Moderation of online content through Natural Language Processing, Machine Learning and Deep Learning

By Théo Simier

Under the direction of Prof. Dr. Christophe Croux

Master of Science Data Analytics and Artificial Intelligence

June 2019

Table of Contents

Table of figures

1. Abstract

The objective of this master project is to illustrate how techniques of Natural Language Processing and Machine Learning can be used to gain meaningful information from text. During this thesis, I explain the most commonly used techniques and apply them on a concrete example: the creation of an algorithm capable of monitoring questions by checking if a question respects or not the terms and conditions of a question-and-answer website named Quora. I start by creating statistical features, normalizing the questions and transforming them into a format that classification algorithms can handle. Finally, I use a Logistic Regression, a Random Forest and a Deep Learning model to predict if the questions are compliant or not. The best algorithm reaches an accuracy of 87%.

2. Short introduction to Natural Language Processing

2.1. What is Natural Language Processing?

2.1.1. Quick overview of Natural Language Processing

2.1.1.1. Definition

Natural Language Processing (NLP) is a subfield of computer science, artificial intelligence and linguistics. The end goal of this subfield is to allow a computer to process and understand human languages in order to perform useful tasks.

2.1.1.2. History

We can trace back the first ideas about mechanizing translation processes back to the 17th century, but it is in the 20th that it became realistic (Hutchins, 2005). After the creation of the first electronic calculators, researchers started to use those computers as aids for translating natural languages. The early 1990s was a major turning point. Candide, a system built by IBM, was created and was based not on hard-coded rules but purely on statistical methods. With the development of new algorithms such as deep learning and the exponential increases in Data, NLP made huge improvement in the 2010s.

2.1.1.3. Applications of NLP

Applications of NLP are multiple and various. Sentimental Analysis is one of them. It consists in identifying, extracting and studying the affective states of some data. For example, by classifying movie reviews into positive or negative reviews in the famous IMDB data set. Speech Recognition deals with the understanding of human speech. Cortana or SIRI are good examples of applications. Chatbots are Artificial Intelligence that conduct a conversation. Finally, using NLP techniques to transform the data from one language to another for applications, such as Google Translate or Reverso, is called Machine Translation.

2.1.2. Natural Language Understanding

2.1.2.1. Definition

NLP has two major components: Natural Language Understanding (or NLU) and Natural Language Generative (or NLG). NLU is the process of mapping language inputs to useful representations. NLU is considered harder than NLG because NLU cannot control the complexity of the language structured that it received in inputs while NLG can.

2.1.2.2.Difficulties of NLU

The high complexity of NLU can be explained by the fact that human language is highly complex. Lexical Ambiguity is the first reason of this complexity (Eduraka, 2018). Indeed, a single word can have two or more possible meanings. For example, the word *fine* can be used in different situations: as a sanction in *I forgot my train ticket and had to pay a fin*e. or as a sentiment in *I went to the park to enjoy the fine weather*.

Without context, it can sometimes be difficult or even impossible to be sure of the meaning. For example: *The fisherman went to the bank.* Does *bank* refer to the financial institution or does it refer to the bank of the river ?

Syntactic ambiguity, also called structural ambiguity, is the second reason why a language can be highly complex. Due to ambiguous sentence structure, a sentence can be interpreted in more than one way. For example: *The chicken is ready to eat.*

Lastly, referential ambiguity, arises when a subject of a sentence is not clear.
For example: *The child told his father that he had won the football game. He was very happy*. Does *He* refer to the child or to the father?

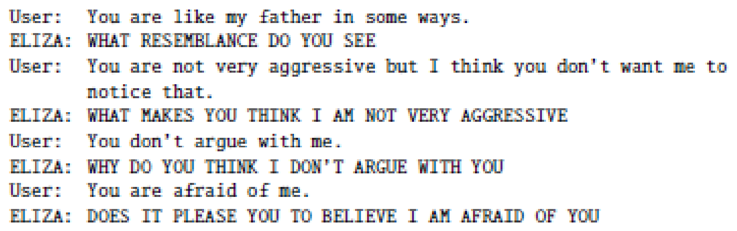
2.1.3. Natural Language Generation

2.1.3.1.Definition

Natural Language Generation is the second major component of NLP. It can be defined as "*the process of constructing natural language outputs from non-linguistic inputs."* (Semaan, 2012). Thus, it is the inverse of NLU.

2.1.3.2.Examples of NLG

Many examples of NLG exist. Here are the most famous examples (Demichelis, 2019).
In 1964, Joseph Weizenbaum created a chatbot called ELIZA that could simulate the answers of a psychotherapist.



```
User:   You are like my father in some ways.
ELIZA:  WHAT RESEMBLANCE DO YOU SEE
User:   You are not very aggressive but I think you don't want me to
        notice that.
ELIZA:  WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
User:   You don't argue with me.
ELIZA:  WHY DO YOU THINK I DON'T ARGUE WITH YOU
User:   You are afraid of me.
ELIZA:  DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
```

Figure 1: Extract of the discussion between ELIZA and a human (Joseph, 1966)

In 1983, the software ANA started to report stock market in English. Lastly, in 2011, Watson, an AI developed by IBM, won the famous game Jeopardy after facing human champions. In this game show, the contestants are presented with clues in the form of answers and must phrase their responses in the form of questions.

Figure 2: IBM-Watson and its human opponents in Jeopardy!

2.2. Why Natural Language Processing is important?

2.2.1. Overview of the different kinds of data

There are 3 types of data: structured, semi-structured and unstructured. During a long time, only structured data was used. Structured data can be organized in a table and are easy to analyze. However, this kind of data only represents 20% of the total data that it collected (Dan, 2017). Semi-structured data is more flexible than structured data and contains semantic tags. Cookies are a huge source of semi-structured data. Finally, unstructured data is not organized in pre-defined manner. They cannot be stored in a traditional relational database. Texts, pictures, videos and sounds are examples of unstructured data.

2.2.2. Importance of NLP

If we limit analysis to structured or semi-structured data that can directly be analyzed by traditional techniques, we lose an immense part of the data being collected. We must find ways to analyze unstructured data. Knowing that most unstructured data exist in the textual form or can be transformed into textual form, for example an audio recording can be transcribed in text, the use Natural Language Processing is crucial. Indeed, NLP allows to prepare the data by transforming it in a format that can be used by traditional techniques. Then, Text Mining and Machine Learning can create high quality information from the text.

3. Data

3.1. Origin of the data set

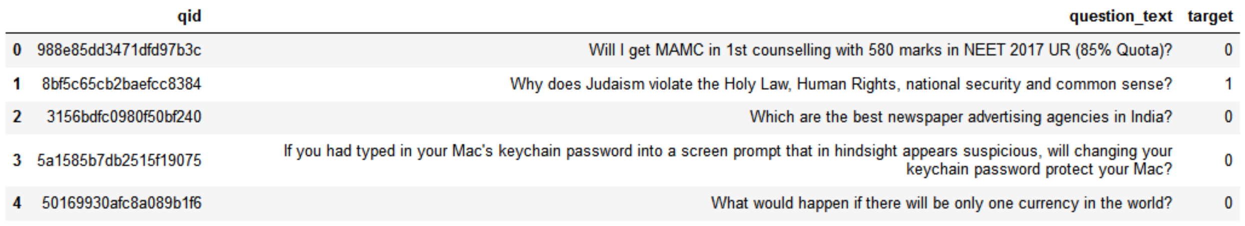
The data set that I used for my master thesis is originated from *quora.com*. Quora is a question-and-answer website that was created in June 2009 (Wikipedia, 2019). The content of Quora is directly created by its users that can ask, answer, and edit questions. Quora states that its mission is to share and grow the world's knowledge by connecting the people who have the knowledge to those who need it (Quora, Inc., 2019). Opinions are welcomed on the Quora platform and debate is fostered.
However, Quora monitors the contents that are considered as toxic or divisive. One of the key challenges for this company was to create a scalable method to identify and flag insincere questions (Quora, Inc., 2019). In order to achieve this goal, they asked help to the community of Kagglers, on the famous website Kaggle.com, through an online competition. Knowing that the data that was shared by Quora for this competition can be used for academic research and education, I used it to illustrate my subject.

3.2. Problem to answer

To illustrate my finding about NLP, I used the same goal as the Kaggle's competition, meaning finding a scalable way to identify insincere questions. Quora defined insincere questions as questions that have one of the following characteristics: "*Has a non-neutral tone"* or *"Is disparaging or inflammatory"* or *"*Is *not grounded in reality"* or *"Uses sexual content (incest, bestiality, pedophilia) for shock value, and not to seek genuine answers".* (Quora, Inc., 2019)
The end goal of this master thesis is to create an algorithm than can correctly classify a question into one of these two classes (sincere or insincere).

3.3. Description of the data set

The data set is composed of 2 main files. A training set and a test set of respectively: 1310000 and 376000 rows and 3 columns each. Each row corresponds to an observation, meaning a question asked by a Quora user. The 3 variables are the qid (a unique question identifier), the question, and the label of the question. A question flagged as "insincere" has a value of 1, or 0 otherwise.



	qid	question_text	target
0	988e85dd3471dfd97b3c	Will I get MAMC in 1st counselling with 580 marks in NEET 2017 UR (85% Quota)?	0
1	8bf5c65cb2baefcc8384	Why does Judaism violate the Holy Law, Human Rights, national security and common sense?	1
2	3156bdfc0980f50bf240	Which are the best newspaper advertising agencies in India?	0
3	5a1585b7db2515f19075	If you had typed in your Mac's keychain password into a screen prompt that in hindsight appears suspicious, will changing your keychain password protect your Mac?	0
4	50169930afc8a089b1f6	What would happen if there will be only one currency in the world?	0

Figure 3: Extract of the data set

As we can see below, the number of questions flagged as insincere is much lower than the number of questions defined as sincere (6% versus 94%).

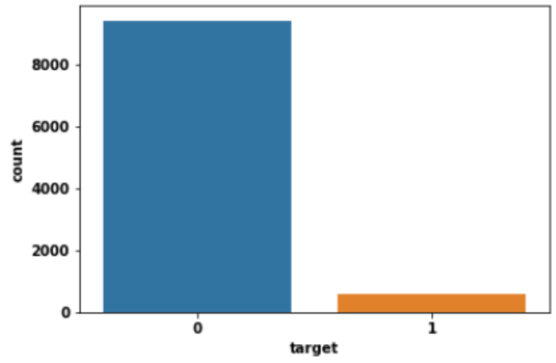


Figure 4: Distribution of the target

In order to train our model in a balance way, we balanced the two classes by selecting 50 000 observations of each class.

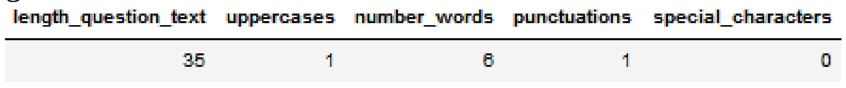
4. Feature Engineering through NLP techniques

In this part, we will see some techniques that I have used to transform the text data into data that can be analyzed by an algorithm. This transformation can also be called Feature Engineering. Indeed, Feature Engineering is the process of creating features from raw data in order to better represent the underlying problem so that it improves the accuracy of the predictive models (Brownlee, Discover Feature Engineering, How to Engineer Features and How to Get Good at It, 2014).
The following techniques will have two main goals: creating features that are significant to predict whether or not a question is sincere and reducing computational needs.

4.1. Statistical Features

The first features that can be created are descriptive features of our text. While many features could be created, I have selected five of them that are the most relevant to describe the question asked. They are the following: the length of the question, the number of words in the question, the proportion of capital letters in the question, the proportion of punctuation marks, the proportion of special characters such as + or $ (see Figure 18: Statistical features).

With the question: "What makes you start your business?", we would have the following value:



length_question_text	uppercases	number_words	punctuations	special_characters
35	1	6	1	0

Figure 5: Example for one row of the statistical features that were created

4.2. Cleaning

Before to use techniques that are specific to NLP, we need to clean the data from all information that is not useful.

4.2.1. Preliminary Cleaning

Thanks to the statistical features that we have previously created, we can now transform the text to remove the lowercase, punctuation, special characters, digits, extra spaces. If those characteristics are significant to predict the sincerity of a question, they will be considered thanks to the statistical features that we have just created (see Figure 19: Data Cleaning).
For example, if we have the sentence:

  I don't buy   cheap cheese! Never less than 10$/kg.   Do you ? We will have 6 steps to clean it, thanks to the module “re”.

First step: Replace special spaces (ex: line break) by a simple space (not visible).
  I don't buy   cheap cheese! Never less than 10$/kg.   Do you ?
Second step: Remove apostrophe.
  I dont buy   cheap cheese! Never less than 10$/kg.   Do you ?
Third step: Replace all characters that are not letters by a space.
  I dont buy   cheap cheese Never less than kg   Do you
Fourth step: Replace double (or more) spaces by a single space.
 I dont buy cheap cheese Never less than kg Do you
Fifth step: Remove starting and ending spaces
I dont buy cheap cheese Never less than kg Do you
Sixth Step: Convert all characters to lowercases.
i dont buy cheap cheese never less than kg do you

After those steps, sentences are standardized. We are now ready to go to the next step of the normalization of the data.

4.2.2. Tokenization

In order to facilitate the processing of our question, for example by counting the frequency of appearance of a word, we need to transform our data from a sentence or phrase to individual words. This process is called Tokenization.

For example, the sentence: “what makes you start your business” will be transformed to a list [‘what’, ‘makes’, ‘you’, ‘start’, ‘your’, ‘business’].

4.2.3. Deletion of the stopwords

Stopwords are words that are not useful for the algorithms because they are not linked to the dependent variable that we want to estimate.
When we look at the frequency of those words in our data set, they often have a very high frequency. For example, the word ‘the’ is present in 26% of the questions and the word ‘what’ is present in 17%.

In order to reduce the number of words and linked features that will be created in the next steps, the stopwords have to be removed.

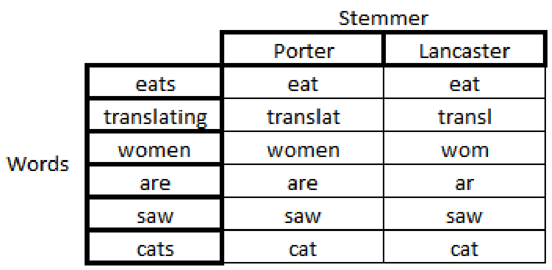
We remove 179 stopwords that are already defined in NLTK. We add 5 of our own stopwords ('like', 'would', 'best', 'get', 'people'). Those additional stopwords were selected based on their frequency in our data set.

The result of the deletion of the stopwords would transform the list: [‘what’, ‘makes’, ‘you’, ‘start’, ‘your’, ‘business’] into [‘makes’, ‘start’, ‘business’].

4.2.4. Stemming

Stemming is the process of normalizing a word into its stem (root form). For example, the words *translates*, *translation*, *translating* have for stem *translate*. If we reduce stemming to its basic process, it chops off the ends of the words in order to recover the stem (Cambridge University Press, 2009).

There are several ways to do stemming. The module NLTK has for example one stemmer (an object that does stemming) called Porter and another one called Lancaster.



		Stemmer	
		Porter	Lancaster
Words	eats	eat	eat
	translating	translat	transl
	women	women	wom
	are	are	ar
	saw	saw	saw
	cats	cat	cat

Figure 6: Comparison of the stemmers: Porter and Lancaster

As we can observe, Lancaster is a more radical stemmer than Porter.
Yet, stemming is not ideal. In this example, *women* and *are* were not transformed with Porter ’stemmer and transformed into *wom* and *are* with Lancaster ’stemmer. It would have been better to recover the words *woman* and *be*. Lemmatization could be the solution.

4.2.5. Lemmatization

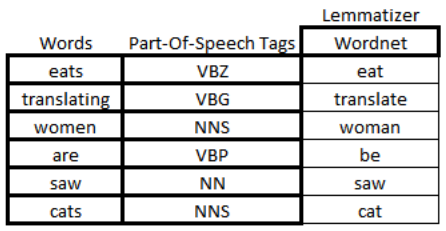
Lemmatization is the process of normalizing a word into its lemma (dictionary form). Lemmatizers (objects that do lemmatization) use the function of the word in a sentence (ex: pronoun, verb) and a list of vocabulary to find its lemma.

4.2.5.1.Part-Of-Speech Tagging

The role of a word in a sentence is also called Part-Of-Speech (POS). 8 Parts-Of-Speech are commonly used: the verb (VB), the noun (NN), the pronoun (PR+DT), the adjective (JJ), the adverb (RB), the preposition (IN), the conjunction (CC), and the interjection (UH) (CLiPS, 2019). Additional parts based on those have been created to go deeper with NLP (see Figure 17: Comprehensive list of Part-Of-Speech). Each POS has a unique tag that can be used in different NLTK functions.

4.2.5.2.Result of the lemmatizer

Several lemmatizers exist. For our project we have used the lemmatizer called Wordnet. This lemmatizer has an important list of vocabulary. Thanks to the library NLTK, we only have to give the word and the POS to the object for it to return the lemma (see Figure 20: Lemmatization).



		Lemmatizer
Words	Part-Of-Speech Tags	Wordnet
eats	VBZ	eat
translating	VBG	translate
women	NNS	woman
are	VBP	be
saw	NN	saw
cats	NNS	cat

Figure 7: Examples of Lemmatization

The use of lemmatization reduces the number of unique words in our data set. It will be useful to fight the curse of dimensionality.

4.3. Bag-of-Ngrams

4.3.1. Simple vectorization

The next transformation that I did was to transform the data from sting to a vector of words and numbers. This process is called “Vectorization”.
The result is a dataframe with in columns each word/number of the data set and in value the result of the count of this word/number in the question. If the word is present N times in the question, then the column of this word will have a value equal to N. This dataframe is a sparse matrix, with 99.9% of the values equal to 0.

With our example we would have the following value: what: 1, make: 1, you: 1, start: 1, your: 1, business: 1, and for any other words or number in our data set: 0.
We have now transformed our questions into a vector of numbers that can be analyzed by machine learning algorithms such as logistic regression. Given the set of words defined by the vector, the algorithm will identify patterns that the question

is sincere or insincere. For example, the word “religion” could be associated with a higher number of insincere question that the word “business”. Then: $P(insincere|religion) > P(insincere|business)$.
One of the limits of a simple vectorization is that it does not take into account the context around the word. In order to avoid lexical ambiguity, context has to be taken into account.

4.3.2. Alternatives: Ngrammes with n > 1

Instead of creating simple vectorization, we could create Ngrammes with N equal to the number of adjacent elements that are grouped to form a feature. To illustrate let’s create a Ngrammes with N=2, also called bigram from the sentence: *Paris is the capital of France*.
After standardization and lemmatization, we obtain the bigram [paris_capital=1, capital_france=1, all the other bigrams =0]. With a simple vectorization we would obtain [paris=1, capital=1, France=1, all the other words =0]. Thanks to N-grams, algorithms are able to identify patterns in the associated words. For this thesis, I have created matrix with N=2 and N=3.

4.4. Preparation of the Data for Neural Networks

4.4.1. Matrix

A Neural Network could have been directly used on the matrix previously created. However, in order to fully benefit from Neural Networks, and more precisely of the artificial recurrent neural network architecture (RNN), the data had to be prepared in a different way. Indeed, RNN allows to fully keep the structure of the words in the question. Instead of using Bag-of-Ngrams to remember the context of the question, we can use the specificity of RNN to process sequences of data.

In order to create a matrix that keep the word sequences of the question, we have to follow three steps. Firstly, we need to create a dictionary that associates each word present in our data set to a unique number. Then, we have to have to create a matrix that keeps the structure of the questions by using as values the numbers created in the previous step. Lastly, to uniformize the size of the questions, we have to pad the matrix, meaning limiting the size of the matrix to a number N and filling the empty spaces by a zero.

4.4.1.1.Quick example with three sequences of words

Let’s use the following 3 examples (not normalized): [this, is, an, example], [this, is, a, second, example], [this, is, a, third, and, last, example].

We would have the following dictionary:
{‘this’: 1, ‘is’: 2, ‘an’: 3, ‘example’: 4, ‘a’: 5, ‘second’: 6, ‘third’: 7, ‘and’: 8, ‘last’: 9}

We would have the following matrix at N=5:

0	1	2	3	4
1	2	5	6	4
5	7	8	9	4

As we can see, we have limited the size of the matrix to N=5 for the last sequences of words. In this sequence, this is the word the more on the left that was ignored. The reason is that we can suppose that the most meaningful words to identify if a question is sincere are at the end of the sequence. The first sequence had a size n=4, a zero was added in first position.

4.4.2. Embedding

4.4.2.1. How we can represent words with embedding

As we have seen above, we can represent a sequence of words with a matrix. However, this simple matrix has limitations. Those indexes do not give any indication about the nature of the word. We do not know if the word with the index 1 is close, from a usage point of view, to the word with the index 2. For example, *woman* and *man* are closer to each other than *woman* and *car*. As we can observe in the example below, via an embedding that is in fact a vector at N dimensions, we are able to represent the similarity and dissimilarity of words.

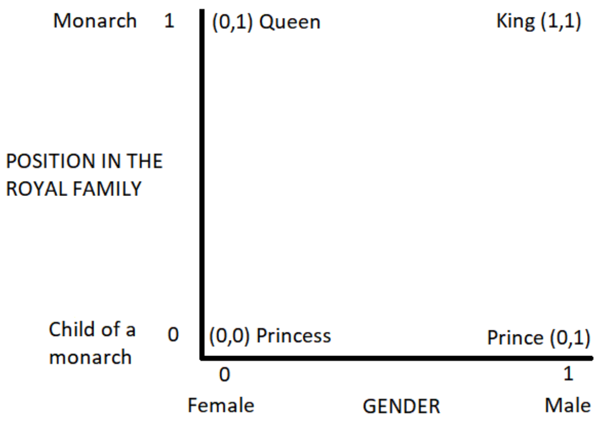


Figure 8: Illustration of an embedding with d=2

In this example, we see that the word *King* is closer to *Queen* or *Prince* that it is to *Princess*.

4.4.2.2.Pre-trained GloVe Embedding

Our embedding was created thanks to an unsupervised algorithm called GloVe (Pennington, Socher, & Manning, 2019). Instead of training our own word representation, we used pre-trained word vectors that represent 400 000 words in 100 dimensions (see Figure 22: Embedding). Those vectors were trained on contents of Wikipedia dated from the year 2014 and on text data from the English Gigaword Fifth Edition (Linguistic Data Consortium, 2019).

5. Methods used to predict the dependent variable

The curse of dimensionality having played a huge role in my choice of algorithms, I will start by explaining what this phenomenon is, then I will present the algorithms that I chose.

5.1. The curse of dimensionality

The curse of dimensionality is a problem that nonparametric methods suffer when the number of dimension *d* increases (Hastie, Tibshirani, & Friedman, 2008). As *d* increases, there are less and less observations in the neighborhoods of a given point, nonparametric methods, such as KNN, become less accurate and thus their convergence rate decrease.

Indeed, let's consider a p-dimensional unit hypercube with inputs uniformly distributed. If we want to capture a fraction *r* of the observations, the expected edge length of the hypercubical neighborhood will be:

$$e_p(r) = r^{(\frac{1}{p})}$$

While the entire range of each input is only 1, if we want to capture 10% of the inputs, we will need to cover:
in one dimension, $e_1(0.1) = 10\%$ of the range of each input variable,
in ten dimensions, $e_{10}(0.1) \approx 80\%$ of the range of each input variable,
in hundred dimensions, $e_{100}(0.1) \approx 98\%$ of the range of each input variable,
in thousand dimensions, $e_{1000}(0.1) \approx 100\%$ of the range of each input variable.

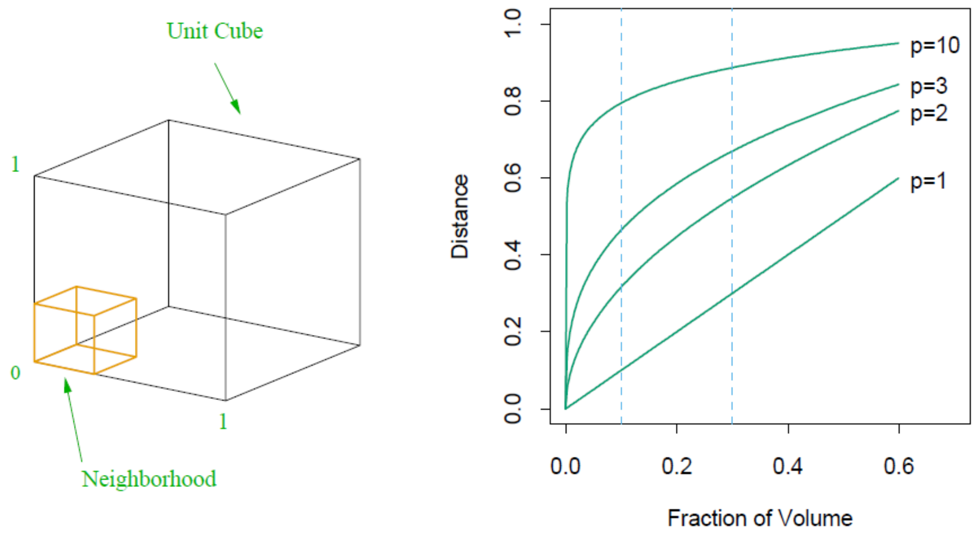


Figure 9: Illustrations of the curse of dimensionality
The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. Retrieved from The Elements of Statistical Learning (Hastie, Tibshirani, & Friedman, 2008)

5.1. Machine Learning Models

Compared to non-parametric methods, parametric models rely on the assumption that there is a linear relationship between *X* and *Y*. They are more restrictive. However, they also have huge advantages. Firstly, their learning rate or parametric rate is fast. It is equal to *1/n*. The second advantage is also visible in the learning rate. Indeed, the parametric rate does not depend on the number of dimensions *d*. As d increases, there is no impact on the learning, thus the curse of dimensionality is not a problem for parametric methods. That is the reason why, in order to predict if a question is sincere or insincere, I used two parametric methods that are very popular for classification: Logistic Regression and Random Forest (see Figure 21: Logistic Regression and Random Forest).

5.1.1. Logistic Regression

In a context of binary classification (where $Y = \{0,1\}$), a Logistic Model is a good choice. This parametric method models the probabilities of the 2 classes via a linear function in x, meaning it modelizes the parameter of the distribution Y|X = x. It has to be noted that logistic regression can also be used with more than 2 classes. In that case, there are several linear functions. The probability of each class is between 0 and 1 and the sum is equal to 1. However, in order to keep the result of the linear function into the range [0,1] it uses the below logit transformation of the probability p(x):

$$logit(x) = \log \frac{p(x)}{1 - p(x)} = \sum_{i=1}^{d} \beta_d x_d$$

The logistic regression model can be fitted by maximum likelihood (Hastie, Tibshirani, & Friedman, 2008). Indeed, in order to estimate the unknown parameters β, we need to maximize the log-likelihood:

$$l(\beta) = \sum_{i=1}^{n} \{y_i x_i^t \beta - \log(1 + \exp(x_i^t \beta))\}$$

5.1.2. Tree based algorithm

Tree based algorithms are very effective algorithms for classification or regression. Those algorithms are based on the use of several trees. I used one of them called Random Forest.

5.1.2.1.Tree

A Tree divides a node of observations into subgroups called children nodes based on a split variable *j* and a split point *s*. The goal is to find rules that can create subgroups where the observations *Y* are identical. *j* and *s* are selected by minimizing a criterion which measures the impurity of the two children nodes. This impurity function of a node *N* is defined by:

$$\tau(n) = \sum_{j=1}^{K} f(p_j(N))$$

$p_j(N)$ stands for the proportion of class *j* in *N*. f is a concave function such that $f(0) = f(1) = 0$.

Several functions exist but for my binary case, I selected the function called Gini:

$$\tau(N) = 2p(1 - p)$$

The impurity of the node will be small if the node is homogeneous and high otherwise. A tree chooses *j* and *s* which minimizes:

$$P(N_1)\tau\big(N_1(j,s)\big) + P(N_2)\tau\big(N_2(j,s)\big)$$

By doing this subdivision several times, it creates subgroups where the observations Y are identical. It has to be noted that as a tree grows larger, meaning as the number of nodes increases, the variance increases but the bias decreases and always becomes null when each node contains only one observation (because $f(1) = 0$).

5.1.2.2. Random Forest

A random forest is a collection of trees. It results from a technique called Bootstrap Aggregating. The idea is instead of fitting a very precise machine, we can fit a lot of simple machines (in the case of random forest a tree) and aggregate them.

However, in order to fit the same machine but to have different results, we need to fit the algorithm on different data set. A technique, called Bootstrap, will randomly draw data sets with replacement from the training data and fit the machine on it. The resulting training data being different, the parameters of the machine will be different.

The bagging process does not impact the bias. However, the variance is reduced when the correlation between the machines decreases. Thus, we want machines that are as less correlated as possible. Simple machines, such as trees, being very sensitive to small disturbances of the data, are ideal to reduce the variance. In order to reduce the bias, we will use trees that are as large as possible. Once a lot of trees have been fitted (most of the time 500), the prediction is the average the result of the trees.

In order to measure the performances of a random forest in a classification setting, we can use the classical misclassification error: $P(Y \neq \hat{T}_B(X))$. We can also use a misclassification error called Out of Bag error. This error is equal to:

$$\frac{1}{n}\sum_{i=1}^{n} 1_{\hat{Y}_i \neq Y_i}$$

In order to calculate this error, we only use the result of the trees that were not trained on this specific observation. It avoids overfitting.

5.2. Focus on Deep Learning

Deep Learning techniques are machine learning methods based on artificial neural networks. The name comes from the fact that in order to learn, the neural networks use several layers to progressively extract higher level features from raw input. To facilitate understanding, I will not go into the details of the mathematics behind those techniques.

5.2.1. Structure of the model

In order to simplify the comprehension of the model, I used a simple model structure (see Figure 23: Structure of the Deep Learning model).

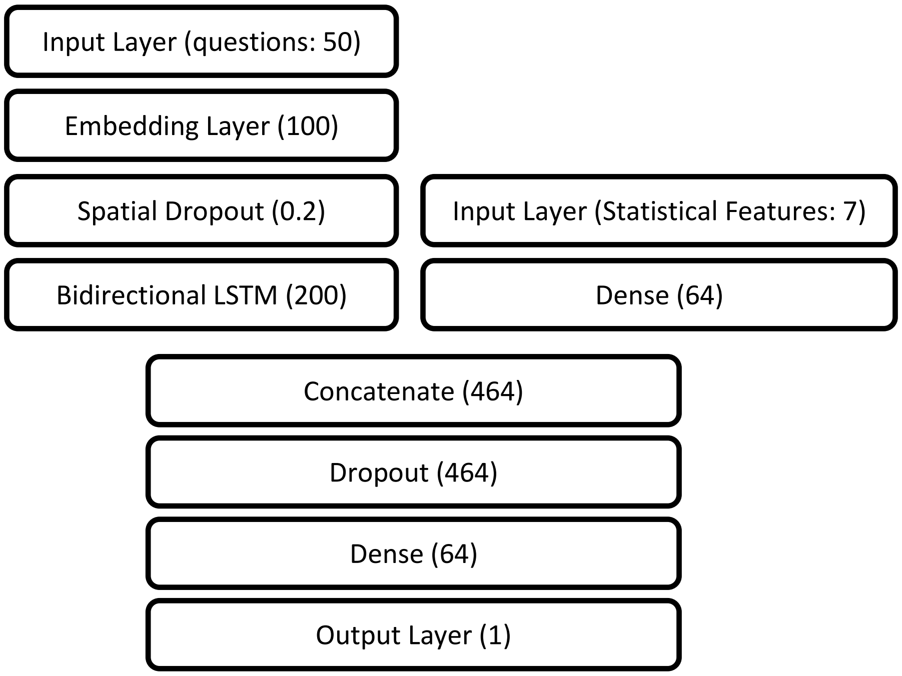


Figure 10: Structure of the Deep Learning model

Input layers are layers that allow to pass data to the Neural Network. For our model, there are two sources of data: the matrix that contains the indexes of the words and the statistical features that we have created. They are combined thanks to the concatenate layer. The Embedding Layer turns the indexes of the words into vectors of 100 dimensions. Spatial Dropout and Dropout layers avoid overfitting by setting a fraction of input units to 0 at each update during training time. Output Layer transforms the input of 64 dimensions into an output of 1 dimension that is the probability than a question is insincere. In total, with this structure, 511 937 parameters were trained (see Figure 24: Summary of the Deep Learning model and Figure 25: Training of the Deep Learning model).

5.2.1.1.Dense Layer

A dense layer is a multitude of perceptron. A perceptron is a weighted sum of inputs to which you add a bias (to avoid a null sum) and then apply a non-linear function in order to be able to represent non-linear data (Lagandula, 2019).

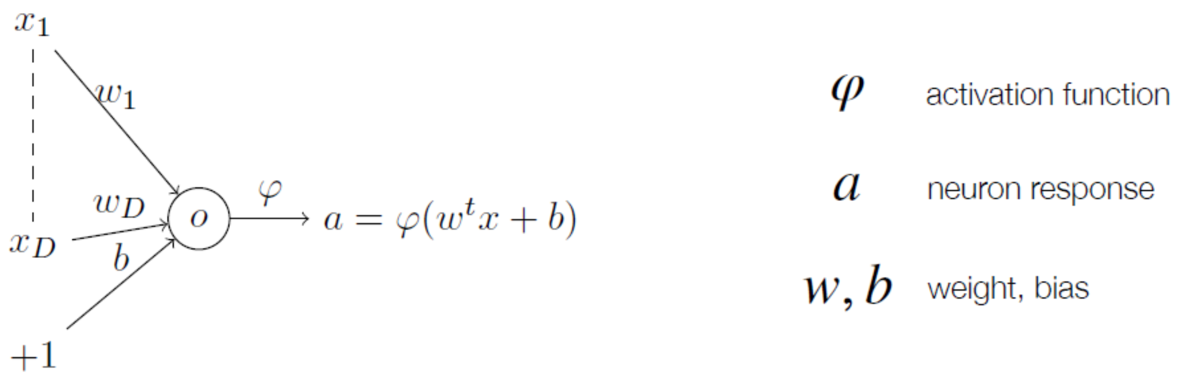


Figure 11: Basic Perceptron

The Neural Networks will train the weights of this layer in order to reduce the loss of the model.

5.2.1.2.LSTM layer

LSTM stands for Long Short Term Memory. It is a special case of Recurrent Neural Networks (RNN). This layer can identify which information is important or not, and by using loops they allow information to persist. They are thus very useful in our case because they can remember the previous word vectors.

5.3. Methods used to evaluate the prediction

5.3.1. Accuracy score

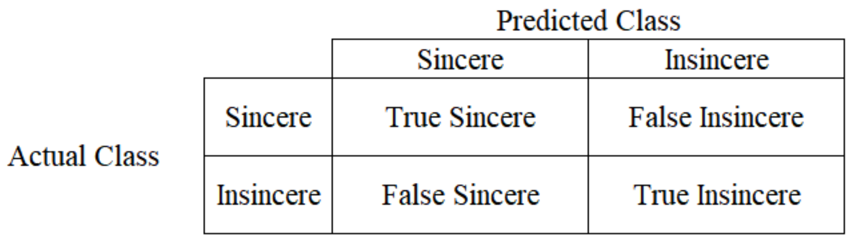
The easiest metric to evaluate the accuracy of a classification model is the Accuracy classification score. It is the percentage of correctly classified observations.

$$Accuracy = \frac{Number\ of\ correct\ classification}{Number\ of\ classification}$$

However, this metric has some limits. Firstly, it considers that each classification has the same importance. It is not always the case. For the Quora problem, maybe the company prefers to create an algorithm that almost never classifies a sincere question as insincere. In this case, if they use the accuracy score, they would not be able to judge if an algorithm performs according to their preferences.

5.3.2. Confusion Matrix

A confusion matrix allows to visualize for each class the predicted classifications as well as their actual classes.



		Predicted Class	
		Sincere	Insincere
Actual Class	Sincere	True Sincere	False Insincere
	Insincere	False Sincere	True Insincere

Figure 12: Confusion Matrix

Famous metrics such as Precision, Recall and F1-Score can be calculated based on this matrix. For this master project, we used a balanced data set. Having no information on the importance of each class, I considered that both classes were as much important. That is why I have decided to only use the accuracy score and the confusion matrix.

6. Results

6.1. Results of the different models

6.1.1. Best classification of the Logistic Regression

By using a logistic regression, we obtained the best accuracy score by training the algorithm on the matrix of the Ngram equal to 1. We obtained a 86.5% accuracy score.

		Predicted Class	
		Sincere	Insincere
Actual Class	Sincere	45%	5%
	Insincere	8%	42%

Figure 13: Confusion matrix of the logistic regression

6.1.2. Best classification of the Random Forest

By using a random forest, we obtained the best accuracy score by training the algorithm on the matrix of the Ngram equal to 1. We obtained a 80.8% accuracy score.

		Predicted Class	
		Sincere	Insincere
Actual Class	Sincere	44%	6%
	Insincere	13%	36%

Figure 14: Confusion matrix of the random forest

As we can observe on the confusion matrix, the random forest is less accurate than the logistic regression to identify the questions that are insincere.

6.1.3. Classification of the Deep Learning Model

By using our Deep Learning model, we obtained the best accuracy of all the algorithms. The accuracy is 87.57%. We have the following confusion matrix:

		Predicted Class	
		Sincere	Insincere
Actual Class	Sincere	43%	8%
	Insincere	4%	45%

Figure 15: Confusion matrix of the Deep Learning Model

Compared to the logistic regression, this model is less accurate to classify the sincere question. However, there is a huge improvement in the classification of the insincere questions.

We can also observe that we did not overfit. Indeed, the test accuracy continues to decrease as the number of epochs increases. For computational reasons, we did only 10 epochs.

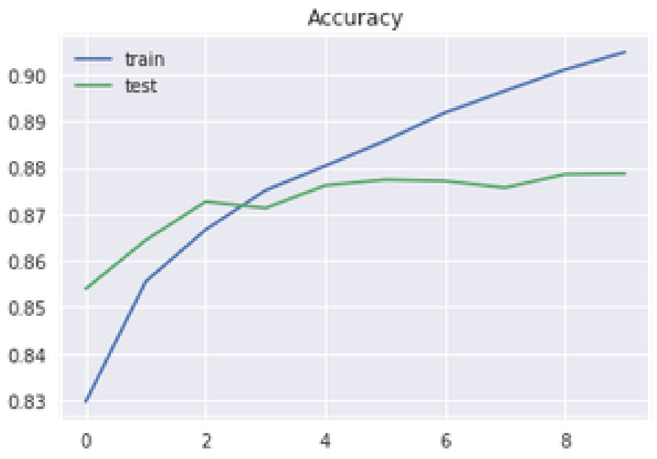


Figure 16: Accuracy of the train and test data set

6.2. Ways to improve the results

Several techniques could have been tried to improve the accuracy of our models. Below is a selection of the most promising techniques based on the results of the Kaggle competition.

6.2.1. Spell Correction

During the Kaggle competition, several Kaggler used spelling correction in order to reduce the number of unique words by identifying words that are similar in intent.

The first step of the spell correction mechanism is to calculate a minimum edit distance and decide according to this distance if two words are similar (Jurafsky & Martin, 2018). This distance is the minimal number of editing operations needed to make one word identical to another word. The editing operations include insertion, deletion and substitution of a letter. Each of these operations can have different weights. As the minimum edit distance increases, the probability that two words are similar decreases.
For example, the minimum edit distance between the words *constitution* and *constition* is 2 (insert a *t* and a *u*). Thanks to this distance we can guess that those two words are quite similar.

The second step is to predict the probability of a word to appear in a sentence. If the probability of this word is very low, we can replace this word by a more probable word. This replacement also has to have a minimum edit distance that is low.
To continue with our previous example, we will use the sentence: *The French government organized a referendum to modify the constition*. In this example, the probability that the word *constition* appears is very low but the probability for the word *constitution* is much higher. *constitution* is also closed to *constition* in terms of minimum edit distance. Thus, *constition* is identified as a misspelled word and can be replaced by *constitution*.

6.2.2. Improve our embedding

6.2.2.1. Select a better suited embedding

As mentioned previously, our embedding was a pre-trained word vector. For computational reason and memory limitations, I selected a simple embedding with 100 dimensions and 400 000 words. However, it exists embedding that were trained on larger data sets. Those embedding have more dimensions and more vocabulary. For example, the larger pre-trained embedding available on the GloVe website has 1.9 million words and 300 dimensions.

6.2.2.2. Create our own embedding for missing words

Another way to improve our embedding would have been to combine pre-trained embedding with an embedding that we would have trained on the vocabulary of the data set. It would have captured the specificity of the vocabulary used on Quora and created additional vectors for words that are not in the pre-trained embedding.

6.2.3. Grid Search to tune the hyperparameters

For this master thesis, the hyperparameters of our models were not tuned. We could have used a grid search to tune some of those hyperparameters such as the number of estimators in our Random Forest or the batch size or dropout rate of our Neural Network.

6.2.4. Combining several models

Last but not least, we could have combined the classification results of different models to generate a new model (Gorman, 2019). Most of the time, the resulting model called stacked model or 2nd-level model will outperform each of the individual models. In our case, we could have run a k-nearest neighbors algorithm on the results of the classification of the Logistic Regression, Random Forest and Neural Network models.

7. General Conclusion

Even if for a human textual content is easily understandable, it is a form of data that cannot be used directly in traditional statistical methods. That is why, during a long time, this source of data was left untapped. With the emergence of new techniques, increased computer capabilities and larger amount of textual data, important innovations were recently created.

During this master thesis, I have explained a set of techniques, called Natural Language Processing, and shown how it can be used to transform text into a format that can be handled by Machine Learning models. As I have proved on the Quora data set, standard models, such as Logistic Regression or Random Forest, can then make great predictions. Deep Learning, a field of study that recently gain attention from the public, is even more fit. Its ability, to progressively identify important elements of the text and keeps them in memory, allows to take into account the context of a word, which improve greatly the accuracy of the prediction.

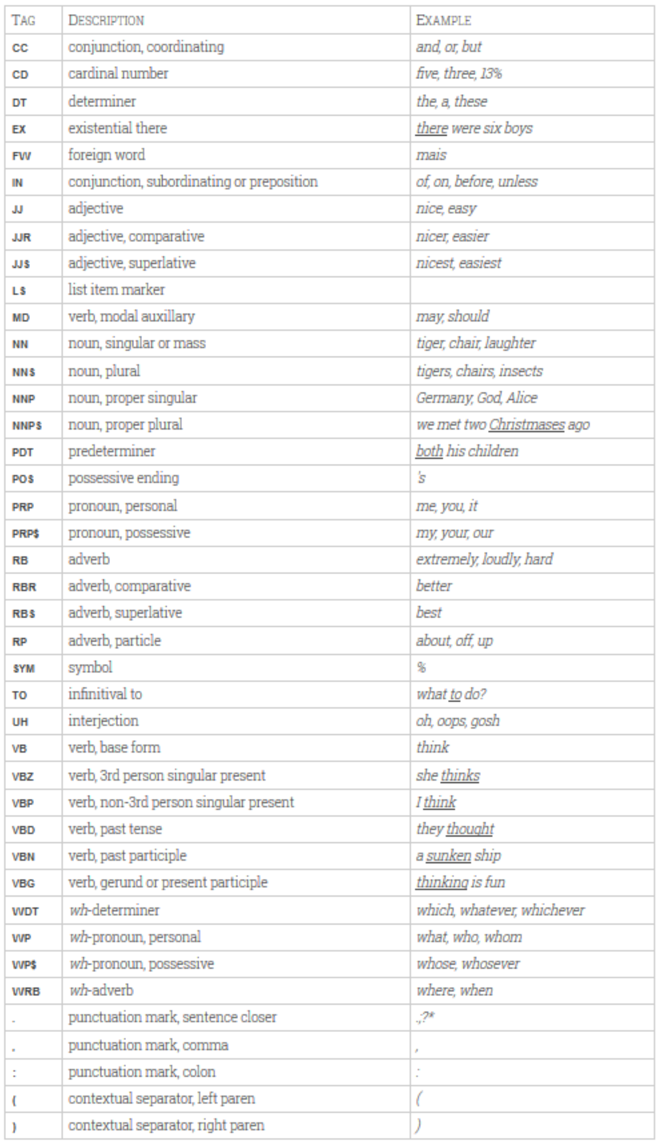
As those techniques become more and more democratized, we can expect that textual data will be less and less neglected.

8. Appendices

8.1. Lexicons

- Bigrams, Trigrams, Ngrams: Tokens of (Two, Tree, N) consecutive written words
- Bag of Words: multiset of the words of the phrase.
- Corpora: large body of text
- Chunking: Picking up individual pieces of information and grouping them into bigger pieces. In NLP, grouping tokens or tokens into chunks.
- Chunk: The result of chunking
- Embedding:
- Lemma: a word without any morphology (ex: eat, horse, etc.)
- Lemmatizing: replace a word by its lemma.
- Morphology: how a word can be modified to change their meaning, suffixes and prefixes are often used (ex: eat ==> eating).
- Named Entity Recognition (NER)
- NLG: Natural Language Generation
- NLP: Natural Language Processing
- NLU: Natural Language Understanding
- Normalization: removal of the elements that are not considered as important: uppercase, punctuation.
- Part-Of-Speech (POS): function of a word in a sentence (ex: pronoun, verb, etc.)
- Phrase: group of words that forms a unit. Several phrases can create a sentence (ex : noun phrase, verb phrase, etc.).
- Pluralization: transform a singular noun to a plural noun.
- Sentence: a set of words that is complete in itself
- Stemming: Normalization of a word into its base form or root form.
- Stem: Base form of the word.
- Stopwords: Useful for a construction of a phrase, not useful in the processing of language
- Syntax Tree: A tree representation of syntactic structure of sentences or strings
- Tagging: associating a word with a part of speech.
- Tokenization: cutting the sentence or phrases in individual words.
- Token: Result of tokenization
- Vectorize: Transform a token into a vector.

8.2. POS: Tags and Descriptions



Tag	Description	Example
CC	conjunction, coordinating	*and, or, but*
CD	cardinal number	*five, three, 13%*
DT	determiner	*the, a, these*
EX	existential there	*there were six boys*
FW	foreign word	*mais*
IN	conjunction, subordinating or preposition	*of, on, before, unless*
JJ	adjective	*nice, easy*
JJR	adjective, comparative	*nicer, easier*
JJS	adjective, superlative	*nicest, easiest*
LS	list item marker	
MD	verb, modal auxillary	*may, should*
NN	noun, singular or mass	*tiger, chair, laughter*
NNS	noun, plural	*tigers, chairs, insects*
NNP	noun, proper singular	*Germany, God, Alice*
NNPS	noun, proper plural	*we met two Christmases ago*
PDT	predeterminer	*both his children*
POS	possessive ending	*'s*
PRP	pronoun, personal	*me, you, it*
PRP$	pronoun, possessive	*my, your, our*
RB	adverb	*extremely, loudly, hard*
RBR	adverb, comparative	*better*
RBS	adverb, superlative	*best*
RP	adverb, particle	*about, off, up*
SYM	symbol	*%*
TO	infinitival to	*what to do?*
UH	interjection	*oh, oops, gosh*
VB	verb, base form	*think*
VBZ	verb, 3rd person singular present	*she thinks*
VBP	verb, non-3rd person singular present	*I think*
VBD	verb, past tense	*they thought*
VBN	verb, past participle	*a sunken ship*
VBG	verb, gerund or present participle	*thinking is fun*
WDT	*wh*-determiner	*which, whatever, whichever*
WP	*wh*-pronoun, personal	*what, who, whom*
WP$	*wh*-pronoun, possessive	*whose, whosever*
WRB	*wh*-adverb	*where, when*
.	punctuation mark, sentence closer	*.;?**
,	punctuation mark, comma	*,*
:	punctuation mark, colon	*:*
(	contextual separator, left paren	*(*
)	contextual separator, right paren	*)*

Figure 17: Comprehensive list of Part-Of-Speech
Source: https://www.clips.uantwerpen.be/pages/mbsp-tags

8.3. Techniques used to simplify the processing

8.3.1. Sparse matrix

After vectorization, we have a sparse matrix, meaning a matrix that is comprised of mostly zero values (Brownlee, A Gentle Introduction to Sparse Matrices for Machine Learning, 2018). The problem with sparse matrices is that it can be as computationally expensive to work with sparse matrices as to work with dense matrix. For example, in a deep learning model, if some specific techniques are not used, the weight of each perceptron will be multiplied by the input x even if this input is zero. Specific ways exist to avoid those useless computations and result in computations that require much less memory and training time.

Instead of storing the values of the sparse matrix in a traditional matrix, it is possible to store the non-zero values. Two techniques almost identical exist: the Compressed Sparse Row also called CSR and the Compressed Sparse Column also called CSC. The CSR stored the non-zero values by using three one-dimensional arrays: one for the value, one for the row indices and one for the column indices. With CSC the column indices are compressed before the row indices.

For example, with the following sparse matrix:

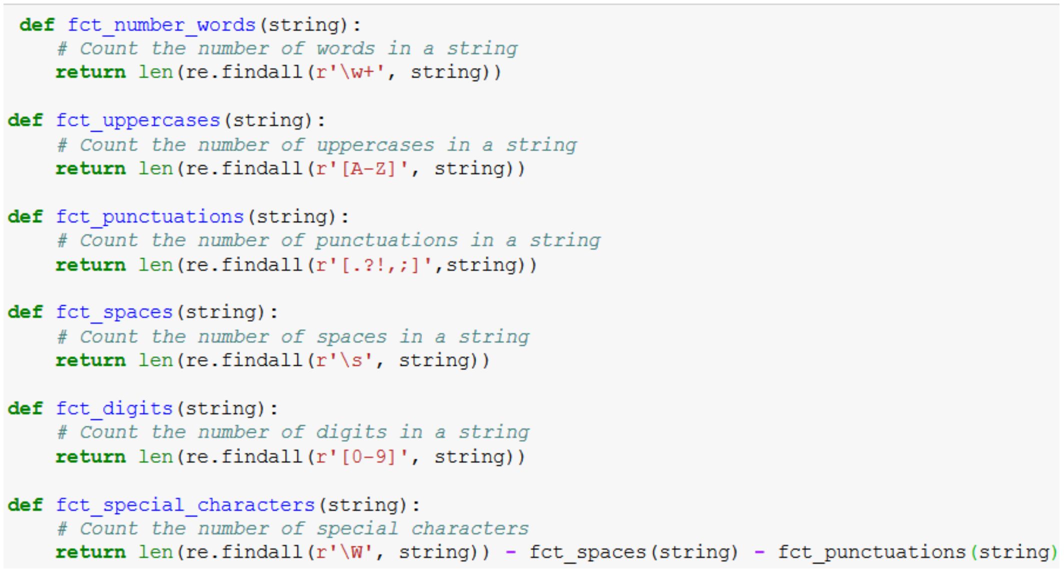
0	0	2	0	0
0	1	0	0	0
0	0	0	5	0

We would have the following CSR representation (in python the count starts at zero):

2	0	2
1	1	1
5	2	4

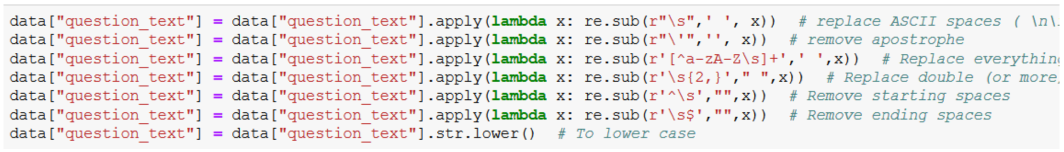
The sparse matrix can be retrieved from this compressed matrix. There is no loss of information.

8.4. Code extracts



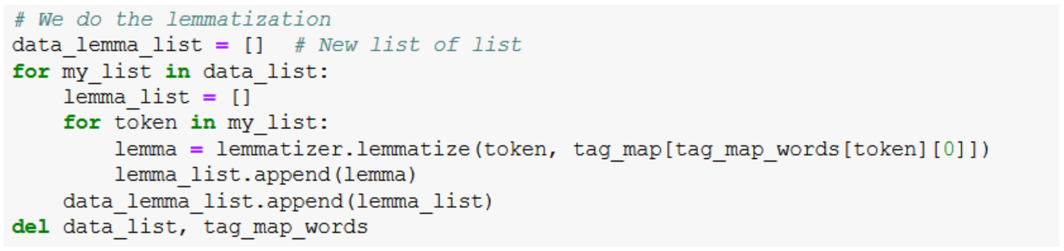
```python
 def fct_number_words(string):
    # Count the number of words in a string
    return len(re.findall(r'\w+', string))

def fct_uppercases(string):
    # Count the number of uppercases in a string
    return len(re.findall(r'[A-Z]', string))

def fct_punctuations(string):
    # Count the number of punctuations in a string
    return len(re.findall(r'[.?!,;]',string))

def fct_spaces(string):
    # Count the number of spaces in a string
    return len(re.findall(r'\s', string))

def fct_digits(string):
    # Count the number of digits in a string
    return len(re.findall(r'[0-9]', string))

def fct_special_characters(string):
    # Count the number of special characters
    return len(re.findall(r'\W', string)) - fct_spaces(string) - fct_punctuations(string)
```

Figure 18: Statistical features



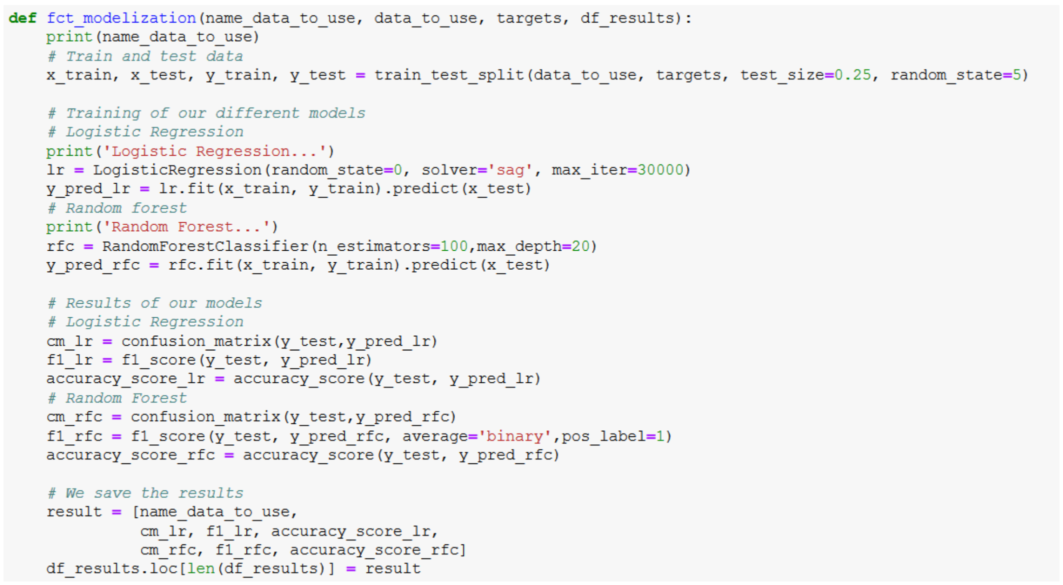
```python
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r"\s",' ', x))  # replace ASCII spaces ( \n\
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r"\'",'', x))  # remove apostrophe
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r'[^a-zA-Z\s]+',' ',x))  # Replace everythin
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r'\s{2,}'," ",x))  # Replace double (or more
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r'^\s',"",x))  # Remove starting spaces
data["question_text"] = data["question_text"].apply(lambda x: re.sub(r'\s$',"",x))  # Remove ending spaces
data["question_text"] = data["question_text"].str.lower()  # To lower case
```

Figure 19: Data Cleaning



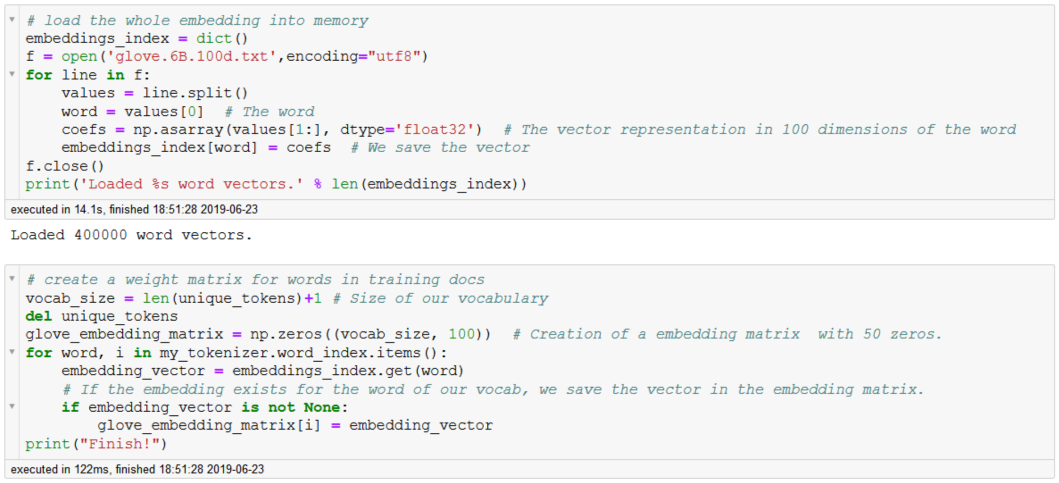
```python
# We do the lemmatization
data_lemma_list = []  # New list of list
for my_list in data_list:
    lemma_list = []
    for token in my_list:
        lemma = lemmatizer.lemmatize(token, tag_map[tag_map_words[token][0]])
        lemma_list.append(lemma)
    data_lemma_list.append(lemma_list)
del data_list, tag_map_words
```

Figure 20: Lemmatization



```python
def fct_modelization(name_data_to_use, data_to_use, targets, df_results):
    print(name_data_to_use)
    # Train and test data
    x_train, x_test, y_train, y_test = train_test_split(data_to_use, targets, test_size=0.25, random_state=5)

    # Training of our different models
    # Logistic Regression
    print('Logistic Regression...')
    lr = LogisticRegression(random_state=0, solver='sag', max_iter=30000)
    y_pred_lr = lr.fit(x_train, y_train).predict(x_test)
    # Random forest
    print('Random Forest...')
    rfc = RandomForestClassifier(n_estimators=100,max_depth=20)
    y_pred_rfc = rfc.fit(x_train, y_train).predict(x_test)

    # Results of our models
    # Logistic Regression
    cm_lr = confusion_matrix(y_test,y_pred_lr)
    f1_lr = f1_score(y_test, y_pred_lr)
    accuracy_score_lr = accuracy_score(y_test, y_pred_lr)
    # Random Forest
    cm_rfc = confusion_matrix(y_test,y_pred_rfc)
    f1_rfc = f1_score(y_test, y_pred_rfc, average='binary',pos_label=1)
    accuracy_score_rfc = accuracy_score(y_test, y_pred_rfc)

    # We save the results
    result = [name_data_to_use,
              cm_lr, f1_lr, accuracy_score_lr,
              cm_rfc, f1_rfc, accuracy_score_rfc]
    df_results.loc[len(df_results)] = result
```

Figure 21: Logistic Regression and Random Forest



```python
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.100d.txt',encoding="utf8")
for line in f:
    values = line.split()
    word = values[0]  # The word
    coefs = np.asarray(values[1:], dtype='float32')  # The vector representation in 100 dimensions of the word
    embeddings_index[word] = coefs  # We save the vector
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

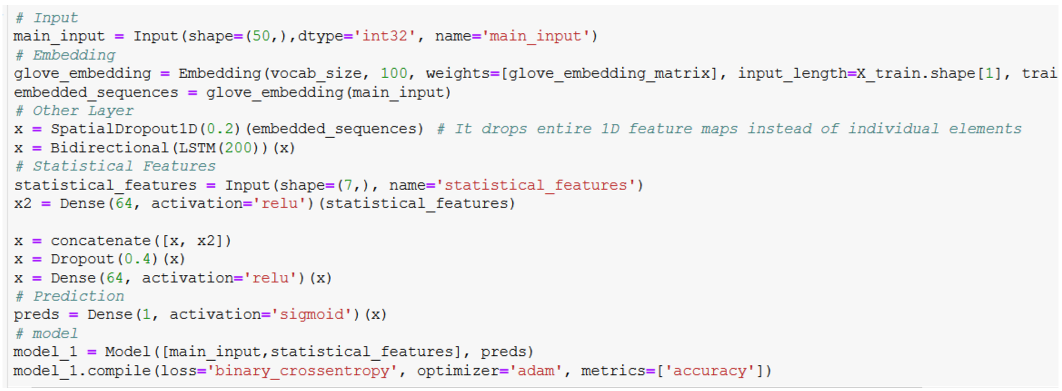
```
executed in 14.1s, finished 18:51:28 2019-06-23
```

```
Loaded 400000 word vectors.
```

```python
# create a weight matrix for words in training docs
vocab_size = len(unique_tokens)+1 # Size of our vocabulary
del unique_tokens
glove_embedding_matrix = np.zeros((vocab_size, 100))  # Creation of a embedding matrix  with 50 zeros.
for word, i in my_tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    # If the embedding exists for the word of our vocab, we save the vector in the embedding matrix.
    if embedding_vector is not None:
        glove_embedding_matrix[i] = embedding_vector
print("Finish!")
```

```
executed in 122ms, finished 18:51:28 2019-06-23
```

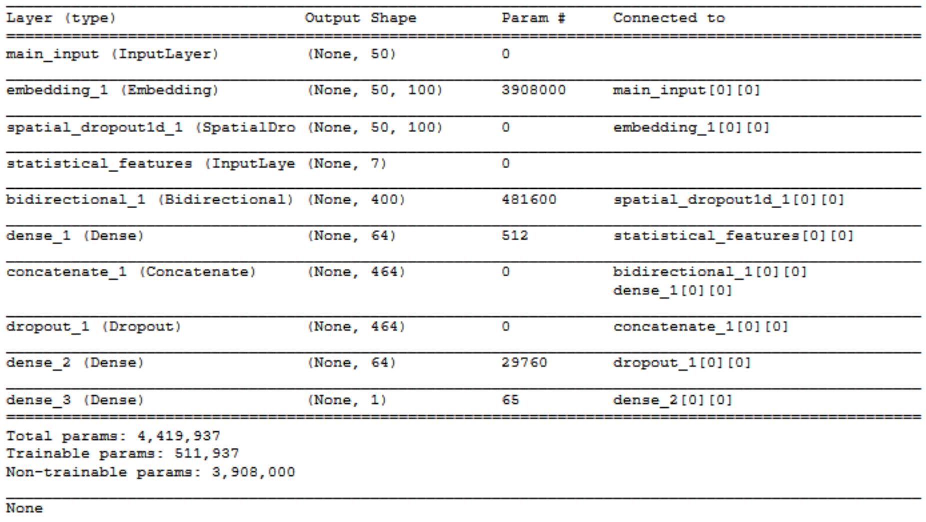
Figure 22: Embedding



```
# Input
main_input = Input(shape=(50,),dtype='int32', name='main_input')
# Embedding
glove_embedding = Embedding(vocab_size, 100, weights=[glove_embedding_matrix], input_length=X_train.shape[1], trai
embedded_sequences = glove_embedding(main_input)
# Other Layer
x = SpatialDropout1D(0.2)(embedded_sequences) # It drops entire 1D feature maps instead of individual elements
x = Bidirectional(LSTM(200))(x)
# Statistical Features
statistical_features = Input(shape=(7,), name='statistical_features')
x2 = Dense(64, activation='relu')(statistical_features)

x = concatenate([x, x2])
x = Dropout(0.4)(x)
x = Dense(64, activation='relu')(x)
# Prediction
preds = Dense(1, activation='sigmoid')(x)
# model
model_1 = Model([main_input,statistical_features], preds)
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

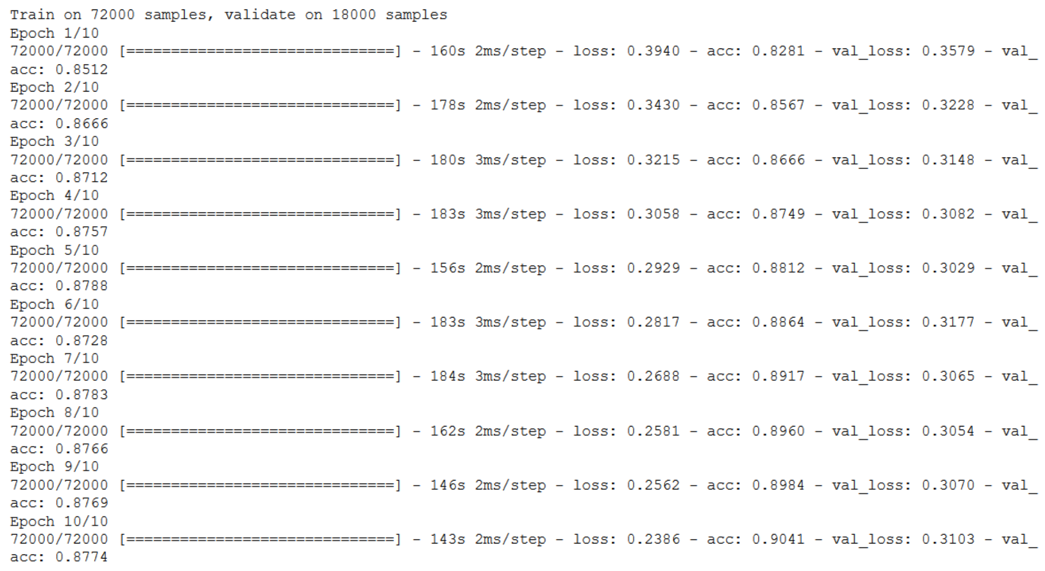
Figure 23: Structure of the Deep Learning model

We can observe that more than 500 000 parameters had to be trained. However, thanks to the pre-trained embedding, we spare the training of an additional 3.9 million parameters.



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
main_input (InputLayer)          (None, 50)            0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 50, 100)       3908000     main_input[0][0]
____________________________________________________________________________________________________
spatial_dropout1d_1 (SpatialDro  (None, 50, 100)       0           embedding_1[0][0]
____________________________________________________________________________________________________
statistical_features (InputLaye  (None, 7)             0
____________________________________________________________________________________________________
bidirectional_1 (Bidirectional)  (None, 400)           481600      spatial_dropout1d_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 64)            512         statistical_features[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 464)           0           bidirectional_1[0][0]
                                                                   dense_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 464)           0           concatenate_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 64)            29760       dropout_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 1)             65          dense_2[0][0]
====================================================================================================
Total params: 4,419,937
Trainable params: 511,937
Non-trainable params: 3,908,000
____________________________________________________________________________________________________
None
```

Figure 24: Summary of the Deep Learning model



```
Train on 72000 samples, validate on 18000 samples
Epoch 1/10
72000/72000 [==============================] - 160s 2ms/step - loss: 0.3940 - acc: 0.8281 - val_loss: 0.3579 - val_
acc: 0.8512
Epoch 2/10
72000/72000 [==============================] - 178s 2ms/step - loss: 0.3430 - acc: 0.8567 - val_loss: 0.3228 - val_
acc: 0.8666
Epoch 3/10
72000/72000 [==============================] - 180s 3ms/step - loss: 0.3215 - acc: 0.8666 - val_loss: 0.3148 - val_
acc: 0.8712
Epoch 4/10
72000/72000 [==============================] - 183s 3ms/step - loss: 0.3058 - acc: 0.8749 - val_loss: 0.3082 - val_
acc: 0.8757
Epoch 5/10
72000/72000 [==============================] - 156s 2ms/step - loss: 0.2929 - acc: 0.8812 - val_loss: 0.3029 - val_
acc: 0.8788
Epoch 6/10
72000/72000 [==============================] - 183s 3ms/step - loss: 0.2817 - acc: 0.8864 - val_loss: 0.3177 - val_
acc: 0.8728
Epoch 7/10
72000/72000 [==============================] - 184s 3ms/step - loss: 0.2688 - acc: 0.8917 - val_loss: 0.3065 - val_
acc: 0.8783
Epoch 8/10
72000/72000 [==============================] - 162s 2ms/step - loss: 0.2581 - acc: 0.8960 - val_loss: 0.3054 - val_
acc: 0.8766
Epoch 9/10
72000/72000 [==============================] - 146s 2ms/step - loss: 0.2562 - acc: 0.8984 - val_loss: 0.3070 - val_
acc: 0.8769
Epoch 10/10
72000/72000 [==============================] - 143s 2ms/step - loss: 0.2386 - acc: 0.9041 - val_loss: 0.3103 - val_
acc: 0.8774
```

Figure 25: Training of the Deep Learning model

9. References

Brownlee, J. (2014, September 26). *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

Brownlee, J. (2018, March 14). *A Gentle Introduction to Sparse Matrices for Machine Learning*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/sparse-matrices-for-machine-learning/

Cambridge University Press. (2009, April). *Stemming and lemmatization*. Retrieved from nlp.stanford.edu: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

CLiPS. (2019, May). *Penn Treebank II tag set*. Retrieved from CLiPS - Computational Linguistics & Psycholinguistics Research Center: https://www.clips.uantwerpen.be/pages/mbsp-tags

Dan, M. (2017, June). *Structured Data*. Retrieved from Datamation: https://www.datamation.com/big-data/structured-data.html

Demichelis, R. (2019, April 2). Quand les machines apprennent à écrire. *Les Echos*. Retrieved from https://www.lesechos.fr/tech-medias/intelligence-artificielle/quand-les-machines-apprennent-a-ecrire-1005790

Eduraka. (2018, October 7). *Natural Language Processing (NLP) & Text Mining Tutorial Using NLTK | NLP Training | Edureka.* Retrieved from YouTube: https://www.youtube.com/watch?v=05ONoGfmKvA

Gorman, B. (2019, June 23). *A Kaggler's Guide to Model Stacking in Practice*. Retrieved from blog.kaggle: http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The Elements of Statistical Learning* (2nd ed.). Springer Series in Statistics.

Hutchins, J. (2005, November). The history of machine translation in a nutshell.

Joseph, W. (1966, January). *ELIZA--A Computer Program For the Study ofNatural Language Communication Between Manand Machine.* Cambridge. Retrieved from http://www.universelle-automation.de/1966_Boston.pdf

Jurafsky, D., & Martin, J. H. (2018). *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*

Lagandula, A. C. (2019, June). *Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works*. Retrieved from Towards Data Science: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

Linguistic Data Consortium. (2019, June 19). *English Gigaword Fifth Edition*. Retrieved from catalog LDC University of Pennsylvania: https://catalog.ldc.upenn.edu/LDC2011T07

Pennington, J., Socher, R., & Manning, C. (2019, June 19). *GloVe: Global Vectors for Word Representation*. Retrieved from nlp.stanford: https://nlp.stanford.edu/projects/glove/

Quora, Inc. (2019, April). *About*. Retrieved from Quora: https://www.quora.com/about

Quora, Inc. (2019, February). *Quora Insincere Questions Classification*. Retrieved from Kaggle: https://www.kaggle.com/c/quora-insincere-questions-classification/data

Semaan, P. (2012, June). Natural Language Generation: An Overview. *Journal of Computer Science & Research (JCSCR)*, 50-57.

Wikipedia. (2019, April). *Quora*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Quora