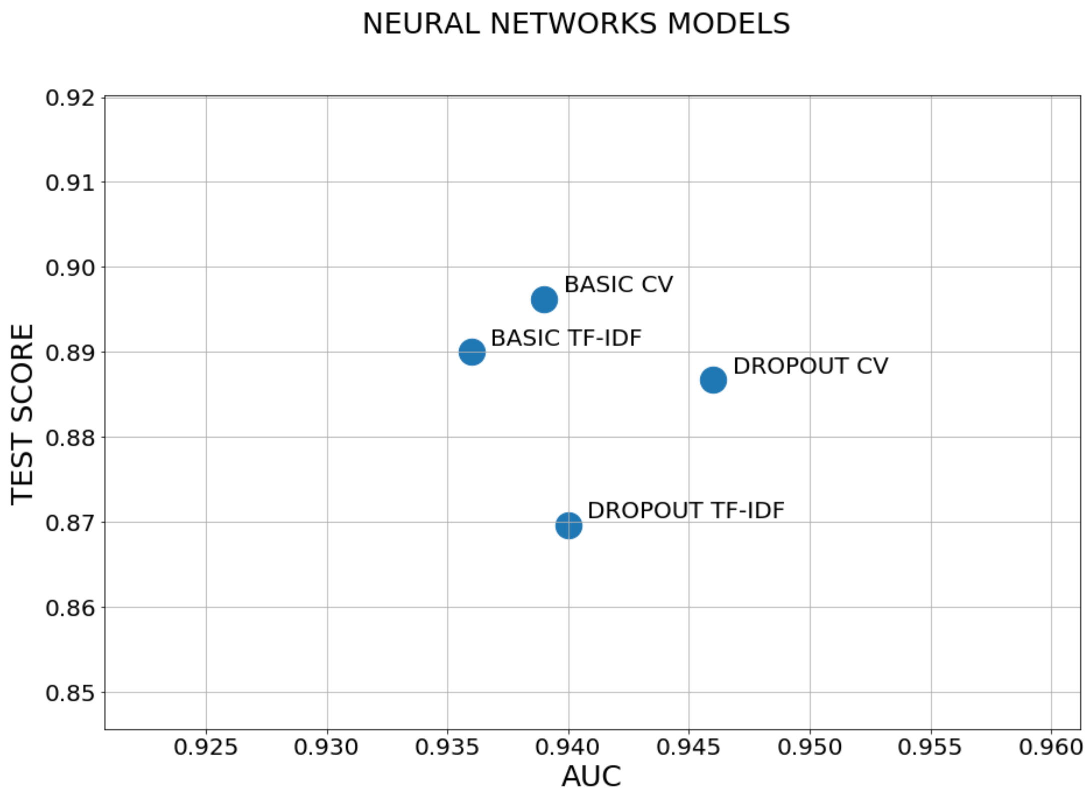
Source: own elaboration

It is possible to conclude that the Stochastic Gradient Descent classifier (an implementation of support vector machine in python), fitted with the TF-IDF vectorizer performs the best compared to other algorithms. From the plot is possible to see that the random forest classifier tends to strongly overfit compared with the logistic regression model.

6.2 Deep learning models evaluation

This section contains plots, results, consideration and evaluation about the deep learning models trained in this study. It compares results from the two neural networks presented in section 5.2. Every neural network is trained with the same amount of samples as for machine learning models, and for each model two versions are trained: one implementing the bag of words vectorizer and the second one the TF-IDF vectorizer The first tangible advantage of neural networks is that if an NVIDIA GPU is available it is possible to train models on GPU reducing training time up 80%, , making the training phase much faster than for machine learning models. The following figure gives a graphical representation of the result obtained:

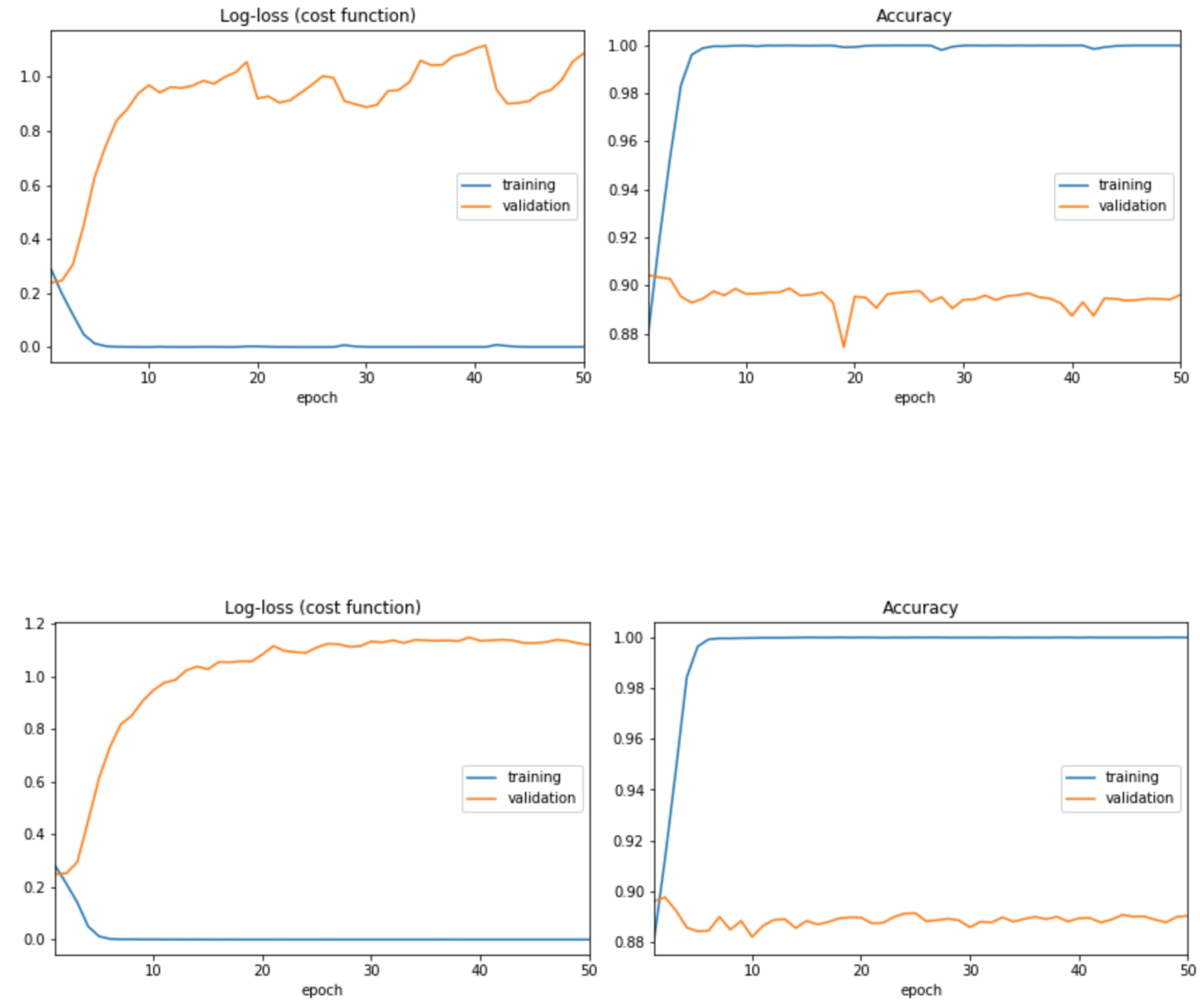
Figure 6.2 Graphical comparison of Neural Networks models results



Source: own elaboration

From the plot it is possible to see how overall all the four models perform very similarly, with a slight increase for the AUC of the dropout models. However, the overfitting is very strong:

Figure 6.3 Cost function and accuracy VS epochs CV model on top and TF-IDF model bottom no dropout applied

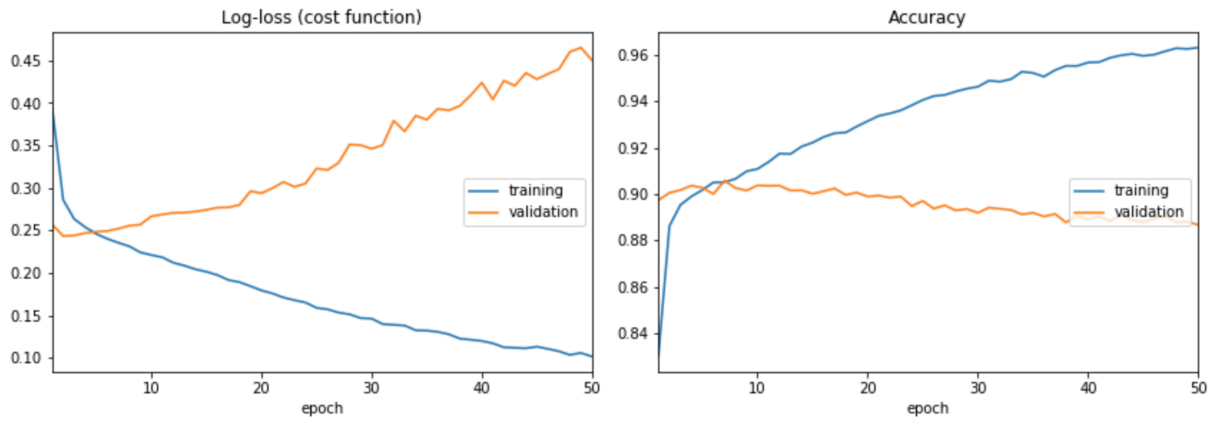


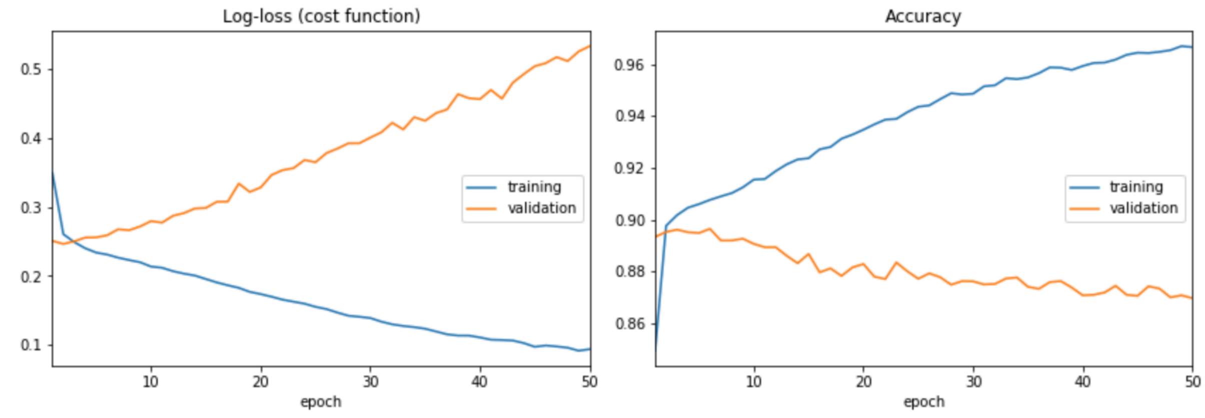
Source: own elaboration

Both CV and TF-IDF models without dropout tends to overfit very quickly, in less than 10 epochs cost function drops to 0 and accuracy to 1 while the change in cost function and accuracy for validation data tends not to change.

However, comparing the same models results using dropout gives different cost function plots:

Figure 6.4 Cost function and accuracy VS epochs CV model on top and TF-IDF model bottom using dropout



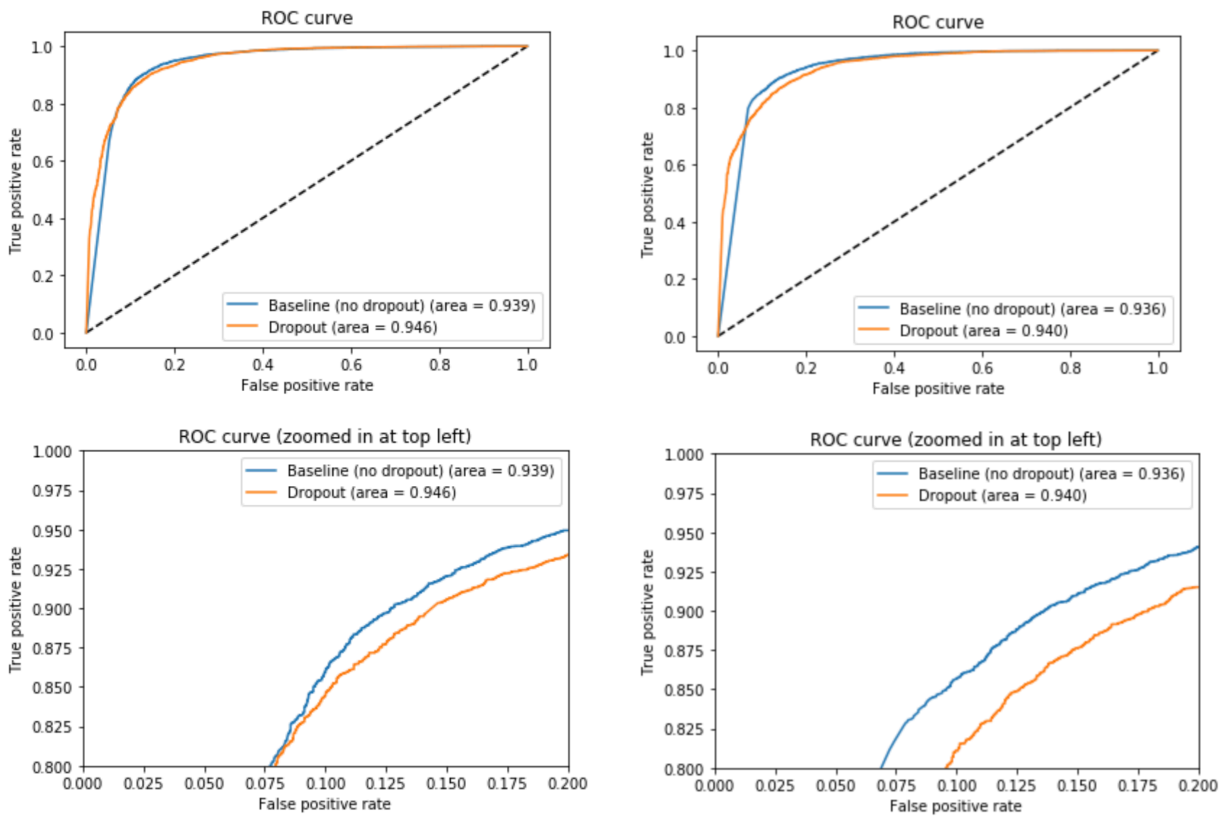


Source: own elaboration

The cost function for dropout models keeps improving by each epoch, proving that dropout technique prevents overfitting and the model is able to learn new features each epoch. It is worth considering that even if both models with dropout and without perform similarly on accuracy an overfitted model is a model that is not able to generalize well on unseen data because it learnt patterns only present in the training set.

Considering the ROC-AUC curves it is furthermore evident how all the models perform very similarly:

Figure 6.5 Comparing of ROC curves for CV models (left) and TF-IDF models (right)



Source: own elaboration

ROC shows that CV vectorizer performs slightly better and TF-IDF and that overall dropout models have a better score with both vectorizers, proving that dropout is a good technique to prevent overfitting and improve AUC on test data.

7 Conclusions

This section's aim is to define the conclusions of this study. The objective defined at the beginning of th study have been reached, however, it is necessary to state how those objective were fullfilled based on the metrics defined at the beginning of the study.

Moreover, few consideration about a production version of this study are included, as the main goal of building a model is to deploy it to the final user. It is also important to throw some considerations on how this technology can be used and what are the ethical boundaries of those tools.

7.1 Model performances

Following the models trained in this study it is possible to come up with several considerations based on different aspects:

- The resources needed to train a model
- The time needed to train a model
- The final results (accuracy score, AUC)

Analyzing the first aspect, the resources needed to train a model, for the machine learning models is very relevant, the decision to sample 50000 observations was taken because of the nature of some algorithms such as logistic regression, decision trees, that needs to take the dataset as a whole and it is not possible to batch train. Hence, a lot of memory errors where encountered and that is why a subsample of the whole dataset was used.

While other algorithms such as stochastic gradient descent, naïve Bayes, neural networks give the possibility to batch train. However, while vectorizing the dataset, both with count vectorizer and TF-IDF vectorizer the same problems where encountered.

Having available a machine that is capable to train neural networks on GPU reduced the training time significantly, making those models the fastest models to train. Moreover, the hyperparameters selection is significantly faster, and almost ready *out of the box*, while machine learning models could not be trained on GPU.

The time needed to train a model is strictly related to the previous point: while using grid search for hyperparameters selection in machine learning models, the training time increased a lot, as for each parameters combinations a model must be trained, this lead to very long training time, especially for stochastic gradient descent and logistic regression, while performances of the latter are similar to other models that needed much less training time.

Considering the third aspect about the test score and AUC, neural networks models, trained on GPU outperformed every other model in terms of training time, providing very good scores. The stochastic gradient descent classifier performed better than any other model, but the training time was significantly longer than neural networks models, which performed slightly worse (1-2% in AUC).

7.2 Exploratory analysis

The explorative analysis gave relevant insights about the dataset, few conclusions can be made: the most popular restaurants are the restaurant with the highest number of reviews, definitely because those location are very busy. Not every restaurant kept their standard high over time, the average number of positive reviews per year vary from restaurant to restaurant, however there is a strong bias toward positive reviews in the whole dataset. According to the word clouds both in positive and negative reviews the most common word that appears are related to customer service, suggesting that this is a very important aspect for a business to pay attention.

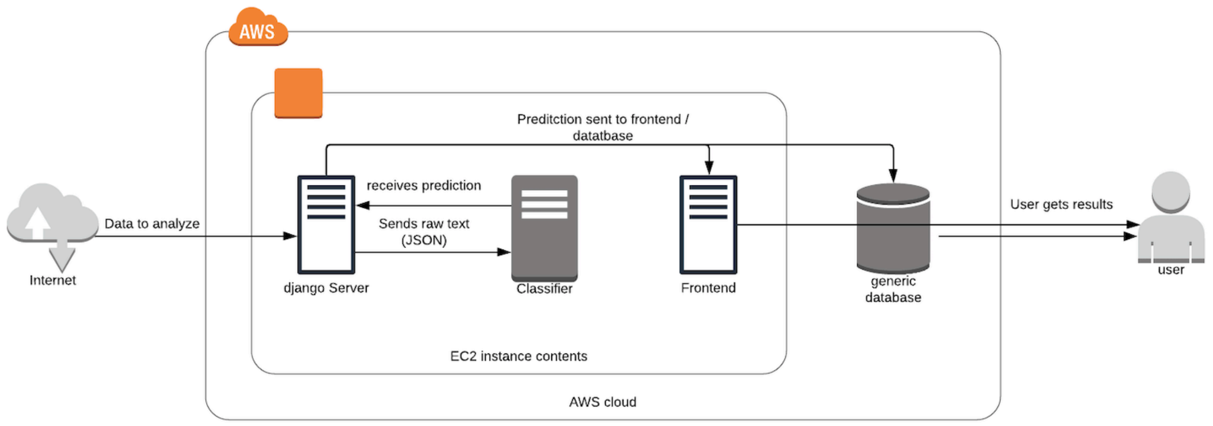
7.3 Going into production

As many customers based their buying decisions on reviews a classifier like the ones trained in this study would be very useful for a business that receives many reviews. The management would be able to immediately bring its attention to the negative reviews, and hence to solve problems faster. It is furthermore possible to apply to a specific set of reviews other NLP techniques in order for example, to tag a review based on content, this could help a business's management to understand better insights of its activity, highlighting strengths and weaknesses.

At the time of writing this thesis Natural Language Processing is becoming more and more popular, very often new tools and algorithms comes out, and older gets better, less “coding work” is needed and different tools offers built-in solutions to analyze natural language. Companies are understanding the real potential behind those techniques and the market is also starting to provide B2C solutions.

Nowadays many solutions for backend development and as well frontend development are available, the REST protocol makes easy to exchange data between different places, and the creation of an MVP is possible.
In order to build a production version of a classifier, where customers can load or get from some other sources reviews a backend server is needed. Python offers different solution for backend development, one of the most popular backend frameworks is Django, which makes it fast and scalable building a server that accepts and sends requests, in advance many cloud platforms offers different serverless architectures, where the backend development is almost reduced to zero.
The following figure (7.1) shows a basic implementation of a web application using Amazon Web Service’s EC2 computing platform.

Figure 7.1 An example of a server architecture for deploying a web application using AWS



Source: own elaboration using draw.io software

The plain text reviews are sent via POST request to the server, which are then being sent to the classifier pipeline, where preprocessing and predition are made. The final prediction is then sent to the frontend, a graphical Interface where the user can see predictions or saved to a database for further analyses.

7.4 Ethical Considerations

The Natural Language Processing topic is a big and fast changing topic, this study covered a relatively narrow topic: binary classification, while NLP in combination with generative adversarial neural networks can be used to generate text. This opens a widely discussed debate on ethics, such tools could be used to write fake news based on circumstances, and often creators of those systems do not release it to the public (Hern, 2019). Moreover, different techniques are nowadays able to create “deep fakes”, a technique for human image synthesis; the combination of those technologies could lead to the misuse of it, generating problematic outcomes.

8 Appendix

8.1 List of figures

8.2 List of tables

8.3 List of equations

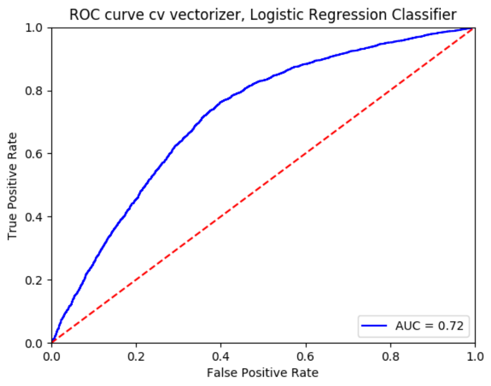
8.4 Bibliography

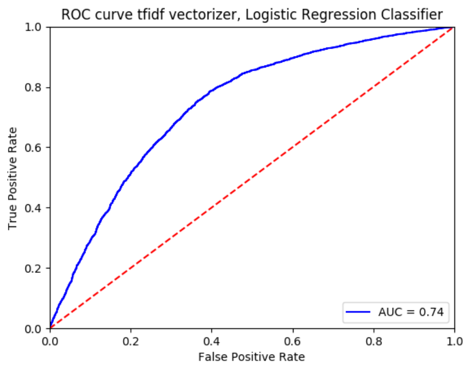
1. Anon., 2019. *HOW TO GET YELP REVIEWS WITH 5 STARS THAT DON'T GET FILTERED.* [Online]
   Available at: https://risingstarreviews.com/yelp-ratings/how-to-get-yelp-reviews/
   [Accessed 10 September 2019].
2. Bohn, D., 2019. *The Verge.* [Online]
   Available at: https://www.theverge.com/2019/1/4/18168565/amazon-alexa-devices-how-many-sold-number-100-million-dave-limp
   [Accessed 1 June 2019].
3. Brownlee, J., 2018. *How and When to Use ROC Curves and Precision-Recall Curves for Classification in Python.* [Online]
   Available at: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
   [Accessed 23 August 2019].
4. Buitinck, L. et al., 2013. *API design for machine learning software: experiences from the scikit-learn project.* s.l., European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases.
5. Chollet, F., 2018. Overfitting and underfitting. In: *Deep Learning with Python.* Shelter Island: Manning Publications Co., pp. 104-111.
6. CrowdFlower, 2016. *2016 Data Science Report,* s.l.: s.n.
7. David H. Wolpert, W. G. M., 1997. *No Free Lunch Theorems for Optimization,* s.l.: IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.
8. Facebook, 2017. *Forecasting at scale,* Menlo Park: PeerJ Preprints.

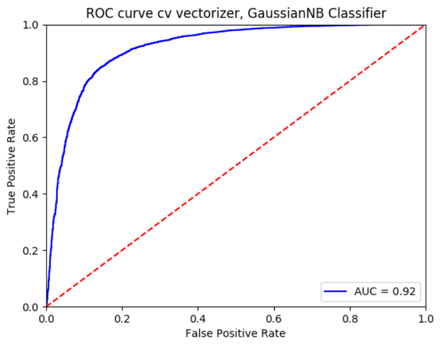
9. Flöck, F., 2016. *K-fold cross validation EN.* [Online]
Available at: https://commons.wikimedia.org/wiki/File:K-fold_cross_validation_EN.jpg
[Accessed 27 August 2019].
10. Fountas, Z., 2011. *Imperial College Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment,* London: Imperial College London Department of Computing.
11. Gesenhues, A., 2013. *Marketing Land.* [Online]
Available at: https://marketingland.com/survey-customers-more-frustrated-by-how-long-it-takes-to-resolve-a-customer-service-issue-than-the-resolution-38756
[Accessed 4 March 2019].
12. Goldbloom, A., 2016. *What algorithms are most successful on Kaggle?.* [Online]
Available at: https://www.kaggle.com/antgoldbloom/what-algorithms-are-most-successful-on-kaggle
[Accessed 7 August 2019].
13. Google Brain, 2015. *TensorFlow.* [Online]
Available at: https://www.tensorflow.org/
[Accessed 7 March 2019].
14. Google, 2013. *Google Code.* [Online]
Available at: https://code.google.com/archive/p/word2vec/
[Accessed 7 March 2019].
15. Hern, A., 2019. New AI fake text generator may be too dangerous to release, say creators. *The Guardian Online* , 14 February.
16. Hutchins, W. J., 2004. The Georgetown-IBM Experiment Demonstrated in January 1954. *Lecture Notes in Computer Science*, pp. 102-114.
17. IBM, 2018. *IBM.* [Online]
Available at: https://www.ibm.com/watson/think-2018/
[Accessed 1 June 2019].
18. Jivani, M. A. G., 2011. A Comparative Study of Stemming Algorithms. *International Journal of Computer Applications in Technology,* 2(6), pp. 1930-1938.
19. Kassabgi, G., 2017. *Text Classification using Neural Networks.* [Online]
Available at: https://machinelearnings.co/text-classification-using-neural-networks-f5cd7b8765c6
[Accessed 12 August 2019].
20. Kinsella, B., 2018. *Voicebot.ai.* [Online]
Available at: https://voicebot.ai/2018/12/24/rbc-analyst-says-52-million-google-home-devices-sold-to-date-and-generating-3-4-billion-in-2018-revenue/
[Accessed 1 June 2019].
21. Luca, M., 2016. *Harvard Business School.* [Online]
Available at: https://www.hbs.edu/faculty/Pages/item.aspx?num=41233
[Accessed 4 March 2019].
22. Mansfield, M., 2018. *Small Business Trends.* [Online]
Available at: https://smallbiztrends.com/2016/12/social-media-marketing-statistics.html
[Accessed 4 March 2019].
23. Markoff, J., 2011. Computer Wins on 'Jeopardy!': Trivial, It's Not. *New York Times.*
24. Mayo, M., 2018. *KDnuggets.* [Online]
Available at: https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html
[Accessed 5 March 2019].

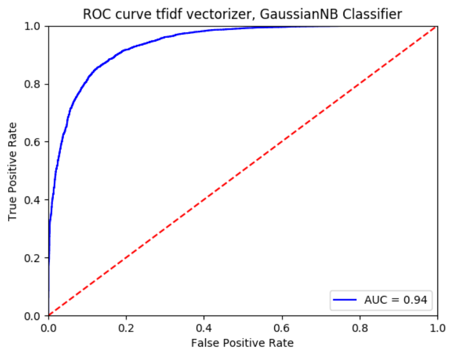
25. Porter, M., 1980. An algorithm for suffix stripping. *Program,* 14(3), pp. 130-137.
26. Quoc V. Le, T. M., 2014. *Cornell University.* [Online]
Available at: https://arxiv.org/abs/1405.4053v2
[Accessed 7 March 2019].
27. Rish, I., 2001. *An empirical study of the naive Bayes classifier,* s.l.: T.J. Watson Research Cente.
28. Schalkwyk, J. et al., 2010. *Google Search by Voice: A case study,* Mountain View: s.n.
29. Sebastian Raschka, V. M., 2017. A roadmap for building machine learning systems . In: *Python Machine Learning.* Birmingham - Mumbai: Packt, p. 11.
30. Srivastava, N. et al., 2014. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting,* Toronto: University of Toronto.
31. Subramanian , P. & Palaniappan , S., 2015. Telecom Data Integration and Analytics – Proposed Model to Enhance Customer Experience. *International Journal of Conceptions on Computing and Information Technology*, 01 October, p. 1.
32. Tang, A. et al., 2018. - Canadian Association of Radiologists White Paper on Artificial Intelligence in Radiology. *Canadian Association of Radiologists Journal*, 01 April, p. 123.
33. TensorFlow, 2019. *Case studies and mentions.* [Online]
Available at: https://www.tensorflow.org/about/case-studies
[Accessed 22 08 2019].
34. Tomas Mikolov, K. C. G. C. J. D., 2013. *Efficient Estimation of Word Representations in Vector Space.* [Online]
Available at: https://arxiv.org/pdf/1301.3781.pdf
[Accessed 7 March 2019].
35. Turing, A. M., 1950. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind,* LIX(236), pp. 433-460.
36. Wong, R., 2017. *Mashable.* [Online]
Available at: https://mashable.com/article/how-to-order-pizza-with-amazon-alexa-google-home/?europe=true
[Accessed 1 June 2019].
37. Yelp, 2018. *An Introduction to Yelp Metrics as of December 31, 2018.* [Online]
Available at: https://www.yelp.com/factsheet
[Accessed 4 March 2019].

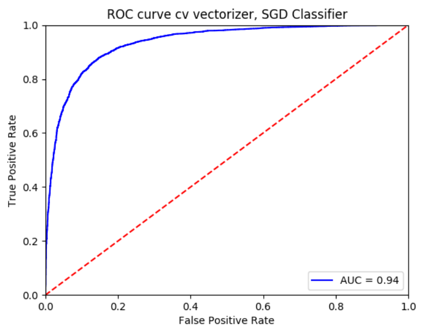
8.5 ROC – AUC plots for machine learning models

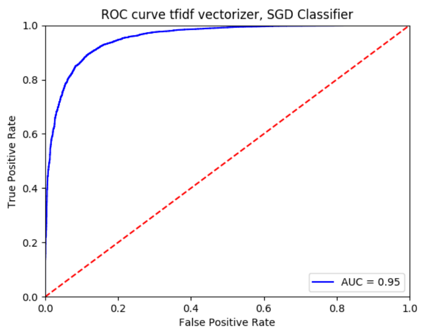


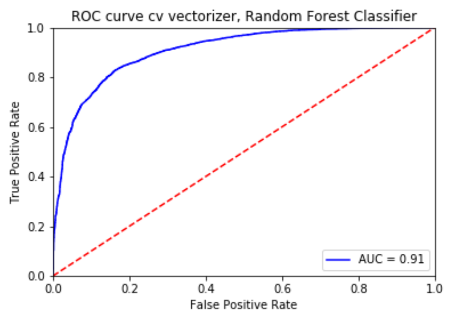


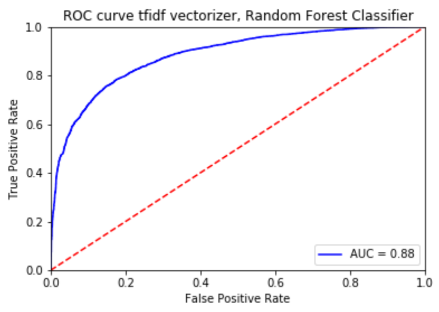












8.6 Python Scripts Used

Explorative analysis

```
# In[1]:
import pandas as pd
# In[2]:
yelp_review = pd.read_csv('../yelp_dataset/yelp_academic_dataset_review.csv')
# In[3]:
yelp_review = yelp_review.dropna()
yelp_review.date = pd.to_datetime(yelp_review.date)
yelp_review['year'] = yelp_review.date.dt.year

yelp_review.info()
# In[4]:
top_reviewed = yelp_review[yelp_review["stars"] > 3]
top_reviewed.head()
# In[5]:
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

top_reviewed.stars.value_counts().plot(kind="bar", rot=45, figsize=(10,5), grid=True, colormap='summer')
plt.title('Barplot of positive and negative reviews')
# ## Average and median of reviews per place: this metric gives an insight on which are the most reviewed places

# In[6]:
top_reviewed = yelp_review[yelp_review["stars"] >= 0]
```

```
top_reviews_dict = {}

for business_id in top_reviewed["business_id"].values:
    try :
        top_reviews_dict[business_id] = top_reviews_dict[business_id] + 1
    except:
        top_reviews_dict[business_id] = 1

topbusiness = pd.DataFrame.from_dict(data= top_reviews_dict,orient="index")
topbusiness.reset_index(inplace=True)
topbusiness.columns = ['business_id', 'rated']
del(top_reviews_dict)
# In[7]:
yelp_business = pd.read_csv('yelp_academic_dataset_business.csv')
top_count = 20
right=pd.DataFrame(yelp_business[['business_id',"name","categories", 'stars']].values, # remove stars for old plot
                   columns=['business_id',"Location name","categories",'stars'])

top_business_data = pd.merge(topbusiness,right=right, how= "inner",on='business_id')
top_business_data.sort_values("rated")[::-1][ : top_count].plot(x = "Location name",
                                          kind= "bar", figsize=(14,6),
                                          title= 'Total reviews per place', colormap='summer').set_ylabel("Total ratings")
del(topbusiness)
del(right)
# In[8]:
top_business_data.sort_values("rated")[::-1][ : top_count].plot(kind='bar',
                                                     x='Location name',
                                                     y='stars',
                                                     rot=80,
                                                    figsize=(14,6),
                                                    title='Top 20 reviewed businesses VS average of stars received')
# In[9]:
top_business_data.stars = top_business_data.stars.astype('int64')
top_business_data.corr().style.background_gradient(cmap='coolwarm').set_precision(2)
# No correlation between number of reviews and average number of stars received
# ## Positive reviews by place over time
# In[10]:
num_business = 2
business_ids = top_business_data.sort_values("rated")[::-1][:num_business].business_id.values
business_names = top_business_data.sort_values("rated")[::-1][:num_business]["Location name"].values
for i, business_id in enumerate(business_ids):
    useful_b = yelp_review.loc[yelp_review['business_id'] == business_id]
    useful_b = useful_b.groupby(['year']).size().reset_index(name='counts')
    series = pd.Series(useful_b["counts"].values, index=useful_b["year"].values, name='Review trend')
    axes = series.plot(kind="bar",figsize=(10, 7))
    plt.xlabel('Year', axes=axes)
    plt.ylabel('Total positive reviews', axes=axes)
    plt.title('Review trend of {}'.format(business_names[i]), axes=axes)
    plt.show()
# ## Frequency of reviews by date and time series decomposition for positive reviewed locations
# In[11]:
top_reviewed['y'] = 1
results = top_reviewed.groupby(['date']).count()
#results = results[['y']]
# In[12]:
idx = pd.date_range(results.index.min(), results.index.max())
results.index = pd.DatetimeIndex(results.index)
results = results.reindex(idx, fill_value=0)
results['ds'] = results.index
# In[13]:
from fbprophet import Prophet
m = Prophet()
m.fit(results)
# In[14]:
future = m.make_future_dataframe(periods=0)
# In[15]:
forecast = m.predict(future)
# In[16]:
fig1 = m.plot(forecast)
# In[17]:
fig2 = m.plot_components(forecast)
# ## Most reviewed businesses
# In[18]:
num_cat =10 # to show top 10 catrgories
top_business = 30 # choose categories of top 30 businesses
cat_data = top_business_data.sort_values("rated")[::-1][:top_business]
#cat_data.categories
Categories={}
for cat in cat_data.categories.values:
    all_categories= cat.split(",")
    for x in all_categories:
        try :
            Categories[x] =Categories[x]+1
        except:
                        Categories[x]=1
            top_categories = pd.DataFrame.from_dict(data= Categories,orient="index")
            top_categories.reset_index(inplace=True)
            top_categories.columns = ['category', 'occurance']
            x_val=top_categories.sort_values("occurance")[::-1][:num_cat].occurance.values
            labels=top_categories.sort_values("occurance")[::-1][:num_cat].category.values
            series = pd.Series(x_val, index=labels, name='Top business types')
            series.plot.pie(figsize=(10, 10),startangle=90)
            # ## Negatively reviwed businesses
            # In[19]:
            bottom_reviewed = yelp_review[yelp_review["stars"] < 3]
            bottom_reviews_dict ={}

            for business_id in bottom_reviewed["business_id"].values:
                try :
                    bottom_reviews_dict[business_id] =bottom_reviews_dict[business_id]+1
                except:
                    bottom_reviews_dict[business_id]=1

            bottombusiness = pd.DataFrame.from_dict(data= bottom_reviews_dict,orient="index")

            bottombusiness.reset_index(inplace=True)
            #bottombusiness.head()
            bottombusiness.columns = ['business_id', 'rated']
            # In[20]:
            top_count= 20
            right=pd.DataFrame(yelp_business[['business_id',"name","categories"]].values,
                               columns=['business_id',"Business name","categories"])

            bottom_business_data = pd.merge(bottombusiness,right=right, how="inner",on='business_id')
            bottom_business_data.sort_values("rated")[::-1][:top_count].plot(x="Business name",y="rated",
                                                      kind="bar",figsize=(14,6),
                                                      title='Negative reviews',
                                                                 rot=80,
                                                                colormap='summer').set_ylabel("Total 1 star ratings")
            del(bottom_reviewed)
            del(bottom_reviews_dict)
            del(bottombusiness)
            del(right)
            # ## Most common words in positive and negative reviews
            # In[4]:
            import re
```

```
from nltk.corpus import stopwords
from wordcloud import WordCloud
import collections
def tokenize(s):
    word_list = re.findall(r'\w+', s.lower())
    filtered_words = [word for word in word_list if word not in stopwords.words('english')]
    return filtered_words
def count_ngrams(lines, min_length= 2, max_length= 4):
    lengths = range(min_length, max_length + 1)
    ngrams = {length: collections.Counter() for length in lengths}
    queue = collections.deque(maxlen=max_length)

    def add_queue():
        current = tuple(queue)
        for length in lengths:
            if len(current) >= length:
                ngrams[length][current[:length]] += 1

    for line in lines:
        for word in tokenize(line):
            queue.append(word)
            if len(queue) >= max_length:
                add_queue()

    while len(queue) > min_length:
        queue.popleft()
        add_queue()

    return ngrams

def print_word_cloud(ngrams, num=5):
    words = []
    for n in sorted(ngrams):
        for gram, count in ngrams[n].most_common(num):
            s = ' '.join(gram)
            words.append(s)

    cloud = WordCloud(width=1440, height= 1080,max_words= 200).generate(' '.join(words))
    plt.figure(figsize=(20, 15))
    plt.imshow(cloud)
    plt.axis('off');
    plt.show()
                print('')
            # In[ ]:
            n_grams = count_ngrams(yelp_review['text'].values)
            # ### Negative reviews wordcloud
            # In[ ]:
            negative = yelp_review[yelp_review['stars'] < 3]
            negative_n_grams = count_ngrams(negative['text'])
            print_word_cloud(negative_n_grams)
            # ### Positive reviews wordcloud
            # In[ ]:
            positive = yelp_review[yelp_review['stars'] >=3]
            positive_n_grams = count_ngrams(positive['text'])
            print_word_cloud(positive_n_grams)
```

Random forest classifier

```
# Importing the libraries
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn import metrics
import matplotlib.pyplot as plt

nrows = 50000

# Importing the dataset
yelp_text = pd.read_csv('./yelp_stemmed_labelled.csv', nrows=nrows)

yelp_labels = yelp_text.labels.values

yelp_text = yelp_text.text.astype('U')

# Splitting dataset in train / test
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(yelp_text,
                                                    yelp_labels,
                                                    test_size=0.2,
                                        stratify=yelp_labels)

X_train

tfidf = TfidfVectorizer()
cv = CountVectorizer(max_features=1500)

# Hyperparameters grid
param_grid = {
    'clf__n_estimators': [100, 200, 500],
    'clf__max_features': ['auto', 'sqrt', 'log2'],
    'clf__max_depth' : [4,5,6,7,8],
    'clf__criterion' :['gini', 'entropy']
}

# Pipeline for CountVectorizer
lr_cv = Pipeline([('vect', cv),
                  ('clf',
                   RandomForestClassifier())])

# Pipeline for TfidfVectorizer
lr_tfidf = Pipeline([
                  ('tfidf', tfidf),
                  ('clf', RandomForestClassifier())])

pipelines = [{'cv' : lr_cv}, {'tfidf': lr_tfidf}]

results = {}

for index in range(len(pipelines)):
    for key in pipelines[index]:

        gs_sgd_tfidf = GridSearchCV(pipelines[index][key],
                            param_grid,
                            scoring='accuracy',
                            cv=10,
                            verbose=100,
                            return_train_score=True,
                            n_jobs=-1)

        gs_sgd_tfidf.fit(X_train, y_train)
    print('Best paramenter set: %s ' % gs_sgd_tfidf.best_params_)

    print('CV accuracy: %.3f' %gs_sgd_tfidf.best_score_)

    clf = gs_sgd_tfidf.best_estimator_
    print('Test accuracy: %.3f' %clf.score(X_test, y_test))

    df_name = next(iter(pipelines[index]))

    y_true = y_test
    y_scores = gs_sgd_tfidf.predict_proba(X_test)[:,1]
    auc = roc_auc_score(y_true, y_scores)

    results['{0}'.format(df_name)] = pd.DataFrame(gs_sgd_tfidf.best_params_, index=[0])
    results['{0}'.format(df_name)]['Vectorizer'] = df_name


    results['{0}'.format(df_name)]['best_score'] = gs_sgd_tfidf.best_score_
    results['{0}'.format(df_name)]['test_accuracy'] = clf.score(X_test, y_test)
    results['{0}'.format(df_name)]['AUC'] = auc


    # calculate the fpr and tpr for all thresholds of the classification
    fpr, tpr, threshold = metrics.roc_curve(y_test, y_scores)
    roc_auc = metrics.auc(fpr, tpr)

    # method I: plt
    title = 'ROC curve %s vectorizer, Random Forest Classifier' %df_name
    plt.title(title)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig('./' + title + '.png')
    plt.clf()
```

Neural Networks Models

```
# NN CV vectorizer
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from livelossplot.keras import PlotLossesCallback

from keras.models import Sequential
from keras.layers import Dense

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('retina')

get_ipython().run_line_magic('matplotlib', 'inline')
nrows = 50000

# Importing the dataset
yelp_text = pd.read_csv('./yelp_stemmed_labelled.csv', nrows=nrows)
yelp_labels = yelp_text.labels.values
yelp_text = yelp_text.text.values.astype('U')

from sklearn.feature_extraction.text import CountVectorizer

tf = CountVectorizer(max_features=1500)
X = tf.fit_transform(yelp_text)
del(yelp_text)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    yelp_labels,
                                                    test_size=0.2,
                                                    stratify=yelp_labels)
# # MODEL BASIC

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(1500,)))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy', auc])
history = model.fit(X_train, y_train,
          epochs=50,
          batch_size=32,
          validation_split=0.2,
          callbacks=[PlotLossesCallback()])
from sklearn.metrics import roc_curve
```

```
y_pred_keras = model.predict(X_test).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)

from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)
scores = model.evaluate(X_test, y_test)
print('Test accuracy:', scores[1])

# # MODEL DROPOUT
import tensorflow as tf
from keras import backend as K
from sklearn.metrics import roc_auc_score

def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    K.get_session().run(tf.local_variables_initializer())
    return auc

from keras.layers import Dropout
model_dropout = Sequential()
model_dropout.add(Dense(512,input_shape=(1500,)))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(128,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(64,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(32,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(1, activation='sigmoid'))
model_dropout.summary()

model_dropout.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy', auc])
model_dropout.fit(X_train, y_train, batch_size=32, epochs=50, validation_split=0.2, callbacks=[PlotLossesCallback()])
score_dropout = model_dropout.evaluate(X_test, y_test)
print('Test accuracy:', score_dropout[1])

from sklearn.metrics import roc_curve
y_pred_dropout = model_dropout.predict_proba(X_test).ravel()
fpr_dropout, tpr_dropout, thresholds_dropout = roc_curve(y_test, y_pred_dropout)
auc_dropout = auc(fpr_dropout, tpr_dropout)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Baseline (no dropout) (area = {:.3f})'.format(auc_keras))
plt.plot(fpr_dropout, tpr_dropout, label='Dropout (area = {:.3f})'.format(auc_dropout))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()

# Zoom in view of the upper left corner.
plt.figure(2)
plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Baseline (no dropout) (area = {:.3f})'.format(auc_keras))
plt.plot(fpr_dropout, tpr_dropout, label='Dropout (area = {:.3f})'.format(auc_dropout))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve (zoomed in at top left)')
plt.legend(loc='best')
plt.show()

# NN TFIDF Vectorizer
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from livelossplot.keras import PlotLossesCallback

from keras.models import Sequential
from keras.layers import Dense

nrows = 50000
# Importing the dataset
yelp_text = pd.read_csv('./yelp_stemmed_labelled.csv', nrows=nrows)
yelp_labels = yelp_text.labels.values
yelp_text = yelp_text.text.values.astype('U')
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer(max_features=1500)
X = tf.fit_transform(yelp_text)
del(yelp_text)

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    yelp_labels,
                                                    test_size=0.2,
                                                    stratify=yelp_labels)
# # MODEL BASIC
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(1500,)))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

model.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy', auc])
history = model.fit(X_train, y_train,
          epochs=50,
          batch_size=32,
          validation_split=0.2,
          callbacks=[PlotLossesCallback()])
from sklearn.metrics import roc_curve
y_pred_keras = model.predict(X_test).ravel()
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)

from sklearn.metrics import auc
auc_keras = auc(fpr_keras, tpr_keras)
model.evaluate(X_test, y_test)

scores = model.evaluate(X_test, y_test,
              batch_size=512)

print('Test accuracy:', scores[1])
# # MODEL DROPOUT
import tensorflow as tf
from keras import backend as K
from sklearn.metrics import roc_auc_score

def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    K.get_session().run(tf.local_variables_initializer())
    return auc
from keras.layers import Dropout

model_dropout = Sequential()
model_dropout.add(Dense(512,input_shape=(1500,)))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(128,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(64,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(32,activation='relu'))
model_dropout.add(Dropout(0.5))
model_dropout.add(Dense(1, activation='sigmoid'))
model_dropout.summary()

model_dropout.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy', auc])
model_dropout.fit(X_train, y_train, batch_size=32, epochs=50, validation_split=0.2, callbacks=[PlotLossesCallback()])

model_dropout.evaluate(X_test, y_test)
score_dropout = model_dropout.evaluate(X_test, y_test,
                        batch_size=512)
print('Test accuracy:', score_dropout[1])

from sklearn.metrics import roc_curve

y_pred_dropout = model_dropout.predict_proba(X_test).ravel()
fpr_dropout, tpr_dropout, thresholds_dropout = roc_curve(y_test, y_pred_dropout)
auc_dropout = auc(fpr_dropout, tpr_dropout)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Baseline (no dropout) (area = {:.3f})'.format(auc_keras))
plt.plot(fpr_dropout, tpr_dropout, label='Dropout (area = {:.3f})'.format(auc_dropout))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()

# Zoom in view of the upper left corner.
plt.figure(2)
plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_keras, tpr_keras, label='Baseline (no dropout) (area = {:.3f})'.format(auc_keras))
plt.plot(fpr_dropout, tpr_dropout, label='Dropout (area = {:.3f})'.format(auc_dropout))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve (zoomed in at top left)')
plt.legend(loc='best')
plt.show()
```

SGD CLASSIFIER

```
# Importing the libraries
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
import matplotlib.pyplot as plt

nrows = 50000
# Importing the dataset
yelp_text = pd.read_csv('./yelp_stemmed_labelled.csv', nrows=nrows)
yelp_labels = yelp_text.labels.values
yelp_text = yelp_text.text.values.astype('U')
```

```
# Splitting dataset in train / test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(yelp_text,
                                                    yelp_labels,
                                                    test_size=0.2,
                                                    stratify=yelp_labels)
tfidf = TfidfVectorizer()
cv = CountVectorizer(max_features=1500)
# Hyperparameters grid
param_grid = [{
                'clf__penalty': ['l2'],
                'clf__learning_rate': ['optimal'],
                'clf__loss' : ['log', 'squared_hinge', 'perceptron',
                               'squared_loss'],
                'clf__early_stopping' : [True],
                'clf__fit_intercept': [False],
                'clf__eta0': [0.01]
               }
               ]
# Pipeline for CountVectorizer
lr_cv = Pipeline([('vect', cv),
                  ('clf',
                   SGDClassifier(verbose=10))])
# Pipeline for TfidfVectorizer
lr_tfidf = Pipeline([
                  ('tfidf', tfidf),
                  ('clf', SGDClassifier(verbose=10))])
pipelines = [{'cv' : lr_cv}, {'tfidf': lr_tfidf}]

results = {}
for index in range(len(pipelines)):
    for key in pipelines[index]:
        gs_sgd_tfidf = GridSearchCV(pipelines[index][key],
                                    param_grid,
                                    scoring='accuracy',
                                    cv=10,
                                    verbose=100,
                                    return_train_score=True)
        gs_sgd_tfidf.fit(X_train, y_train)
        print('Best paramenter set: %s ' % gs_sgd_tfidf.best_params_)
        print('CV accuracy: %.3f' %gs_sgd_tfidf.best_score_)

        clf = gs_sgd_tfidf.best_estimator_
        print('Test accuracy: %.3f' %clf.score(X_test, y_test))
        df_name = next(iter(pipelines[index]))

        y_true = y_test
    y_scores = gs_sgd_tfidf.predict_proba(X_test)[:,1]
    auc = metrics.roc_auc_score(y_true, y_scores)

    results['{0}'.format(df_name)] = pd.DataFrame(gs_sgd_tfidf.best_params_, index=[0])
    results['{0}'.format(df_name)]['Vectorizer'] = df_name
    results['{0}'.format(df_name)]['best_score'] = gs_sgd_tfidf.best_score_
    results['{0}'.format(df_name)]['test_accuracy'] = clf.score(X_test, y_test)
    results['{0}'.format(df_name)]['AUC'] = auc

    # calculate the fpr and tpr for all thresholds of the classification
    fpr, tpr, threshold = metrics.roc_curve(y_test, y_scores)
    roc_auc = metrics.auc(fpr, tpr)
    # method I: plt
    title = 'ROC curve %s vectorizer, SGD Classifier' % df_name
    plt.title(title)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig('./' + title + '.png')
    plt.clf()
```

NAÏVE BAYES CLASSIFIER

```
# Importing the libraries
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn import metrics
import matplotlib.pyplot as plt

nrows = 50000
# Importing the dataset
yelp_text = pd.read_csv('./yelp_stemmed_labelled.csv', nrows=nrows)
yelp_labels = yelp_text.labels.values
yelp_text = yelp_text.text.astype('U')
# Splitting dataset in train / test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(yelp_text,
                                        yelp_labels,
                                        test_size=0.2,
                                        stratify=yelp_labels)
tfidf = TfidfVectorizer()
cv = CountVectorizer(max_features=1500)
# Hyperparameters grid
param_grid = [{'clf__alpha': [1, 0]}]
# Pipeline for CountVectorizer
lr_cv = Pipeline([('vect', cv),
                ('clf',
                 MultinomialNB())])
# Pipeline for TfidfVectorizer
lr_tfidf = Pipeline([
                ('tfidf', tfidf),
                ('clf', MultinomialNB())])
pipelines = [{'cv' : lr_cv}, {'tfidf': lr_tfidf}]
results = {}
for index in range(len(pipelines)):
    for key in pipelines[index]:
        gs_sgd_tfidf = GridSearchCV(pipelines[index][key],
                            param_grid,
                            scoring='accuracy',
                            cv=10,
                            verbose=100,
                            return_train_score=True,
                            n_jobs=-1)

        gs_sgd_tfidf.fit(X_train, y_train)

        print('Best paramenter set: %s ' % gs_sgd_tfidf.best_params_)
        print('CV accuracy: %.3f' %gs_sgd_tfidf.best_score_)
        clf = gs_sgd_tfidf.best_estimator_
        print('Test accuracy: %.3f' %clf.score(X_test, y_test))
        df_name = next(iter(pipelines[index]))
        y_true = y_test
        y_scores = gs_sgd_tfidf.predict_proba(X_test)[:,1]
        auc = roc_auc_score(y_true, y_scores)

        results['{0}'.format(df_name)] = pd.DataFrame(gs_sgd_tfidf.best_params_, index=[0])
        results['{0}'.format(df_name)]['Vectorizer'] = df_name
        results['{0}'.format(df_name)]['best_score'] = gs_sgd_tfidf.best_score_
        results['{0}'.format(df_name)]['test_accuracy'] = clf.score(X_test, y_test)
        results['{0}'.format(df_name)]['AUC'] = auc

        # calculate the fpr and tpr for all thresholds of the classification
        fpr, tpr, threshold = metrics.roc_curve(y_test, y_scores)
        roc_auc = metrics.auc(fpr, tpr)

        # method I: plt
        title = 'ROC curve %s vectorizer, GaussianNB Classifier' %df_name
    plt.title(title)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig('./' + title + '.png')
    plt.clf()
```

LOGISTIC REGRESSION CLASSIFIER

```
# Importing the libraries
import pandas as pd

nrows = 50000

# Importing the dataset
yelp_text_full = pd.read_csv('./yelp_stemmed_labelled.csv')
yelp_text = yelp_text_full.sample(n=nrows)
del(yelp_text_full)
yelp_labels = yelp_text.labels.values
yelp_text = yelp_text.text.values.astype('U')

# Splitting dataset in train / test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(yelp_text,
                                                    yelp_labels,
                                                    test_size=0.2,
                                                    stratify=yelp_labels)
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
import matplotlib.pyplot as plt

tfidf = TfidfVectorizer()
cv = CountVectorizer(max_features=1500)
# Hyperparameters grid
param_grid = [{
        'clf__penalty': ['l2'],
        'clf__C': [1.0, 10.0, 100.0],
        'clf__solver' : ['newton-cg', 'sag'],
        'clf__multi_class' : ['ovr', 'multinomial', 'auto'],
        'clf__fit_intercept': [False, True]
        }
```

```
         ]

# Pipeline for CountVectorizer
lr_cv = Pipeline([('vect', cv),
                ('clf',
                 LogisticRegression(verbose=0))])
# Pipeline for TfidfVectorizer
lr_tfidf = Pipeline([
                ('tfidf', tfidf),
                ('clf', LogisticRegression(verbose=0))])
pipelines = [{'cv' : lr_cv}, {'tfidf': lr_tfidf}]
results = {}

for index in range(len(pipelines)):
    for key in pipelines[index]:
        gs_lr_tfidf = GridSearchCV(pipelines[index][key],
                               param_grid,
                               scoring='accuracy',
                               cv=10,
                               verbose=5,
                               return_train_score=True,
                               n_jobs=-1)
        gs_lr_tfidf.fit(X_train, y_train)
        print('Best paramenter set: %s ' % gs_lr_tfidf.best_params_)
        print('CV accuracy: %.3f' %gs_lr_tfidf.best_score_)
        clf = gs_lr_tfidf.best_estimator_
        print('Test accuracy: %.3f' %clf.score(X_test, y_test))
        df_name = next(iter(pipelines[index]))

        y_true = y_test
        y_scores = gs_lr_tfidf.predict_proba(X_test)[:,1]
        auc = metrics.roc_auc_score(y_true, y_scores)

        results['{0}'.format(df_name)] = pd.DataFrame(gs_lr_tfidf.best_params_, index=[0])
        results['{0}'.format(df_name)]['Vectorizer'] = df_name
        results['{0}'.format(df_name)]['best_score'] = gs_lr_tfidf.best_score_
        results['{0}'.format(df_name)]['test_accuracy'] = clf.score(X_test, y_test)
        results['{0}'.format(df_name)]['AUC'] = auc

        # calculate the fpr and tpr for all thresholds of the classification
        fpr, tpr, threshold = metrics.roc_curve(y_test, y_scores)
        roc_auc = metrics.auc(fpr, tpr)

        # method I: plt
        title = 'ROC curve %s vectorizer, Logistic Regression Classifier' % df_name
        plt.title(title)
        plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
        plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig('./' + title + '.png')
plt.clf()
```

Abstract

In the era of user generated contents unstructured text represents a very wide source of information. The main goal of this study is to build different models, in order to recognize positive and negative labelled text. The text is processed using a series of Natural Language Processing techniques.
This study is conducted on the Yelp dataset, a collection of unstructured text containing around six-hundred thousands text reviews labelled with a score from 0 to 5 stars and posted by the users on the platform. An exploratory analysis is conducted, together with the implementation of different machine learning and deep learning techniques such as Logistic Regression, Random Forest, Naïve Bayes Classifier, Stochastic gradient descent classifier and Neural Networks.
The algorithms in the study are implemented in Python, using modern machine (and deep) learning packages such a scikit-learn and TensorFlow and modern data manipulation and visualization libraries such as Pandas, NumPy and Matplotlib.
All codes, snippets, plots and notebook are available in the appendix of this thesis and on the webpage http://robertosannazzaro.github.io under the section "Master Thesis".

DECLARATION OF THE AUTHOR OF THE CHAPTERS OF THE MASTER'S THESIS

titled ....

....

written by: ....Student No. ....

under the supervision of ....

Subjecting myself to legal liability, I hereby declare that I have written these thesis chapters independently and that no content included therein was obtained in contravention of the applicable regulations.

I also represent that the submitted thesis chapters have not been used before in any procedure conducted for the purpose of acquiring a professional academic title.

Equally, I declare this copy of the thesis chapters to be identical with the attached digital version.

I agree to have the thesis chapters checked and verified, also using plagiarism detection software, hereinafter referred to as 'software', as well as to have the tent of the thesis chapters stored in the comparative base to ensure protection against unauthorized use and passing the thesis to the National Diploma Thesis Repository

I also give my consent to the Warsaw School of Economics to process my personal data included in the thesis chapters to the extent which is necessary to perform verification with the software and to have them archived for free-of-charge access pursuant to the rules expressed in the order.

……………………………… ……………………………………..

(Date) (Author's signature)