Heriot-Watt University

Masters Thesis

Visualising Customer Reviews

Author:
Mael Antonin Evan Abgrall (H00295823)

Supervisor:
Dr. Stephano Padilla

A thesis submitted in fulfilment of the requirements
for the degree of Msc. Artificial Intelligence.
in the
School of Mathematical and Computer Science

April 2019

Declaration of Authorship

I, Mael Antonin Evan Abgrall, declare that this thesis titled, "Visualising Customer Reviews" and the work presented in itis my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form(e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Mael Abgrall

Date: 03/04/2019

Abstract

Online reviews are a very important section of websites for businesses and customers: it allows customers to make wiser choices and businesses to get feedback from their customers. However, some of those reviews are biased, either because the rating is too complex or too simple, or because they do not provide enough information, resulting in rating different from the review, or extreme rating with shorts reviews that lack useful information.

This project aims to find a way to replace the rating with artificial intelligence: an algorithm will analyse a review and automatically associate a rating with it, thus making the rating more in line with the text generated by the users. It will also leverage reviews that are too short since they usually lack useful information. The goal of the project being to propose a tool easy to use in python, that will provide this automated rating and weighting. This tool will allow to import already created models or to use the ones created during this project.

Acknowledgements

I would like to thank my supervisor, Stephano Padilla, for his motivation, guidance, advices and support during this project.

I would like to thank also my parents and my friends who supported me during the project.

Table of content

Table of figures

Table of formulas

Abbreviations

NLP: Natural Language Processing

UGC: User Generated Content

RNN: Recurrent Neural Networks

LSTM: Long Short-Term Memory

VSM: Vector Space Model

TF: Term Frequency

TF-IDF: Term Frequency Inverse Document Frequency

BOW: Bag of Word

ML: machine learning

CHAPTER 1: Introduction

1. 1. Motivation

Studies founds that user generated content (UGC) is extremely valuable for consumers: 95% of the customers read online reviews before making a purchase [1], 72% don't take action until they read reviews [2] and 97% of those customers are influenced by those reviews [3]. A study also found that customers prefers to trust UGC instead of traditional advertising [4].

Most of the UGC is designed in two parts: A text section where the customer can provide complex and detailed feedback, and a simplified visual feedback such as stars, thumbs up/down, or less often, Likert scales. The visual feedback being extensively used for filtering reviews or categorise them (for example showing only texts associated with 5 stars).

However, from personal experience and some web research, the simplified feedback suffers from a lot of distortion: the distribution of those opinions being polarized with many extreme positive and negative reviews, and a few moderate opinions [5][6] (example: someone without experience with a certain type of cuisine would rate a restaurant with the maximum number of stars, even if this restaurant is not actually very good, or an angry customer would put the less number of stars only because of over-reacting).

Fig. 1: two examples of distortion: On the left (From Amazon) a top rating without any real argument to support this rating. On the right (From Trip advisor) a review stating an "average" restaurant, but the user put the lowest rating.

The aim of this thesis is to reduce this bias by replacing user selected scale by an automatic scale: using artificial intelligence to analyse the text part, which can reveal a biased opinion, and attribute a score to this text: less positive texts having a low score, more positive having a higher score. On top of this, other simple text processing methods can be applied, weighting each review based on how detailed the text is.

An example on the figure 1 would be to have the following outcome:

- The amazon review should have a small weight due to the lack of details (thus impacting less the average score of the product).
- The restaurant rating should have a medium/high weight since it is more detailed, and a higher star rating that would match the user opinion.

1. 2. Aim & Objectives

The aim of this project is to use artificial intelligence to classify texts with a good accuracy as positive or negative.

Multiple objectives will allow to achieve this:

- Test multiple well-known methods of pre-processing.
- Test two of the most used representation of words.
- Test a simple but easy to understand ML algorithm (Decision trees).
- Test a widely used ML algorithm (Naïve Bayes).
- Test a more complex ML algorithm (Logistic Regression).
- And optionally try a very complex but performant ML algorithm (deep neural networks).
- Compare the result of each combination of pre-processing / word model / ML algorithm.

1. 3. recapitulative

Customer reviews are a very important part of a website, most of the engagement of a user depend on those reviews. Those reviews are split in two: a short text and a score. However, those reviews may have some issues such as being biased, not very accurate or having a score unrelated to the text. This project aims to solve part of those issues using artificial intelligence to classify reviews and automatically attribute a score to a given text.

CHAPTER 2: Literature Review

2. 1. Feature extraction

2. 1. a. Introduction

Feature extraction is a process of modifying a dataset to ease or improve the accuracy of a machine learning algorithm. It can be done by removing information that can be useless or harmful for the learning algorithm (ex: noise) or modifying it to make it easier to use (ex: replace words by numbers).

Machine learning algorithms can only be used with numbers. Since we are using text documents, the goal is to transform those texts into vectors of numbers. This thesis will focus on the three most used: Term frequency (2. 1. c.), term frequency inverse document frequency (2. 1. d.) and word2vec (2. 1. e.).

The conversion can create huge data structure: when converting the texts, usually the number of unique words will dictate the size of the data structure, pre-processing techniques will reduce the number of unique words, reducing the size of the data structure and possibly improve the accuracy of the ML algorithm. This thesis will focus on three of the most used methodologies: Lowercase texts (2. 1. h), Lemmatization/Stemming (2. 1.i) and removal of stop words (2. 1. j).

2. 1. b. Bag of word

Bag of word[7] is a representation of all the word present in a text in an unordered list. Bag of word in itself does not provide any other information and must be used with other methods to create a vector. Term frequency (2. 1. c.) and Term frequency inverse document frequency (2.1.d.) will add weights to the words presents in the representation given by the bag of words.

2. 1. c. Term Frequency

This method will simply count how many times a term occurred in a document and calculate its frequency.

$$Frequency_{term} = \frac{number\ of\ occurence_{term}}{total\ number\ of\ words\ in\ the\ document}$$

Formula. 1: The frequency formula for a single term (or word). This operation will be done for each word present in the bag of words model.

The frequency will be used as a weight for each term, thus creating a vector.

The advantage of this methods is its simplicity to use. However, this methodology has two drawbacks: It can create a huge data structure if there is a lot of unique words.

2. 1. d. Term Frequency – inverse document frequency

The term frequency inverse document frequency or TF-IDF [8] is a method to evaluate how important are words in a sample. The issue with term frequency is that highly frequent words in a sample will dominate even if they do not carry any useful information.

The methodology will rescale the weight depending on the frequency and how often the word appears in all samples. Words that are often in a text and present in a lot of other samples will be penalized (ex: if the word “method” have a high frequency in a certain document, and this word is used in a lot of documents, it will be penalized).

$$Inverse\ document\ frequency_{term} = log\frac{Total\ number\ of\ samples}{Number\ of\ samples_{term}}$$

Formula. 2: The inverse document frequency of a term (word).

Using the previous formula, we can construct the TF-IDF as follow:

$$Weight = Frequency \times Inverse\ document\ frequency$$

Formula. 3: TF-IDF

TFIDF share the same advantages and drawbacks as the TF algorithm.

2. 1. e. Word2Vec

Word2Vec [9] is a two-layer neural network (it is not a deep neural network). As the name implies, Word2vec will convert a word into a vector. Machine learning use two steps to be trained (See 2.2.a. and 2.3.a.): Prediction and correction. There is two way to realise the prediction step with word2vec: the continuous bag of words and skip-gram.

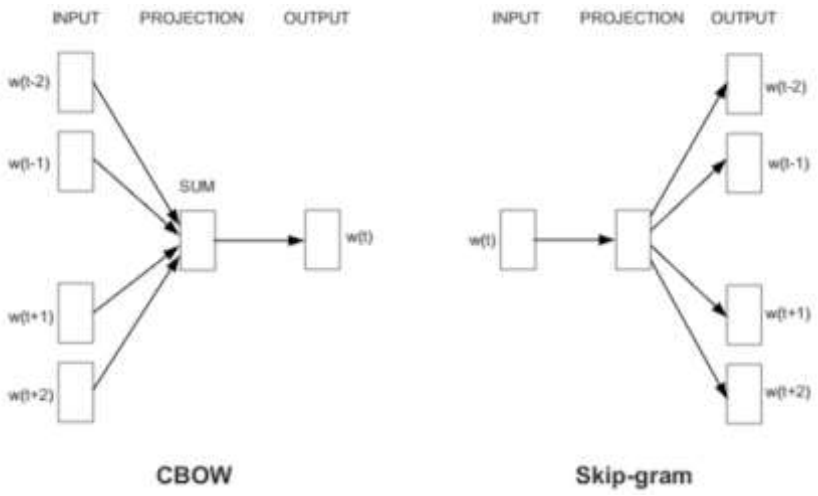


Fig. 2: Continuous bag of word and skip-gram training [10]

- Continuous bag of word: will take a full text, and given the context of this text, will try to predict a word.
- Skip-Gram: will take a word and try to predict the context.

The output of the word2vec is a vector with multiple dimensions (this is defined by hyper parameters; thus, it can change. By default, it is a 100 dimensions vector). The main feature of word2vec is the ability to cluster words using this vector: words that are usually of the same kind (ex: king and queen) will have a vector of numbers similar or very close.

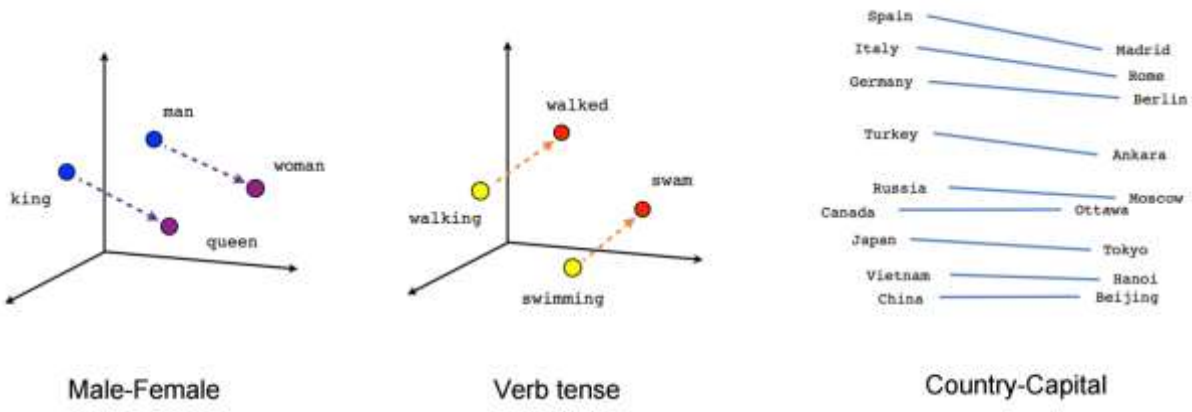


Fig. 3: Word2Vec clusters of words [11]

The advantage of word2vec over the bag of word models is the fixed shape: the dataset may have a huge number of unique words, the size of the data structure will still be as big as specified in the hyper parameters.

2. 1. f. Tokenization

Tokenization is the simpler word model of all: it simply builds a dictionary of the words present in a dataset. This dictionary will associate a unique integer with a unique word and allow later to convert a sentence into a sequence of integers (replacing each word by its associated integer.

Sample of the dictionary:
i : 2
had : 5
service : 12
great : 20
a : 23

Original	Converted
"I had a great service"	[2, 5, 23, 20, 12]

Fig. 4: Example of tokenization

The advantage of this method is its simplicity: it is fast to set up and easy to understand.

The main drawback of this method is that when used with large datasets, the dictionary be huge. The second drawback is its inconsistency in the output shape of the word model: ML algorithms needs a fixed shape as an input, while tokenization provides a shape depending on how many words there is in a sentence. The answer to this disadvantage is to use a fixed length, and crop the sentence, or add zeros to the output in order to match the fixed length.

This methodology is usually used with RNN (or any ML algorithm that make uses of time series), therefore it is not tested in this project.

2. 1. g. note about the word models

It is important to note a huge difference between the Bag of word models (2. 1. b., 2. 1. c. & 2. 1. d.) and the Tokenizer (2. 1. g.): the bag of word removes any temporality in the text (in other words, the specific organisation of words, if a certain word is placed before another in a sentence).

For example, the positive sentence “I’m happy, but not sad” and the negative one “I’m not happy, but sad” will be represented exactly the same way when used by a bag of word model as they have the same words and the same number of unique words.

Word2Vec can be used with or without temporality: It can be used as a more complex tokenizer (and keeping the temporality) or by averaging the overall review (each word is converted using word2Vec and the sentence is averaged). Only the second method is used since the project do not uses time series.

2. 1. h. Pre-processing with lower case

Before converting the texts into vector of numbers, an easy but useful pre-processing methodology is to convert all words to lowercase, thus reducing the number of unique words.

example: “Hello”, “hello”, “heLLo” are three unique words, converting those string to lower case (“hello”, “hello”, “hello”) will create a single unique word instead of three.

2. 1. i. Lemmatization/Stemming

The goal of stemming and lemmatization [13] is to replace words that have different forms (ex: analysis, analysing, analyse, …) by another one: Stemming will only “cut” the word (ex: analysis will become analys) while lemmatization will replace this word with the corresponding form in a dictionary (ex: analysing will become analyse).

As with lower case conversion, this methodology decreases the number of unique words present in a dataset.

2. 1. j. Removing stop words

Removing stop words from a text is simply removing a pre-defined set of words. Those words are the most common, most used words, usually used to have a correct grammar (“the”, “a”, “at”, “to”, etc). Those words, even if they are used a lot, do not add any meaningful information to the text.

This type of pre-processing can leverage the issues of some methodologies such as Term frequency.

2. 2. Machine learning

2. 2. a. Introduction

Machine learning is a subfield of artificial intelligence. It refers to algorithms able to learn patterns from data and classify it.

The process of learning is divided in two major steps:

- **Prediction:** The statistical model (or model) will take a set of inputs (for example pixels of an image) and predict a class associated with those inputs (ex: a dog).
- **Correction :** Given the predicted output, the algorithm will calculate the error, this error will then be used to correct the statistical model used for prediction.

The process of learning is to repeat those two steps with a lot of data and reduce the error as much as possible. Once the training is sufficient, only the first step is used (ex: what is this image -> a cat).

There is multiple way the correction can be done, and this is the main differentiation between the type of machine learning algorithms:

- **Supervised :** In this category of ML algorithms, the correction is made using a ground truth (the data has a label indicating what the prediction should be).

Fig. 5: Examples of supervised data in image segmentation. The left image is the data fed to the ML algorithm, while the right image is the "label" where red pixels represent the correct area for the plane. (Image from google)

- **Unsupervised:** In this category of ML algorithms, the correction is made by exploring the data and inferring classes from it (there is no need for labels).

Fig. 6: an example of unsupervised dataset: no labels are provided, and the ML algorithm will classify "hot-dog" or "not hot-dog" by itself. (image from google)

- **Semi-supervised:** this method is a mix of the two previous ones, a large dataset with only a small part of it labelled. Usually used when labelling is difficult, it can reduce a lot the volume of labelled data without impacting too much the performances of the ML model.

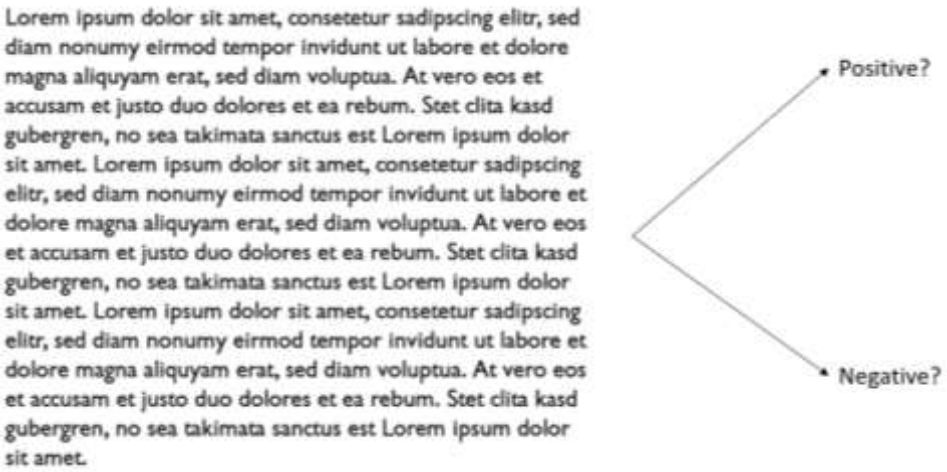


Fig. 7: an example of semi-supervised learning: the algorithm chooses some data from the dataset and ask that data to be labelled. (image from google)

- **Reinforcement:** this method will correct the model by interacting with its environment (trials and errors).

Fig. 8: an example of reinforcement learning: training a self-driving car: the goal for the agent is to avoid going out of the road. (image from google)

During the correction part, there is multiple way to calculate the error, each having perks and drawbacks depending on the item to classify (for example in image segmentation the Jaccard function can be very useful). Since there is plenty of functions to calculate the error, only the most common will be used in the project.

Only supervised learning machine learning will be used in this project: it is easy to implement and less time consuming compared to the other methodologies.

2. 2. b. Naïve Bayes

Naïve Bayes[14] is a model that work on the Bayes' theorem: the probability of an event based on prior knowledge.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Formula. 4: Bayes' theorem.

Where A is the class, and B the prior knowledge. Here is an example using text classification:

$$P(happy|text) = \frac{P(text|happy) \times P(happy)}{P(text)}$$

Formula. 5: Example of naive Bayes for text classification. Where P(Happy|text) is the probability that the text is happy.

The advantage of naïve Bayes is its low requirement in computational cost (it can be very efficient on a large dataset). Its drawback is when predicting a class without posterior probability: in this case the model is unable to make predictions

2. 2. c. Logistic regression

Logistic regression is one of the most used models for binary classification: it is a machine learning model that come from the logistic function (formula 7).

Logistic regression is made from another formula: the linear regression (formula 6). The linear regression is an easy to understand model: it just combines inputs (X) with weights (w) to output a value (Z).

$$Z = w0 + w1 * X1 + w2 * X2 + \cdots + wn * Xn$$

Formula. 6: The linear regression, w is a weight, and X an input value

This is very useful in some case of machine learning such as predicting a continuous value (ex: house prices depending on the surface for sale).

The issue with linear regression is when we are looking for discrete values (is it good or not good ?). This is where Logistic regression is used: to cap this continuous value within the range [0; 1]. The logistic regression simply uses the output of the linear regression (Z) and use it in the Sigmoid function (formula 7).

$$S(z) = \frac{1}{(1 + e^{-z})}$$

Formula. 7: Logistic function, where S(z) is in between 0 and 1, and Z being the input

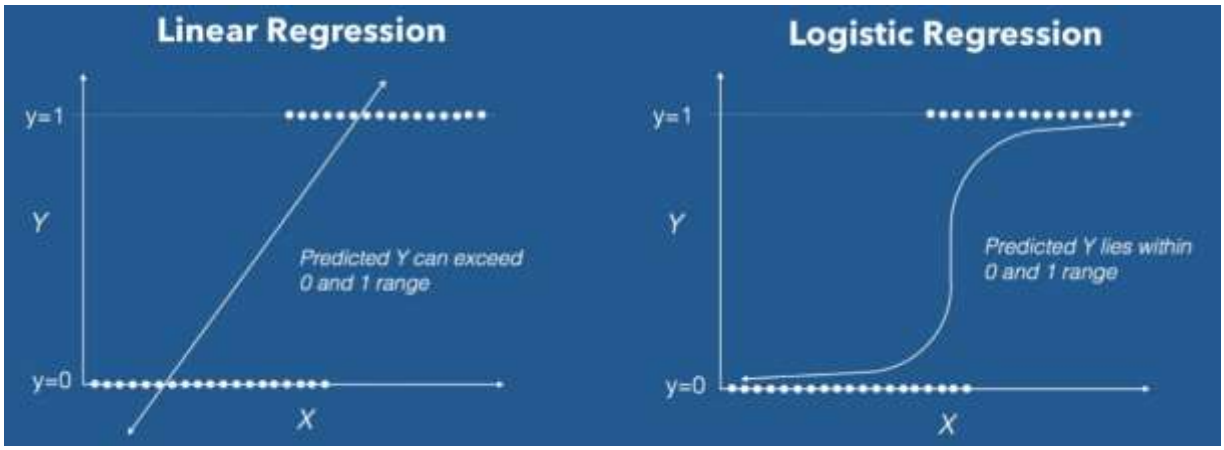


Fig. 9: Comparison of the linear & logistic regression (source: google)

The advantage of logistic regression is mainly it's low need in computation and relatively easy to understand. It is also used a lot by analysts and data scientists.

The drawback of logistic regression is that it does not handle very well a large number of features.

2. 2. d. Decision tree

Decision tree[15] is a simple tree model of decisions : at each condition (also called an internal node) the algorithm will choose a path to follow depending on this condition (each path is called a branch, or edge). Then end of the tree is the final decision (also called a leaf).

The difference between a “simple” decision tree and the machine learning model is the fact that the ML model is built automatically.

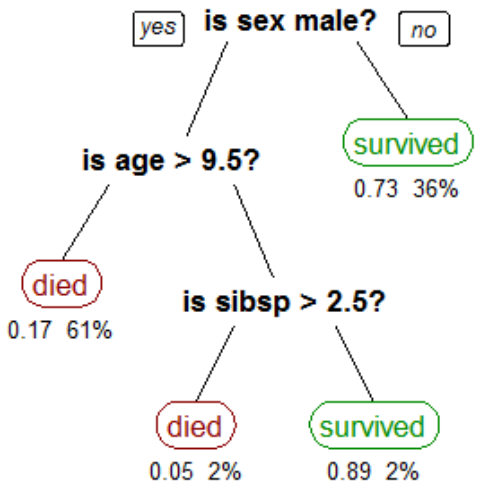


Fig. 10: A example of a decision tree on the Titanic data (image from Wikipedia)

Decision trees are used a lot in machine learning due to their simplicity: they are very easy to explain and to someone not involved in computer science and they can be visualised (see fig. 10 where a shallow decision tree can be seen graphically). The second big advantage of decision tree is the fact that they do not need a lot of data and data preparation.

Decision trees also have huge drawbacks: they are inefficient. Either because they are creating huge trees and overfit, or the opposite and lacking accuracy.

2. 3. Deep learning

2. 3. a. Introduction

Deep learning is a subfield of machine learning that use multiple layers of learning models, with each output of a layer being the input of the next layer.

The first deep network being a multilayer perceptron[12] (a perceptron is a model very close to the linear regression).

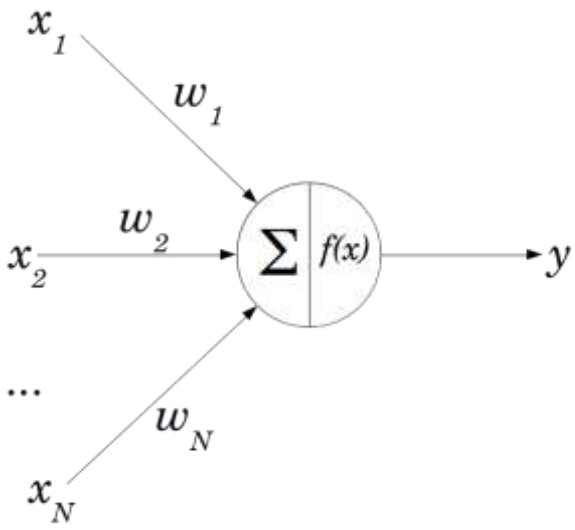


*Fig. 11: a perceptron. As in linear regression, the output is the sum of each weighted input (w1*x1 + w2*x2 + … + wn*xn) [16]*

The difference between a linear regression and a perceptron is the presence of an activation function (f(x)): as in real neurons, the activation function will decide is the neuron will be activated (and propagate information) or not.

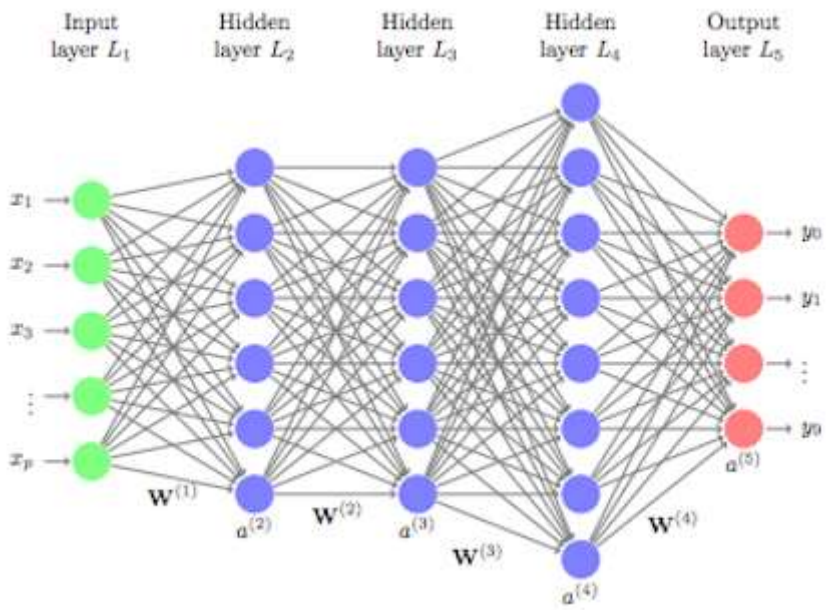


Fig. 12: a multiplayer perceptron (also called a feed forward network) [17]

There is no rule about how many / which type of layer to use. However, the type of layer will impact the result expected (for example convolutional layers are “learning” kernels. Kernels being originally used in image processing – edge detection, sharpening, etc – this type of layer is usually used in neural networks that process images).

A common idea is that each layer will learn a level of abstraction (see fig.) mapping the input (raw data) to the output (abstract idea).

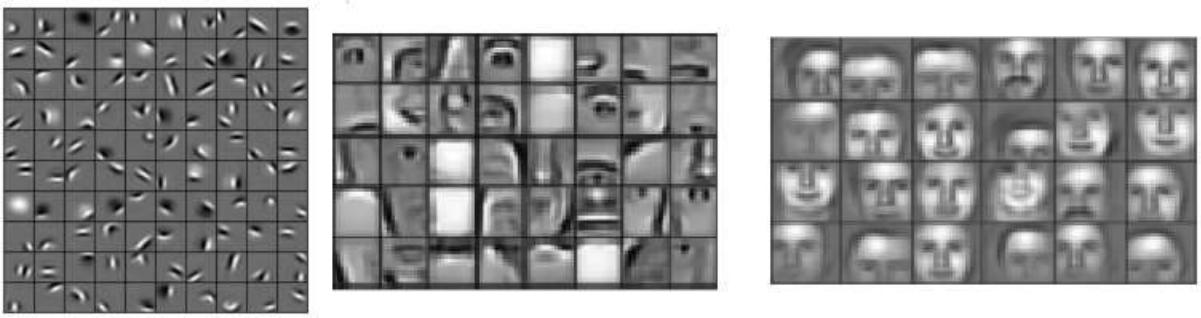


Fig. 13: output of multiple convolutional layers. The level of abstraction increases from left to right (simple shapes to faces). [18]

Since deep learning is a subfield of machine learning, the learning process (prediction / correction) is the same.

Usually the main advantage of deep learning is its accuracy: deep neural networks are very accurate and showed a huge leap forward when used correctly. For example, in the ImageNet challenge[19], the best “traditional” ML algorithm where struggling around 25% of errors, when AlexNet[20] was introduced, it achieved a 16% of errors, and by today most of the team competing have models with greater than 95% accuracy.

However, deep neural network have multiple big inconvenient: they need a lot of data to perform well, they are computationally expensive to put in place (a GPU is required, and even then, a training can takes days), and extremely difficult to understand (there is no rule of thumb for organising layers, what type of layer or the many hyperparameters needed for those layers).

2. 3. b. Recurrent Neural Networks

Recurrent neural networks (RNN) are a special kind of neural networks that take advantage of time series (data at t+1 is correlated to the data present at t). This is useful in text analysis.

This type of neural networks is using special layers that will “keep information in memory”: The Long short-term memory [21] and the Gated recurrent unit [22].

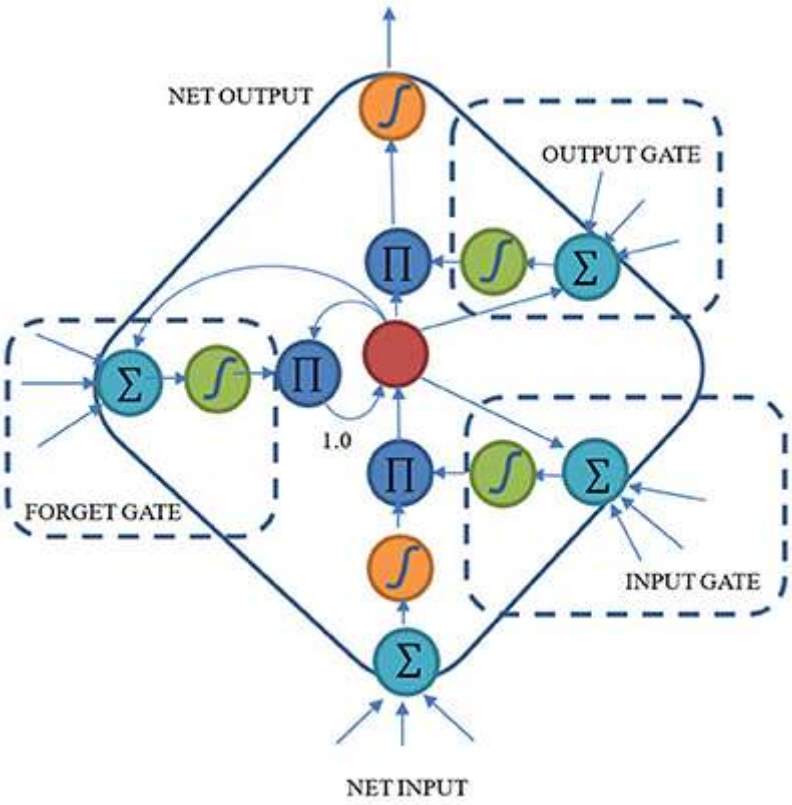


Fig. 14: LSTM cell [23]

The difference between a perceptron and a LSTM cell is the presence of 3 gates:

- Input gate: will control if the cell should accept the input information or not
- Output gate: control if the cell should output the content stored in the cell
- Forget gate: reset the content of the cell

The gated recurrent unit (GRU) is a simpler cell than LSTM that can have similar performances. The main difference is the fusion of the input and output gate into an “update gate”. This will allow to train the cell faster.

LSTM are better in huge datasets and/or when training time is not an issue, since they are more complex.

The advantage of RNN is their uses of time series, this is why it is used in video recognition, text to speech, and NLP.

The disadvantages are, as with feed forward neural networks, the need of a lot of data and the fact that it is resource intensive.

It is also important to note that when using bag of words model, RNN doesn't have any advantage over a simpler and easier to used feed forward network since the bag of word remove the temporality (see 2. 1. g.)

2. 4. Recapitulative

Feature extraction is commonly used in machine learning to simplify the process of learning and get better results. The methodology will "extract" meaningful information from the complete dataset and leave behind less important data. In this case it will help to reduce the size of a data structure, improve the accuracy and make the data understandable for a ML algorithm.

Different learning algorithm exist, this thesis focuses on Naïve Bayes, Logistic regression, decision trees and deep neural networks. They work in two steps: prediction and correction, which combined and repeated helps the algorithm to "learn".

Traditional machine learning algorithm perform well, however deep learning, which is a subfield of machine learning, can outperform traditional machine learning, if there are the required resources (data, and calculation). They can be difficult to tweak and tune too.

The process of sentiment analysis is a combination of feature extraction that will clean and organise the data and a machine learning algorithm that will learn what sentiment to predict with a given text.

CHAPTER 3: Methodology

3. 1. The data

3. 1. a. Data sources

There are multiple data sources available. The source used will be the large movie view dataset [24] because it is widely used and well explained. It is a collection of 50,000 labelled and 50,000 unlabelled reviews from IMDB. This dataset is split evenly between positive and negative reviews. No more than 30 reviews per film are present in the dataset since reviews about the same film tend to be similar.

The reviews with a rating >= 7 are considered positives and <=4 negatives (there is no neutral reviews). The rating is still accessible in order to try multiple classification instead of binary classification.

3. 2. b. Overfitting

A common issue in machine learning and deep learning is overfitting. Overfitting is an error that will make a model very accurate on a training data, but not very good (or sometime bad) when used on real data. In other words, the model does not generalise. To counter act overfitting (and also reveal it) we usually split the dataset in two different parts: a big part that will be used during training, and a smaller one used to validate the model (thus if the model is overfitted, it will not be able to predict validation data as well as train data).

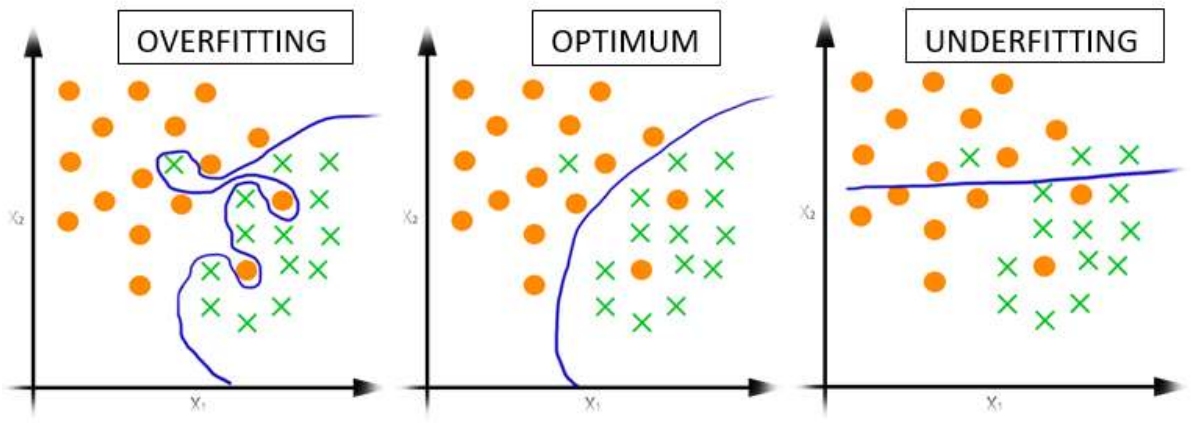


Fig. 15: Underfitting (the model doesn't learn anything), Desired model, and overfitted model. [25]

There is two main type of overfitting: lack of data and test overfitting

The first type of overfitting is simple to handle: you only need more data. And when it is not possible, multiple techniques can artificially increase the size of the dataset (for example image augmentation).

The second type of overfitting occurs when too much trial has been done on the validation dataset: every time a change has been made to the hyper-parameters to increase the accuracy on validation data, the model slightly overfit on validation data. To counter act this type of overfitting, the solution is to use either a dataset split in three or cross validation.

3. 2. c. Three split dataset

Splitting the dataset in three will produce the following subset: a training set used for training, a validation set used to see how the model is performing on “unseen” data, and at last, a test set used only once to see if the model has good performance on data never seen before.

The advantage of splitting the dataset in three is simplicity: setting up cross validation can be difficult and time-consuming

The disadvantage is that it is still possible to over fit a little bit on the validation set after a lot of trial and error when tweaking the hyper parameters

3. 2. d. Cross validation

Cross validation extends the concept of the three-split dataset: we keep three different subsets (training, validation and test), however this method does not keep the train and validation set static.

On K-fold cross validation, the dataset is divided on folds. At each run, the train set will be composed of all folds except one. The remaining fold will be used for validation. A run is a complete training/validation of the model, and a run will be executed for each train/validation configuration possible.

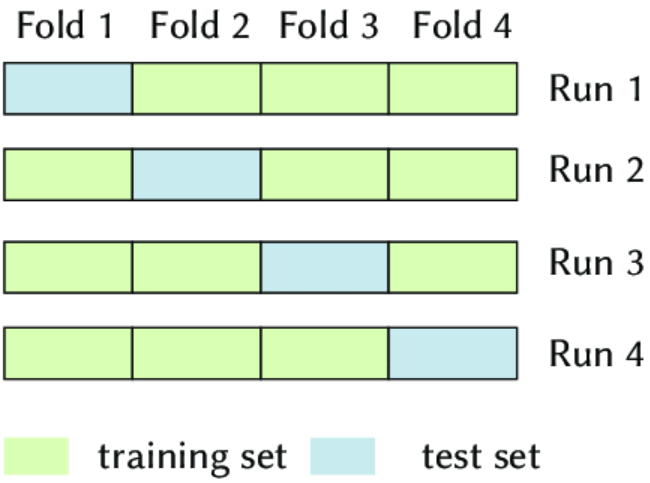


Fig. 16: K-fold cross validation [26]

The advantage of cross validation is a model more robust against overfitting. However, it can be difficult to set up, and it will increase a lot the time needed to create and evaluate model

3. 2. Evaluation & testing

3. 2. b. Evaluation

Accuracy will be used to evaluate the models. Simple to calculate, extensively used, the accuracy is the number of correctly classified samples divided by the total number of samples

$$\text{Accuracy} = \frac{Correct}{Total}$$

Formula 8: Accuracy is calculated using the number of correctly classified samples divided by the total number of samples classified.

Other types of evaluation exist (Precision and Recall). However, those types of evaluations are not very suited for this task: They each focus on a specific type of misclassification to make a model efficient on important tasks.

- **Precision:** focuses on reducing false positives, ex: Mail categorised as spams is bad, so too much mail misclassified as spam will reduce the precision of the model, even if the accuracy is good.
- **Recall:** focuses on false negatives, ex: a fraud categorised as a normal transaction. Recall will be low on a model where too many frauds has been misclassified as normal transactions, even if the model accuracy is good.

Since sentiment analysis do not need to focus on a certain type of badly classified sample, those metrics will not be used.

When using cross validation, each run will produce a model with a different accuracy, however those are usually similar (very low standard deviation). The average accuracy is commonly used as a "global" accuracy for a model when trained with cross validation.

3. 2. b. Testing

The testing will try every combination of pre-processing, feature extraction and machine learning model possible.

For each combination of pre-processing (feature extraction on words) a pre-processed dataset will be created:

- **1st dataset:** No pre-processing
- **2nd dataset:** Stemming only
- **3rd dataset:** stop words removal only
- **4th dataset:** stemming and stop word removal
- **5th dataset:** intelligent lower-case conversion only (added later to the project, more about the intelligent conversion in the project changes, 7. 1. b.)
- **6th dataset:** intelligent conversion and stemming
- **7th dataset:** intelligent conversion and stop word removal
- **8th dataset:** intelligent conversion, stop word removal and stemming
- **9th dataset:** lower case conversion only
- **10th dataset:** lower case conversion and stemming
- **11th dataset:** lower case conversion and stop word removal
- **12th dataset:** lower case conversion, stop word removal and stemming

For the twelve datasets created, each word model (TF, TF-IDF and word2vec) will be created and trained on the dataset (so there is three word model for each pre-processed dataset).

For each word model created, the script will create and train each ML model (traditional ML models: Naïve Bayes, Logistic Regression and Decision Tree) and then evaluate it. The training/evaluating will be done using a 10-fold cross validation, and the data saved will be the average and the standard deviation of the score of each ML model.

There is a total of 1080 tests done (each pre-processed dataset * each word models * each ML algorithm * 10-fold cross validation).

CHAPTER 4: Requirements

The goal is to propose a python module usable in a real environment such as movie rating, restaurant rating, etc.

4. 1. Mandatory

4. 1. a. Interface & ease of use

The development will follow as much as possible the UNIX philosophy, especially those ideas: functions should do only one thing, functions, classes and methods needs to be clear and intuitive.

The module has to be as flexible as possible: for example, let the user select a pretrained model, or use one of his own, this will also ease tests and comparison of the different models.

Here are some basic design ideas:

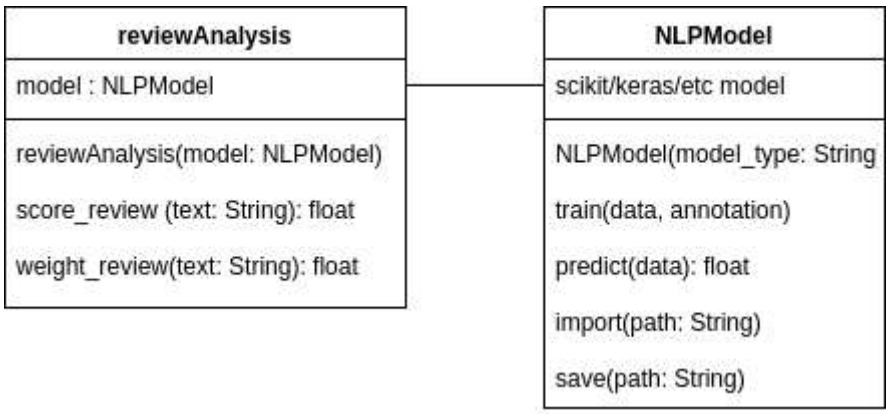


Fig. 17: UML mock-up

The class NLP model would be used to add a layer and unify different python modules used to train or use a model (example: access a scikit or a keras model via a simple train/predict method).

4. 1. b. Mandatory requirement checklist

The following table lists all the mandatory requirements and the estimation of time for each requirement.

1. **Project setup (Virtual environment, Git, Dependencies, …):** in order to track the project advancement and comply with the professional issues (5.1), the first step of the project is to create a virtual environment and set up git.
2. **Dataset split:** In order to have results as objective as possible, and avoid Overfitting (3.2.b), two various way of splitting the dataset should be implemented: The three split style and cross validation.
3. **Term Frequency:** The first Bag of Word model to convert texts into vector of numbers, with an interface to make easy the creation, usage and save of the model.
4. **TF-IDF:** The second Bag of Word model, using the same interface as TF.
5. **Word2Vec:** Implementation of word2Vec, using the same interface as TF and TF-IDF.
6. **Lemmatization/Stemming:** create a function to process batch of texts.
7. **Removing stop words:** create a function to remove stop words in a batch of texts.
8. **Lower case conversion:** create a function to convert a batch of text to lower case.
9. **Machine learning models:** The three machine learning models presented in 2.2 should be usable using the library SKLearn.
10. **Automatic testing:** create a script to automatically train and evaluate each combination of pre-processing and ML model.
11. **Modularity:** the system should be usable without the automatic script (see requirement 9), with custom models or only part of the program

4. 2. Optional

4. 2. a. Testing deep learning

Once the mandatory requirements will be fulfilled, I will try to implement deep learning techniques to sentiment analysis: From simple networks (fully connected) to more complex ones (RNN).

Deep learning is an optional requirement since setting up the libraries, training and fine tuning the neural network are all requiring a lot of time and adequate hardware.

4. 2. b. Adversarial attacks

When all mandatory requirements will be fulfilled and all optional requirement too, I will try to generate adversarial examples to attack the models and try to defend them.

CHAPTER 5: Professional, Legal, Ethical and Social issues

5. 1. Professional issues

The language used will be Python for its simplicity and abundance of artificial intelligence libraries: the language allows to create prototypes very fast and easily.

The project will be using a virtual environment to unsure that there is no library conflict and that the program can be run on another machine and/or another operating system without any issue.

The program will be heavily commented for clarity and documentation strings will be as extensive as possible.

Git and GitHub will be used for version management, and the repository will be public.

The IDE used is Visual studio code, and the code will be runnable & tested on Fedora and CentOS.

5. 2. Legal issues

Since the project will use various open source libraries, all licenses and copyright will be respected and mentioned in the “readme.md” file.

Every software used to help creating the project will be open source (SSH, SSHFS, Visual studio Code, Python, virtualEnv, Git)

At last, the project will be present on GitHub with the open source MIT licence.

List of libraries used (and why):

- **NumPy[27]:** powerful & fast data structures
- **SKLearn[28]:** ML algorithm & shuffling the data during loading
- **Beautiful soup[29]:** data cleaning (noise removal)
- **NLTK[30]:** data cleaning (stop words & stemming)
- **Gensim[31]:** word embedding model (Word2Vec)
- **Pandas[32]:** easy manipulation of csv files
- **Keras[33]:** ease the use of TensorFlow
- **Tenforflow-gpu[34]:** deep learning
- **IPDB:** interactive python debugger
- **Pylint:** python linter
- **autoPep8:** automatic code formatting

5. 3. Social & Ethical issues

This project does not involve users, so there is not a lot of social and ethical issues. However, there is still some issues related to the dataset used.

There can be some issue when using machine learning: working with sensitive data is not a good idea, studies shown that it is possible to retrieve training data from a neural network [35], or some data would create an ethical issue (ex: using ethnicity in the inputs).

The IMBD movie review is anonymous and it is coming from a public website, by common sense, sensitive data should not be present since people do not share this in public.

CHAPTER 6: Plan

The Kanban methodology combined to Agile will be used to track the progress of the project. Each sprint has a duration of one to two weeks.

The main deadlines are stated here:

13 May 19 May	Setup of the project & git, understanding / opening the dataset
20 May 2 June	Pre-processing of the dataset (Feature extraction)
3 June 16 June	Bag of Word (TF & TF-IDF)
17 June 30 June	Word2vec
1 July 14 July	Implement traditional ML models
15 July 28 July	if no issues have been raised, start working on deep learning & optional requirements
29 July 8 August	Work on the draft of the dissertation
8 August 15 August	Review & corrections of the draft, hand down

Tracking of the project has been made using a whiteboard (see appendix 1). Each week is represented as a case, and each line represent a month. This allows to track very fast and easily how many weeks are left for a specific deadline, and how late/early deadline impact on the whole project.

CHAPTER 7: Progress of the project & results

7. 1. progress & ameliorations on the project

7. 1. a. using Regex and cleaning the dataset

The first stage of the project was to clean the data: some html code was still present in the dataset, multiple spaces, etc. The library “Beautiful soup” was used to clean the text and remove html code.

Another problem was the usage of words with recurring letters (example: “yaaaaaawwwnn”), increasing the number of unique words. Regular expression where used to remove multiple spaces and replace words with too many recurring letters (such as in the example) with the correct word (here “yawn”). This little change allowed to significantly reduce the number of unique words.

Once cleaned, the dataset was saved in another format than the original: the original dataset was organised with one file for one review, the rating being in the title of the file. For faster load, the dataset was saved in a csv format containing all reviews and their labels (one file for testing and one file for training), reducing the number of files, and reducing the time needed to open the entire dataset.

On the code below, the html removal is done on line 80, removal of punctuation & non letters on line 83, and the removal of words with recurring letters on line 88.

```
70  def noise_removal(text):
71      """remove html tags and non letter characters from a given text and return a list of cleaned words
72
73      Arguments:
74          text {string} -- text
75
76      Returns:
77          list -- cleaned text
78      """
79      # Remove HTML
80      text = BeautifulSoup(text, "html.parser").get_text()
81
82      # Remove non-letters
83      text = re.sub("[^a-zA-Z]", " ", text)
84
85      # remove letters that are used more than three times in a row
86      # sources: https://en.oxforddictionaries.com/explore/words-with-same-letter-three-times-in-a-row/
87      # https://stackoverflow.com/questions/4574509/remove-duplicate-chars-using-regex
88      text = re.sub(r'([\w])\1{2,}', r'\1', text)
89
90
91      word_list = text.split()
92      return word_list
```

This step added to the project is called noise removal and is used on all the tested cases.

7. 1. b. intelligent conversion to lower case

The second step in the project was to convert all reviews to lower case. A small change was done during the portion of code made to convert the dataset to lower case: the script now has a choice of doing either nothing, converting all words to lower case, or do a “smart” conversion. The idea was that sometimes people write reviews or words with all letters capitalised to emphasise on the word (this is commonly considered as yelling on internet).

The smart conversion will simply look at the number of upper-case letters in a word, if the entire word is upper case, the word will not be converted to lower case.

Original	**Convert everything**	**“smart” conversion**
Hello my name is Marc	hello my name is marc	hello my name is marc
I am ANGRY	i am angry	I am ANGRY

Fig. 18: comparison of full conversion versus “smart” conversion

The code below is analysing each word in a review, and will either convert it to lower case, or keep it in all case if the “intelligent” parameter is set to True.

```
def lowercasing(list_of_words, intelligent=False):
    """lowercase a given text. The "intelligent mode" will not lowercase words fully uppercase (ex: STOP and Stop)

    Arguments:
        list_of_words (list) -- list of words in a text

    Keyword Arguments:
        intelligent (bool) -- use the intelligent mode or not (default: (False))

    Returns:
        list -- list of words lowercase
    """
    curated_list = []

    for word in list_of_words:
        # if intelligent mode is on and the word fully uppercase
        if(intelligent and word.isupper()):
            curated_list.append(word)

        # if the intelligent mode is on, and the word is not fully uppercase
        if(intelligent and word.isupper() == False):
            curated_list.append(word.lower())

        if(not intelligent):
            curated_list.append(word.lower())
    # end of loop

    return curated_list
```

7. 1. c. Creation of pre-processed datasets

Since the pre-processing is a bit time consuming, multiple files where created, each of those files corresponding to a specific combination of feature extraction (as stated in 3.2.b). Those datasets where identified by their name (reused in the results):

- The first letter (T/F/I) is the conversion to lower case (True, False or Intelligent)
- The second letter correspond of the removal of stop words (True/False)
- The last letter corresponds to the application of stemming on words (True/False)

The cleaning, conversion/pre-processing of the dataset is done only once, and is handled by a separate script (convert_dataset.py, located in the root of the source code, see Appendix 5). This script takes also care of downloading/updating the data needed by NLTK for stemming and removing stop words.

Fig. 19: All the cleaned sets

7. 1. d. creation of folds for cross validation

The first big issue came from the size of the dataset: the code is working at the moment using data in RAM and not from the disk, the entire dataset is loaded in the memory. The issue is when using cross validation, creating all the folds at the same time resulted in an out of memory error. The change made was to create folds on the fly for training: the script will have at disposition X number of subsets (where X is the number of folds desired) that will be concatenated before training/evaluating

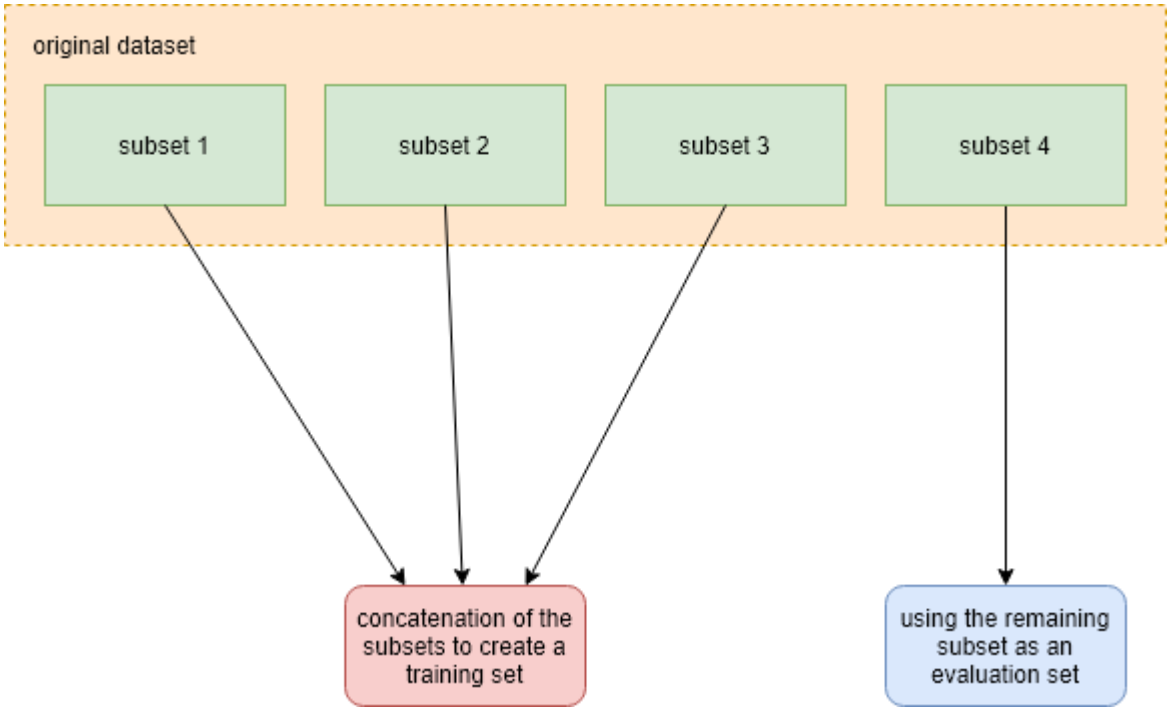


Fig. 20: Creation of folds on the fly. The scrip will use the increment in the for loop to concatenate the subsets (example: on the first pass, training will be composed of 2, 3 and 4, on the second pass, training will be composed of 1, 3 and 4, etc. This figure represents the 4th pass)

7. 1. e. Automatic training and evaluating

The process of training and evaluating each fold is long, even for a simple machine learning program. The need to automate training and evaluating for all test case was required.

The main file is a script is taking care of this (see appendix 5). This script is divided in four section for more visibility (each of this section called by a loop):

- **The first part** will load sequentially each dataset present in the source folder. And call the function to test each word models (second part).
- **The second part** will create a word model for each word model available (TF, TFIDF and word2vec). When the word model has been created and trained, this function will call the third part of the program.
- **The third part** of the program will test every ML model available in the code (Naïve Bayes, decision tree and logistic regression) and call the 4th part of the program.
- **The fourth part** of the program will simply save a string containing all the relevant data (all accuracy results obtained during cross validation, the mean and standard deviation). This string is shown in the figure 21.

The overall script took 4 to 5 days to do the 1080 tests and was run on the "Jove" cluster of MACS.

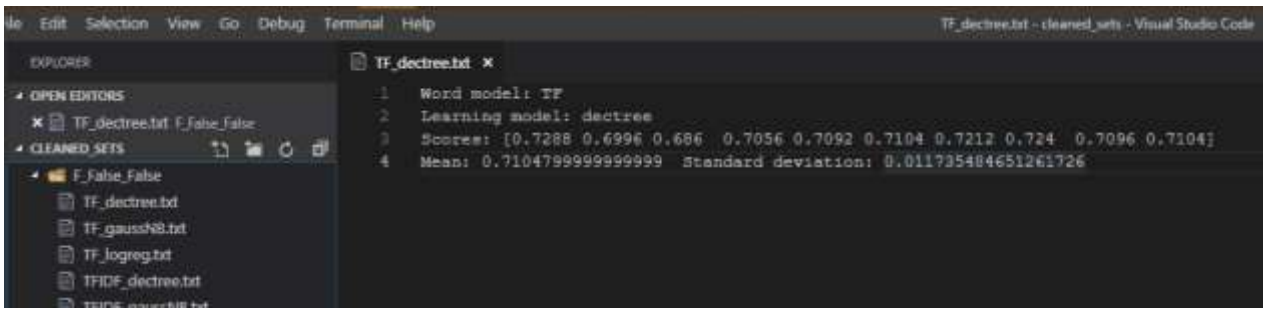


Fig. 21: a sample of the saved result

7. 1. f. modules created

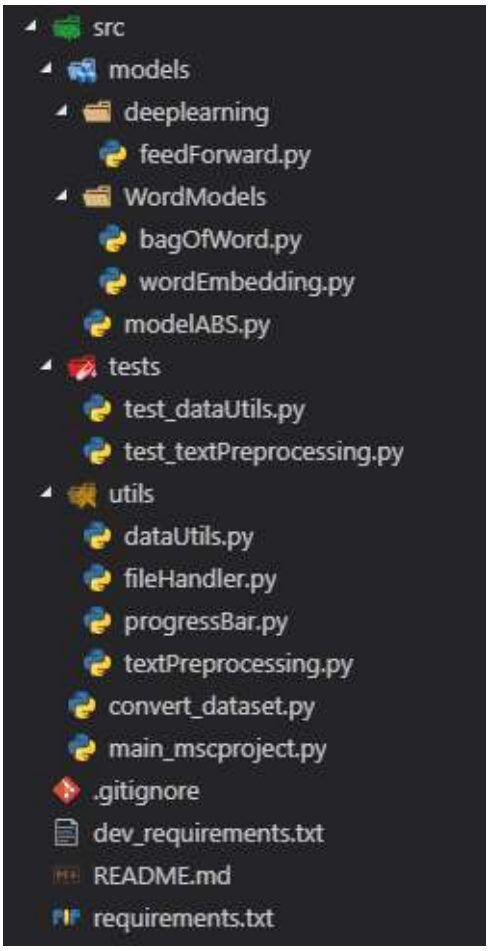


Fig. 22: List of the files created.

Project root: .gitignore, readme, requirements and dev_requirements are all the files related to understanding the project, set up the python environment and git.

src/ : convert dataset and main file are located here (see appendix 4 & 5), those are the scripts run to either clean and convert the dataset or run all test case.

src/utils/ :

- **dataUtils.py:** this module contains functions related to data conversion (three split, cross validation, cleaning the dataset, etc.
- **fileHandler.py:** this module contains the functions related to opening or saving csv files, NumPy files, pickle files and the function to load the original (non csv) dataset.
- **progressBar.py:** used during debugging to see the progression of a loop. Should not be used when logs are saved to a text file (nohup).

- **textPreprocessing.py:** this module has all functions related to text processing (lower case, stemming noise removal and stop word removal).

src/tests/ : contain the unit tests for dataUtils.py and textPreprocessing.py.

src/models/ : contain the modelABS.py, abstract class for word models.

src/models/wordmodels/ :

- **bagOfWard.py:** class to create a bag of word model (TF or TF-IDF).
- **wordEmbedding.py:** class to create a word embedding model (word2vec).

src/models/deeplearning/:

- **feedForward.py:** class to create a feed forward neural network.

There is no module for traditional machine learning because the library SKLearn is already providing a unified interface to multiple ML models. Deep learning is implemented, but the code is not working with the current dataset. Since the script using the deep learning model is feeding the entire dataset at once to the GPU: the program crashes because of a memory error.

7. 3. Results

pre-processing		Decision tree	Naïve Bayes	Logistic regression
TF	F F F	0.7104	0.6721	0.8819
	F F T	0.7207	0.6641	0.8741
	F T F	0.721	0.6651	0.8797
	F T T	0.7196	0.6629	0.874
	I F F	0.7111	0.6725	0.8824
	I F T	0.723	0.6634	0.8748
	I T F	0.7232	0.6651	0.8818
	I T T	0.7184	0.6619	0.875
	T F F	0.7138	0.674	0.8843
	T F T	0.7203	0.6637	0.8741
	T T F	0.7272	0.6655	0.8792
	T T T	0.7202	0.6647	0.8726
TFIDF	F F F	0.704	0.6739	0.8893
	F F T	0.7123	0.6726	0.8878
	F T F	0.7163	0.6724	0.8925
	F T T	0.7107	0.6727	0.887
	I F F	0.6967	0.6734	0.8886
	I F T	0.7158	0.6718	0.8878
	I T F	0.7196	0.6732	0.8922
	I T T	0.7183	0.6713	0.8874
	T F F	0.7025	0.6757	0.8892
	T F T	0.7146	0.6716	0.8872
	T T F	0.7185	0.6729	0.8928
	T T T	0.7138	0.6722	0.887
W2V	F F F	0.7041	0.6836	0.8583
	F F T	0.741	0.7694	0.865
	F T F	0.733	0.7786	0.8673
	F T T	0.721	0.7708	0.8647
	I F F	0.6939	0.688	0.8582
	I F T	0.7322	0.7675	0.8643
	I T F	0.7206	0.7732	0.87
	I T T	0.7341	0.7695	0.8628
	T F F	0.6868	0.683	0.8563
	T F T	0.7233	0.7777	0.8649
	T T F	0.7253	0.773	0.8668
	T T T	0.7233	0.7722	0.8654

Fig. 23: The results of the script.
The pre-processing letters are respectively lower case, stop words and stemming (see 7. 1. c.)

It is important to note that since the process is automatic (and very long), no hyper parameter tweaking has been done: all the results are achieved using the default parameters of SKLearn: those results could be better (or worse) with fine tuning of each model. The same has been done with Word2Vec: the default parameters of using a 100-dimension vector is used and no fine tuning has been made.

Looking at those results and the chart below (see fig. 23 & appendix 2), the first thing that can be seen is the huge difference between logistic regression and the other two ML algorithm: Logistic regression has an average of 16 – 17 % more accuracy than the other two, and the worst result was still performing at 85.63% while the bests from the other two were 74.1% (Decision trees) and 77.86% (Naïve Bayes). The second thing that can be seen is the impact of lemmatization on logistic regression: training on a dataset without lemmatization gave almost always the best results. At last, logistic regression seems to perform better using bag of word models.

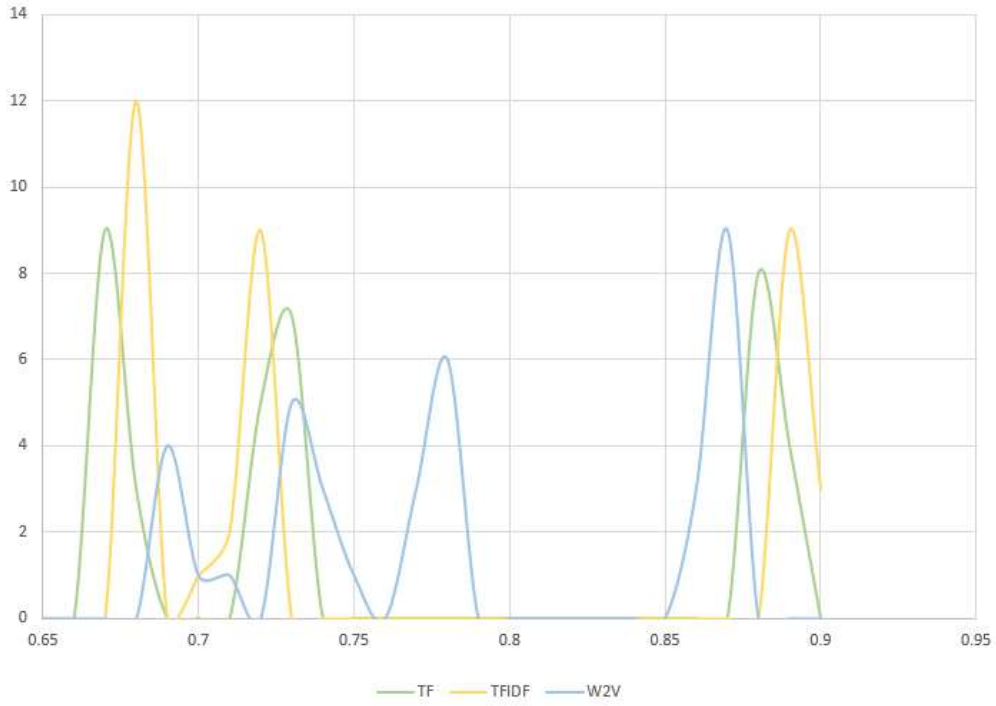


Fig. 24: Distribution of each word model results. The X axis represent the accuracy, the Y axis represent the number of samples. This figure was done using excel charts and a bin size of 0.01

When looking at the sorted results by word model (Appendix 3), excluding logistic regression, decision trees are performing better than naïve Bayes when combined with TF, while naïve Bayes is performing better than decision trees when combined with Word2Vec. However, when looking only at naïve Bayes and decision trees, the dataset pre-processing do not show any significant impact on the accuracy.

CHAPTER 8: Conclusion & perspectives

8. 1. Perspectives & Future work

Every mandatory requirements and objectives of this project have been completed. However, two more weeks could have helped to get results on the optional requirements (Deep neural networks).

Due to the lack of time, the most powerful machine learning techniques haven't been tested (the current code is overloading the GPU memory, so the deep neural network cannot train). However, we can theorise that even a simple feed forward neural network will be very accurate: as seen in the section 2.3.a, feed forward networks are "more powerful" version of the linear regression, which is almost the same as a logistic regression, and the project show that logistic regression already gave very good results.

Another simple improvement to the script would be to replace the logistic regression by a linear regression (same for the output of a deep neural network) in order to have percentages instead of a binary output: this could allow a more granular rating instead of only 0 and 1. The issue is it will also need either a dataset labelled the same way, or the use of unsupervised/semi supervised learning.

RNN neural networks should give even better results since they take advantage of time series, thus not confusing the two sentences "I'm happy, but not sad" and "I'm not happy, but sad". Since RNN needs tokenization, An interesting improvement could be to reduce the drawback of cropping very long reviews (which usually also carry more important information) using compressed reviews (for example using a dictionary generated by the LZW algorithm[36] instead of a simple dictionary), thus preventing the crop of big reviews.

8. 2. Conclusion

The aim of this project has been achieved: a working pipeline of sentiment analysis was implemented using a combination of pre-processing techniques and a learning algorithm.

Even if there is still exciting future work that can be done to improve those results, every mandatory requirement has been completed: the main script tested every combination possible of pre-processing techniques and machine learning model as stated in the requirement, and all module are still usable independently (modularity), as expressed in the requirements no. 11.

The results of this script showed that logistic regression is outperforming the other two algorithms, and that some of the cleaning techniques are impacting the model in a bad way (Lemmatization) while the others did not have an impact on the accuracy.

At last, the best combination (TF-IDF, lower case sentences and removal of stop words with logistic regression) gave a very good result, even without a single tweaking of hyper parameters, and could be used in a real case environment.

Bibliography

[1]: http://spiegel.medill.northwestern.edu/online-reviews/ Spiegel Research Center, 2017

[2]: http://mytestimonialengine.com/how-reviews-affect-purchasing-decisions-and-seo/ Testimonial engine

[3]: https://fanandfuel.com/no-online-customer-reviews-means-big-problems-2017/ Fan and fuel, 2016

[4]: The Nielsen Company. 2015. Recommendations from friends remain most credible form of advertising among consumers; branded websites are the second-highest-rated form. https://www.nielsen.com/us/en/press-room/2015/recommendations-from-friends-remain-most-credible-form-of-advertising.html

[5]; https://www.reviewtrackers.com/online-review-bias/ Are reviews fundamentally biased? Migs Bassig, ReviewTrackers

[6]: https://academic.oup.com/jcr/article/45/3/471/4925297 Seeing Stars: How the Binary Bias Distorts the Interpretation of Customer Ratings, Matthew Fisher, George E Newman, Ravi Dhar, Journal of Consumer Research

[7] Zhang, Yin & Jin, Rong & Zhou, Zhi-Hua. (2010). Understanding bag-of-words model: A statistical framework. International Journal of Machine Learning and Cybernetics.

[8]: Introduction to modern information retrieval, Salton, Gerard and McGill, Michael J., 1986

[9]: Distributed Representations of Words and Phrases and their Compositionality, Tomas Mikolov et. al., 2013

[10]: Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, 2013.

[11]: https://www.tensorflow.org/tutorials/representation/word2vec, word2vec representation

[12] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.

[13] Introduction to information retrieval, Christopher D. Manning et. al., 2008

[14] Witten et al., Naïve Bayes, 2011

[15] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and Regression Trees. Wadsworth, Belmont, CA, 1984.

[16]: https://www.lucidarme.me/simplest-perceptron-update-rules-demonstration/, schema of a Perceptron

[17]: https://uc-r.github.io/feedforward_DNN, schema of a multilayer perceptron.

[18]: Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng, Unsupervised learning of hierarchical representations with convolutional deep belief networks, Communications of the ACM, 2011

[19]: Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

[20]: Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2012

[21]: Long short-term memory, Hochreiter, S., and Schmidhuber, 1997

[22]: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, kyunghyun cho et al. 2014

[23]: Yu, Z., Moirangthem, D.S. and Lee, M., 2017. Continuous timescale long-short term memory neural network for human intent understanding.

[24]: Large movie Review Dataset, Learning Word Vectors for Sentiment Analysis, Maas et. al., 2011

[25]: https://medium.com/@srjoglekar246/overfitting-and-human-behavior-5186df1e7d19, Overfitting

[26]: Pedregosa, Fabian. (2015). Feature extraction and supervised learning on fMRI: from practice to theory.

[27]: Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[28]: Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

[29]:

[30]: Edward Loper, Steven Bird, NLTK the natural language toolkit, ETMTNLP '02 Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1, 2002

[31]: Radim Rehurek and Petr Sojka, Software Framework for Topic Modelling with Large Corpora, Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010

[32]: Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[33]: François Chollet and other, Keras, 2015

[34] : Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

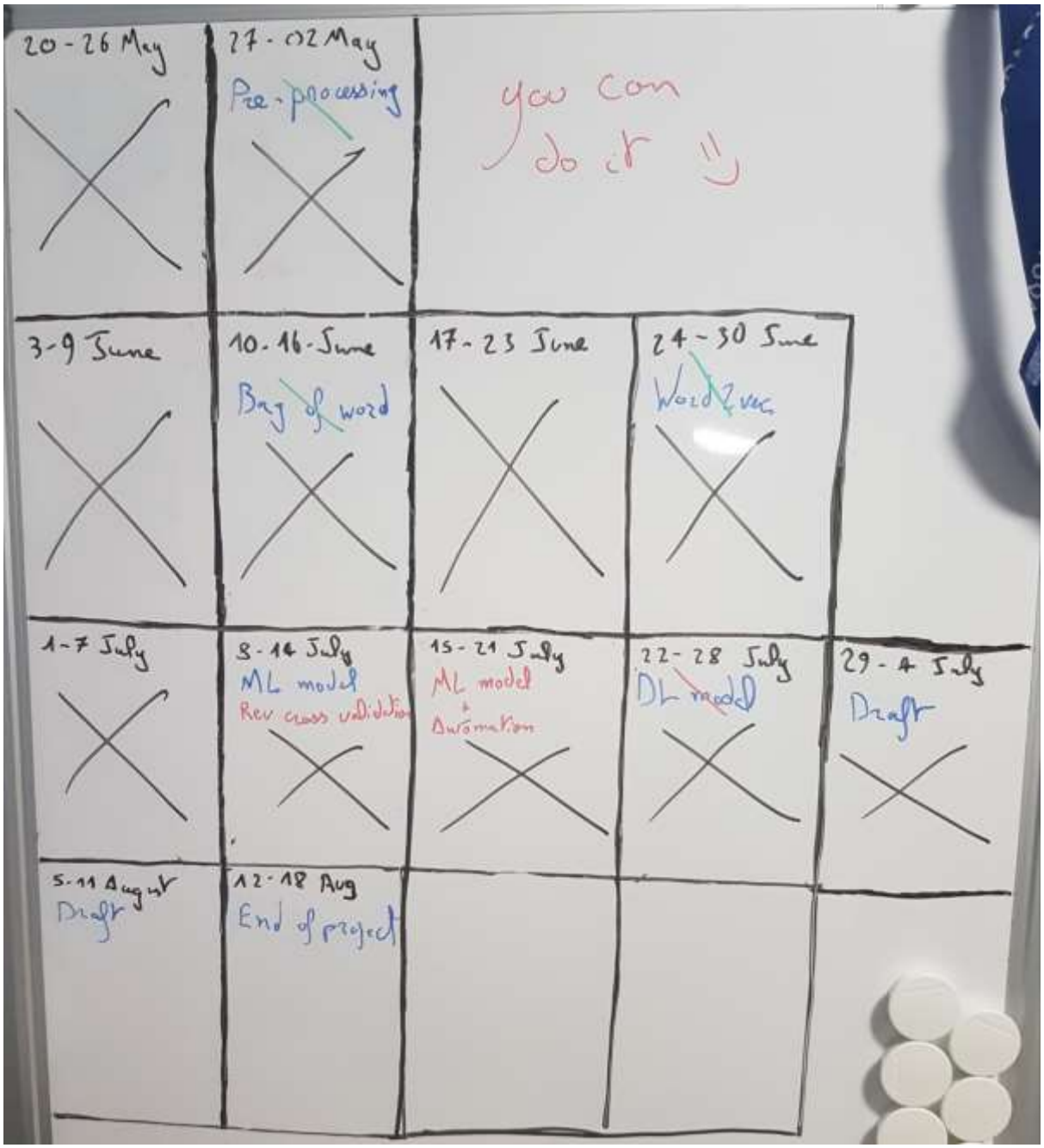
[35]: Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, Dawn Song, The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks, 2018

[36]: Ziv, Jacob, Lempel, Abraham (1977). A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory.

Appendix

Appendix 1: Project plan

Each case represents a week, each line represents a month



Appendix 2: Results sorted by accuracy

Logistic regression			Decision tree			Naive Bayes		
TFIDF	T T F	0.8928	W2V	F F T	0.741	W2V	F T F	0.7786
TFIDF	F T F	0.8925	W2V	I T T	0.7341	W2V	T F T	0.7777
TFIDF	I T F	0.8922	W2V	F T F	0.733	W2V	I T F	0.7732
TFIDF	F F F	0.8893	W2V	I F T	0.7322	W2V	T T F	0.773
TFIDF	T F F	0.8892	TF	T T F	0.7272	W2V	T T T	0.7722
TFIDF	I F F	0.8886	W2V	T T F	0.7253	W2V	F T T	0.7708
TFIDF	F F T	0.8878	W2V	T T T	0.7233	W2V	I T T	0.7695
TFIDF	I F T	0.8878	W2V	T F T	0.7233	W2V	F F T	0.7694
TFIDF	I T T	0.8874	TF	I T F	0.7232	W2V	I F T	0.7675
TFIDF	T F T	0.8872	TF	I F T	0.723	W2V	I F F	0.688
TFIDF	F T T	0.887	TF	F T F	0.721	W2V	F F F	0.6836
TFIDF	T T T	0.887	W2V	F T T	0.721	W2V	T F F	0.683
TF	T F F	0.8843	TF	F F T	0.7207	TFIDF	T F F	0.6757
TF	I F F	0.8824	W2V	I T F	0.7206	TF	T F F	0.674
TF	F F F	0.8819	TF	T F T	0.7203	TFIDF	F F F	0.6739
TF	I T F	0.8818	TF	T T T	0.7202	TFIDF	I F F	0.6734
TF	F T F	0.8797	TFIDF	I T F	0.7196	TFIDF	I T F	0.6732
TF	T T F	0.8792	TF	F T T	0.7196	TFIDF	T T F	0.6729
TF	I T T	0.875	TFIDF	T T F	0.7185	TFIDF	F T T	0.6727
TF	I F T	0.8748	TF	I T T	0.7184	TFIDF	F F T	0.6726
TF	F F T	0.8741	TFIDF	I T T	0.7183	TF	I F F	0.6725
TF	T F T	0.8741	TFIDF	F T F	0.7163	TFIDF	F T F	0.6724
TF	F T T	0.874	TFIDF	I F T	0.7158	TFIDF	T T T	0.6722
TF	T T T	0.8726	TFIDF	T F T	0.7146	TF	F F F	0.6721
W2V	I T F	0.87	TFIDF	T T T	0.7138	TFIDF	I F T	0.6718
W2V	F T F	0.8673	TF	T F F	0.7138	TFIDF	T F T	0.6716
W2V	T T F	0.8668	TFIDF	F F T	0.7123	TFIDF	I T T	0.6713
W2V	T T T	0.8654	TF	I F F	0.7111	TF	T T F	0.6655
W2V	F F T	0.865	TFIDF	F T T	0.7107	TF	I T F	0.6651
W2V	T F T	0.8649	TF	F F F	0.7104	TF	F T F	0.6651
W2V	F T T	0.8647	W2V	F F F	0.7041	TF	T T T	0.6647
W2V	I F T	0.8643	TFIDF	F F F	0.704	TF	F F T	0.6641
W2V	I T T	0.8628	TFIDF	T F F	0.7025	TF	T F T	0.6637
W2V	F F F	0.8583	TFIDF	I F F	0.6967	TF	I F T	0.6634
W2V	I F F	0.8582	W2V	I F F	0.6939	TF	F T T	0.6629
W2V	T F F	0.8563	W2V	T F F	0.6868	TF	I T T	0.6619

Appendix 3: Results sorted by word model

tfidf			tf			w2w		
logreg	T T F	0.8928	logreg	T F F	0.8843	logreg	I T F	0.87
logreg	F T F	0.8925	logreg	I F F	0.8824	logreg	F T F	0.8673
logreg	I T F	0.8922	logreg	F F F	0.8819	logreg	T T F	0.8668
logreg	F F F	0.8893	logreg	I T F	0.8818	logreg	T T T	0.8654
logreg	T F F	0.8892	logreg	F T F	0.8797	logreg	F F T	0.865
logreg	I F F	0.8886	logreg	T T F	0.8792	logreg	T F T	0.8649
logreg	F F T	0.8878	logreg	I T T	0.875	logreg	F T T	0.8647
logreg	I F T	0.8878	logreg	I F T	0.8748	logreg	I F T	0.8643
logreg	I T T	0.8874	logreg	F F T	0.8741	logreg	I T T	0.8628
logreg	T F T	0.8872	logreg	T F T	0.8741	logreg	F F F	0.8583
logreg	F T T	0.887	logreg	F T T	0.874	logreg	I F F	0.8582
logreg	T T T	0.887	logreg	T T T	0.8726	logreg	T F F	0.8563
dectree	I T F	0.7196	dectree	T T F	0.7272	nb	F T F	0.7786
dectree	T T F	0.7185	dectree	I T F	0.7232	nb	T F T	0.7777
dectree	I T T	0.7183	dectree	I F T	0.723	nb	I T F	0.7732
dectree	F T F	0.7163	dectree	F T F	0.721	nb	T T F	0.773
dectree	I F T	0.7158	dectree	F F T	0.7207	nb	T T T	0.7722
dectree	T F T	0.7146	dectree	T F T	0.7203	nb	F T T	0.7708
dectree	T T T	0.7138	dectree	T T T	0.7202	nb	I T T	0.7695
dectree	F F T	0.7123	dectree	F T T	0.7196	nb	F F T	0.7694
dectree	F T T	0.7107	dectree	I T T	0.7184	nb	I F T	0.7675
dectree	F F F	0.704	dectree	T F F	0.7138	dec tree	F F T	0.741
dectree	T F F	0.7025	dectree	I F F	0.7111	dec tree	I T T	0.7341
dectree	I F F	0.6967	dectree	F F F	0.7104	dec tree	F T F	0.733
nb	T F F	0.6757	nb	T F F	0.674	dec tree	I F T	0.7322
nb	F F F	0.6739	nb	I F F	0.6725	dec tree	T T F	0.7253
nb	I F F	0.6734	nb	F F F	0.6721	dec tree	T T T	0.7233
nb	I T F	0.6732	nb	T T F	0.6655	dec tree	T F T	0.7233
nb	T T F	0.6729	nb	I T F	0.6651	dec tree	F T T	0.721
nb	F T T	0.6727	nb	F T F	0.6651	dec tree	I T F	0.7206
nb	F F T	0.6726	nb	T T T	0.6647	dec tree	F F F	0.7041
nb	F T F	0.6724	nb	F F T	0.6641	dec tree	I F F	0.6939
nb	T T T	0.6722	nb	T F T	0.6637	nb	I F F	0.688
nb	I F T	0.6718	nb	I F T	0.6634	dec tree	T F F	0.6868
nb	T F T	0.6716	nb	F T T	0.6629	nb	F F F	0.6836
nb	I T T	0.6713	nb	I T T	0.6619	nb	T F F	0.683

Appendix 4: Code for cleaning & pre-processing of datasets

```
#python integrated
import pathlib

# project files
import utils.fileHandler as fileHandler
import utils.dataUtils as dataUtils

# libs + update
import nltk
nltk.download('wordnet')
nltk.download('stopwords')

##########
# Static #
##########
FOLDER = "../dataset/cleaned_sets"
pathlib.Path(FOLDER).mkdir(parents=True, exist_ok=True)

TRAIN_SET = fileHandler.import_dataset("../dataset", "train")
TEST_SET = fileHandler.import_dataset("../dataset", "test")


######################
# preprocessing types #
######################
clean_patterns = []
# F F
clean_patterns.append({"lowercasing":"T", "stopword":False, "stemming":False})
clean_patterns.append({"lowercasing":"intelligent", "stopword":False, "stemming":False})
clean_patterns.append({"lowercasing":"F", "stopword":False, "stemming":False})

# T F
clean_patterns.append({"lowercasing":"T", "stopword":True, "stemming":False})
clean_patterns.append({"lowercasing":"intelligent", "stopword":True, "stemming":False})
clean_patterns.append({"lowercasing":"F", "stopword":True, "stemming":False})

# F T
clean_patterns.append({"lowercasing":"T", "stopword":False, "stemming":True})
clean_patterns.append({"lowercasing":"intelligent", "stopword":False, "stemming":True})
clean_patterns.append({"lowercasing":"F", "stopword":False, "stemming":True})

# T T
clean_patterns.append({"lowercasing":"T", "stopword":True, "stemming":True})
clean_patterns.append({"lowercasing":"intelligent", "stopword":True, "stemming":True})
clean_patterns.append({"lowercasing":"F", "stopword":True, "stemming":True})
```

```
54
55  ##########
56  # Script #
57  ##########
58  for pos in range(len(clean_patterns)):
59      name = clean_patterns[pos]["lowercasing"] + "_" +
60          str(clean_patterns[pos]["stopword"]) + "_" +
61          str(clean_patterns[pos]["stemming"])
62      print(name)
63
64      # cleaning
65      train_cleanset = dataUtils.clean_array(TRAIN_SET, clean_patterns[pos], debug=True)
66      test_cleanset = dataUtils.clean_array(TEST_SET, clean_patterns[pos], debug=True)
67
68      # saving
69      location = FOLDER + "/" + name
70      print("saving to: " + location)
71      pathlib.Path(location).mkdir(parents=True, exist_ok=True)
72      """ too huge files
73      fileHandler.save_np_array(location + "/train", train_cleanset)
74      fileHandler.save_np_array(location + "/test", test_cleanset)"""
75      fileHandler.save_toCSV(location + "/train.csv", train_cleanset)
76      fileHandler.save_toCSV(location + "/test.csv", test_cleanset)
```

Appendix 5: Script to automate all tested cases

```
# python integrated
import ipdb
import os

# libraries
import numpy

# project files
import utils.fileHandler as fileHandler
import utils.dataUtils as dataUtils
import models.WordModels.bagOfWord as bagOfWord
import utils.textPreprocessing as tp

import models.WordModels.wordEmbedding as wordEmbedding

# model types:
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
# those strings are used to create the file name
LEARNING_MODEL = {
    "logreg":LogisticRegression(solver='lbfgs'),
    "dectree":DecisionTreeClassifier(),
    "gaussNB":GaussianNB()}
#cross validation
FOLDS = 10


# preprocessing
# as to update: just add a string to the correct word model
BAG_OF_WORD = ["TF", "TFIDF"]
WORD_EMBEDDING = ["w2v"]
# those strings are used to create the file name
PREPROCESS_MODEL = BAG_OF_WORD + WORD_EMBEDDING


########################
# Loading & locations #
########################
SOURCE_FOLDER = "../dataset/cleaned_sets"

##########
# Script #
##########
"""
for more visibility, the script is split in three sections:

- the first part (not in functions) will load sequentially each datasets present in the source folder,
    and call the function to test each word models

- the second part (def test_each_wordmodel) will create a word model as specified in the PREPROCESS_MODEL constant.
    When the word model has been created and trained, this function will call the third part of the program

- the third part of the program will test every ML and DL model as specified in the LEARNING_MODEL constant,
    and save the result in the source folder

roughly:
for each dataset -> load
    for each preprocess -> create word model
        for each ML model -> train, evaluate and save
"""

def save_result(save_path, result_string, wordmodel_type, learningmodel_type):
    with open(save_path + "/" + wordmodel_type + "_" + learningmodel_type + ".txt", "w+") as file:
        file.write(result_string)
```

```python
def test_each_learning_model(wordmodel_type, wordmodel, labels, processed_reviews, save_path):
    global LEARNING_MODEL
    # no idea why it forces me to do that the issue comes from the line:
    # model = LEARNING_MODEL[learningmodel_type]

    """Not used anymore, replaced by cross validation
    train_labels, validation_labels = dataUtils.three_split(labels, 0.75)
    train_array, validation_array = dataUtils.three_split(processed_reviews, 0.75)
    """

    print("\t\tCreating cross validation subsets for labels")
    CV_labels = dataUtils.cv_split(labels, folds=FOLDS)
    print("\t\tLABELS done...")

    print("\t\tCreating cross validation subsets for reviews")
    CV_reviews = dataUtils.cv_split(processed_reviews, folds=FOLDS)
    print("\t\tdone...")

    for learningmodel_type in LEARNING_MODEL:
        crossvalidation_models = []
        crossvalidation_results = numpy.array([])

        for set_number in range(FOLDS):
            print("\t\tCross validation passage " + str(set_number) + " of " + str(FOLDS))

            ####
            # Creating datasets for the fold
            ####
            print("\t\t\tCreating fold")
            train_labels, validation_labels = dataUtils.create_sets_forCV(CV_labels, set_number, FOLDS)
            train_feature, validation_feature = dataUtils.create_sets_forCV(CV_reviews, set_number, FOLDS)
            print("\t\t\tdone...")

            ###
            # training
            ###

            # initialisation
            model = LEARNING_MODEL[learningmodel_type]
            print("\t\t\tTraining learning model")
            model.fit(train_feature, train_labels)
            print("\t\t\tdone...")

            print("\t\t\tEvaluating model")
            predictions = model.predict(validation_feature)
            # calculate score : number of correct predictions / total of validation reviews
            score = 0
            for position in range(validation_labels.shape[0]):
                if(validation_labels[position] == predictions[position]):
                    score += 1
            result = score / validation_labels.shape[0]

            # saving
            crossvalidation_models.append(model)
            crossvalidation_results = numpy.append(crossvalidation_results, result)


        # end of "for set_number in range(folds):"

        result_string = "Word model: " + wordmodel_type + "\nLearning model: " + learningmodel_type
        result_string += "\nScores: " + str(crossvalidation_results)
        result_string += "\nMean: " + str(numpy.mean(crossvalidation_results))
        result_string += "  Standard deviation: " + str(numpy.std(crossvalidation_results))

        # only strings are saved ATM TODO: save wordmodel, learning model
        save_result(save_path, result_string, wordmodel_type, learningmodel_type)
```

```
132
133
134 def test_each_wordmodel(review_array, save_path):
135     # for each word model
136     for wordmodel_type in PREPROCESS_MODEL:
137
138         if(wordmodel_type in BAG_OF_WORD):
139             wordmodel = bagOfWord.bagOfWord(wordmodel_type)
140
141         # if we are dealing with word embedding, the dataset need to change
142         if(wordmodel_type in WORD_EMBEDDING):
143             wordmodel = wordEmbedding.wordEmbedding(wordmodel_type)
144             review_array = dataUtils.split_text(review_array)
145
146         print("\ttraining " + wordmodel_type)
147         wordmodel.train(review_array[:, 0])
148         print("\tdone...")
149
150         print("\tProcessing reviews using the wordmodel")
151         processed_reviews = wordmodel.process(review_array[:, 0])
152         print("\tdone...")
153
154         # not dealing with images -> numpy array type is 'object', which is not usable by scikit and keras
155         labels = review_array[:, 1]
156         labels = labels.astype(numpy.int)
157
158         # calling part 3 of the program
159         test_each_learning_model(wordmodel_type, wordmodel, labels, processed_reviews, save_path)
160
161
162 for root, _, files in os.walk(SOURCE_FOLDER):
163     if(files):
164         # loading the dataset
165         print("\nloading and shuffling " + str(root))
166         review_array = fileHandler.load_fromCSV(root + "/train.csv")
167         review_array = dataUtils.shuffle_set(review_array)
168         print("done...")
169         # call part 2 of the program
170         test_each_wordmodel(review_array, root)
```