Republic of Tunisia
Ministry of Higher Education and Scientific Research
University of Tunis El Manar
National Engineering School of Tunis

INDUSTRIAL ENGINEERING DEPARTMENT

Final year project

Presented by
Mayara LATRECH

To obtain the

National engineering diploma in Modeling for Industry and Services

A web browser plugin for offensive and hate speech detection

Within
Biware Consulting & U2S

Supported 18/06/2019

Examination Board :

President	: Mr. Mourad BELLASSOUED
Protractor	: Ms. Anissa RABHI
Examiner	: Mr. Azmi MAKHLOUF
Host company supervisor	: Mr. Amine BOUSARSAR
ENIT supervisor	: Mr. Jalel EZZINE
ENIT supervisor	: Ms. Chiraz BEN ABDELKADER

2018/2019

Acknowledgements

First, I would like to express my deepest thanks to Biware Consulting and U2S for offering me this opportunity to have this memorable experience and to work in such encouraging conditions.

I would like to thank Mr. Amine Bousarsar, my supervisor at Biware Consulting, and Mr. Jalel Ezzine and Ms. Chiraz Ben Abdelkader, my supervisors at ENIT for their guidance and their precious remarks during the internship period that made of this experience comfortable and interesting.

I would also like to thank Ms. Chiraz Ben Abdelkader for her insight and her sacrifices, especially the long working sessions we had in Biware Consulting.

I would definitely express my gratitude to ENIT for this rich and fruitful training and courses during those 3 years.

Last but not least, I would never forget the support and the encouragement that I had from my family and my friends during my studies.

Abstract

The purpose of this work is to automatically detect offensive and hate speech using machine learning algorithms. The work includes three main tasks : data extraction, data processing and data modeling .

We have used a data-set that contains twitter comments and their labels from a previous study : *"Automated Hate Speech Detection and the Problem of Offensive Language"* to train our model. We have used preprocessing techniques on the data-set such as tokenization and stemming then, we used text factorization techniques to have a numerical value representing words from the vocabulary. We implemented then various supervised machine learning models. A data rebalacing technique was introduced then changed the modeling architecture for better results.

Key Words : offensive and hate speech detection, text mining, machine learning, data rebalacing, feature engineering.

Résumé

Le but de ce travail est de détecter automatiquement les discours offensants et haineux à l'aide des algorithmes d'apprentissage automatique. Le travail comprend trois tâches principales : l'extraction de données, le traitement de données et la modélisation de données.

Nous avons utilisé un ensemble de données contenant des commentaires de Twitter et leurs libellés d'une étude précédente : *"Automated Hate Speech Detection and the Problem of Offensive Language"* pour entrainer notre modèle. Nous avons utilisé des techniques de prétraitement sur les données comme la tokénisation et le troncage. Ensuite, nous avons utilisé des techniques de factorisation de texte pour obtenir des valeurs numériques qui représentent les mots du vocabulaire. Nous avons mis en place différents modèles d'apprentissage automatique supervisé. Une technique de rééquilibrage des données a été introduite puis, l'architecture de modélisation a été modifiée pour avoir de meilleurs résultats.

Mots clés : détection de discours offensifs et haineux, analyse de texte, apprentissage automatique, rééquilibrage des données, ingénierie des caractéristiques.

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

With the rise of digital technology, virtually everyone has access to social media. A communication platform is then given to any user. Thus, comments on social media may contain offensive language, sexist or racist speech. Many users are disturbed by that kind of language and wish to have it automatically filtered. Concerned parents may want to filter all kinds of offensive speech from their children's social media pages. Other users may not be bothered by blacklisted words but do not want to see any hate speech in their social media feed. Therefore, offensive and hate speech detection is becoming an emerging mission that holds many challenges.

Before identifying hate and offensive speech, we need to differentiate them first. According to Cambridge dictionary, offensive language is the kind of language that causes someone to be upset or angry[1]. This definition can be subjective because not everyone gets angry or upset by the same speech. Hate speech is *"public speech that expresses hate or encourages violence towards a person or group based on something such as race, religion, sex, or sexual orientation"* Cambridge Dictionary[1]. By these definitions, we can consider speech that contain a certain blacklisted word as offensive in a context but hateful in an another context.

In this project, text mining was employed to detect offensive and hate speech. A labelled data-set was taken from a previous study. It was cleaned then encoded into numerical values using the bag-of-words model. Then, the presentation of the previous data-set was injected into a model. After obtaining the modeling results, we tweaked each step of the previous pipeline to obtain better results.

This report is organized as follows. The first chapter is a general introduction that contains a presentation of the host company and laboratory and the project motivation. The second chapter is the state of the art where the techniques used in this project are detailed. The third chapter details the project contribution by describing the methodology of deployment of the techniques previously described in the second chapter, presenting experimental results. The final chapter is the application development part of the project.

1.1 Project context

This project is a joined project between Biware Consulting and U2S in order to create a bridge between the academic side and the industrial side.

1.1.1 Host company presentation

Founded in 2011, Biware Consulting is a company that specialises in implementation of decision support solutions for financial services, telecommunications, distribution and industry. It offers an expertise in collecting, structuring and restituting data in the operational and strategic area of it's clients[2]. Biware's expertise can be divided into three themes:

i. Data Management which contains three main sections.
- Business Intelligence: the establishment of data warehouses and multidimensional analysis systems, business performance measures based on key indicators.
- Customer Intelligence: the proposition of customer behavior analysis and segmentation solutions as well as operational and analytical CRMs (Costumer Relationship Management)
- Market Intelligence: The establishment of strategic watch systems and risk management systems[2].

ii. Advanced Analytics where Biware focuses on Predictive Analytics, Optimizing solutions, Personalized services and Social Media Analytics[2].

iii. Data discovery the proposal of analytical toolboxes to deal with precise and variant topics over time[2].

1.1.2 Host lab presentation

1.1.2.1 ENIT

ENIT (Ecole Nationale d'Ingenieurs de Tunis) is a public Tunisian university that was founded in 1968 by Mr. Mokhtar Laatiri. Each year, nearly 400 national engineering exam laureates choose ENIT to pursue their three-year course in one of its nine sectors. The recruitment of engineering students at ENIT takes place mainly on three (3) national entrance exams for engineering training courses. These training courses are: Mathematics and Physics (MP), Physics and Chemistry (PC) and Technology (T). They consist of written tests for candidates who have completed two (2) years of preparatory classes. The Specific entrance exam (CS), which is a an entrance exam based on files, also allows to recruit students in the first year among the good candidates holding a license degree. In addition, the school recruits foreign students under the same conditions as Tunisian students. Over the past five years, enrollment of ENIT students has averaged 60% MP, 22% PC, 16% T and 2% CS.

On average over the last five years, ENIT has 45% of its students who received the mention "Good" at the baccalaureate and 47% of mentions "Very Good" which confirms the good level of students recruited, 31.4% are originating from the four governorates of the Tunis region, 67.3% from other regions of Tunisia and 1.3% from foreign countries (mainly from sub-Saharan Africa). The female gender represents a large part of the enrollment of school admissions: 41% on average over the last five years.

The nine sectors of ENIT are: Civil Engineering (GC), Electrical Engineering (GE), Hydraulic Engineering and Environment (GHE), Industrial Engineering (GI), Mechanical Engineering (GM), Computer Science (Info), Modeling for Industry and Services (MIndS), Advanced Techniques (AT) and Telecommunications (Telecom). These nine sectors are managed by five departments: Civil Engineering, Electrical Engineering, Industrial Engineering, Mechanical Engineering, and Information and Communication Technologies[3].

1.1.2.2 U2S

Signals and Systems Research Unit is a research unit within the Information and Communication Technology Department in the National School of Engineers of Tunis (ENIT).

The research areas of the unit are based on the study of signals and the theory of systems applied to ICT. This part includes several disciplines including Analysis and Design of Aloghorithms of Filtering, Analysis and Processing of Signals and Non-stationary Systems, Control of Nonlinear Systems, Decision Support Models for Energy Systems. Other research topics are developed in parallel is that mainly focus on telecom aspects such as the design of Radiocommunication systems, ...

The members of the unit belong to two categories according to the scientific level. On the one hand, we find the class of confirmed researchers that is to say those who have already obtained their thesis and who are rightly the Assistant Masters, the Masters of Conferences and the professors. On the other hand, the "junior researchers" constitute the other half of the members, thus designating the PhD students and the masters students. Currently, the unit has 38 members including 16 PhD students.

The majority of projects undertaken within the unit are located within the framework of national and / or international cooperation. This is facilitated by the development of a spirit of openness expressed through cotutelles or even CMCUs. The aim is to develop theoretical tools that have utility on a practical scale[4].

1.2 Project motivation

The heavy use of social media has become the norm for most people. In fact, in 2011 it was estimated that 70% of teens use social media on a daily basis. Nearly one in four teens visits their favorite social media sites more than 10 times a day[5]. It is true that children benefit from the use of social media by interacting and learning from others. However, they become exposed to large amounts of offensive content online.

ScanSafe's monthly "Global Threat Report" found that up to 80% of blogs contained offensive contents. As adolescents are more likely to be negatively affected by biased and harmful contents than adults, detecting online offensive contents to protect adolescent online safety becomes an urgent task[5].

Simply flagging offensive words in social media is effective to an extent. However, it does not take into account that hate speech is not dependant on offensive words. Some comments can be extremely hateful without resorting to any offensive words. Furthermore, some offensive words are used as slang. They are used by children in their daily conversations. If we simply flag all offensive words, we will restrict the normal flow of conversations children have online. Hence, we would like to have a tool to distinct between offensive speech and hate speech.

Chapter 2

State of the art

2.1 Existing products

Before presenting the techniques used in this project, let us see what products exist in the market for filtering offensive and/or hate speech.
There are plenty of Android , iOS and computer applications to filter offensive words. In fact, most mobile keyboard applications offer the option to censor blacklisted words.

For example, Swype is an Android application that replaces the on-screen keyboard for smartphones and tablets. It offers many option including the option to turn enable/disable curse-words. It stars them when blocked[20].

CleanSpeak is an application that the user integrates in their website to filter speech. They offer smart identifiers of blacklisted words as well as a kid's chat filter that is even stricter. I only allows vocabulary from a whitelist they defined[21].

Music Bleeper is an iOS application that bleeps blacklisted words from the user's music library. The user defines the blacklist. This app is mostly targeted for parents and children to use. The parent will define the blacklisted words and the child will use their phone with this application running in the background[22].

Nofanity is a desktop application that censors blacklisted words in YouTube videos as well as YouTube comments. The user chooses how strict the swear filter is (Light, Moderate, or Aggressive)[23].

Many other products exist in the market that are similar to the ones we presented. We notice that those products are dictionary based and they do not make the distinction between hate and offensive speech.
The academic world has plenty of research paper on detecting and classifying offensive language. The industrial world has plenty of application to filter words. The prospects of this project is to build a bridge between those two words and come up with an application that uses more advanced techniques to classify offensive language.

2.2 Text mining pipeline

Text mining is a branch of data mining used to process text corpora in order to analyse its content and extract knowledge. Information on the text is typically derived through the devising of patterns and trends through means such as statistical pattern learning.
Text mining tasks include spam filtering in e-mails, fraud detection by insurance companies, prediction and prevention of crime[6]. In this project, text mining is used to classify tweets into offensive speech, hate speech and other.
The text mining pipeline will be followed during this project.

Figure 2.1 – Text mining pipeline

2.3 Text cleaning

The first step of the text mining pipeline is cleaning the raw data. This step is used to transform the corpus we have in the form of sequences of characters into a clean corpus in the form of sequences of words[6].
In this project, the following sub-tasks of text cleaning were used:
- Removing useless characters
- Tokenization
- Converting to lowercase
- Removing stop words
- Word normalization

2.3.1 Removing useless characters

Tweets contain many useless characters such as emojis, hyperlinks, email addresses, punctuation marks, numbers, math symbols and user names. In order to find those characters, I used regular expressions to detect and remove them.
A regular expression is a string character that describes a set of possible character strings[7]. For example the string "\w" refers to [A-Z] and [a-z] characters and the string "\d" refers to [1-9] characters.

2.3.2 Tokenization

Tokenization is the act of breaking a string into pieces. The pieces, also called tokens, can be words, keywords, phrases or symbols[8]. I used the tokenization technique to break

each document (which is a single tweet) into a list of words. The tokenizer used here cuts the string when it finds a white space, it also deletes punctuation marks that were not deleted in the previous task using regular expressions.

2.3.3 Converting to lowercase

Now that we have our documents in the form of lists of words, we then convert all the words to lowercase. This is done so that the words ”Hello”, ”hello” and ”HELLO” are considered as the same word.

2.3.4 Removing stop words

Stop words are words that are too frequent in sentences but add no real value to the overall meaning. In English, the list of stop words contains ”i”, ”the”, ”or”, ”is”, ”he”, ”who”, ”to”, ”my”, ”off” etc.. These words are removed to reduce the overall number of words in the corpus.

2.3.5 Word normalization

Text normalisation is an essential task for natural language processing. It is the transformation of a text into a standard and more simplified form that makes the language processing more efficient.
Natural language contains many derivative words, combined forms and conjugated terms. The purpose of word normalisation is to attribute one word to all the derivatives of the latter[9].
The most common word normalisation technique is lemmatization.

2.3.5.1 Lemmatization

Lemmatization is the task of grouping different variations of the same word with the use of a vocabulary and morphological analysis of words. It normally aims to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. For example, the words ”try”, ”trying”,”tried”, ”tries” have the same lemma which is ”try”[9].
Since lemmatization uses morphological analysis to determine the lemma of words, it can be time consuming. Hence, I used a fast heuristic method for word normalisation which is stemming.

2.3.5.2 Stemming

Stemming has the same purpose as lemmatization which is to group words that have the same root into one string called stem. The stem of a word is achieved by chopping off the end of words in the hope that it coincides with those words’ affixes[9]. I have tried

four different stemmers from the NLTK (Natural Language ToolKit) library.

i. Porter Stemmer

The Porter Stemmer algorithm was developed by the mathematician Martin Porter in 1980. This algorithm does not require a stem dictionary but it does require an explicit list of suffixes. The algorithm has certain rules to know if it should remove the suffix from the word or not.[10] The result of this algorithm is a stem of a word that might not be a word itself. For example, the stem of "trying" and "try" is "tri". The details of this algorithm are in the appendix.

ii. Regexp Stemmer
Regular expressions can be used for stemming. In fact, We can define a string of regular expressions then, the stemmer removes any prefix or suffix that matches the expression[10].

2.3.6 Conclusion

After cleaning the corpus by removing useless characters and stop words, tokenizing and normalising the words, our clean text is now ready for document representation. This phase is important because text documents lack the imposed structure of a traditional database. Therefore, the corpus, that corresponds to unstructured data, needs to be transformed to structured data.

2.4 Document representation

Document representation is the task of transforming the corpus that is in the form of unstructured data into a representation of that corpus that corresponds to structured data. There are various models that are currently used for document representation. In this project we used the Bag-of-words model.

2.4.1 Bag-of-words

The bag-of-words model is considered as one of the most popular methods for document representation. In fact, the simplest way to represent document is to consider each unique word as a feature.

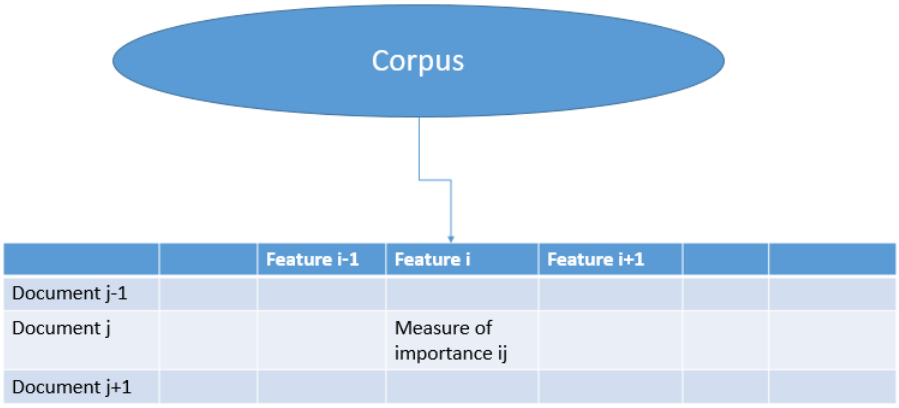


Figure 2.2 – Feature extraction

The corpus is a set of documents. It can also be considered as set of N key words called vocabulary or features. Each document is transformed to a vector of length N called feature vector where the value of i-th element of the vector is the measure of importance of the i-th feature in the document.

When the documents are stacked together they form a matrix. It is then considered as structured data.
The bag-of-words model has many variants that differ in calculating the measure of importance.
Let X_i be the i-th element of the feature vector X. The value of X_i can be calculated using different methods.

2.4.1.1 Existence

If the feature exists in the document then $X_i = 1$. If the feature does not exist in the document then $X_i = 0$ [12].

2.4.1.2 Occurrence

The value of X_i is the number of occurrences of of the feature in the document[12].

2.4.1.3 Term Frequency

Term Frequency measures how frequently a feature occurs in a document. The documents have different lengths. Hence, it is possible for a feature to occur more frequently in long documents than shorter ones. The term frequency is often divided by the document length as a way of normalizing occurrences. The value of X_i is the term frequency or TF.

$$TF = \frac{number\ of\ occurrences\ of\ the\ feature\ in\ the\ document}{number\ of\ words\ in\ the\ document} [12]$$

2.4.1.4 TF-IDF

TF-IDF weight or *term frequency-inverse document frequency* weight is a statistical measure that evaluates the level of importance of a word to a document in a the corpus.The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. [12] TF-IDF is the product of two statistics term frequency and inverse document frequency.
IDF is a measure of the rarity of the feature, i.e. how frequently it appears in the corpus. It is known that certain terms in the English language, such as "the", "of", and "how", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones. The most common variation of IDF is

$$IDF = log(\frac{M}{1 + d \in D : t \in d})[6]$$

Where, **M** is the number of documents in the corpus.
$(d \in D : t \in d)$ number of corpus documents that contain the feature t. Other variations of IDF can be found in the literature[12].

Weighting scheme	IDF weight $n_t = \|d \in D : t \in d\|$
Unary	1
Inverse document frequency	$log(\frac{M}{n_t})$
Inverse document frequency max	$log(\frac{max_{t' \in d} n_{t'}}{1+n_t})$
Probabilistic inverse document frequency	$log(\frac{M-n_t}{n_t})$

Table 2.1 – Variations of IDF

2.4.2 Conclusion

Now that the corpus is cleaned and transformed into the form of structured data, it is ready to be the input for the modeling phase in order to achieve the classification task.

2.5 Modeling

2.5.1 Introduction

Machine learning is a sub-field of Artificial Intelligence that gives computers the ability to automatically discover patterns in given data. Machine learning can be classified into 3 types of algorithms[13].

- Supervised Learning where algorithms learn from a labeled data-set. The algorithm detects patterns after it matches given inputs to given outputs. It infers a function based on labelled training data which is a pair of data points and targets.
- Unsupervised Learning where algorithms infer patterns from a data-set without having a given output.
- Reinforcement learning where an algorithm performs actions based on a reward and

punishment system. It receives rewards by performing correctly and penalties for performing incorrectly.

Machine learning can also be categorised by the output[13].
- Classification where inputs are divided into two or more classes. The algorithm then assigns new inputs into the classes.
- Regression where the outputs are continuous.
- Clustering where the algorithm divides inputs into groups. Unlike classification, those groups are not known beforehand.
- Density estimation where the algorithm finds the distribution of inputs in a space.
- Dimensionality reduction where the algorithm simplifies inputs by mapping them into a lower-dimensional space.

In this project, supervised machine learning algorithms are used to classify data into 3 categories.

Let us denote an input by $x \in \mathbb{R}^D$ where D is the dimension of the data. The training set contains N data points. It is represented by the matrix [14]

$$\mathbf{X} := (x_1, ..., x_N)^T \in \mathbb{R}^{N \times D}$$

The training set has associated targets $\mathbf{t}$ where [14]

$$\mathbf{t} := (t_1, ..., t_N)^T$$

The algorithm we want to build is one that takes new data $\mathbf{x}$ and predicts it's corresponding target $\mathbf{t}$. In order to predict the target, we estimate it by a function [14]

$$y(x) = y(x, \mathbf{w})$$

where $\mathbf{w} := (w_1, ..., w_M)^T \in \mathbb{R}^M$ is a vector of parameters or weights previously determined from the training set[14].
The following sections will presents the various machine leaning algorithms used in this project.

2.5.2 Logistic regression

Suppose that we have K classes $\mathcal{C}_1, ..., \mathcal{C}_K$ where $K \geq 2$ and a vector $\phi \in \mathbb{R}^M$ of M features. Logistic regression consists of finding the most likely vector of parameters such that for $k = 1..K$,

$$p(\mathcal{C}_k|\phi) = y_k(\phi) := \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$$

[14]

where the activations a_k are

$$a_k := \mathbf{w}_k^T \phi$$

The target $\mathbf{t} = (t_1, ..., t_N)^T \in \{0, 1\}^K$ is encoded by "one-of-K" which means that $t_k = 1$ only if the corresponding input $\mathbf{x}$ comes from the class $\mathcal{C}_k$. Data are:

$$(\phi_n) := (\phi(\mathbf{x}_n))$$

$$y_{n,k} := y_k(\phi_n)$$

$$\mathbf{T} := (t_{n,k}) \in \mathbb{R}^{N \times K}$$

The likelihood is the given by

$$\begin{aligned} p(\mathbf{T}|\mathbf{w}_1, ..., \mathbf{w}_K) &= \prod_{n=1}^{N} \prod_{k=1}^{K} p(t_{n,k}|\mathbf{w}_k) \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\phi_n)^{t_{n,k}} [14] \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} y_{n,k}^{t_{n,k}} \end{aligned}$$

We have to minimize the cross-entropy that is given by

$$\begin{aligned} E(\mathbf{w}_1, ..., \mathbf{w}_K) &:= -\log p(\mathbf{T}|\mathbf{w}_1, ..., \mathbf{w}_K) \\ &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \log y_{n,k} \quad [14] \end{aligned}$$

For $j = 1..K$,

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, ..., \mathbf{w}_K) = \sum_{n=1}^{N} (y_{n,j} - t_{n,k}) \phi_n$$

The Hessian H of E consists of a $(MK \times MK)$ matrix with K blocks by K, of size $(M \times M)$ each, and each $(j, k)^{th}$ block is given by

$$\nabla_{\mathbf{w}_k} \nabla_{\mathbf{w}_j} E(\mathbf{w}_1, ..., \mathbf{w}_K) = \sum_{n=1}^{N} y_{n,k}(I_{k,j} - y_{n,j}) \phi_n \phi_n^T [14]$$

where we used

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{k,j} - y_j)$$

The matrix H is definite positive, which implies the existence of a unique minimum for the cross-entropy error E.

2.5.3 Support Vector Machine

In this section, the SVM classifier is introduced for binary classification. Then, it is modified for multi-class classification.
For an input $\mathbf{x}$ we consider a target $t \in \{-1, 1\}$ such that

$$t := \begin{cases} +1, & \text{if } y(\mathbf{x}) > 0 \\ -1, & \text{if } y(\mathbf{x}) < 0 \end{cases} [14]$$

where $y(\mathbf{x}) := \mathbf{w}^T\phi(\mathbf{x}) + b$ For the training data $(\mathbf{x}_1, t_1, ..., \mathbf{x}_N, t_N)$ we have

$$t_n y(\mathbf{x}_n) = |y(\mathbf{x}_n)| > 0, \text{for } n = 1..N[14]$$

We assume that the training data-set is linearly separable in the feature space.
To train the model, SVM chooses the decision boundary $D := \{y(\mathbf{x}) = 0\}$ with maximum margin. Where the margin is the smallest distance between D and any input $\mathbf{x}_n$.

$$\text{margin } = \min_n \frac{|y(\mathbf{x}_n)|}{||\mathbf{w}||} = \min_n \frac{t_n y(\mathbf{x}_n)}{||\mathbf{w}||}$$

Hence,

$$\text{SVM } \equiv \max_{\mathbf{w},b} \{ \frac{1}{||\mathbf{w}||} \min_n \{t_n(\mathbf{w}^T\phi(\mathbf{x}) + b)\}\}[14]$$

Since there is invariance by $\mathbf{w} \mapsto const * \mathbf{w}$ and $b \mapsto const * b$ we can equivalently choose to

$$\begin{cases} \min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \\ \text{s.t } t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) \geq 1, \ n = 1..N \end{cases}$$

An optimization problem has a dual form if we evolve in a convex space. which is the case. We apply then by the Lagrangian formula.

$$\mathbf{L}_p(\mathbf{w}, b, a) = \frac{1}{2}||b||^2 - \sum_{n=1}^{N} a_n(t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) - 1)$$

Where a_n are the multipliers of Langrange.

By setting the first partial derivatives to zero we have

$$\begin{cases} \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) = 0 \\ \frac{\partial L}{\partial b} = \sum_{n=1}^{N} a_n t_n = 0 \\ \frac{\partial L}{\partial a_n} = -t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) + 1 \leq 0, \forall n \end{cases} [15]$$

The solution has to satisfy the conditions of Karush-Khun-Tucker (KKT)

$$a_n(t_n(\mathbf{w}^T\phi(\mathbf{x}_n) + b) - 1) = 0, \forall n[15]$$

By introducing the information resulting from the cancellation of partial derivatives of the Lagrangian, we obtain an optimization depending only on multipliers.

$$\begin{cases} \max_{\mathbf{a}} \tilde{L}(\mathbf{a}) := \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{s.t } a_n \geq 0, \ n = 1..N \\ \sum_{n=1}^{N} a_n t_n = 0 \end{cases} [14]$$

with

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n)$$

Then,

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b[14]$$

Karush-Khun-Tucker (KKT) conditions give us, for $n = 1..N$ either

$$a_n = 0 \text{ or } t_n y(\mathbf{x}_n) = 1$$

Then for those data $\mathbf{x}_n$ such that $a_n = 0$, they do not entre in the above sum. For the others, $t_n y(\mathbf{x}_n) = 1$, i.e. $y(\mathbf{x}_n) = \pm 1$ meaning that these $(\mathbf{x}_n)$ or support vectors lie on the maximum margin hyperplanes in feature space.

In our case, SVM is applied of multi-class classification. The approach used is *one-versus-all*. It consists of having K separate SVMs where K is the number of classes. For every $k = 1..K$, data from class $\mathcal{C}_k$ are the positives and all the rest are the negatives.

2.5.4 Naive Bayes

Naive Bayes are a set of supervised learning algorithms based on applying Bayes' theorem. It is called naive because it assumes independence between every pair of features given the value of the class variable.
Let y be a class variable and $x_1, ..x_n$ are feature vectors. Bayes' theorem states the following relationship

$$P(y|x_1, ..x_n) = \frac{P(y)P(x_1, ..x_n|y)}{P(x_1, ..x_n)}[16]$$

Using the naive conditional independence assumption

$$P(x_i|y, x_1, ..x_{i-1}, x_{i+1}, ..x_n) = P(x_i|y), \forall i = 1..n$$

The relationship is then simplified

$$P(y|x_1, ..x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..x_n)}[16]$$

$P(x_1, ..x_n)$ is constant given the input, so we get

$$P(y|x_1, ..x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

We can use the following classification rule

$$\hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_i|y)[16]$$

We can use the Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i|y)$, the former is then the relative frequency of class y in the training set[16].

2.5.5 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the data-set and uses averaging to improve the predictive accuracy and control over-fitting.[17]

2.5.5.1 Decision Tree

Decision Tree is a rule based supervised learning method. The model makes prediction based on decision rules inferred from data features. To build a decision tree, we perform classification of an input through a series of tests on attributes that describe it and organize all the tests possible as a tree. A leaf of this tree designates one of the C classes (but at each class can match several leaves). Each node is associated with a test on one or more attributes. The classification is carried out starting from the root to pursue recursively the process until we meet a leaf.
Building a decision tree X:
-If all the points of X belong to the same class then create a leaf bearing the name of this class.
Else
- choose the best attribute to create a node the test associated with this node separates X into parts: X_1....X_n
- build-tree (X_1)
- ...
- build-tree (X_n)
End

We stop the deepening of the tree when the stop conditions are met. Stop conditions may include:
- Depth of the tree reaches a fixed limit (= number of variables used)
- Number of leaves reaches a fixed maximum
- The number of each node is less than a fixed threshold

to find the variables that best separate individuals from each class we have various choice criterion. One of the most popular criterions is CART (Classification And Regression Tree). To apply it we calculate Gini's index

$$I = 1 - \sum_{1}^{n} f_i^2$$

Where n is the number of classes and f_i is the frequency of the class in the node.
When Gini's index is lower, we say that the node is more pure.[18]

2.5.5.2 Random forest algorithm

1. For b = 1 to B:
(a) Draw a bootstrap sample $Z*$ of size N from the training data.
(b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the

following steps for each terminal node of the tree, until the minimum node size $n_{\min}$ is reached.
i. Select m variables at random from the p variables.
ii. Pick the best variable/split-point among the m.
iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x: Let $\hat{C}_b(x)$ be the class prediction of the b^{th} random-forest tree.

$$\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B[19]$$

2.6 Evaluation

To evaluate the performance of a model we use various metrics such as accuracy, precision, recall and f1-score. In binary classification a confusion matrix would be

Data class	Classified as positive	Classified as negative
positive	true positive (TP)	false negative (FN)
negative	false positive (FP)	true negative (TN)

Table 2.2 – Confusion matrix for binary classification

For multi-class classification we will have TP_i, TN_i, FN_i and, FP_i that correspond to the class $\mathcal{C}_i$.

Accuracy: Formally, the accuracy has the following definition

$$\text{Accuracy } = \frac{\text{number of correct predictions}}{totalnumberofpredictions}[24]$$

In multi-class classification we have

$$\text{Accuracy } = \frac{\sum_{i=1}^{K} \frac{\text{TP}_i+\text{TN}_i}{\text{TP}_i+\text{TN}_i+\text{FN}_i+\text{FP}_i}}{K}[24]$$

Precision: Precision is a measure of a classifier's exactness. It is defined for each class $\mathcal{C}_i$ by

$$P_i = \frac{\sum_{i=1}^{K} \text{TP}_i}{\sum_{i=1}^{K}(\text{TP}_i + \text{FP}_i)}[24]$$

Recall: Recall is a measure of a classifier's completeness. It is defined for each class $\mathcal{C}_i$ by

$$R_i = \frac{\sum_{i=1}^{K} \text{TP}_i}{\sum_{i=1}^{K}(\text{TP}_i + \text{FN}_i)}[24]$$

F1-Score: F1-Score is a weighted average of precision and recall. It is defined for each class $\mathcal{C}_i$ by

$$F1_i = 2 * \frac{P_i * R_i}{P_i + R_i}[24]$$

2.7 Conclusion

In this chapter, we introduced the text mining pipeline. After collecting the data, we preprocess it by removing noise, tokenization and stemming. After cleaning the document, we vectorize them using the bag-of-words model. The corpus is now ready for classification. We presented four classifiers: Logistic regression, Support Vector Machine, Naive Bayes, and Random forest. Once we get the classification results we can evaluate the model using various metrics.

Chapter 3

Back-end part: Text mining pipeline

In this chapter, the work achieved in the internship will be presented. We will compute the concepts of chapter 2 then give the results. We will start by presenting the workflow adopted to build the final model.

3.1 The workflow

In this project, we chose to start simple, see the results given then add complexity. We followed the workflow presented by the figure below.

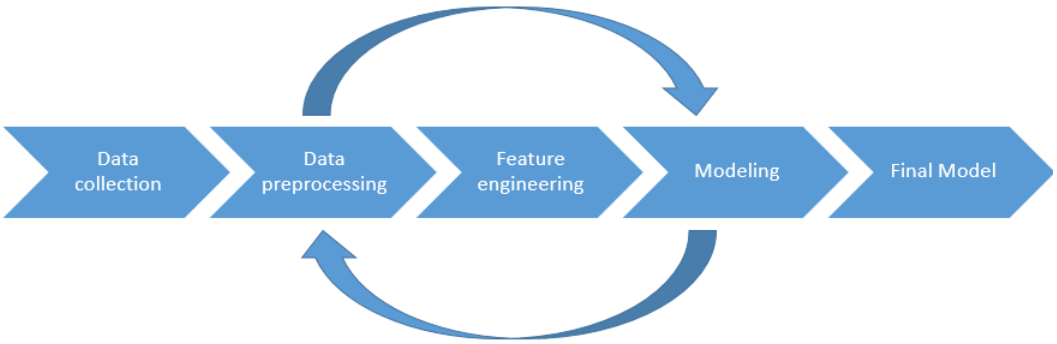


Figure 3.1 – The workflow adopted

The first step is to collect data. Once we have the data-set we can proceed with preprocessing. Now that we have a clean document we can start the phase of feature engineering. When features are ready, they are fed into a model. We repeat this loop using various preprocessing, feature engineering and modeling techniques to obtain the final model.

3.2 Data Collection

Before collecting the data, a hate speech lexicon is formed containing words and phrases identified by internet users as hate speech. This lexicon is compiled by hatebase.org which is a platform built to help detecting and monitoring hate speech. They analyse public conversations using vocabulary based on nationality, sexual orientation, gender, ethnicity, religion, class and disability with data across 90+ languages and 175+ countries.
They used the twitter API to search for tweets that contain terms from that lexicon. They obtained a corpus of 85.4 million tweets. From this corpus they then took a random sample of 25k tweets containing terms from the lexicon and had them manually coded. The workers then labelled each tweet as hate speech, offensive but not hate speech, or neither offensive nor hate speech. Each tweet was coded by three or more people. The final label of each tweet is assigned with a majority decision. Some of the tweets had no majority class so no label was assigned to them. As a result, they obtained a final sample of 24,783 labelled tweets. [25] This data-set is now put in their GitHub page for free use. I have downloaded it as a starting point in this project.

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

Figure 3.2 – The data-set

The data-set contains 7 columns:
- An unnamed column that represents the position of each tweet.
- A count column that represents the number of people who labelled the tweets.
- A hate_speech column that represents the number of people who tagged a certain tweet as hate speech.
- An offensive_language column that represents the number of people who tagged a certain tweet as offensive language.
- A neither column that represents the number of people who tagged a certain tweet not offensive language nor hate speech.
- A tweet column that contains the tweet texts.

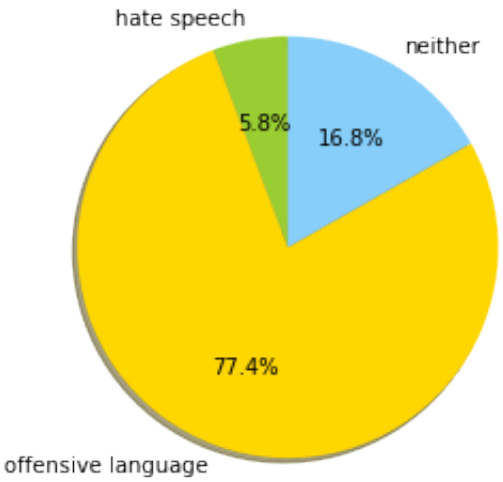


Figure 3.3 – Pie chart for the data-set

Only, 5.8% of the tweets were labelled as hate speech. The majority of the tweets (77.4%) were labelled as offensive language. The rest (16.8%) was labelled as neither.

3.3 First Pipeline

3.3.1 Data Preprocessing

As a starting point, we follow a simple preprocessing pipeline. We remove mentions which are usernames starting with "@", urls which are expressions starting with http and punctuation marks. We also replace any space pattern with white space.
After that, each tweet is split into tokens and lower cased. The tokens are then stemmed using the Porter stemmer.
For example, the following tweet:

"!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. amp; as a man you should always take the trash out..."

is transformed into:

'woman', 'shouldn', 't', 'complain', 'about', 'clean', 'up', 'hous', 'amp', 'man', 'should', 'alway', 'take', 'trash', 'out'

3.3.2 Feature engineering

To extract features, we used the bag-of-words model. Each feature is three consecutive words. The model is designed to have the number of maximum features as 10000. We ignore features that are too rare ie. occur less than 5 times in the corpus. We also ignore features frequently occurring with document frequency greater than 0.75. The first tweet is transformed into the following form.

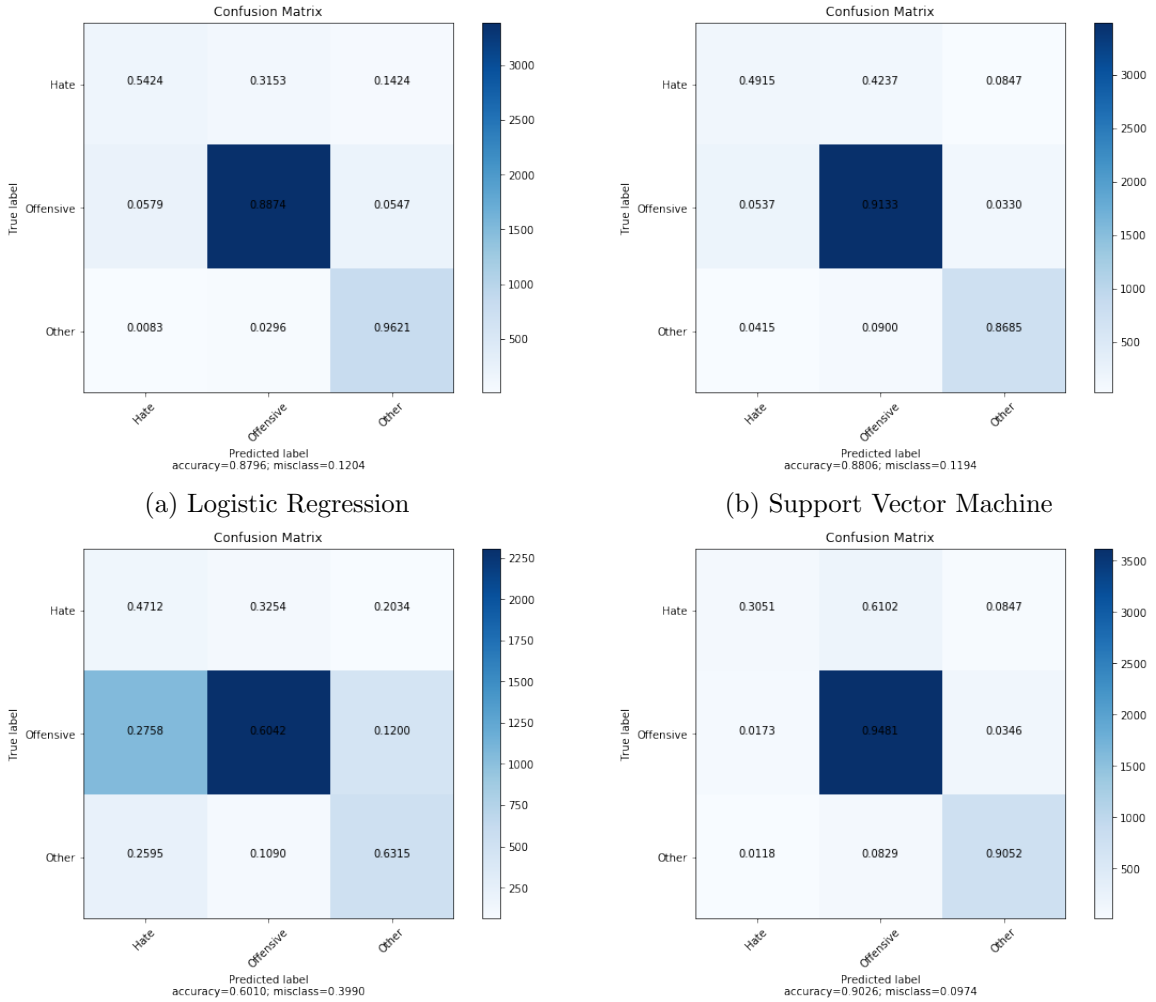
```
woman complain clean hous amp man alway take trash take trash
  (0, 6894)     6.483184225000979
  (0, 1373)     7.28927181674152
  (0, 1293)     7.166669494649188
  (0, 3111)     6.250378762775033
  (0, 130)      4.413498858266507
  (0, 3918)     4.807994935004098
  (0, 109)      5.479558543896869
  (0, 5924)     5.183019017611027
  (0, 6305)     4.094154258492171
  (0, 5939)     8.345324490990834
```

Figure 3.4 – TF-IDF transformation

This figure represents the feature names of the first document. The document matrix is a sparce matrix. It is stored in the form of couples. Each couple contains the index of the feature in the matrix and its tf-idf score.

3.3.3 Model selection and evaluation

After having our corpus transformed into a sparce matrix. We feed them into four different models. We use logistic regression with penalty="l1" and C=0.01, Support Vector Machine with C=0.01 and penalty='l2', Naive Bayes with the assumption that the likelihood of the features is assumed to be Gaussian, and Random Forest with n_estimators = 50, max_features='auto' and criterion = 'gini'. These parameters are determined using grid search algorithm. After training the models, we test them and obtain the following confusion matrices.



(a) Logistic Regression

(b) Support Vector Machine

(c) Naive Bayes

(d) Random Forest

Figure 3.5 – Confusion matrices

Logistic regression, SVM and Random forest perform significantly better than Naive Bayes. Both logistic regression and SVM have an accuracy of 88% and random forest has an accuracy of 90% whereas, Naive Bayes' accuracy only reaches 60%. However, all classifiers perform badly when classifying hate speech. We can look further at the classification report for each classifier.

	precision	recall	f1-score	support
0	0.41	0.54	0.47	295
1	0.97	0.89	0.93	3818
2	0.76	0.96	0.85	844
avg / total	0.90	0.88	0.89	4957

(a) Logistic Regression

	precision	recall	f1-score	support
0	0.38	0.49	0.43	295
1	0.95	0.91	0.93	3818
2	0.83	0.87	0.85	844
avg / total	0.89	0.88	0.89	4957

(b) Support Vector Machine

	precision	recall	f1-score	support
0	0.10	0.47	0.16	295
1	0.92	0.60	0.73	3818
2	0.51	0.63	0.56	844
avg / total	0.80	0.60	0.67	4957

(c) Naive Bayes

	precision	recall	f1-score	support
0	0.54	0.31	0.39	295
1	0.94	0.95	0.94	3818
2	0.83	0.91	0.87	844
avg / total	0.89	0.90	0.90	4957

(d) Random Forest

Figure 3.6 – Classification reports

Precision, recall and consequently f1-score are significantly lower for the ”hate speech” class than they are for the ”offensive speech” and ”neither” class in all of the classifiers. While those metrics easily reach 90% for the offensive speech and neither classes, they do not exceed 55% for the hate speech class.
These results were expected from the start. In fact, when we look at the pie chart of the data distribution we find that only 5.8% of the data are hate speech and the overwhelming majority is offensive speech. Hence, it is expected that the model would perform the worst when classifying hate speech.

3.3.4 Conclusion

In this section, we tried a straightforward pipeline where we cleaned the corpus then transformed it into a matrix using the bag-of-words model. The matrix then is fed into a classifier. We tried four different classifiers: Logistic regression, Support vector machine, Naive Bayes, and Random Forest. Hence, the model consist of a vectorizor and a classifier. This approach gave high accuracy but bad classification for hate speech.
In the next section we will try adding more features and see the performance of the models with that change.

3.4 Second approach: Adding more features

In the previous sections, we considered that the features are stemmed words. This way, we lose the overall sentence structure. The idea is that if two sentences contain similar words, one might be just offensive and the other might indicate hate speech. In this chapter we will change the approach, we will add more features in the hope that it will change the results.

3.4.1 Word based representation

Similarly as the previous section, after removing, tokenizing, and stemming the words, each word is considered a feature. We use the bag-of-words model to transform the documents into a feature matrix.

3.4.2 Part of Speech tagging (POS)

POS tagging is a linguistic technique used in Natural Language Processing. POS tagging basically assigns a tag to each word in a text. This tag is a morphological category and it might be a noun, a verb an adjective etc.[26]
Example:

```
>>> text = word_tokenize("They refuse to permit us to obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Figure 3.7 – Example of POS tagging

We notice in this example that ”refuse” is tagged as a verb then ”refuse” is tagged as a noun. POS filtering permits us to know the different morphological categories of the same word in a sentence.

To add tags as features, we changed the preprocessing technique. In fact, now we remove URLs, mentions and punctuation marks from the documents then tokenize them. We do not stem the words nor remove stop words. When the documents are preprocessed, we apply POS tagging to get tuples containing a word and it's corresponding tag.
The tuples are now considered features. We transform each document (a document here is a list of tuples that contain words and their tags) into a feature matrix using the bag-of-words model. We ignore features that occur less than 5 times in the corpus and features that have term frequency greater than 0.75.

3.4.3 Sentiment analysis

Sentiment analysis is a subfield of Natural Language Processing that is used for opinion mining. Here, we use the VADER Sentiment Intensity analyser to get polarity scores. VADER (Valence Aware Dictionary for sEntiment Reasoning) takes a tweet as an input then gives a polarity score as an output. Polarity score is a matrix with 4 rows and two columns. The row are: Positive, Negative, Neutral and Compound, and their corresponding values. [27]
VADER is a lexical approach for sentiment analysis. It look at the sentiment category or score of each word in the sentence and decide what the sentiment category or score of the whole sentence is. Emotion intensity for each word is measured on a scale from -4 to +4, , where -4 is the most negative and +4 is the most positive. The midpoint 0 represents a neutral sentiment. For example “horrible” gets mapped to to -2.5, ”okay” gets mapped to 0.9, and the emoticons “/-:” get mapped to -1.3. To construct VADER's

dictionary, a number of human raters rated manually words and averaged their ratings for each word. VADER sentiment analysis returns a sentiment score in the range -1 to 1, from most negative to most positive. To calculate this score, we sum the scores of each word of the sentence then apply Hutto's normalisation.

$$\text{Hutto} = \frac{x}{\sqrt{x^2 + \alpha}}$$

where x is the sum of the sentiment scores of each word of the sentence and α is a normalization parameter that we set to 15. [27]

3.4.4 Other features

In addition to POS tagging and Sentiment analysis, we decided to add other features:

3.4.4.1 The Flesch/Flesch-Kincaid readability tests

The Flesch/Flesch–Kincaid readability tests are tests designed to indicate the difficulty of comprehending an English text upon reading it.[28]

In the Flesch reading-ease test, texts that are easier to read have higher scores while texts that are more difficult to read have lower scores. We calculate the score using the formula

$$\text{FRE} = 206.835 - 1.015\frac{\text{total words}}{\text{total sentences}} - 84.6\frac{\text{total syllables}}{\text{total words}} \text{[29]}$$

Scores correspond various degrees of difficulty.

Score	School level	Notes
100.00-90.00	5th grade	Very easy to read. Easily understood by an average 11-year-old student.
90.0–80.0	6th grade	Easy to read. Conversational English for consumers.
80.0–70.0	7th grade	Fairly easy to read.
70.0–60.0	8th & 9th grade	Plain English. Easily understood by 13- to 15-year-old students.
60.0–50.0	10th to 12th grade	Fairly difficult to read.
50.0–30.0	College	Difficult to read.
30.0–0.0	College graduate	Very difficult to read. Best understood by university graduates.

Figure 3.8 – Degrees of difficulty for Flesch reading-ease test

The Flesch-Kincaid grade level is used extensively in the field of education. It translates the 0–100 score to a United States grade level, making it easier for teachers, parents, librarians, and others to judge the level of readability of various books and texts. It can also mean the number of years of education generally required to understand this text, relevant when the formula results in a number greater than 10. For example, a score of 8.2 would indicate that the text is expected to be understandable by an average student in year 8 in the United Kingdom or an 8th grade student in the United States. [30] To calculate Flesch–Kincaid grade level of a text we use the following formula

$$\text{FK} = 0.39\frac{\text{total words}}{\text{total sentences}} + 11.8\frac{\text{total syllables}}{\text{total words}} - 15.59\text{[30]}$$

3.4.4.2 Content based features

We also calculate some characteristics for documents and add them as features. We calculate the number of characters in words of a document, the total number of characters of a document, the number of terms in a document, the number of words in a document, and the number of unique terms.

3.4.4.3 Twitter objects count

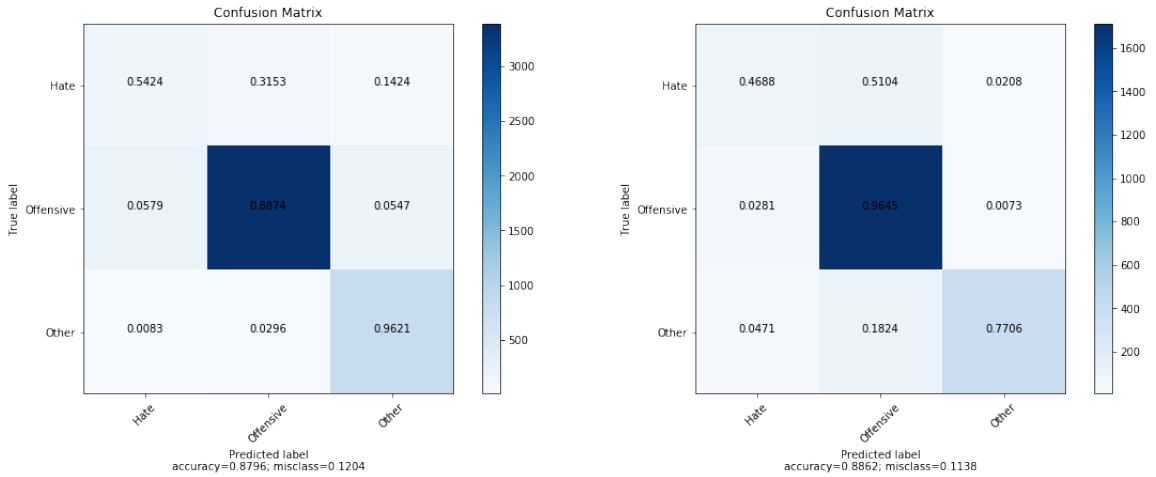
We use the frequency three types of objects in the tweets: URLs, Mentions, and Hashtags. We detect those objects using regular expressions then count their number for each tweet.

3.4.5 Feature Matrix

The final feature matrix is composed of all the features previously mentioned. We join the word feature matrix, the POS feature matrix, and the vectors calculated with other features. We obtain then our final feature matrix that is ready to be fed into a model.

3.4.6 Model selection and evaluation

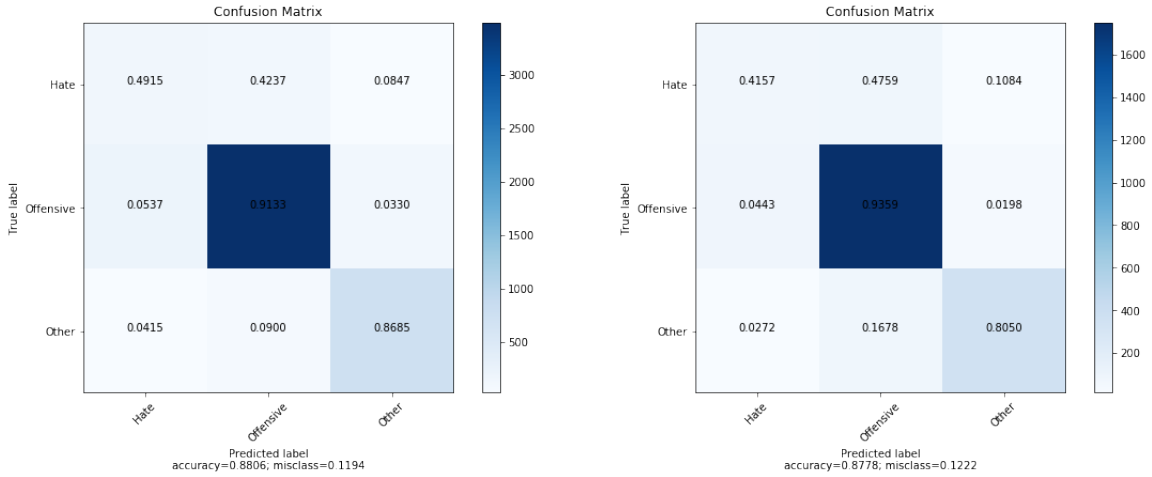
Now that we have our feature matrix ready, we feed into three different models. We use logistic regression with penalty="l1" and C=0.01, Support Vector Machine with C=0.01, penalty='l2'and loss='squared_hinge', and Random Forest with n_estimators = 50, max_features='auto' and criterion = 'gini'. These parameters are determined using grid search algorithm. After training the models, we test them and obtain the following confusion matrices.



(a) With word based representation (b) With multi feature representation

Figure 3.9 – Confusion matrices for Logistic regression

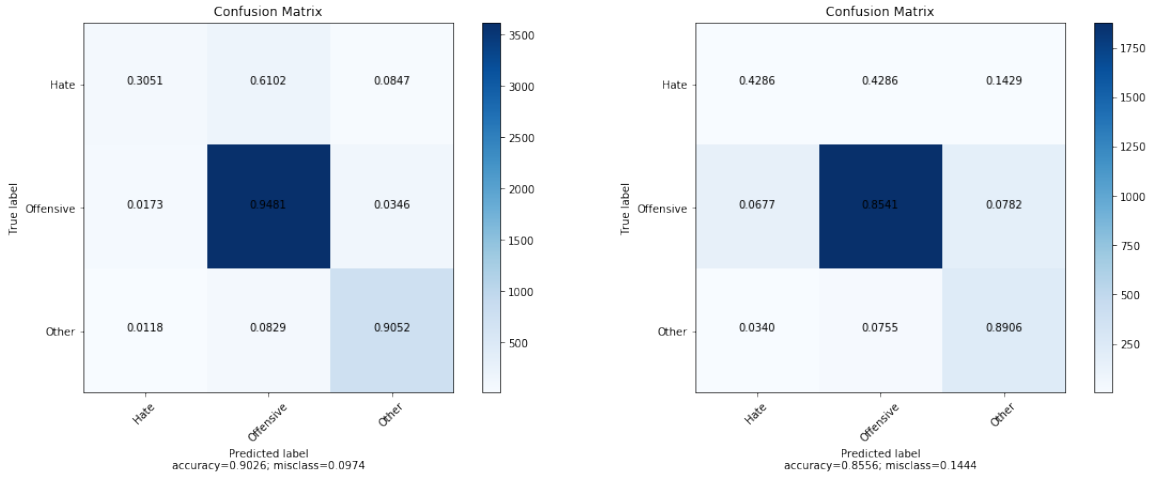
With logistic regression, the accuracy stayed the same with adding additional features at 88%. The model got better at classifying offensive speech but worse at classifying hate speech and neither.



(a) With word based representation (b) With multi feature representation

Figure 3.10 – Confusion matrices for Support vector machine

With Support Vector Machine, the accuracy stayed the same with adding additional features at 88%. There is no significant difference between the two models' performances.



(a) With word based representation (b) With multi feature representation

Figure 3.11 – Confusion matrices for Random Forest

With Random forest, the accuracy decreased from 90% to 86%. The model got better at classifying hate speech but got worse at classifying offensive speech and neither.

3.4.7 Conclusion

In this section we decided to add different features to the feature matrix. We added POS tags, Sentiment analysis polarity scores, Flesch/Flesch–Kincaid readability test, twitter object counts and other features. This approach added complexity to the model but did not change significantly the performance of the model. We still have the same problem as the previous section, the model does not perform well when trying to classify hate speech.

We suspect that those results are due to the unbalanced data-set. In order to ameliorate the results, we introduce a data rebalancing technique and change the model architecture.

3.5 Third Approach: Data rebalancing

In the previous section, we suspected that the bad classification is due to the unbalanced data-set. In this section we introduce the approach we adopted to solve this problem.
We first separate the three classes. We divide the offensive speech class into 10 sections and the neither class into 2 sections. The purpose of this division is to obtain 20 mini data-sets. Each of them contain a hate speech section, an offensive speech section and a neither section.

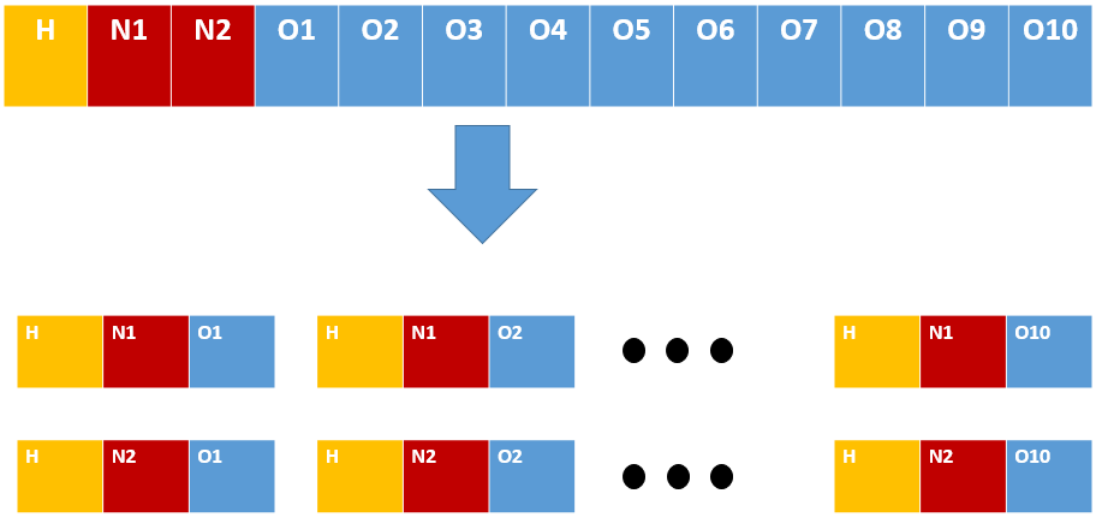


Figure 3.12 – Data sectioning

This figure represents how we sectioned our data-set to obtain 20 new data-sets.
H refers to hate speech, $N_i, i = 1, 2$ refers to the i^{th} section of the Neither class, and $O_j, j = 1..10$ refers to the j^{th} section of the Offensive speech class.
Each data-set has the following composition

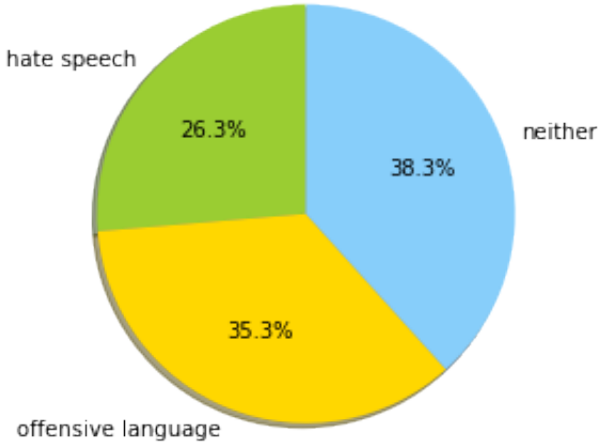


Figure 3.13 – Pie chart of a new data-set

Each data-set now is balanced. It contains 26.3% of hate speech, 35.3% of offensive language and 38.3% of neither.
Now that we have 20 data-sets, we follow a pipeline.

3.5.1 Data preprocessing

For each mini data-set we follow the same preprocessing techniques as the last section. We remove mentions which are usernames starting with "@", urls which are expressions starting with http and punctuation marks. We also replace any space pattern with white space.
After that, each tweet is split into tokens and lower cased. The tokens are then stemmed using the Porter stemmer.

3.5.2 Feature engineering

For each mini data-set we also follow the same factorization technique as last section. We use the Bag-of-words model where we remove words that occur less than 3 times in the corpus (the corpus now refers to the training set of each mini data-set). We also remove words with term frequency higher than 0.75.
Now that we have each of our 20 data-sets transformed into a matrix, we feel them into a model.

3.5.3 Modeling and evaluation

Now that we have the 20 transformed data-sets ready. We feed them into a model with a new architecture. The model is now composed of 20 verctorizors and 20 classifiers. Each data-set will be fed into a vectorizor then a classifier. We will then calculate the f1-score for each classifier. To classify a new data point, we will get it's classification result for

each classifier. The final result will be the majority vote of the classifiers appended with their f1-score.

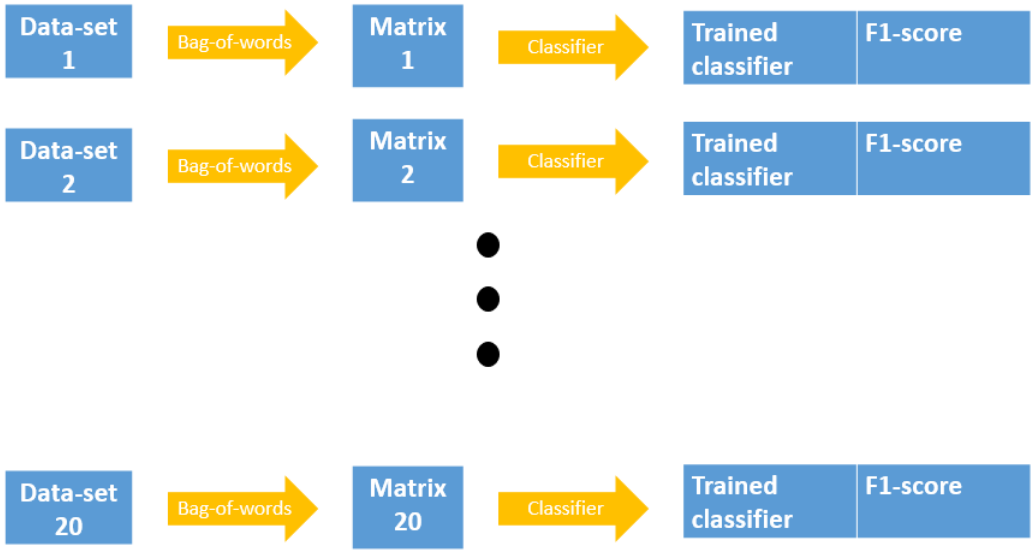


Figure 3.14 – Training the model

This figure represents how we train the model. We split the data into a training set and a testing set: the training set is 80% of the data and the testing set is 20% of the data. The training set is then used to train the vectorizor and the classifier. The testing set is then introduced in order to test the model and calculate the f1-score. F1-score is a vector of $\mathbb{R}^3$. Each row represents the f1-score for each class.

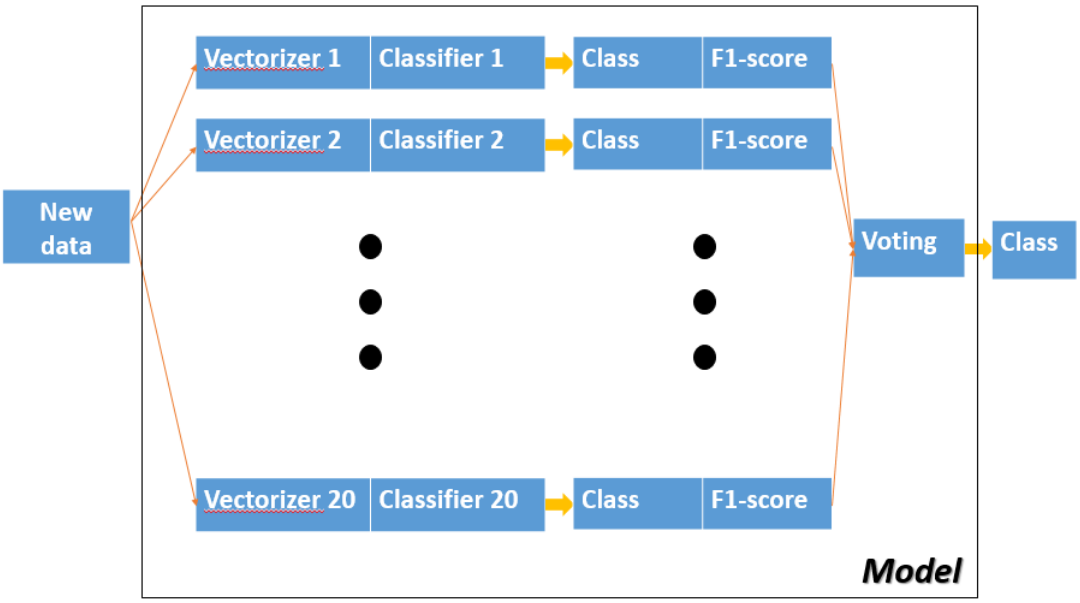


Figure 3.15 – Testing the model

Now that the f1-score is calculated for each classifier, we can implement the model.

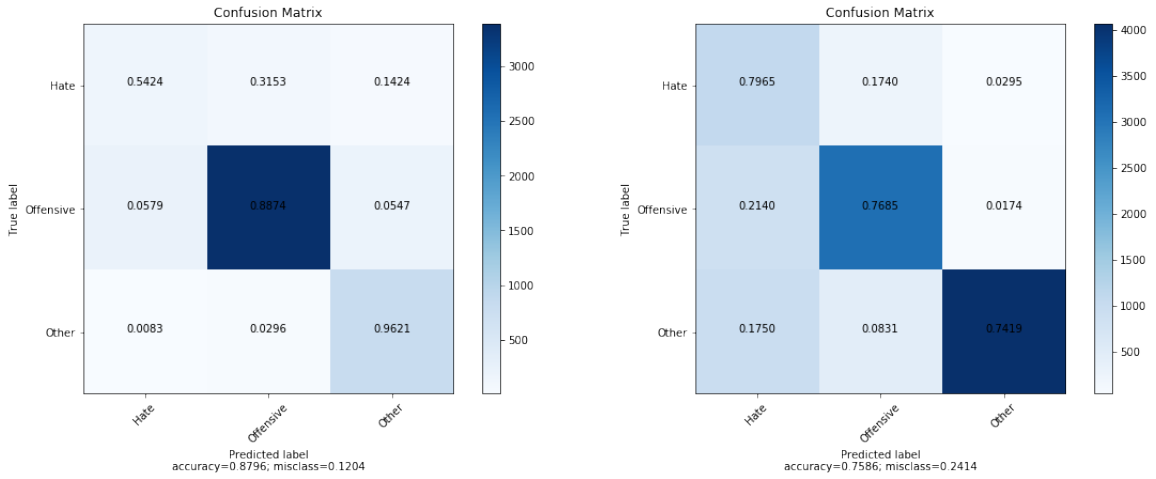
When a new data point arrives, it is transformed into a vector then classified with 20 different classifiers. The result of each classifier along with it's f1-score are then the input of a voting system that will determine the final class.
Let H be the number of classifiers that decided the new data point is actually hate speech, O be the number of classifiers that decided the new data point is offensive speech,and N be the number of classifiers that decided the new data point is neither.

$$\text{Vote } = \max(\sum_{n=1}^{H} \text{f1-score}[0]_{n\in\mathbb{H}}, \sum_{n=1}^{O} \text{f1-score}[1]_{n\in\mathbb{O}}, \sum_{n=1}^{N} \text{f1-score}[2]_{n\in\mathbb{N}})$$

Where $\mathbb{H}$ is the set of the classifiers that decided that the new data point is hate speech. $\mathbb{O}$ is the set of the classifiers that decided that the new data point is offensive speech. $\mathbb{N}$ is the set of the classifiers that decided that the new data point is neither.

If the vote is equal to $\sum_{n=1}^{H} \text{f1-score}[0]_{n\in\mathbb{H}}$ then the final class will be hate speech. If max is equal to $\sum_{n=1}^{O} \text{f1-score}[1]_{n\in\mathbb{O}}$ then the final class will be offensive speech. If max is equal to $\sum_{n=1}^{N} \text{f1-score}[2]_{n\in\mathbb{N}}$ then the final class will be neither.

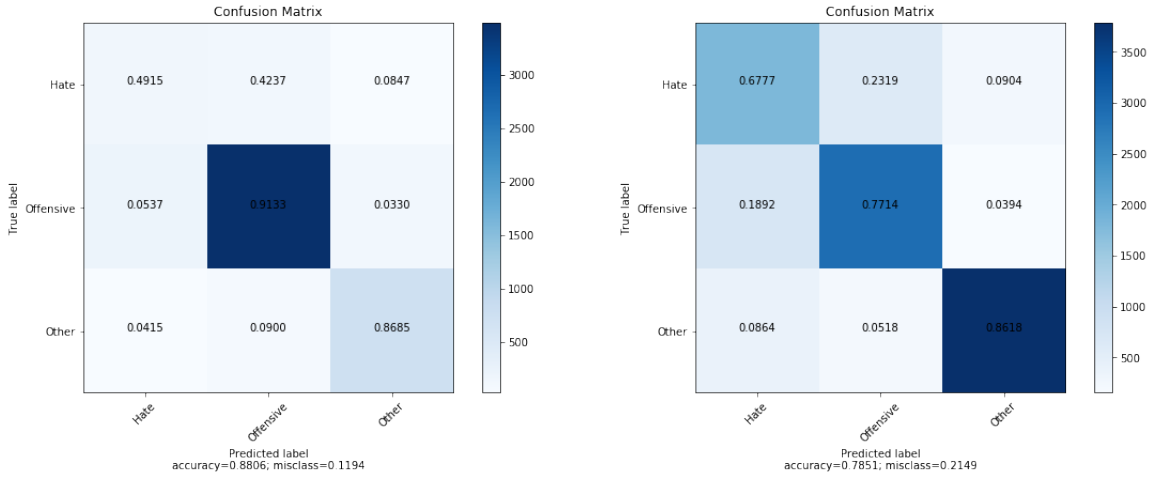
We tried this architecture with the same three classifiers from the previous sections: Logistic regression, Support vector machine, and Random Forest.



(a) With unbalanced data (b) With rebalanced data

Figure 3.16 – Confusion matrices for Logistic regression

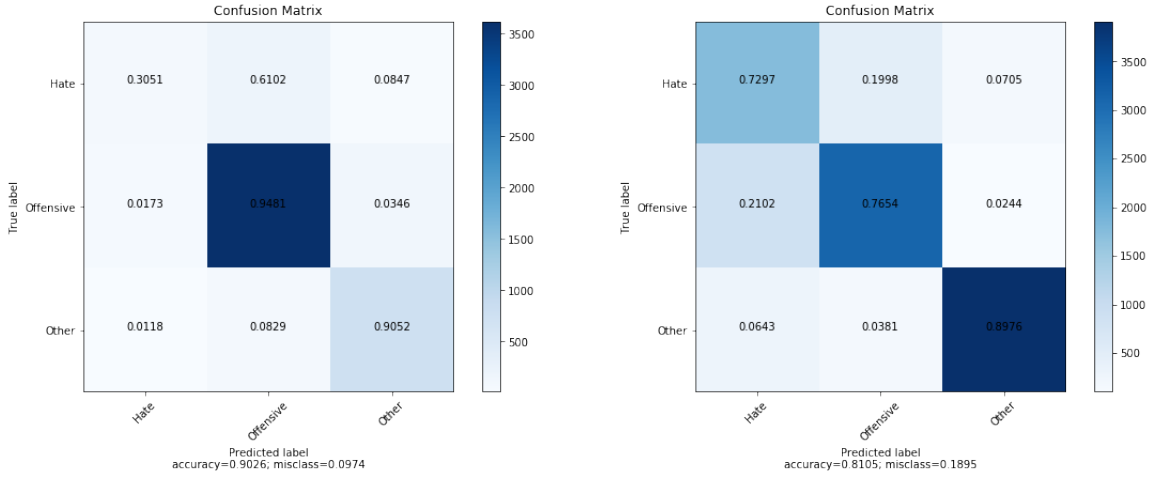
We notice that after applying the data rebalancing technique, the accuracy of the logistic regression decreased from 88% to 76%. However, this result does not mean that the old model is performing better. If we take a look at the confusion matrices of both models we notice that the new model classifies hate speech better. The trade off is that now it is biased more to misclassify offensive speech and neither as hate speech.



(a) With unbalanced data (b) With rebalanced data

Figure 3.17 – Confusion matrices for Support vector machine

For the Support vector machine classifier, the accuracy dropped from 88% to 79%. However, similarly as logistic regression, the classification of hate speech got better while the classification of offensive speech got worse.



(a) With unbalanced data (b) With rebalanced data

Figure 3.18 – Confusion matrices for Random forest

With the Random forest classifier, the accuracy dropped from 90% to 81%. However, correctly classifying hate speech increased significantly from 31% to 73%. Misclassification of offensive speech increased from 5% to 23%. Similarly to the previous classifiers, the model is now biased toward misclassifying documents into hate hate speech.

3.5.4 Conclusion

In this section, we introduced a technique for data rebalancing. It consequently modified the structure of the model used. We tried 3 different classifiers. They performed

better than the simple model when classifying hate speech. However, the accuracy decreased. In the next section we will try to modify the vectorization technique and see if it gives better results.

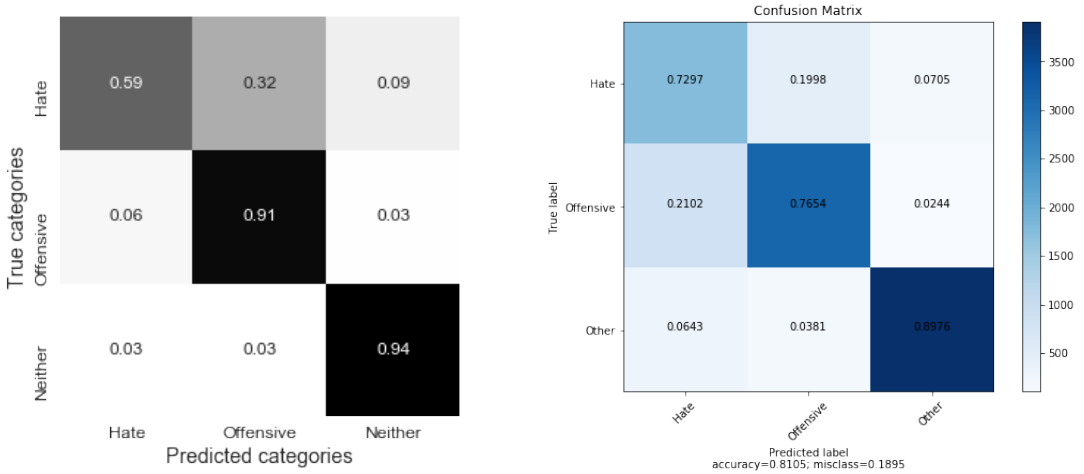
3.6 Final model

After applying the data rebalancing technique and for each of the 20 steps, we choose the best classifier among logistic regression, SVM and random forest. Here, the best classifier is the one with the highest accuracy.
Once we calculate the individual classifiers' accuracies and compare them we get

```
Classifier 0 is Random Forest
Classifier 1 is SVM
Classifier 2 is Random Forest
Classifier 3 is Random Forest
Classifier 4 is Random Forest
Classifier 5 is Random Forest
Classifier 6 is Random Forest
Classifier 7 is Random Forest
Classifier 8 is Random Forest
Classifier 9 is Random Forest
Classifier 10 is Random Forest
Classifier 11 is Random Forest
Classifier 12 is Random Forest
Classifier 13 is Random Forest
Classifier 14 is Random Forest
Classifier 15 is Random Forest
Classifier 16 is Random Forest
Classifier 17 is Random Forest
Classifier 18 is Random Forest
Classifier 19 is Random Forest
```

Figure 3.19 – Chosen classifiers

We conclude that Random forest outperforms the other classifiers. Hence, we keep the final model gotten from the previous section.



(a) Original results [25] (b) Final model results

Figure 3.20 – Confusion matrices

In the original study, they applied a pipeline that finished with the logistic regression model. Their model is biased to classify tweets as less hateful than what the human coders decided. With data rebalancing to create a meta model using the random forest classifier, the model performs better when classifying hate speech. It also is biased to classify tweets as more hateful than what the human coders decided.

3.7 Conclusion

In this chapter we tried 3 different approaches to obtain the final model. We tried a simple pipeline where we implemented a bag-of-words model to vectorize words and four classifiers. The second approach was to add more features for the model. The third approach was to introduce a data rebalancing technique that changed the architecture of the model.

Perspectives

In this project, we touched on machine learning algorithms. For the rest of the internship, it would be interesting to tackle deep learning approaches for both feature engineering and classification. This will be done using a different data-set. In fact, we will work with comments in the Arabic language, specifically the Tunisian dialect.

Chapter 4

Front-end part: Plugin development

In the previous chapter we built a model using 20 vectorizors, 20 classifiers and a voting system. The trained model is now ready to be put in an application. The proposed idea is to develop a browser plugin that detects offensive and hate speech in social media. Since the data-set we have is extracted from Twitter, we decided to choose Twitter as our social media target. We will start first with describing the product to develop using an IEEE Requirement Specification template.

4.1 Requirements Specification

4.1.1 Introduction

4.1.1.1 Purpose

The purpose of this document is to present a detailed description of a browser plugin that classifies Twitter pages based on the levels of offensiveness and hate speech they contain.

4.1.1.2 Document Conventions

This Document was created based on the IEEE template for System Requirement Specification Documents.

4.1.1.3 Intended Audience and Reading Suggestions

This document is written for the Biware/U2S research team as well as the potential users of the software: social media users.

4.1.1.4 Product Scope

The final product will be activated when a social media page is visited. It will be able to detect and classify offensive language and hate speech and give the page a score for the level of offensiveness it has. The user will be able then to choose whether to block the page or not based on its score.

4.1.1.5 References

- Twitter website:
https//twitter.com/
- IEEE Template for System Requirement Specification Documents:
https://goo.gl/nsUFwy

4.1.2 Overall Description

4.1.2.1 Product Perspective

This product is developed for every social media user who wished to avoid offensive or hate speech while using social media.
As soon as the user visits a social media page, the plugin will be activated. It will:
• Detect offensive and hate speech.
• Give a score to the page based on the degree of offensiveness and hate it contains.
• Give the user the option either to block his access to the page or to continue visiting it.

4.1.2.2 Product Functions

This product will:
• Collect comments on a social media page.
• Classify comments into hate speech, offensive speech or neither.
• Score the page based on the classification results.
• Block the page if the user chooses to.

4.1.2.3 User Classes and Characteristics

• Typical social media users who want to avoid offensive and hate speech.
• Parents who want to control the level of exposure their kids get to offensive and hate speech.

4.1.2.4 Operating Environment

• Windows 2000
• Windows XP
• Windows Vista
• Windows 7
• Windows 8
• Windows 10

4.1.2.5 Design and Implementation Constraints

This product is developed using HTML, CSS and JavaScript for the visual plugin part of the application and Python for modelling and classification.

4.1.2.6 Assumptions and Dependencies

This product is a browser plugin so it requires Google Chrome to be installed. It also requires a Windows operating system to be installed.

4.1.3 External Interface Requirements

4.1.3.1 User Interfaces

The product's interface will have a score section: it will indicate level 1 to 5 of offensiveness of the page. It will also contain a block button the will take the user to a blank page instead of the page he is visiting.

4.1.3.2 Hardware Interfaces

CPU: Pentium 4 processor or higher
RAM: 128MB or higher

4.1.3.3 Software Interfaces

This product requires Google Chrome to be installed.

4.1.4 System Features

This section demonstrates the plugin's most prominent features and explains how they can be used and the results they will give back to the user.

4.1.4.1 Page scoring

This plugin will score the pages of a social media networks.

Description and Priority
The plugin will collect comments from social media pages and it will run then through the python classification pipeline. It will then calculate the score of the page.

Stimulus/Response Sequences
The plugin will be stimulated when the user visits a social media page.

4.1.4.2 Page blocking

This plugin will change the current page to a blank page.

Description and Priority
The plugin will change the current page to a blank page when the user decided to block

the social media page.

Stimulus/Response Sequences
The plugin will be stimulated when the user clicks on the block button.

4.2 Implementation of the plugin

The front-end of the browser plugin was developed using HTML and JavaScript. First the user needs to upload the application to the extensions section of their chrome browser.

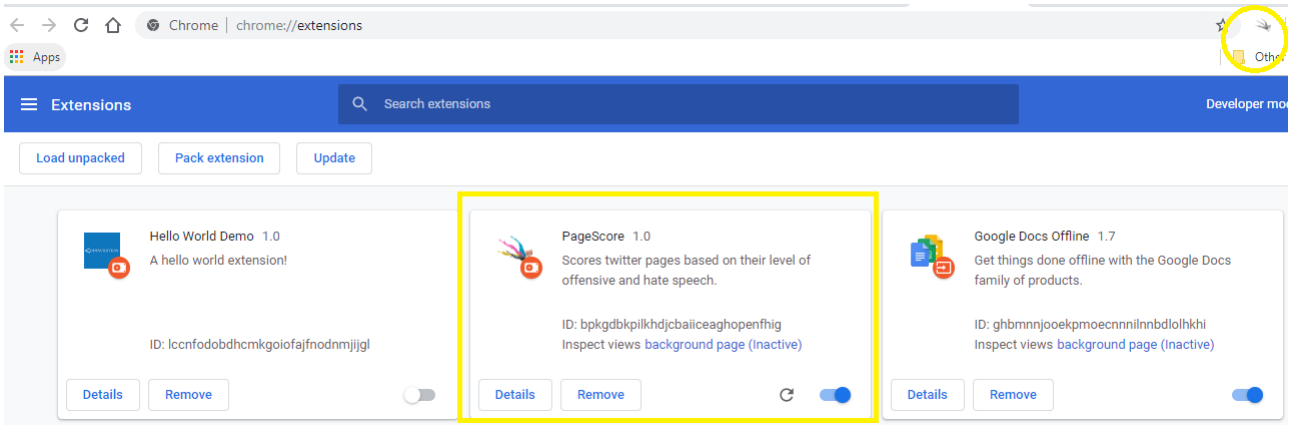


Figure 4.1 – Plugin uploaded

The application is made to run only when the user visits Twitter. Hence, it stays inactive until it is triggered to be activated.

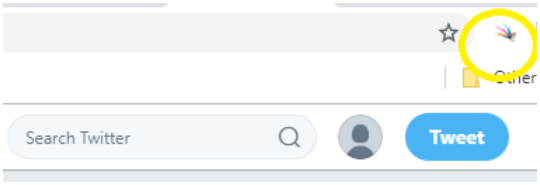


Figure 4.2 – Plugin activated

As we can see in the figure above, once the user visits the Twitter website, the plugin gets activated.

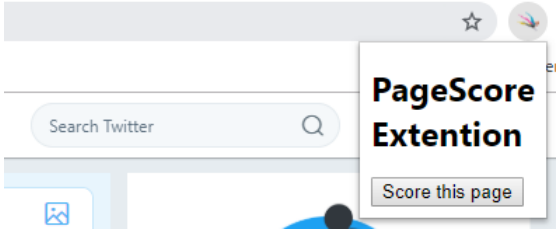


Figure 4.3 – Plugin interface

The interface of the plugin is simple. Once the user clicks on "Score this page", a table will appear having the total number of tweets in that page, the number of offensive tweets, and the number of hate tweets. Based on these numbers, the Twitter page will be given a score from 1 to 5. Based on that score, the user is then given the choice to block the page. Blocking the page means that the browser will redirect the active page to a blank page.

In the background, clicking "Score this page" button will trigger a JavaScript code to get the active page's URL. This URL will be the input of a python script that contains a twitter API. This API downloads tweets using a filter: the URL of the active page. Once we have the tweets, we transform then into a data-set then send them as an input to the trained model. The output of the model is prediction of classes from each tweet. We then count the total number of the tweets, the number of offensive tweets, and the number of hateful tweets. This table is then sent back to the plugin where it will be shown in the front-end.

Problem encountered

The main problem we encountered is how to manage the communication between the JavaScript code and the Python code. We proposed a file based solution: When JavaScript gets the URl it stores it into a file. At the operating system level we put a listener. This listener triggers the activation of the Python code as soon as the file is created. Python's output will also be stored in a file. Another listener is put at the operating system level to trigger the activation of JavaScript's code as soon as the file is created.

As we tried to implement this solution, we discovered that it cannot be done. There is a security measure that prevents web applications from automatically downloading files. For the rest of the internship, we will try to find other solutions.

Perspectives

Since the file based solution did not work, we will try another solution next. We will try to put the outputs for JavaScript's code and Python's code in a server. We will try a client-server architecture and see if it will work.

Conclusion

In this internship, the problem of distinguishing between offensive and hate speech was tackled. We started by presenting the general problem. Then we discussed the techniques used in Natural Language Processing and Classification.
We implemented those techniques using 3 different approaches. First, we implemented a simple pipeline and had a model that contains one vectorizor and one classifier. Second, we added more features to the vectorizor. Third, we implemented a data rebalancing technique that changed the architecture of the model. We presented the results for each approach. Finally, we chose a final model that has 20 vertorizors and 20 classifiers as well as a voting function.
After training the final model, we considered it the back-end part of a browser plugin. We developed the front-end of the plugin separately.

This internship was definitely a big learning experience. I was introduced to text mining and Natural Language Processing as well as web applications development. For the rest of the internship, we will tackle deep learning solutions for vectorization and classification. We will also tackle the problem of working with corpora in the Tunisian dialect.

Appendices

.1 Tools used

In this project we used various tools for implementing the model and developing the browser plugin.

.1.1 Python

Python is high-level, general purpose programming language. It was created Guido van Rossum and released in 1991. Python adopts an object-oriented approach for programming. It is heavily used in the field of Natural Language Processing and many other fields.[31]

.1.2 HTML

HTML (HyperText Markup Language) is the language for describing the structure of Web pages. It gives the possibility to publish structured online documents that contain text, photos, videos, etc..[32]

.1.3 JavaScript

JavaScript is an object-oriented programming language used for web development. It is used to enable interactive web pages. [33]

.1.4 NLTK

NLTK (Natural Language Toolkit) is a platform used to work with human language data. It is a built in package in Python. It offers easy-to-use interfaces, various corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.[34]

.1.5 Scikit learn

Scikit learn is a Python library used for machine learning. It offers many tools for regression, classification, clustering, dimensionality reduction, model selection, and preprocessing. It is widely used and easy to implement. [**?**]

.2 Porter Stemmer Algorithm

.2.1 Notations

Before presenting the Porter Stemmer algorithm we will need few definitions first.
- "A consonant in a word is a letter other than A, E, I, O, U and other than Y preceded by a consonant". A consonant is denoted by c.
- A vowel in a word is a letter other than consonants. It is denoted by v.
- C: a list of consecutive consonants of lenght greater than 0.

- V: a list of consecutive vowels of lenght greater than 0.[11]
By these notations, any word has one of these four forms:

CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V

Table 1 – Possible word forms[11]

Hence, any word can be represented as

$$[C](VC)^m[V]$$

Where the square brackets denote arbitrary presence of their contents.
The rules to removing a suffix will be given in the form

$$(condition)S1 \longrightarrow S2 \text{ [11]}$$

This form means that if a word ends with the suffix S1, the latter is replaced by S2 if the stem before S1 satisfies the given condition.
The condition part may contain the following notations:

Notation	Meaning
*A	The stem ends with A (and similarly for the other letters)
v	The stem contains a vowel
*d	The stem ends with a double consonant
*o	The stem ends cvc, where the second c is not W, X or Y

Table 2 – Notations and their meanings [11]

.2.2 Algorithm

Now that we have our definitions and notations, we can introduce the Porter Stemmer Algorithm. It contains five steps and it is implemented using a switch case[11].

.2.2.1 Step 1

- Step 1 a

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

Table 3 – Step 1 a rules and examples

- Step 1 b

Rule	Example
($m > 0$) EED → EE	feed → $feed$
	agreed → $agree$
(*v*) ED →	plastered → $plaster$
	bled → $bled$
(*v*) ING →	motoring → $motor$
	sing → $sing$

Table 4 – Step 1 b rules and examples 1

If the second or third of the rules in Step 1b is successful, the following is done:

Rule	Example
AT → ATE	conflat(ed) → $conflate$
BL → BLE	troubl(ed) → $trouble$
IZ → IZE	siz(ed) → $size$
(*d and not (*L or *S or *Z)) → $singleletter$	hopp(ing) → hop
	tann(ed) → tan
	fall(ing) → $fall$
	hiss(ing) → $hiss$
	fizz(ed) → $fizz$
(m=1 and *o) → E	fail(ing) → $fail$
	fil(ing) → $file$

Table 5 – Step 1 b rules and examples 2

- Step 1 c

Rule	Example
(*v*) Y → I	happy → $happi$
	sky → sky

Table 6 – Step 1 c rules and examples

.2.2.2 Step 2

Rule	Example
$(m > 0)$ ATIONAL $\rightarrow$ *ATE*	relational $\rightarrow$ *relate*
$(m > 0)$ TIONAL $\rightarrow$ *TION*	conditional $\rightarrow$ *condition*
	rational $\rightarrow$ *rational*
$(m > 0)$ ENCI $\rightarrow$ *ENCE*	valenci $\rightarrow$ *valence*
$(m > 0)$ ANCI $\rightarrow$ *ANCE*	hesitanci $\rightarrow$ *hesitance*
$(m > 0)$ IZER $\rightarrow$ *IZE*	digitizer $\rightarrow$ *digitize*
$(m > 0)$ ABLI $\rightarrow$ *ABLE*	conformabli $\rightarrow$ *conformable*
$(m > 0)$ ALLI $\rightarrow$ *AL*	radicalli $\rightarrow$ *radical*
$(m > 0)$ ENTLI $\rightarrow$ *ENT*	differentli $\rightarrow$ *different*
$(m > 0)$ ELI $\rightarrow$ *E*	vileli $\rightarrow$ *vile*
$(m > 0)$ OUSLI $\rightarrow$ *OUS*	analogousli $\rightarrow$ *analogous*
$(m > 0)$ IZATION $\rightarrow$ *IZE*	vietnamization $\rightarrow$ *vietnamize*
$(m > 0)$ ATION $\rightarrow$ *ATE*	predication $\rightarrow$ *predicate*
$(m > 0)$ ATOR $\rightarrow$ *ATE*	operator $\rightarrow$ *operate*
$(m > 0)$ ALISM $\rightarrow$ *AL*	feudalism $\rightarrow$ *feudal*
$(m > 0)$ IVENESS $\rightarrow$ *IVE*	decisiveness $\rightarrow$ *decisive*
$(m > 0)$ FULNESS $\rightarrow$ *FUL*	hopefulness $\rightarrow$ *hopeful*
$(m > 0)$ OUSNESS $\rightarrow$ *OUS*	callousness $\rightarrow$ *callous*
$(m > 0)$ ALITI $\rightarrow$ *AL*	formaliti $\rightarrow$ *formal*
$(m > 0)$ IVITI $\rightarrow$ *IVE*	sensitiviti $\rightarrow$ *sensitive*
$(m > 0)$ BILITI $\rightarrow$ *BLE*	sensibiliti $\rightarrow$ *sensible*

Table 7 – Step 2 rules and examples

.2.2.3 Step 3

Rule	Example
$(m > 0)$ ICATE $\rightarrow$ *IC*	triplicate $\rightarrow$ *triplic*
$(m > 0)$ ATIVE $\rightarrow$	formative $\rightarrow$ *form*
$(m > 0)$ ALIZE $\rightarrow$ *AL*	formalize $\rightarrow$ *formal*
$(m > 0)$ ICITI $\rightarrow$ *IC*	electriciti $\rightarrow$ *electric*
$(m > 0)$ ICAL $\rightarrow$ *IC*	electrical $\rightarrow$ *electric*
$(m > 0)$ FUL $\rightarrow$	hopeful $\rightarrow$ *hope*
$(m > 0)$ NESS $\rightarrow$	goodness $\rightarrow$ *good*

Table 8 – Step 3 rules and examples

.2.2.4 Step 4

Rule	Example
$(m > 1)$ AL $\rightarrow$	revival $\rightarrow$ *reviv*
$(m > 1)$ ANCE $\rightarrow$	allowance $\rightarrow$ *allow*
$(m > 1)$ ENCE $\rightarrow$	inference $\rightarrow$ $infer$
$(m > 1)$ ER $\rightarrow$	airliner $\rightarrow$ *airlin*
$(m > 1)$ IC $\rightarrow$	gyroscopic $\rightarrow$ *gyroscop*
$(m > 1)$ ABLE $\rightarrow$	adjustable $\rightarrow$ *adjust*
$(m > 1)$ IBLE $\rightarrow$	defensible $\rightarrow$ $defens$
$(m > 1)$ ANT $\rightarrow$	irritant $\rightarrow$ *irrit*
$(m > 1)$ EMENT $\rightarrow$	replacement $\rightarrow$ *replac*
$(m > 1)$ MENT $\rightarrow$	adjustment $\rightarrow$ *adjust*
$(m > 1)$ ENT $\rightarrow$	dependent $\rightarrow$ *depend*
$((m > 1) and (*Sor * T))$ ION $\rightarrow$	adoption $\rightarrow$ *adopt*
$(m > 1)$ OU $\rightarrow$	homologou $\rightarrow$ *homolog*
$(m > 1)$ ISM $\rightarrow$	communism $\rightarrow$ *commun*
$(m > 1)$ ATE $\rightarrow$	activate $\rightarrow$ *activ*
$(m > 1)$ ITI $\rightarrow$	angulariti $\rightarrow$ *angular*
$(m > 1)$ OUS $\rightarrow$	homologous $\rightarrow$ *homolog*
$(m > 1)$ IVE $\rightarrow$	effective $\rightarrow$ $effect$
$(m > 1)$ IZE $\rightarrow$	bowdlerize $\rightarrow$ *bowdler*

Table 9 – Step 4 rules and examples

.2.2.5 Step 5

Step 5 a

Rule	Example
$(m > 1)$ E $\rightarrow$	probate $\rightarrow$ *probat*
	rate $\rightarrow$ *rate*
(m=1 and not *o) E $\rightarrow$	cease $\rightarrow$ *ceas*

Table 10 – Step 5 a rules and examples

Step 5 b

Rule	Example
$(m > 1$ and *d and *L$) \rightarrow$ *singleletter*	controll $\rightarrow$ *control*
	roll $\rightarrow$ *roll*

Table 11 – Step 5 b rules and examples

Bibliography

[1] Cambridge dictionary official website, *dictionary.cambridge.org*

[2] Biware Consulting official website, *http://biware-consulting.com/*

[3] ENIT's official website, *http://www.enit.rnu.tn*

[4] U2S's official website, *http://www.u2s.enit.rnu.tn*

[5] Ying Chen, Yilu Zhou, Sencun Zhu, Heng Xu, *Detecting Offensive Language in Social Media to Protect Adolescent Online Safety*
https://faculty.ist.psu.edu/xu/papers/Chen_etal_SocialCom_2012.pdf

[6] Chiraz Ben Abdelkader, *Lesson # 1: Introduction to Data & Text Mining*, TICV Masters, Text Mining Course, October 28, 2018

[7] Jan Goyvaerts, *Regular Expressions: The Complete Tutorial*, 2006, 2007
http://www.regular-expressions.info/print.html

[8] Natual Language Toolkit library's official website
https://www.nltk.org/api/nltk.tokenize.html

[9] Stanford university, *Stemming and Lemmatization*
https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[10] Natual Language Toolkit library's official website
https://www.nltk.org/api/nltk.stem.html

[11] Martin F. Porter, *An algorithm for suffix stripping*, Program, 14 no. 3, pp 130-137, July 1980
https://pdfs.semanticscholar.org/ca20/32154c90c85e3aac3ece5d94fd8e6cad71ce.pdf

[12] Cambridge University, *Scoring, term weighting and the vector space model*, Cambridge University Press, April 1, 2009
https://nlp.stanford.edu/IR-book/pdf/06vect.pdf

[13] Nils J. Nilsson, *INTRODUCTION TO MACHINE LEARNING*, Robotics Laboratory, Department of Computer Science, Stanford University, November 3, 1998
https://ai.stanford.edu/ nilsson/MLBOOK.pdf

[14] Azmi Makhlouf, *Machine Learning Fundamentals: Probabilistic Aspects (Lecture notes)*, August 26, 2018

[15] Ricco Rakotomalala, *SVM Support Vector Machine*, Lumiere University Lyon
https://eric.univ-lyon2.fr/ ricco/cours/slides/svm.pdf

[16] Scikit learn's official website *https://scikit-learn.org*
https://scikit-learn.org/stable/modules/naive_bayes.html

[17] Scikit learn's official website *https://scikit-learn.org*
https://scikit-learn.org/stable/modules/tree.html

[18] J.R. QUINLAN, *Induction of Decision Trees*, Machine Learning 1: 81-106, 1986
http://hunch.net/ coms-4771/quinlan.pdf

[19] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning*, Second Edition, p.587-588, Springer Series in Statistics.

[20] Nuance's official website
https://www.nuance.com/mobile/mobile-applications/swype/android.html

[21] CleanSpeak's official website, *https://cleanspeak.com*

[22] The smartphone app review's official website
https://thesmartphoneappreview.com/iphone/music-bleeper-iphone-review/

[23] Nofanity's official website, *https://nofanity.com/*

[24] Marina Sokolova, Guy Lapalme, *A systematic analysis of performance measures for classification tasks*, Information Processing and Management 45 (2009) 427–437

[25] Thomas Davidson, Dana Warmsley, Michael Macy, Ingmar Weber, *Automated Hate Speech Detection and the Problem of Offensive Language*, Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017)
https://arxiv.org/pdf/1703.04009.pdf
https://github.com/t-davidson/hate-speech-and-offensive-language/tree/master/data

[26] Natural Language Toolkit, *Categorizing and Tagging Words*
https://www.nltk.org/book/ch05.html

[27] C.J. Hutto, Eric Gilbert, *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*, Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014
http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf

[28] SAISD Social Studies Department, *How to Determine Readability - Flesch/Flesch-Kincaid*
https://www.saisd.net/admin/curric/sstudies/resources/teacher_zone/Literacy/reading/lit_read_fletch.pdf

[29] Rudolf Flesch, *How to Write Plain English*
https://web.archive.org/web/20160712094308/http://www.mang.canterbury.ac.nz/writing_guide/writing/flesch.shtml

[30] Kincaid JP, Fishburne RP Jr, Rogers RL, Chissom BS *Derivation of new readability formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy enlisted personnel* Research Branch Report 8-75, Millington, TN: Naval Technical Training, U. S. Naval Air Station, Memphis, TN, February 1975

[31] Python's official website, *https://www.python.org*

[32] W3's official website, *https://www.w3.org/standards/webdesign/htmlcss*

[33] JavaScript's official website *https://www.javascript.com/*

[34] NLTK's official website, *https://www.nltk.org/*

[35] Scikit-learn's offical website, *https://scikit-learn.org*