Detecting Insults in Social Commentary

via Feature Engineering and Recurrent Neural Network

Zongsheng Wang

Student Number: 13001886

MSc Data Science (with Specialisation in Statistics) Graduate Project

in

Statistical Science Department
University College London

Co-supervised by: Dr. Jinghao Xue and Ms Xiaochen Yang

September 2017

Word Count: 10397

Contents

1 Introduction **7**

1.1 Project Overview . . . 7

1.2 Related Works . . . 8

1.2.1 Offensive Comments Detection . . . 8

1.2.2 Sentiment Analysis . . . 9

1.2.3 Approaches in the Kaggle Challenge . . . 9

2 Data Analysis and Preprocessing **11**

2.1 Exploratory Data Analysis . . . 11

2.1.1 Comment/Sentence Length Analysis . . . 12

2.1.2 Preliminary Sentimental Analysis . . . 13

2.1.3 Potential Noise . . . 14

2.2 Data Preprocessing . . . 14

2.2.1 Format Fixing . . . 15

2.2.2 Word Parsing . . . 15

2.2.3 Offensive Words Correction . . . 15

3 Traditional Feature-based Approach **17**

3.1 Features Overview . . . 17

3.1.1 Lexicon-based Feature . . . 18

3.1.2 Lexical Features . . . 18

3.1.3 Syntactic Features . . . 19

3.2 Feature Extraction and Selection . . . 20

3.3 Evaluation . . . 22

3.3.1 Area under the Receiver Operating Curve . . . 22

3.3.2 F1 Score . . . 23

3.4 Model and Training . . . 23

3.5 Results . . . 23

4 Recurrent Neural Network (RNN) Approach **25**

4.1 Introduction . . . 25
4.1.1 Recurrent Neural Network . . . 25
4.1.2 Long-Short-Term-Memory Network . . . 27
4.2 Word Embedding . . . 31
4.3 Models . . . 31
4.3.1 Baseline Model . . . 32
4.3.2 Bidirectional Word-level Model with Weighted Sum Pooling . . . 33
4.3.3 Hierarchical Sentence-level Model with Probability Pooling . . . 36
4.4 Visualisation . . . 38
4.4.1 Word Representations . . . 38
4.4.2 Weighted Sum Pooling . . . 40
4.5 Experiment . . . 41
4.6 Results and Comparison . . . 41

5 Ensemble **45**
5.1 Simple Average Ensemble . . . 46
5.2 Weighted Average Ensemble . . . 46
5.3 Stacking . . . 47

6 Conclusion and Discussion **49**
6.1 Limitations . . . 50
6.1.1 Dataset . . . 50
6.1.2 Test Set . . . 50
6.1.3 Uninterpretability of RNN . . . 51
6.1.4 Hyperparameter Tuning of RNN . . . 51
6.2 Further Works . . . 51
6.2.1 Different Datasets . . . 51
6.2.2 Different Tasks . . . 51
6.2.3 Different Neural Network Structures . . . 52
6.2.4 Different Ensemble Strategies . . . 52

Bibliography **53**

Abstract

In this project, starting from a dataset provided in a Kaggle challenge, I analyse and preprocess the provided dataset, then extract features and construct different traditional feature-based models to detect insulting comments. I also introduce the Recurrent Neural Network (RNN) on insulting comment detection task, and use different ensemble strategies to further improve the prediction accuracy. Finally I compare the performance of all constructed single models, ensemble strategies and the approach proposed by the winner of the Kaggle challenge. The experiment result shows that the RNN models can match with the traditional models and even perform better on F1 score. My best single model (word-level RNN) outperforms the winner's approach by 0.91% on test AUC, and the best ensemble strategy (weighted average ensemble using F1 score as the measurement of performance) exceeds the winner's test AUC by 3.06%.

Chapter 1

Introduction

1.1 Project Overview

With the increasing number of netizens, the size of online communities are growing faster than ever before. Unfortunately the amount of offensive comments is also increasing with the expansion of online communities, which causes unpleasant experience and even leads to negative effects especially toward the adolescents. One report from arstechnica in 2007 stated that ‘up to 80 percent of blogs host offensive content, ranging from "adult language" to pornographic images’[1], and there is no evidence showing this issue has been alleviated during the past decade.

Back to old days, online forum administrators often tried to delete insulting comments manually, which is extremely in-efficient and becomes unfeasible with the increasing number of forum users. With the development of related technology, automatic filtering was used to detect offensive messages and it is still one of the popular offensive content detection methods today. However many of those filters used simple ‘bad-word’ detecting rules which was either too strict or too slack and often can be dodged using the subtlety of languages, since sentences can still be offensive without cursing words, for example using normal or friendly phrases in a sarcastic way.

In 2012, the world’s largest machine learning competition website Kaggle hosted a challenge called ‘Detecting Insults in Social Commentary’, and provided a manually labelled dataset which selected from conversation streams like news commenting sites, message boards, blogs, text messages, etc. This dataset provides us a great foundation to explore the offensive comment detection problem, and the models proposed during the competition can be used as the benchmark of new models.

The goal of this project is to build a classifier through machine learning to detect insulting comments using the above dataset. Both traditional and Recurrent Neural Network (RNN) models are constructed in this project to explore the insulting comment detection. Since the RNN has been widely used in many natural language processing tasks such as language translation and sentiment analysis in recent years, in

this project I spend more attention on constructing RNN models. The traditional feature extraction models are also constructed as the benchmarks of RNN models.

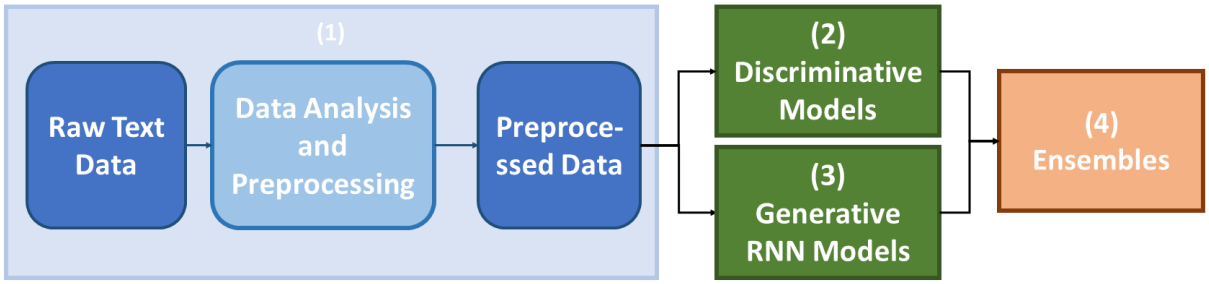


Figure 1.1: Project workflow

The general workflow of this project is shown in figure 1.1. First part is the analysis of dataset, followed by data preprocessing. Next several traditional models with different features are constructed as the benchmarks, which are compared with the later constructed RNN models with different network structures. Finally different ensemble strategies are used to combine the previously constructed single models and further improve the prediction accuracy.

1.2 Related Works

1.2.1 Offensive Comments Detection

Although the insulting text detection task has been studied for decades, there is only limited published papers specifically focused on this topic. The existing approaches mainly consist of either pre-defined offensive text patterns construction or insulting feature engineering:

• **pre-defined offensive text patterns**: This approach uses a pre-defined dictionary of offensive words or language patterns to detect if one text is offensive or not. Its performance mainly depends on the coverage of the pre-defined dictionary, which is relatively rigid and static. Spertu [2] in 1997 proposed a system called Smokey, which included 47 manually defined features based on the syntax and semantics of the training sentence. Smokey were able to categorise 64% of the offensive comments and 98% of the normal comments in a separate small set of 460 messages. However this model often failed when classifying a sentence using friendly phrases sarcastically due to its features limitation.

• **feature engineering**: Gianfortoni et al. [3] and Tsur et al. [4] used two similar feature engineering based approaches containing POS (Part of Speech) and n-grams tf-idf; while Xiang et al. [5] proposed a similar approach but using a large scale Twitter corpus and more sophisticated system architecture. However, although the above feature based models perform well on given datasets, it takes plenty of effort to extract the features manually and the feature-based models often suffer the hazard of over-fitting on different datasets.

In addition, Ying Chen [6] et al. proposed a LSF (Lexical Syntactic Feature) architecture to detect offensive content in social media. In particular, they constructed a user's detection model to predict the user's potential probability of sending out offensive content in a systematic and dynamic way, instead of single offensive comment detection.

1.2.2 Sentiment Analysis

Besides of direct insulting detection, many researches have been done to predict the positive or negative sentimental information in text documents (J. Bollen et al. [7] and D. Davidov et al. [8]). This area of study is similar to insulting comment detection except the datasets are more balanced. Same as insulting comment detection, features like POS and n-grams are often used in this task to capture some specific patterns between positive and negative sentimental information.

Due to the rapid development of deep learning, in recently years models including Recurrent Neural Networks and Convolutional Neural Networks have been widely used in sentiment analysis and achieved breakthrough results. In 2014, Y Kim introduced Convolutional Neural Networks on sentiment analysis and received state-of-art results on certain datasets [9]. Unlikely word-level models, Zhang et al.[10] apply a character-level CNN for text classification and also achieve decent performance. LSTM, a variant of Recurrent Neural Network, is introduced by Tai et al. [11] in text classification in 2015, this approach uses tree-dependency instead of direct word-level sequence, and outperforms many other word-level approaches. Tang et al. [12] and Yang et al. [13] both use hierarchical structure in sentiment classification, by first using a CNN or RNN to get sentence-level vectors and then another RNN to compute the final document-level vectors.

1.2.3 Approaches in the Kaggle Challenge

Since the dataset used in this project is originally provided during one Kaggle machine learning challenge, many candidates during the challenge have created their own models to solve the insulting comment detection task. From the discussion forum on Kaggle, during the competition all candidates manually extracted the features including comment text statistics (length, cursing word count/ratio, etc), tf-idf n-grams and POS , and trained and predicted the data using models such as logistic regression or support vector machine (SVM). A brief summarise of some of the winners' approaches is listed as follows:

• **champion's approach:** The champion's approach contains char and word tf-idf features followed by refined chi-square feature selection, along with the bad word list related features. Logistics regression and support vector regression are chosen as base estimators and stacking ensemble methods to predict final result [14].

• **2nd winner's approach:** The candidate who achieves the 2nd place builds several basic classifiers using logistic regressions and words/stemmed words/POS tags/languge models probabilities as features. Then the results of basic classifiers are combined using stacking ensemble. The random

forest regressor is used to produce the final result [15].

• **3rd winner's approach:** The approach landing the 3rd place in the challenge consists 3 different classifiers: word-nram SVM (n from 1 to 4), character-ngram SVM (n from 4 to 8) and a dictionary-based classifier using a curse words dictionary. The output from each classifier, along with some other features such as the comment length, are fed into a neural network to ensemble the final prediction [16].

• **6th place's approach:** This approach is similar to the winner's model, where four logistic regression models with different word/char n-gram features are ensembled to predict the final results. One novelty in this approach is that the n-gram features are dimension-reduced using principal component analysis (PCA) [17].

Chapter 2

Data Analysis and Preprocessing

Before constructing the models, one important step is to analyse the data first, and then preprocess the data according to what has been discovered. In a supervised classification task like this project, data analysis is a useful tool to identify the distribution difference between each class and correlation within datasets, from either plotting appropriate graphs or implementing statistical test. Data analysis can also help us to find if there exist any deficiencies or potential problems in the datasets due to the nature of datasets, so that we can implement corresponding preprocessing techniques to alleviate the problem. The exploratory data analysis of this project and the details of data proprocessing are described in the following section.

2.1 Exploratory Data Analysis

In total, the Kaggle challenge provided 6594 training comments and 2235 test comments. The ratio of normal comments and insulting comments is around 3:1 in the training set and around 1:1 in the test set. The frequency table of the training and test sets is displayed below:

	Normal	Insulting	Total
Training	4852	1742	6594
Test	1158	1077	2235

Table 2.1: Frequency of normal and insulting comments

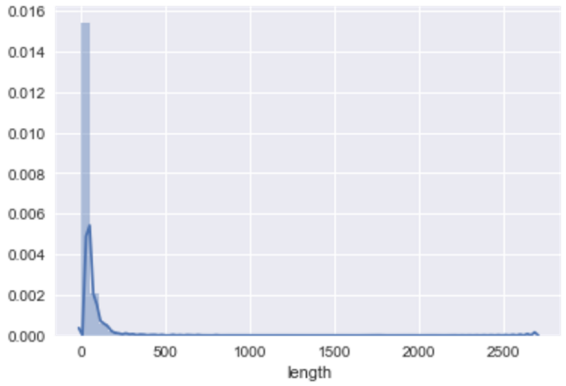
All datasets consist of a label column (0 = normal; 1 = offensive) followed by date and comment content columns. Around half of entries in date column are empty. Majority of comments are written in English, while a small amount of comments contain only emoji or some random and non-sense symbols. Part of training set is shown in Figure 2.1.

Insult	Date	Comment
1	20120618	"You fuck your dad."
0	20120528	"i really don't understand your point.\xa0 It seems that you are mixing apples and or
0		"A\\xc2\\xa0majority of Canadians can and has been wrong before now and will be
0		"listen if you dont wanna get married to a man or a women DONT DO IT. what woul
0	20120619	"C\xe1c b\u1ea1n xu\u1ed1ng \u0111\u01b0\u1eddng bi\u1ec3u t\xecnh 2011 c\xf3
0	20120620	"@SDL OK, but I would hope they'd sign him to a one-year contract to start with. Gi
0	20120503	"Yeah and where are you now?"
1		"shut the fuck up. you and the rest of your faggot friends should be burned at the s
1	20120502	"Either you are fake or extremely stupid...maybe both..."
1	20120620	"That you are an idiot who understands neither taxation nor women's health."
0	20120620	"@jdstorm dont wish him injury but it happened on its OWN and i DOUBT he's injure

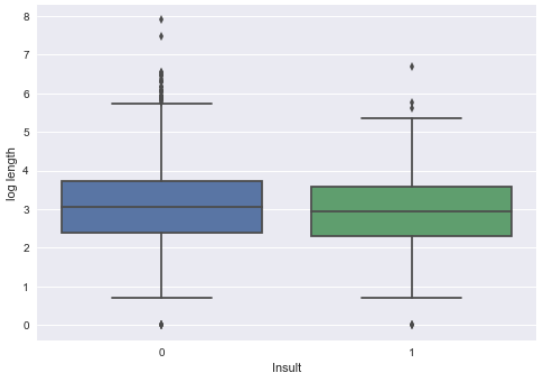
Figure 2.1: Part of the training set

2.1.1 Comment/Sentence Length Analysis

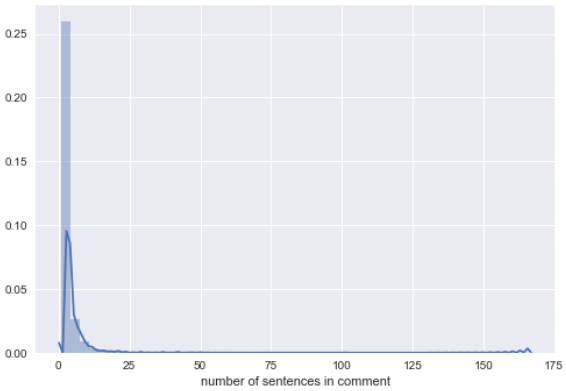
The original dataset consists of only comment text information, therefore without further processing, there is limited space for direct data analysis. Here I first analyse the length related statistics within the training set. The distribution (Figure 2.2a) of comment length (total word in each comment) is left skewed, most comments are less than 200 words while there exist several extreme outliers. From the box-plots of log-transferred comment length of two groups (Figure 2.2b), there exists some slightly difference between the distributions of the length of insulting and normal comments: on average insulting comments are shorter than normal comments.



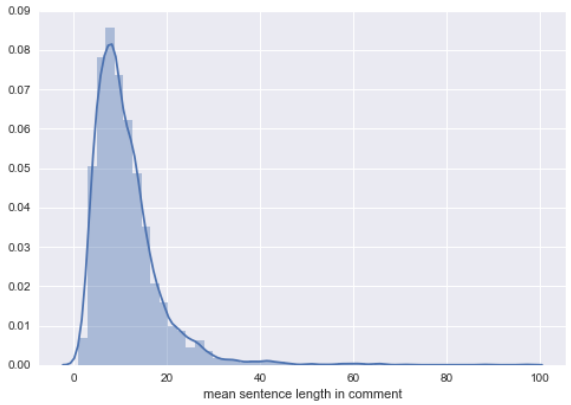
(a) distribution of comment length



(b) log-transferred comment length of two groups



(c) distribution of number of sentences in comments



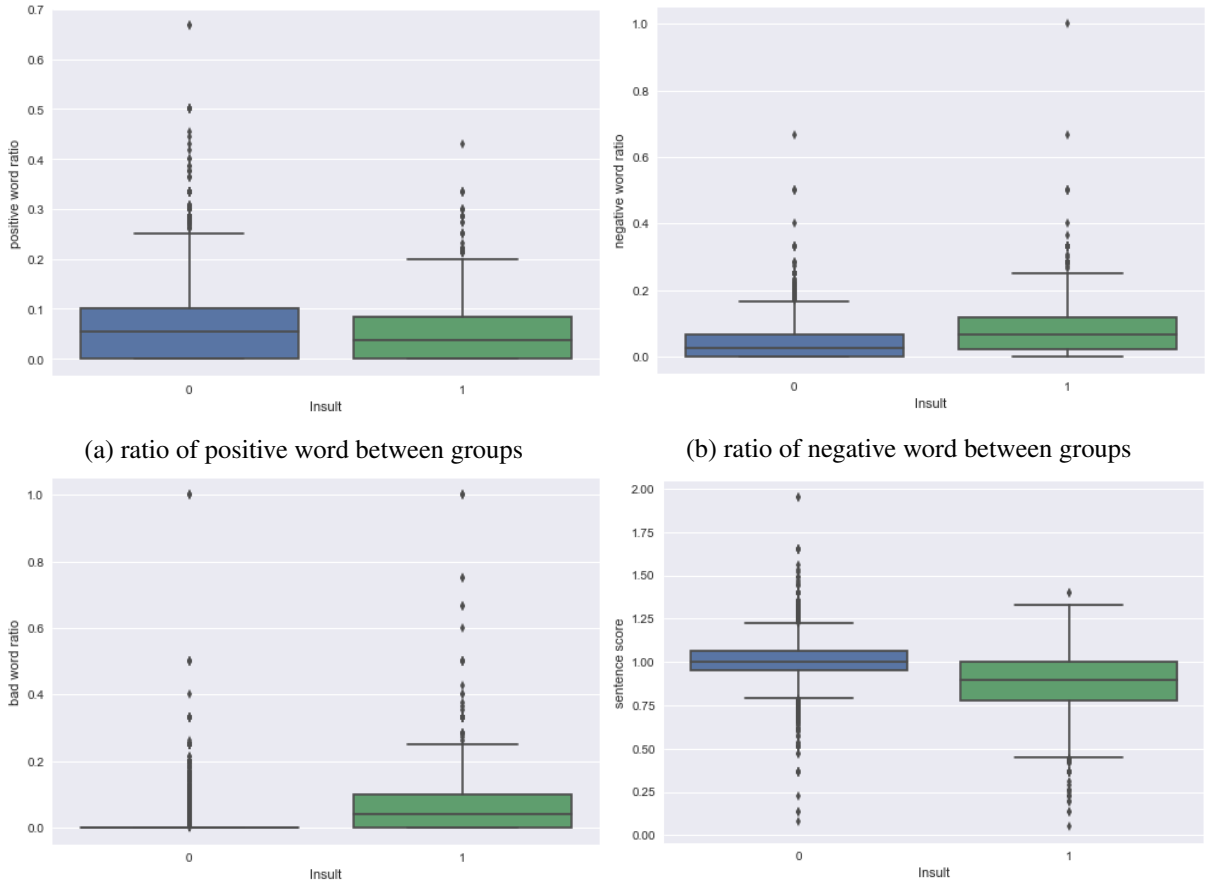
(d) distribution of mean of sentence length in comments

Figure 2.2: comments/sentences length statistics

One comment in the dataset usually consists of several sentences. Here I split each comments in the training set into sentences, using period, comma, colon, semicolon, question marks and exclamation mark as slicing indices. It can be seen from the sentences number histogram that (Figure 2.2c) most comments are composed of less than 25 sentences, and most sentences consists of 1 to 40 words (Figure 2.2d) .

2.1.2 Preliminary Sentimental Analysis

Here the ratio of positive and negative sentimental words between groups is plotted in Figures 2.3a and 2.3b. The positive and negative words are defined using the SentiWordNet [18]. Overall insulting comments have a smaller positive word ratio and larger negative word ratio compare to normal comments. The ratio of offensive words in each comments is computed using an offensive word list from Google [19] and shown in Figure 2.3c. It is obvious that the ratio of offensive word in an insulting comment is much higher than that in a normal comment, which is consistent with our common sense.



(a) ratio of positive word between groups

(b) ratio of negative word between groups

(c) ratio of bad word between groups

(d) sentimental score between groups

Figure 2.3: sentimental analysis between groups

The overall sentimental score between normal and insulting comments is also plotted in Figure 2.3d. The sentimental score of one comment is defined by myself as:

$$\text{sentiment score} = \exp\left(\frac{-2*\text{bad word frequency} - 1*\text{negative word frequency} + 1*\text{positive word frequency}}{\text{comment length}}\right)$$

The sentimental score roughly captures the sentimental trend represented in one comment; the value of score of one comment is proportional to the degree of positive sentiment shown in the comment. Overall the sentiment scores of insulting comments are much lower than that of normal comments. The above analysis indicates that sentiment related features might be useful to classify two groups.

2.1.3 Potential Noise

It has been found that there exist three identical comments 'f**king idiots' in the training set, two of which are labelled as insulting but the other is labelled as normal. This inconsistency indicates the Kaggle dataset contains noise which might restrict a classification model's performance.

Since the comments in the dataset are labelled manually, and each individual has different subjective judgement on the degree of offensiveness, consequently for one comment which is at the edge of flame, it might be classified as either normal or insulting by different individuals. This subjectivity has been reflected in the dataset. For example, in my subjective opinion, those test set comments in Table 2.2 have similar level of offensiveness, however they are classified into different classes. These comments which are hard to label also increase to noise within the dataset, thus further restrict the upper limit of the models' performance.

comment	Label
you f**k their heads	0
Start with impeaching this dumb a*s	0
Actually your version is completely fully of sh*t	1
I am sure she looks better than the sloppy out of shape b**ch you call your gril	1

Table 2.2: Part of comment labels

2.2 Data Preprocessing

The dataset is collected from different online forums, news commenting sites, blogs, text messages, etc. Among the dataset there exist plenty of typos, abbreviations, emojis, unmeaningful symbols and non-ascii characters. Therefore a systematic data proprocessing is need to reduce the ambiguity of text information, which is helpful on improving the performance of traditional models by unifying the extracted features. In addition data preprocessing can also improve the performance of RNN models by removing the non-related and unhelpful word within inputs.

The strategy of data preprocessing is based on what has been discovered in the exploratory data analysis, together with part of the winners' approaches [15] [16]. Because it has been found in the data analysis that offensive word information might be useful, I paid special attention on refining the offensive words in each comment. In general, the data preprocessing consists of mainly three parts: format fixing, word parsing and offensive words correction.

2.2.1 Format Fixing

Since the dataset is crawled from Internet without any processing, it contains many formatting issues and non-ascii characters which might cause trouble when extracting features later. Fixing the format of comments helps to remove some non-useful information. The format fixing includes:

- Formatting white-spaces by removing duplicates and newlines.
- Removing non-break symbols such as /xa0 which represents non-breaking space in Latin1 character encoding.
- Unifying different URL links, hash-tags and email address into specific tokens: '_URL_', '_Hash-Tag_' and '_Email_'.

2.2.2 Word Parsing

Word parsing was implemented on comments after format fixing. It includes lower-case transformation, symbol replacement, abbreviation/synonym replacement, uniform spelling, etc to unify the words with similar meanings. Some details of word parsing are listed as follows:

- Unifying abbreviations and synonyms: e.g.: bbq $\rightarrow$ barbecue, I'll $\rightarrow$ I will.
- Unifying spelling: e.g.: adaptor $\rightarrow$ adapter.
- Converting word with repeated characters into normal form, eg: cooooooool –> cool.

2.2.3 Offensive Words Correction

Last but not least, I spent some extra effort on correcting the offensive words. Sometimes people misspelled the offensive words purposely to avoid being detected. It is of importance to correct those misspelled words into a unified form. The offensive words correction includes:

- Converting words consisted by more than 3 symbols such as %$# to special token "_CR_" representing cursing word.
- Grouping together sequences of one-letter words such as "f * c k", since some internet users try to avoid filtering by splitting the offensive word by characters.
- Grouping together consecutive words (like "f* ck") using the bad word list, since some internet users try to avoid filtering by splitting the offensive word in to 2 parts.
- Removing dots inside words, since it has been discovered in the dataset that adding dot inside an offensive word is also a method of avoiding filtering.

Chapter 3

Traditional Feature-based Approach

Traditional feature-based approach is the most popular approach in offensive comment detection. Overall this approach extracts features from a dataset, and uses traditional classifiers such as logistic regression or support vector machine to classify the comment. In fact all candidates in the Kaggle's 'Detecting Insults in Social Commentary' challenge used traditional approaches to solve the offensive comments classification problem.

In this project, four feature-based models using different features from the preprocessed dataset are constructed, mainly used to:

1: Compare the performance of traditional models trained using different sorts of features and determine which kinds of features are more useful in offensive content detection .
2: Compare the best traditional model I constructed with the winner's model in the Kaggle challenge.
3: Later after creating the RNN models, the performances of the traditional models can be served as the benchmarks.

In the following section, I will in turn introduce the different features typically used in offensive text detection, followed by my feature extraction strategy and the evaluation metrics. Finally I will introduce and discuss the traditional models I constructed in this project and their performance.

3.1 Features Overview

The features used in traditional models can be classified into three types: lexicon-based, lexical and syntactic features. In the following part I will introduce the main characteristics, advantages and disadvantages of these features.

3.1.1 Lexicon-based Feature

Lexicon-based features such as offensive word count/ratio use a pre-defined dictionary to compute the relative statistics of offensive words in a text document. This is one of the most traditional and straight-forward features in offensive content detection and is still used in many modern websites (such as YouTube). Although simple and straight-forward, the main limitation of this kind of features is that the effect of these features purely depends on the coverage of dictionary. Nowadays thousands of offensive Internet slang are created every month, it is hard to expand the pre-defined dictionary with the new Internet slang in time. Companies like Facebook tries to dynamically expand the dictionary by adding the frequently reported words automatically, however this approach still suffers the lagging problem since the dictionary cannot be updated in a timely fashion.

3.1.2 Lexical Features

Lexical features consider the text in whole document instead of simply counting the frequency of specific words or phrases. **Bag of Words** (BoW) is one of the early used lexical features. From its name, it treats the document collection as a 'bag' which contains word tokens, it takes the frequency of each words in each text as features, and the order of word is ignored. Taking two sentences in the Shakespeare Sonnet as an example:

'Let me not to the marriage of true minds Admit impediments.
Love is not love which alters when it alteration finds, or bends with the remover to remove'

their BoW features are:

Term	**admit**	**alters**	**alteration**	**bends**	**finds**	**it**	**let**	**love**	**marriage**	**minds**	...
Sentence 1	1	0	0	0	0	0	1	0	1	1	
Sentence 2	0	1	1	1	1	1	0	2	0	0	...

And the total number of features in BoW is the number of unique words in the Shakespeare Sonnet.

N-gram approach is an variant of BoW, it is an improved approach in that it takes N adjacent words as a token rather than a single word. Offensive comments often start with 'You are' which can be successfully captured by bi-gram features, however BoW might fail to link 'You' with 'are' in this case thus fail to capture the information.

The bi-gram features of the above two sonnet sentences are:

Term	**Let me**	**me not**	**not to**	**...**	**love is**	**is not**	**not love**	...
Sentence 1	1	0	0	...	0	0	1	...
Sentence 2	0	1	1	...	1	1	0	...

TF-IDF (Term Frequency and Inverse Document Frequency) is introduced by Spärck Jones in 1972 [20] to improve the N-gram. The intuition behind it is that word tokens are different in their ability to classify

documents, and one token might be not a good discriminator if it occurs in most documents such as ‘an’, ‘the’ or ‘of’. Therefore we need to set weight to each token which captures its degree of discrimination, and this weight can be automatically calculated using the Inverse Document Frequency. The Inverse Document Frequency of token *t* is:

$$IDF_t = \log_{10} \frac{N}{n_t}$$

where N is the number of documents (comments in this project) in the collection, and n_t is the is the number of documents where token t occurs. For a token t which exists in almost every document such as ‘an’, ‘the’ or ‘of’, its n_t is close to N, thus its

$$IDF_t = \log_{10} \frac{N}{n_t} \approx \log_{10}(1) = 0.$$

Finally, the term frequency of each token t in each document is multiplied by its IDF, so as to indicate its real ability to classify documents.

3.1.3 Syntactic Features

Although it has been proved that lexical features, such as bi-gram and tri-gram, perform well in detecting offensive comments, without considering the grammar information, they often fail to distinguish sentences’ offensive information separated by long distances. In particular, offensive words usually have a clear target, however the target and offensive word sometimes are separated by attributive phrases which increases the difficulty of using lexical features to detect the offensive texts. Therefore, natural language

Rules	Meanings	Examples	Dependency Types
Descriptive Modifiers and complements: A(noun, verb, adj) ←B(adj, adv, noun)	B is used to define or modify A.	you f***ing; you who f***ing; you…the one…f***ing.	• abbrev (abbreviation modifier), • acomp (adjectival complement), • amod (adjectival modifier), • appos (appositional modifier), • nn (noun compound modifier), • partmod (participial modifier)
Object: B(noun, verb) ←A(noun)	A is B’s direct or indirect object.	F*** yourselves; shut the f** up; f*** you idiot; you are an idiot; you say that f***...	• dobj (direct object), • iobj (indirect object), • nsubj (nominal subject)
Subject: A(noun)→B(noun, verb)	A is B’s subject or passive subject.	you f***…; you are **ed… …f***ed by you…	• nsubj (nominal subject), • nsubjpass (passive nominal subject), • xsubj (controlling subject), • agent (passive verb’s subject).
Close phrase, coordinating conjunction: A and B; …A, B…; …B, B…	A and B or two Bs are close to each other in a sentence, but be separated by comma or semicolon.	F** and stupid; you, idiot.	• conj (conjunct), • parataxis (from Greek for “place side by side”)
Possession modifiers: A(noun)→B(noun)	A is a possessive determiner of B.	your f*** …; s*** falls out of your mouth.	• poss (holds between the user and its possessive determiner)
Rhetorical questions: A(noun)←B(noun)	B is used to describe clause with A as root (main object).	Do you have a point, f***?	• rcmod (relative clause modifier)

Table: syntactical intensifier detection rules

parsers [7] such as the Stanford parser are introduced to parse sentences on grammatical structures and then extract syntactic features from the parsed grammatical information. In Chen's paper 'Detecting Offensive Language in Social Media to Protect Adolescent Online Safety'[6], he creates a list of syntactical intensifier detection rules which is displayed in Table 2.1.

3.2 Feature Extraction and Selection

The features are extracted according to what has been discovered in the exploratory data analysis, and catalogued into lexicon-based, lexical and syntactic feature groups. In addition some features which are outside of those three feature groups but turn to be useful are also extracted, such as language model probabilities.

• The extracted lexicon-based features include the frequency and ratio of offensive words in each comment. The offensive words here are defined from a list of bad, swear and offensive words used by Google [19], I also expand the list using the offensive word list shared by the winner in the Kaggle challenge [14]. Since from the exploratory data analysis that sentiment related features might be useful in this task, the positive word count/ratio, negative word count/ratio and sentimental score, which are introduced in section 2.12, are also included as the lexicon-based features. Besides of above sentimental related features, the lexicon features also includes frequencies of hashtag, URL and email address in each comments.

• Lexical features used in this project consist of TF-IDF word-level n-gram (n from 1 to 5) and TF-IDF character-level n-gram (n from 2 to 5). Those terms which have document frequencies lower than three are ignored. This creates in total 93790 features and each of them represents the number of occurrence of corresponding words in comments, after idf weighting. However, the number of created features greatly outnumbers the sample size of the training set, using all these features will lead to an undetermined system, therefore a chi-square feature selection is used to select the 10000 TF-IDF features with the highest values for the test chi-squared statistic relative to the classes. Chi-square test can be used to measure dependence between variables, so using this selection method removes those features that are more likely to be independent from class and therefore irrelevant to classify the offensive comments. In total there are four syntactic features extracted from the dataset.

• After using the Stanford parser to analyse and label the grammar information of each comments, syntactic features are extracted according to the syntactical intensifier detection rules in Table 2.1. Due to the relatively small data size, not all syntactical intensifier detection rules can be found among the dataset; in this project only descriptive modifier and complements, objective, subjective and rhetorical questions rules are used as syntactical features. The quantities of these four features depend on the occurrence of dependency types in each syntactical intensifier detection rule in each comment.

• Another important feature I extracted is the language model probabilities. Language model probability is widely used in many natural language processing task especially the language translation. It is a probability distribution over sequences of words. Given such a sequence $w_1, \ldots, w_n$ of length m, it assigns a probability $P(w_1, \ldots, w_m)$ to the whole sequence [21]. Here the the collection of all comments from the training and test sets are defined as the language. After computing its distribution using certain

language model, for each comments C_i which consists of a sequence of words $w_{i1}, \ldots, w_{in}$, we computes its log-transformed probability divided by comment length $log(P(w_{i1}, \ldots, w_{im}))\frac{1}{N_i}$ as features. The language model chosen here is the modified Kneser-Ney (interpolated) which introduced in Chen & Goodman's paper in 1998 [22]; I modified the model by introducing backoff to handle the unseen words and improve the performance. For word w_i given its n previous sequence w_{i-n+1}^{i-1}, its n-gram probability $P_{KN}(w_i|w_{i-n+1}^{i-1})$ is represented as follows:

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{\max[c(w_{i-n+1}^{i-1}) - D(c(w_{i-n+1}^{i-1})), 0]}{\sum_{wi} c(w_{i-n+1}^{i-1})}$$

$$+\frac{D_1 * N_1(w_{i-n+1}^{i-1}\bullet) + D_2 * N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+1} * N_{3+1}(w_{i-n+1}^{i-1}\bullet)}{\sum_{wi} c(w_{i-n+1}^{i-1})} * \frac{N_{1+}(\bullet w_{i-n+2}^{i})}{N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet)})$$

where

$$N_{1+}(\bullet w_{i-n+2}^{i}) = |\left\{w_{i-n+1} : c(w_{i-n+1}^{i} > 0)\right\}|$$

$$N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet) = |\left\{w_{i-n+1}, w_i : c(w_{i-n+1}^{i} > 0)\right\}| = \sum_{w_i} N_{1+}(\bullet w_{i-n+2}^{i})$$

and

$$D(c) = \begin{cases} 0 & \text{if } c = 0 \\ 1 - 2d\frac{N_2(w_{i-n+1}^{i-1}\bullet)}{N_1(w_{i-n+1}^{i-1}\bullet)} & \text{if } c = 1 \\ 2 - 3d\frac{N_3(w_{i-n+1}^{i-1}\bullet)}{N_2(w_{i-n+1}^{i-1}\bullet)} & \text{if } c = 2 \\ 3 - 4d\frac{N_4(w_{i-n+1}^{i-1}\bullet)}{N_3(w_{i-n+1}^{i-1}\bullet)} & \text{if } c >= 3 \end{cases}$$

The model structure consists of mainly three parts. The first part $\frac{max[c(w_{i-n+1}^{i-1}) - D(c(w_{i-n+1}^{i-1})), 0]}{\sum_{wi} c(w_{i-n+1}^{i-1})}$ estimates the ngram frequency with discount, the second part $\frac{D_1 * N_1(w_{i-n+1}^{i-1}\bullet) + D_2 * N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+1} * N_{3+1}(w_{i-n+1}^{i-1}\bullet)}{\sum_{wi} c(w_{i-n+1}^{i-1})}$ normalise the probability, and the last part $\frac{N_{1+}(\bullet w_{i-n+2}^{i})}{N_{1+}(\bullet w_{i-n+2}^{i-1}\bullet)}$ is the interpolated lower-order distribution which measures the continuation of the word given history. Back-off is introduced in this model in order to improve the precision and solve the problem of unseen history. Given one word and its history, if the count of history equals to zero ($\sum_{wi} c(w_{i-n+1}^{i-1}) = 0$) or its word and history belongs to a set where the continuation of word and history equals zero ($w_{i-n+1}^{i} \in \left\{w_{i-n+1}^{i} where N_{1+}(\bullet w_{i-n+2}^{i}) = 0\right\}$), the model backoffs to a lower-order gram in order to find the none-zero probability of the word and history. The Kneser-Ney probabilities up to 6-gram are constructed as the language probability features here.

• Apart from the above features, several other features are also extracted to help identifying the offensive comment, including:

1. capital word count/ratio: These features compute the frequency and ratio of all capital in each comment, since Internet users often uses capital word to show their strong and furious sentiment when posting offensive comment.

2. comment length: The comment length feature captures the number of word in one comment. From data analysis, insulting comments are on average shorter than normal comments.

3. average word length: This feature captures the average alphabet-length of words in each comment, since on average the offensive words in insulting comment are shorter than normal words.

Feature Type	Features	Feature Selection	Total number
Lexicon-based Features	Offensive Word Count Offensive Word Ratio Positive Word Count Positive Word Ratio Negative Word Count Negative Word Ratio Sentimental Score Hashtag Frequency URL Address Frequency Email Address Fequency	None	10
Lexical Features	Character Bi-gram Character Tri-igram Character Quad-gram Character Penta-gram Word Uni-gram Word Bi-gram Word Tri-gram Word Quad-gram Word Penta-gram	Chi-square selection	10000
Syntactic Features	Descriptive modifier Objective Subjective Rhetorical Questions	None	4
Language Model Features	Uni-gram Kneser-Ney Bi-gram Kneser-Ney Tri-gram Kneser-Ney Quad-gram Kneser-Ney Penta-gram Kneser-Ney Sexa-gram Kneser-Ney	None	6
Other Features	Capital Word Count Capital Word Ratio Comment Length Average Word Length Starting with 'You are'	None	10

Table 3.1: Feature Table

4. staring with 'You are': This is a binary feature which indicates if the comment starts with 'You are', since offensive comments sometimes begin with 'you are' as the insulting target.

The table of features summary is listed in Table 3.1.

3.3 Evaluation

3.3.1 Area under the Receiver Operating Curve

The evaluation metric for the original Kaggle competition is the Area under the Receiver Operating Curve (AUC). A receiver operating characteristic (ROC) is created by plotting the true positive rate (TPR) against the false positive rate (NPR) at different value from 0 to 1, which illustrates the performance of a binary

classifier system when its discrimination threshold is varied [23].

The AUC is calculated from:

$$A = \int_{-\infty}^{\infty} TPR(T)(-FPR^{'}(T))dT$$

This evaluation metric considers both classes fairly in terms of misclassification errors, which is crucial in the classification of two highly unbalanced classes (e.g., one major negative class with one minor positive class). Another benefit of using AUC as evaluation metric is that it avoids threshold setting in classification task, which sometimes greatly affect the calculation of accuracy.

3.3.2 F1 Score

Here I use F1 score as another measure of accuracy. F1 score is the harmonic mean of precision and recall:

$$F1 = 2\frac{precison*recall}{precison+recall}$$

The F1 score of one prediction trends to be closer to the lower value between precision and recall, due to the property of harmonic mean, which makes F1 score another popular evaluation metric when the class distribution is unbalanced.

3.4 Model and Training

The prediction model used in the feature-based approach is logistic regression. Although other more complicated models such as random forest might perform better, their training time is way longer considering the relatively large feature size of lexical features model. Therefore logistic regression is chosen since it achieves an appropriate trade-off between performance and training time. For data point i with feature $X_i = (x_{i1}, x_{i2}, ..., x_{im})^T$, its predicted probability that X_i belongs to class 1, denoted by $P(\hat{y}_i|X_i)$, is:

$$P(\hat{y}_i = 1|X_i) = \frac{1}{1 + e^{-\theta^T X_i}}$$

Balanced weighted loss (3 on insulting comment and 1 on normal comment) is used due to the unbalanced classes in the training set, and l_2 regularisation (C=2) is chose to alleviate the over-fitting.

The training set is randomly split into a training set and a cross-validation set with ratio of 9:1, and the hyper-parameters of logistic regression is chosen by gird search on the cross-validation set. All models are constructed by using Python Scikit-learn 0.190.

3.5 Results

Four models are constructed using logistic regression. Three of them only include lexicon-based, lexical and syntactical features respectively, while the another model contains full features extracted in Table 3.1. The results are shown in Figure 3.2

The lexical feature model is the best single-sort-feature model, but considering its large feature size (10000 after chi-square selection), its performance is within the expectation. The lexicon-based model, although only contains 10 features, achieves a decent results on both F1 and AUC. The syntactic feature model, however, does not perform well compares to the other two single-sort-feature models. The full feature model is the best model among all, in total it contains 10030 features, it slightly outperforms the winner's approach by 0.002. In addition, there exists a large gap between the models' performance on the validation set and the test set, which might indicates the data within the validation and test sets are not consistent.

	Validation		**Test**	
	F1	*AUC*	*F1*	*AUC*
lexicon-based features	0.6359	0.8437	0.6476	0.7215
lexical features	0.7252	0.9101	0.7043	0.8233
Syntactic features	0.5097	0.6753	0.5212	0.6325
Full features	**0.7444**	**0.9191**	**0.7383**	**0.8447**
Winner's model	/	/	/	0.8425

Table 3.2: Performance Comparison

In addition, after computing the $P(\hat{y}_i = 1|X_i)$ of each comment in the test set, it has been discovered that all comments in Table 2.2 are misclassified into wrong class by the full features model (Table 3.3). It indicates that the subjectivity in labelling might affect the model's performance due to the increasing false positive/negative rates on those marginal data points.

	comment	**Label**	$\mathbf{P(\hat{y}_i = 1)}$
False Positive	you f**k their heads	0	0.8351
	Start with impeaching this dumb a**	0	0.9310
False Negative	Actually your version is completely fully of sh*t	1	0.4626
	I am sure she looks better than the sloppy out of shape b**ch you call your gril	1	0.2265

Table 3.3: Part of comments and corresponding $P(\hat{y}_i = 1|X_i$

Chapter 4

Recurrent Neural Network (RNN) Approach

Traditional feature-based approaches, although performs well, require a great amount of work on feature engineering. New approaches such as Recurrent Neural Network (RNN) can save us from time-consuming feature engineering, and in recent years, they achieve outstanding performances on many natural language processing (NLP) tasks.

In the following section, I will in turn introduce the structures of the basic recurrent neural network and its variant the long-short-term-memory (LSTM) network. Next I will explain the structures of all constructed RNN models in the project and display their corresponding visualisations and parameters. Finally I will discuss their performances and the comparison between RNN models and traditional feature-based models.

4.1 Introduction

Human deduce the degree of offensiveness of one text document by reading through it word by word and sentence by sentence, the understanding of a text document is based on all previous word we read. Traditional approaches in offensive detection fail to capture this pattern. Fortunately recurrent neural networks are able to imitate this pattern to some extent, by capturing the sequential information of their input using the the loop property.

4.1.1 Recurrent Neural Network

In machine learning applications, there exist many sequential data with form:

$$x_1, x_2, x_3, x_4, ..., x_t$$

For example:

- In natural language processing, x_1 is the first word, x_2 is the second word and so on.
- In signal processing $x_1, x_2, x_3...$ are the the signal information in each frame.
- In finance, $x_1, x_2, x_3...$ might be the prices of one stock each day/week.

This sort of data set usually varies in length, therefore it is difficult for traditional neural networks to handle the sequential data. However RNN can be used to extract features from sequential data, and transfer them into a fixed length vector, with the help of its loop property. Starting with the first input x_1, the architecture of RNN is shown as follows:

- x_1 ($x_i \in \mathbb{R}^{d_x}$ for i in $1:t$) is a vector which represents the first input. (Later in this chapter, I will discuss the methodology of how to transfer words into vectors). x_1 is first concatenated with the initial hidden state h_0 (usually a zero vector) where $h_0 \in \mathbb{R}^{d_h}$, and then the h_1 is computed by transferring the concatenated vector $[h_0; x_1]$ into a linear layer followed by pointwise non-linear tanh activation function:

$$h_1 = tanh(W \cdot [h_0; x_1] + b)$$

where $W \in \mathbb{R}^{d_h \times (d_h + d_x)}$ and $b \in \mathbb{R}^{d_h}$.

Figure 4.1: RNN architecture 1

- The calculation of h_2 is similar to that of h_1, by first concatenating x_2 with h_1, then computing h_2 through:

$$h_2 = tanh(W \cdot [h_1; x_2] + b)$$

The W and b here are the same as those in the first step, in other words the RNN is used recursively in each steps.

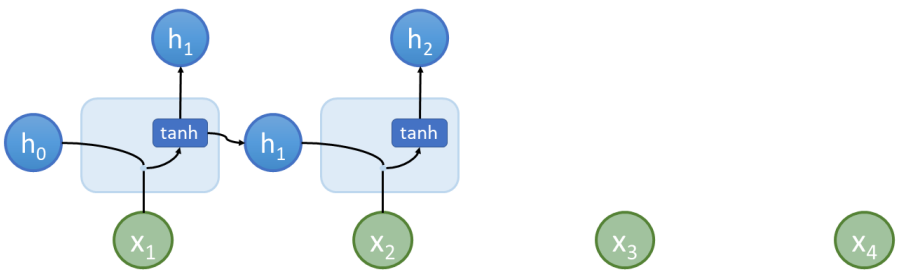


Figure 4.2: RNN architecture 2

• Repeating the above calculation by recursively feeding the current word vector x_t and the previous output h_{t-1} into the RNN, we obtain the last output h_n, where n is the length of the inputs. h_n can be used as the vector of final output feature.

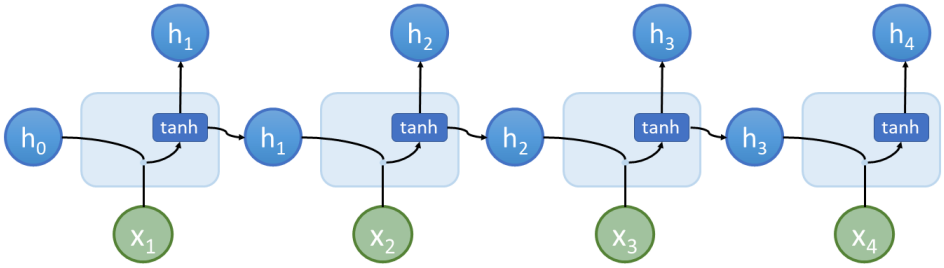


Figure 4.3: RNN architecture final

For convenience, only sequential input x_t of length four is plotted in the above example, but in reality RNN is capable of transferring the sequential inputs of any length into feature vectors of same size, by recursively using its inside parameters. For example, in Figure 4.4 there exists two sentences with length 4 and 6 respectively, and we want to transfer them into feature vectors of same size. RNN can achieve this by using RNN recursively 2 more times for the second sentence. Since the parameters of RNN at all time steps are shared, the parameters needed to computing features vectors of two sentences are the same.

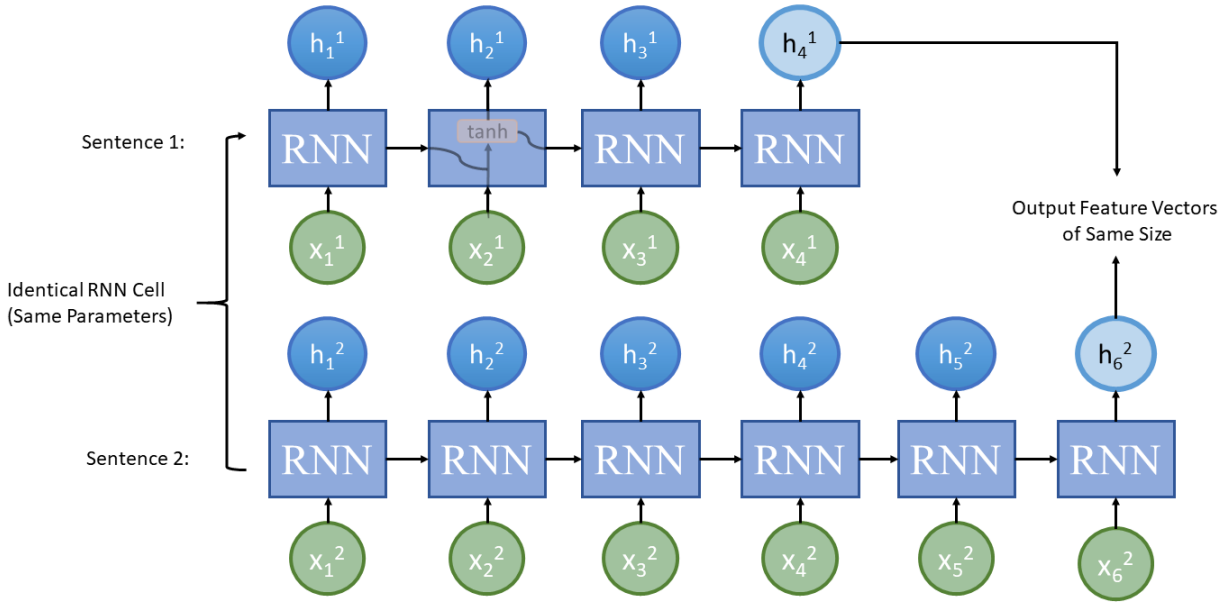


Figure 4.4: Difference length inputs

4.1.2 Long-Short-Term-Memory Network

The biggest drawback of RNN is that it suffers the gradient vanishing problem which is discussed by Hochreiter and Schmidhuber, 1997 [24] and Bengio,et al., 1994 [25]. Consequently, RNN may fail to capture the long-distance correlation (also called as long-term dependency) in a sequential input. If the length

of inputs is too long, the output of RNN may contain little information related to those inputs at initial steps.

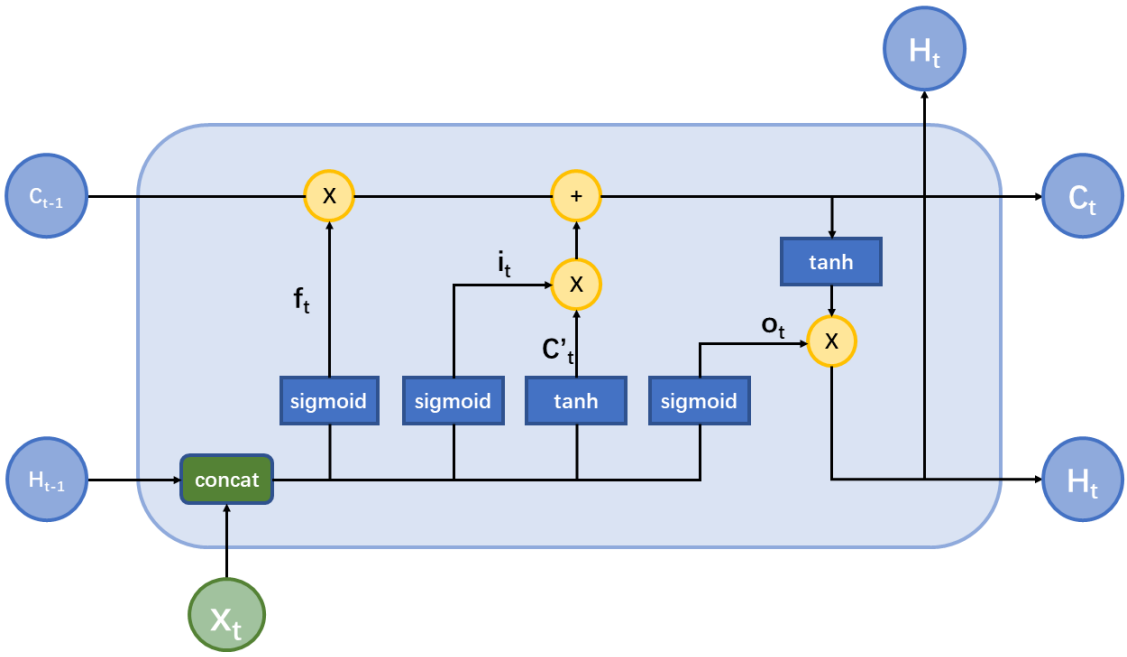


Figure 4.5: LSTM structure

Long Short Term Memory networks (LSTM) is introduced by Hochreiter and Schmidhuber in 1997 [24] to solve the gradient vanishing problem of RNN. LSTM is a variant of RNN, it contains an extra cell state and more complicated inside structure. The structure of LSTM is shown in Figure 4.5. The key intuition behind LSTM is that it introduces a new cell state which acts like a conveyor belt. It controls how much information from the previous step is kept and how much information from current input is transferred into the current step. The step-by-step workflow of LSTM is shown as follows:

• The initial step of LSTM is to decide how much information from previous step is kept, using a sigmoid layer called the 'forget gate layer'. Similar to RNN, x_t is first concatenated with the previous hidden state h_{t-1}, and then the f_t is computed by transferring the concatenated vector $[h_0; x_1]$ into a linear layer followed by pointwise non-linear sigmoid activation function (instead of the tanh activation function in the regular RNN):

$$f_t = sigmoid(W_f \times [h_{t-1}; x_t] + b_f)$$

where $W_f \in \mathbb{R}^{d_h \times (d_h + d_x)}$ and $b_f \in \mathbb{R}^{d_h}$, d_h and d_x are the vector sizes of h_{t-1} and x_t respectively. The output f_t is a vector with same size of c_{t-1} and each element in f_t has value between 0 and 1, after sigmoid transformation. f_t is later used to control what part of and how much information in c_{t-1} is kept in the current step.

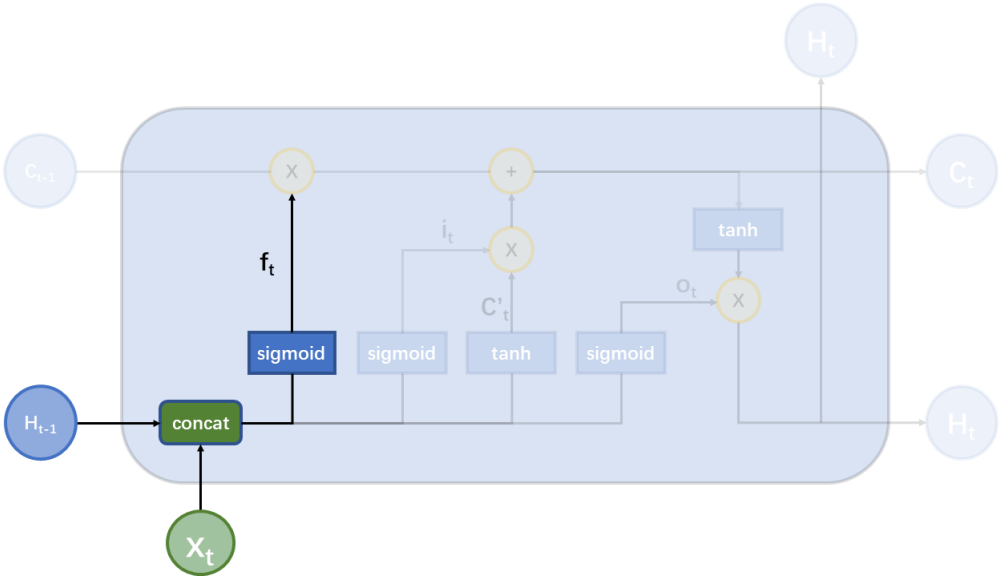


Figure 4.6: LSTM forget gate layer

• Next LSTM decides how much information from current input and previous hidden state is transferred into the current cell state. This is achieved by first applying another sigmoid layer called the "input gate layer", it produces a vector i_t which decides how much current input will be updated. Meanwhile, a tanh layer is applied to the concatenated $[h_{t-1}; x_t]$ to compute the vector of current input C'_t. Mathematically, the above operation is equivalent to:

$$i_t = sigmoid(W_i \times [h_{t-1}; x_t] + b_i)$$
$$C'_t = tanh(W_C \times [h_{t-1}; x_t] + b_C)$$

where W_i, $W_C \in \mathbb{R}^{d_h \times (d_h + d_x)}$ and b_i, $b_C \in \mathbb{R}^{d_h}$

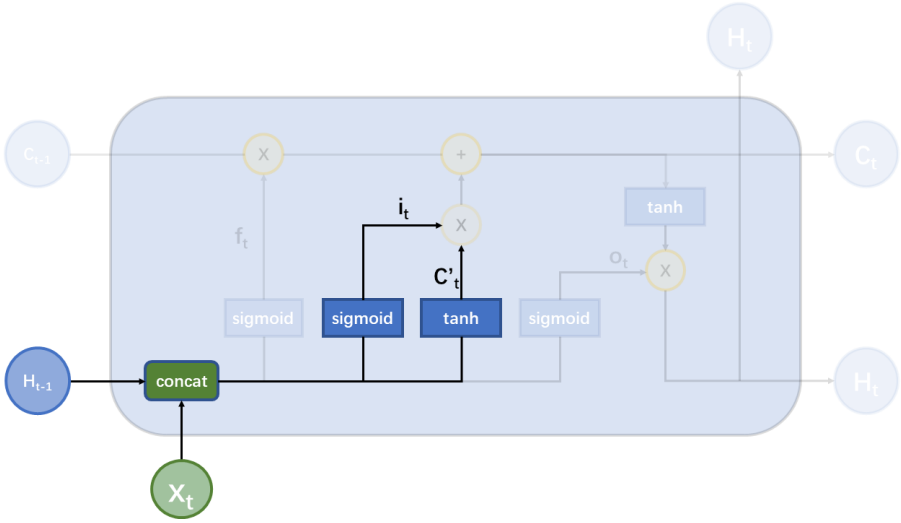


Figure 4.7: LSTM input gate layer

• The third step is to update new cell state C_t from C_{t-1} and the previously computed vectors f_t, i_t, C'_t. C_t is obtained by first point-wise multiplying C_{t-1} with f_t, thus part of information in the previous cell state C_{t-1} is forgotten. Then the vector is updated by adding the scaled current input $C'_t * i_t$. Overall the new cell state C_t is computed through:

$$C_t = f_t * C_{t-1} + C'_t * i_t$$

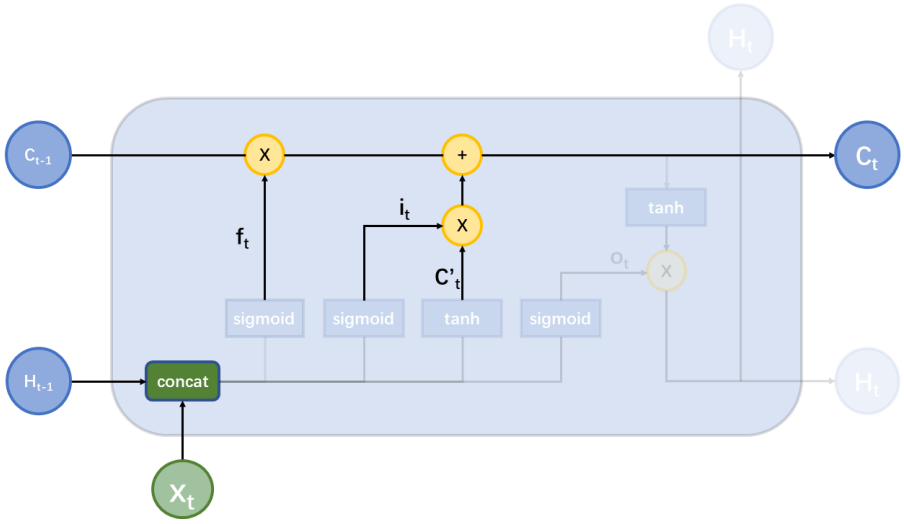


Figure 4.8: Cell state update

• The final step is to calculate the output h_t based on the current cell state output C_t and current input $[h_{t-1}, x_t]$. h_t is obtained by pointwise multiplying tanh activated C_t with linear transformed $[h_{t-1}, x_t]$, followed by a non-linear sigmoid activation function. In this way the output h_t is computed by LSTM, and not all information in the current input is used to compute output. Mathematically:

$$o_t = sigmoid(W_o \times [h_{t-1}; x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

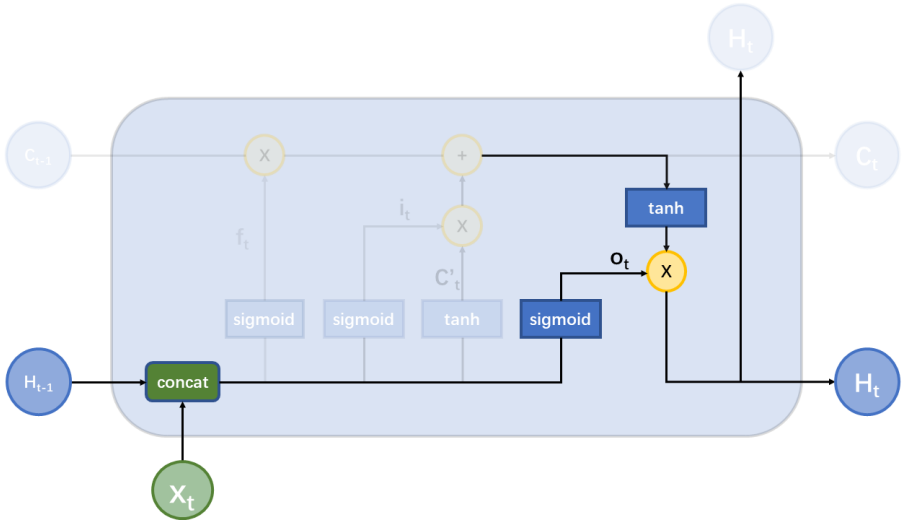


Figure 4.9: LSTM output

In short, the operation of LSTM in one time step can be abbreviated as:

$$h_t, c_t = LSTM(h_{t-1}, c_{t-1}, x_t)$$

or more often

$$h_t = LSTM(h_{t-1}, x_t)$$

since the cell states c_t are transferred within LSTM and only the hidden states h_t are used as the LSTM outputs.

There exist several variants of LSTM including Gated Recurrent Unit (GRU), Grid LSTM, etc. However the intuitions behind them are identical: to control the information inflow and outflow in the current cell, so that not all information from the previous step is inflowed into the current step and not all information from the current step is outflowed as output. In this way the long-distance correlation in a sequential input can be better captured.

4.2 Word Embedding

Before transferring words into a recurrent neural network, we need to first convert each word into word vector representation (also known as word embedding (Bengio et al., 2003 [26])). All the word vector representations can be stacked into a word embedding matrix $M_w \in \mathbb{R}^{V \times d_v}$ where d_v is the pre-defined dimension of word vector and V is the vocabulary size. This word embedding matrix can be included as a part of trainable parameters in the network and optimised during training.

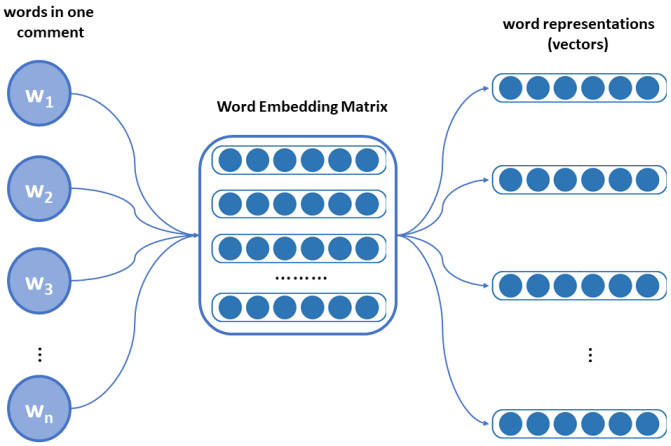


Figure 4.10: Word Embedding

Each word vector representation can be either initialised randomly from a distribution, or using pre-trained corpus such as word2vec (Mikolov et al., 2013 [27]) and Glove (Pennington et al., [28]). In this project the pre-trained word representation method is used using a Glove Twitter dataset, this corpus has in total 1.2M vocabularies and word vectors of size 100. Out-of-vocabulary words in the training set are randomly initialised by sampling values uniformly from [-0.5, 0.5].

4.3 Models

In this project, several recurrent neural network models are constructed to solve the offensive comment detection problem, starting from the simplest baseline model to the most complicated document level model.

4.3.1 Baseline Model

The architecture of the baseline RNN model is straight-forward: the words $w_1, w_2, .., w_n$ in one comment are first converted into word vector representations $x_1, x_2, .., x_n$ and then fed forwardly into RNN. This creates a series of RNN outputs $h_1, h_2, .., h_n$. The last RNN output h_n is used as the final comment representation (feature vector) $Comment$. This series of operation is called as 'comment encoder', where the information in one comment is encoded as a feature vector. Mathematically, for input word representations $x_1, x_2, x_3, ..., x_n$:

$$Comment = h_n \quad \text{where } h_t = LSTM(h_{t-1}, x_t)$$

The next step is the 'classifier', where the comment representation $Comment$ is transformed into a k-dimensional vector through a linear layer followed by a RELU activation function. Then another linear layer is used to transform the k-dimensional vector into the logits vector with length = 2 (the class number). Afterwards, the logits vector is transformed into conditional probabilities through a softmax layer. The above operations can be formalised as follows:

$$MLP_1 = RELU(W_{mlp} * Comment + b_{mlp}) \quad \text{where } W_{mlp} \in \mathbb{R}^{d_k \times d_h} \text{ and } b_{mlp} \in \mathbb{R}^{d_k}$$

$$logits = W_l * MLP_1 + b_l \quad \text{where } W_l \in \mathbb{R}^{2 \times d_k} \text{ and } b_l \in \mathbb{R}^2$$

$$P(\hat{y} = 1|X) = \frac{\exp(logits_1)}{\exp(logits_0 + logits_1)} \quad Softmax$$

The loss function used in this model is a cross-entropy loss between real class y_i and predicted class probability $\hat{P}_i$ of one comment. An Adam optimiser with learning rate = 0.001 is used on the sum of cross-entropy loss of each training data to optimise the model's parameters.

$$loss = \sum_i^N y_i \log(P(\hat{y}_i = 1|X_i)) + (1 - y_i) \log(1 - P(\hat{y}_i = 1|X_i))$$

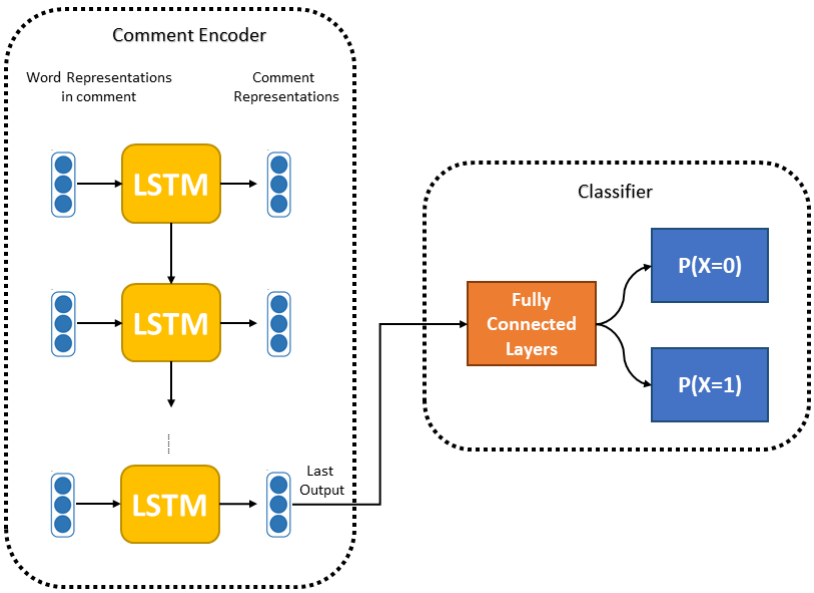


Figure 4.11: Baseline Model Architecture

The structure of the above baseline model is shown in Figure 4.11. The baseline model, although simple and relatively rough, can achieve a decent result in the later experiment. However, there are several limitations in its design:

• The baseline model uses the final LSTM output as the comment representation. Although LSTM can alleviate the long-term dependency loss, it does not fully solve the problem. The model might still miss the information of word representations at the initial time steps if the comment is too long.

• The offensiveness of one comment usually depends on only part of comment (one or couple of words in a sentence), and most part of a comment is relatively normal. However, the baseline model tries to identify the offensiveness from an abstract comment representation instead of focusing on some specific parts of comments. This factor might also be detrimental to the model's performance.

• One comment consists of one or more sentences, and there might exist correlation between sentences which can be exploited to improve the accuracy of classification. However in the baseline model, each comment is considered as a sequential series of words, and the sentence structural information is ignored.

Considering the limitations of the baseline model, in the following section, I propose two other models to improve the baseline model's design and solve its limitations.

4.3.2 Bidirectional Word-level Model with Weighted Sum Pooling

The bidirectional word-level model with weighted sum pooling (for short: word-level model) is an improved version of the baseline model. The general structure of this model is similar to that of the baseline model: we compute the comment representation from a sequential series of word representations. The sentence structure within each comment is still disregarded. The improvement by the word-level model is that:

• The comment representation comes from a **weighted sum pooling** (also called as 'attention') using the LSTM output at all time steps.
• It uses a **bidirectional LSTM** instead of the single forward LSTM.

Weighted Sum Pooling

Theoretically there are mainly three strategies of extracting the comment representation from LSTM outputs. First we can simply use the last LSTM output as the comment representation (Figure 4.12 (a)). This is the strategy used in the baseline RNN, but suffers the risk of long-term dependency loss. Secondly we can take the average of all LSTM outputs as the comment representation (Figure 4.12 (b)). This strategy alleviates the risk of long-term dependency loss compared to strategy (a).

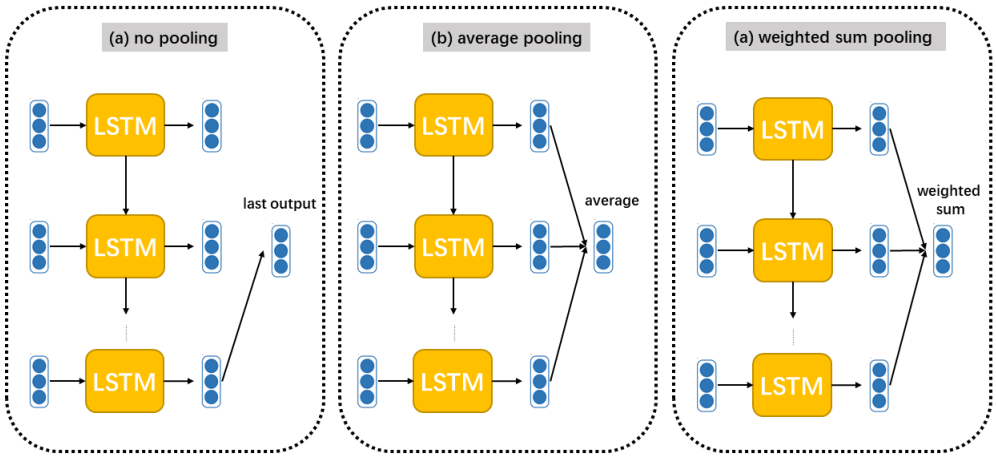


Figure 4.12: Pooling Strategy

However, it is obvious that some words in one comment are more important than the others in the offensiveness detection task. Therefore it would be beneficial if we can assign different weights to different LSTM outputs when pooling. This can be achieved using a weighted sum pooling strategy which is often called 'attention' (Xu, et al. [29]). Specifically:

$$u_t = tanh(W_a h_t + b_a)$$
$$a_t = \frac{\exp(u_t^T u_w)}{\sum_t \exp(u_t^T u_w)}$$
$$C = \sum_t a_t h_t$$

That is, we first transfer each LSTM output h_t into a linear layer with the tanh activation function to get the hidden representation u_t for each h_t. Then the weight of each h_t is measured by a word-level context vector u_w after applying the softmax function. This gives us a series of weights $a_t \in [0, 1]$ which captures the importance of each h_t. Finally the comment representation $Comment$ is computed by the weighted sum of h_t based on their corresponding weights a_t. Here W_a, b_a and u_w are the parameters to be optimised during training.

Bidirectional RNN

The baseline model only read inputs x_t forwardly from x_1 to x_n, thus each h_t can be regarded as a summary of its preceding words. This approach is feasible in the baseline model, since only the last output of LSTM is used as the comment representation which is computed from all inputs $x_1, x_2, ..., x_n$.

However, the above approach might be problematic when using the weighted sum pooling, where the LSTM outputs from all time steps are used to compute the comment representation. Those LSTM outputs h_i at early time steps are computed from words at only early inputs, thus are less informative than the outputs h_i from late time steps. Consequently the weighted sum pooling is likely to assign less weight to

the h_t if t is small, and over-focus on the h_t at late time steps.

In order to solve this limitation, a bidirectional RNN (bi-RNN) structure (Bahdanau et al., 2014 [30]) is introduced in this model. Bi-RNN consists of a forward RNN and a backward RNN. The forward RNN reads the sequential series of inputs $x_1, x_2, ..., x_n$ in forward order from x_1 to x_n, and computes a series of forward RNN outputs $\overrightarrow{h}_1, \overrightarrow{h}_2, .. \overrightarrow{h}_n$; the backward RNN reads the inputs reversely and computes the backward RNN outputs $\overleftarrow{h}_1, \overleftarrow{h}_2, .. \overleftarrow{h}_n$. The h_t at each time step is obtained by concatenating the forward $\overrightarrow{h}_t$ and backward $\overleftarrow{h}_t$, such that:

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$$

In this way, each h_t can be regarded as a summary of its preceding words and its following words, meanwhile focusing on its current input. Therefore bi-RNN is able to improve the performance of weighted sum pooling by alleviating the over-focus tendency.

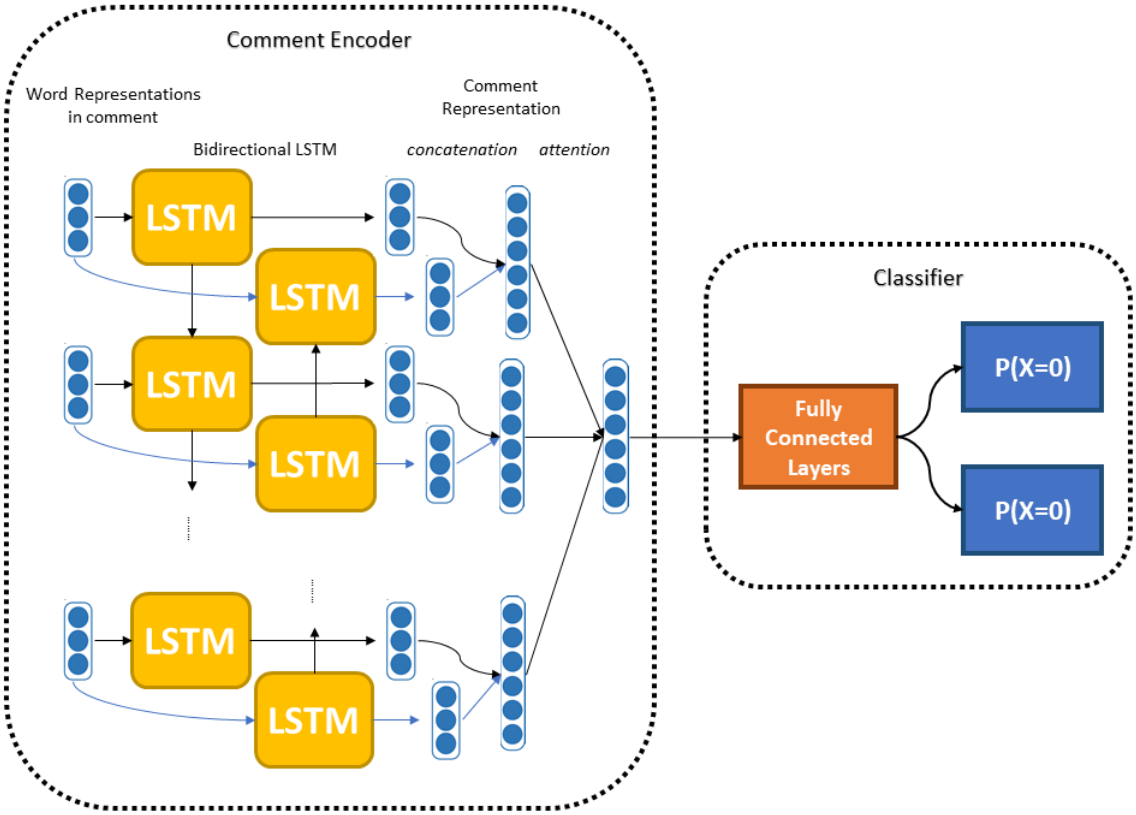


Figure 4.13: Word-Level Model Architecture

The graphic structure of bidirectional word-level model with weighted sum pooling is shown in Figure 4.13. The word-level model can be formulated as follows.

Comment Encoder:

For a comment with word representations $x_1, x_2, .., x_n$, where n is the number of words in the comment:

$$\begin{aligned}
\overrightarrow{h}_t &= \overrightarrow{LSTM}(\overrightarrow{h}_{t-1}, x_t) \\
\overleftarrow{h}_t &= \overleftarrow{LSTM}(\overleftarrow{h}_{t+1}, x_t) \\
h_t &= [\overrightarrow{h}_t; \overleftarrow{h}_t] \\
u_t &= tanh(W_a h_t + b_a) \\
a_t &= \frac{\exp(u_t^T u_w)}{\sum_t \exp(u_t^T u_w)} \\
Comment &= \sum_t a_t h_t
\end{aligned}$$

We obtain the corresponding comment representation.

Classifier and Loss:

The classifier and loss in the word-level model are identical to those in the baseline model:

$$\begin{aligned}
MLP_1 &= RELU(W_{mlp} Comment + b_{mlp}) \\
logits &= W_l MLP_1 + b_l \\
P(\hat{y} = 1|X) &= \frac{\exp(logits_1)}{\exp(logits_0 + logits_1)} \\
loss &= \sum_i^N y_i \log(P(\hat{y}_i = 1|X_i)) + (1 - y_i) \log(1 - P(\hat{y}_i = 1|X_i))
\end{aligned}$$

4.3.3 Hierarchical Sentence-level Model with Probability Pooling

The bi-directional word-level model improves the comment representation and alleviates the long-term dependency loss, but it still ignores the sentence structures within a comment. Therefore hierarchical sentence-level model is introduced to capture the sentence structure within a comment. Tang et al. 2015 [12] and Yang et al. 2015 [13] use similar approaches to solve the sentiment classification problem.

The hierarchical sentence-level model first splits a comment into a sequential series of sentences and uses LSTM to compute the corresponding sentence representations $s_1, s_2, ...s_l$ where l is the number of sentences in the comment. Then the sentence representations are fed into another LSTM to compute the final comment representation $Comment$. The main advantage of this model is that instead of taking a comment as a sequential series of words and disregarding its sentence structure, the hierarchical structure in the sentence-level model first computes the sentence representations in a comment, thus the correlation and relative importance of each sentence can be studied and analysed in this approach.

The hierarchical sentence-level model consists of several sentence encoders (depending on the number of sentences in a comment), one comment encoder and a final classifier. The parameters within each sentence encoder are shared, and their inner structures are identical to the that in the comment encoder in

the word-level model. The only difference is that the inputs in sentence encoders are the words in the each sentence rather than words in whole comment; the structure of comment encoder in hierarchical sentence-level model is also similar to that in word-level model, except now the inputs are the sentence representations $s_1, s_2, ..., s_l$, instead of the word representations $x_1, x_2, ..., x_n$.

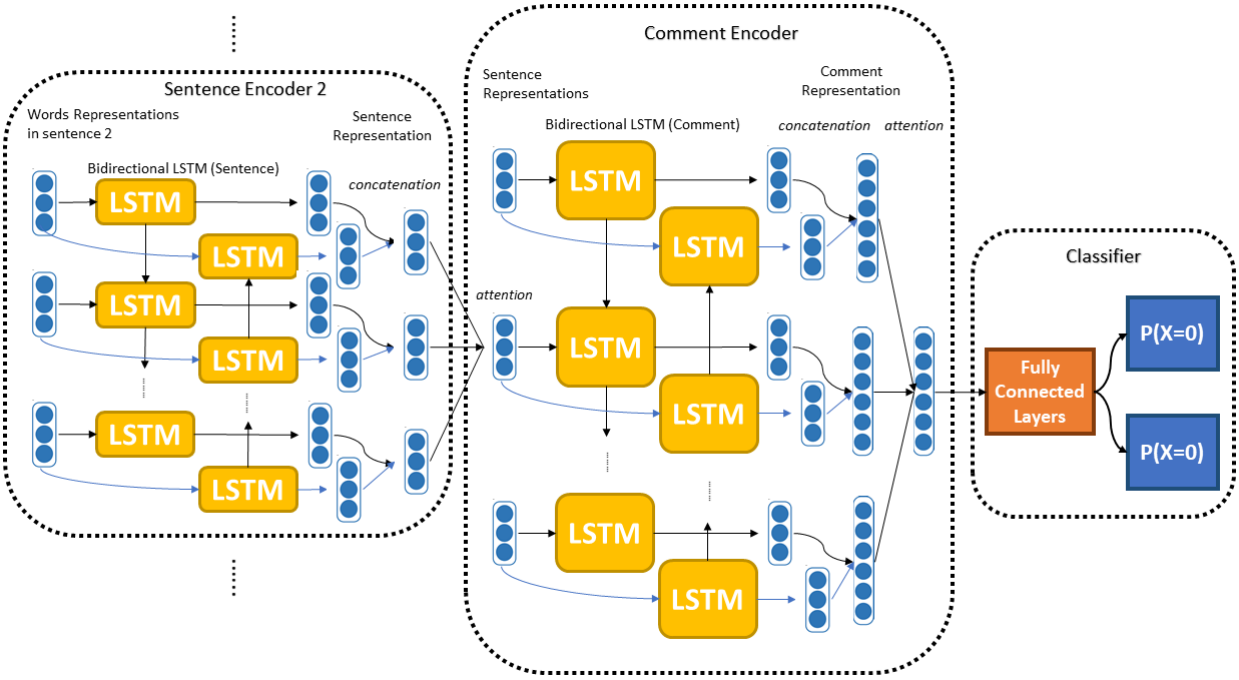


Figure 4.14: Hierarchical Sentence-Level Model Architecture

The graphic structure of hierarchical sentence-level word with weighted sum pooling is shown in Figure 4.14. It can be formulated as follows:

Sentence Encoder i:

For a sentence i in a comment with word representations $x_{i1}, x_{i2}, ..., x_{il_i}$, where l_i is the length of that sentence.

$$
\begin{aligned}
\overrightarrow{h}_{it} &= \overrightarrow{LSTM_s}(\overrightarrow{h}_{i,t-1}, x_{it}) \\
\overleftarrow{h}_{it} &= \overleftarrow{LSTM_s}(\overleftarrow{h}_{i,t+1}, x_{it}) \\
h_{it} &= [\overrightarrow{h}_{it}; \overleftarrow{h}_{it}] \\
u_{it} &= tanh(W_{sa} h_{it} + b_{sa}) \\
a_{it} &= \frac{\exp(u_{it}^T u_{sw})}{\sum_t \exp(u_{it}^T u_{sw})} \\
s_i &= \sum_t a_{it} h_{it}
\end{aligned}
$$

We compute the sentence representation s_i of sentence i.

Comment Encoder:

From the sentence encoders we obtain the sentence representations $s_1, s_2, ..s_n$, of all sentences in the comment, where n is number of sentence in the comment:

$$\begin{aligned}
\overrightarrow{h}_t &= \overrightarrow{LSTM}(\overrightarrow{h}_{t-1}, s_t) \\
\overleftarrow{h}_t &= \overleftarrow{LSTM}(\overleftarrow{h}_{t+1}, s_t) \\
h_t &= [\overrightarrow{h}_t; \overleftarrow{h}_t] \\
u_t &= tanh(W_a h_t + b_a) \\
a_t &= \frac{\exp(u_t^T u_w)}{\sum_t \exp(u_t^T u_w)} \\
Comment &= \sum_t a_t h_t
\end{aligned}$$

We compute the final comment representation in the hierarchical sentence-level model.

Classifier and Loss:

The classifier and loss in hierarchical sentence-level model are identical to those in the word-level and baseline models:

$$\begin{aligned}
MLP_1 &= RELU(W_{mlp} Comment + b_{mlp}) \\
logits &= W_l MLP_1 + b_l \\
P(\hat{y} = 1|X) &= \frac{exp(logits_1)}{exp(logits_0 + logits_1)} \\
loss &= \sum_i^N y_i \log(P(\hat{y}_i = 1|X_i)) + (1 - y_i) \log(1 - P(\hat{y}_i = 1|X_i))
\end{aligned}$$

4.4 Visualisation

One of the biggest criticism of deep learning models including the RNN is that it is hard to interpret. Although RNN approach can achieve decent or even break-through performance on many tasks, its automatic-feature-generating property makes it hard to investigate what factors contribute to its performance and how. Fortunately in this project, some visualisations method can help to indirectly analyse the classification mechanism of the RNN models. Here I focus on visualising the word representations and the weighted sum pooling (attention) method, and how they help to classify the comment.

4.4.1 Word Representations

In the RNN models, each word in comment is transferred into a word vector based on the word embedding matrix, and the word embedding matrix is part of the model's parameters which are optimised during training. Each row in the word embedding matrix corresponds to a word vector representation.

After training, we can visualise the similarity of each word vector using TSNE. TSNE, or t-distributed Stochastic Neighbour Embedding, is a tool to visualise high-dimensional vectors by projecting the them into a 2-d space. It converts similarities between vectors to joint probabilities and tries to minimise the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional vectors (Laurens et,al 2008 [31]).

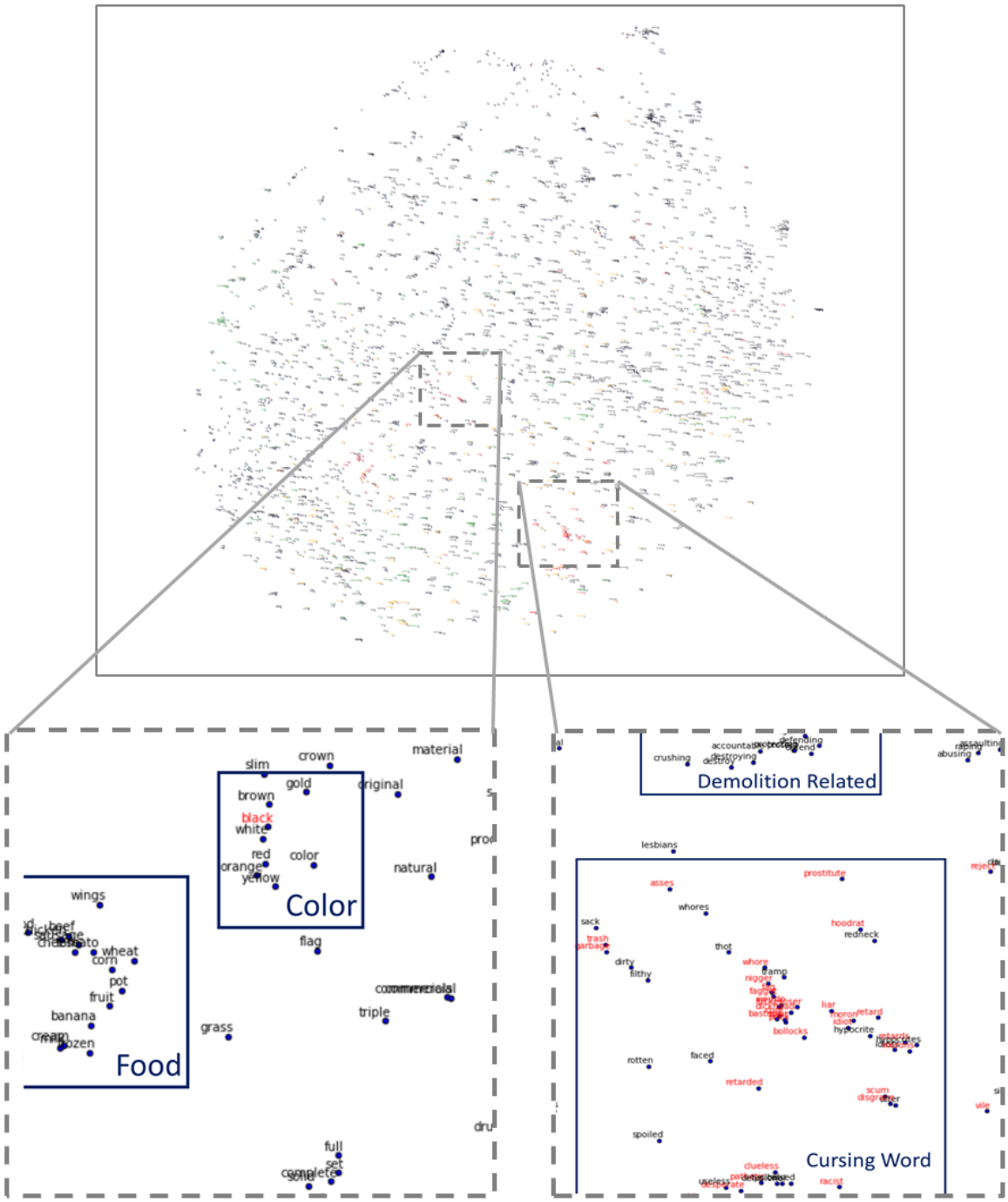


Figure 4.15: Word Representations Visualisation

Part of TSNE results from the optimised word embedding matrix of the word-level model is shown in Figure 4.15, the words marked in red are those offensive words defined in the previously used offensive

word list (section 2.1.2 and section 3.2). It can be observed clearly that those words with similar meanings or same type tend to locate close to each others, for example, words related to food or colour are located together which is shown in the left dash line block. From the right dashed line block, cursing words which marked in red locate within a close range, together with some words related to demolition which often come with negative meanings.

Since these word vectors represent the information related to each word, it is reasonable to deduce that that words with close meanings should have similar vectors after optimisation. The visualisation of word embedding matrix confirm this deduction, which indicates that the RNN models are capable of learning the meanings of each word during training.

4.4.2 Weighted Sum Pooling

Weighted sum pooling in this project is used to calculate the sentence/comment representations. Since the objective of the project is to classify if one comment is offensive or not, ideally the weighted sum pooling should assign higher weights to those LSTM outputs which contain offensive information.

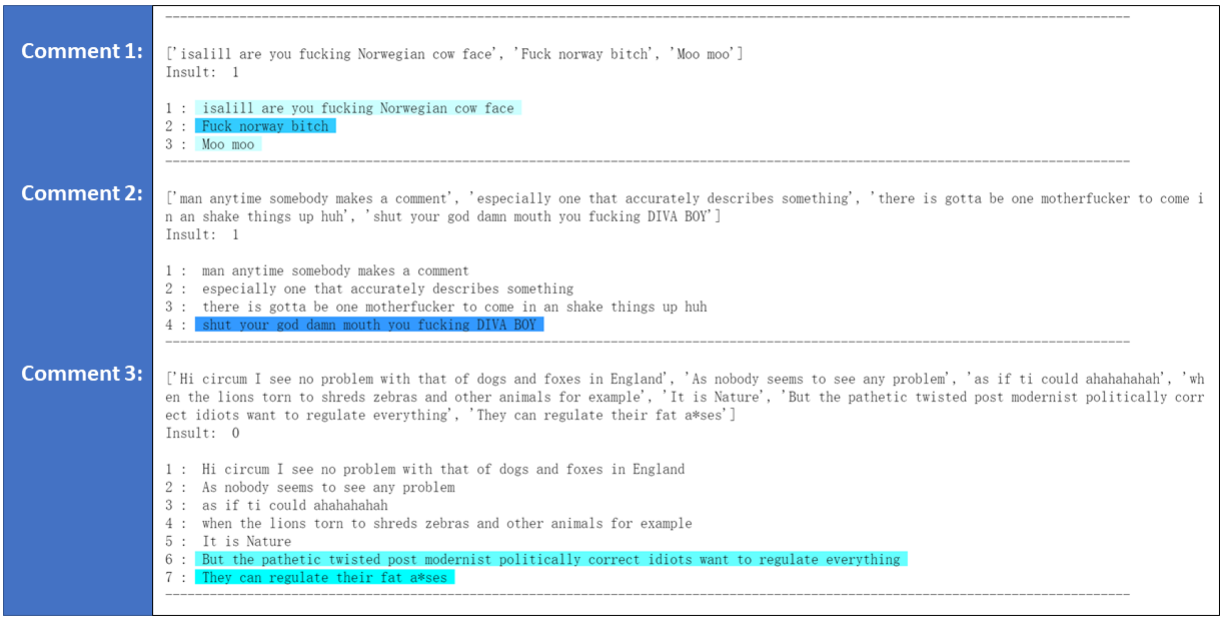


Figure 4.16: Sentence Attention Visualisation

Figure 4.16 is the visualisation of the weights assigned to all comment LSTM outputs h_t and their corresponding sentences in the hierarchical sentence-level model. The depth of blue indicates the size of weight assigned to each sentence. It can be seen from the visualisation that the weight assigned to each sentence is proportional to its degree of offensiveness. In comment 1, all three sentence contains offensive content thus all of them are assigned non-zero weights; in comment 2 only the last sentence is directly offensive with clear target, thus it is the only sentence assigned with a high weight; although comment 3 is labelled as non-offensive, the assigned weights are still related to the offensive degree of each sentence.

4.5 Experiment

All three RNN models are constructed using Tensorflow 1.20. The original training set is randomly split into the training and validation sets with ratio 9:1 (through the same random seed used in the traditional models). The learning rate is set as 0.001 and the models are trained using randomised mini-batch of size 64. Balanced weighted loss (3 on insulting comment and 1 on normal comment) is also used here due to the unbalanced classes in the training set .

The size of word representations is 100. The hidden sizes of LSTM in the comment encoders of all three models are equal to 200, while the hidden sizes of LSTM in the sentence encoders of the sentence-level model is set to be 100. The size of fully connected layer in each classifier is 100. The attention size in each weighted sum pooling is set to be equivalent to its input size. A dropout layer is used on the final comment representations as regularisation, and the dropout rate is 0.25. As a summary:

$$
\begin{aligned}
x_t &\in \mathbb{R}^{100} \\
\overrightarrow{h_{it}}\ ,\ \overleftarrow{h_{it}} &\in \mathbb{R}^{100}\ ;\ h_{it}\ , u_{it}\ , s_i \in \mathbb{R}^{200} \\
\overrightarrow{h_t}\ ,\ \overleftarrow{h_t} &\in \mathbb{R}^{200}\ ;\ h_t\ , u_t\ , C \in \mathbb{R}^{400} \\
MLP &\in \mathbb{R}^{100}
\end{aligned}
$$

4.6 Results and Comparison

The results of all constructed models in this project are shown in Table 4.1. Overall the RNN models perform well. All three RNN models achieve better results than the full features model on the test F1 score, and the word-level RNN model outperforms the other two RNN models. The test AUC of the word-level model greatly outperforms the winner's model from the Kaggle challenge by almost one percent.

The word-level model is constructed based on he baseline model, and the sentence-level model is constructed based on the other two models. The performance of the word-level model is indeed better than the baseline model comprehensively, which indicates the bidirectional RNN and the weighted sum pooling can improve the performance of RNN models on the offensive comment detection. However, the sentence-level model, although is designed to capture the sentence correlation between comments and hopefully improve the accuracy, performs even worse than the baseline model. One possible explanation is that the sentences within the sentence-level model are split from period, comma, colon, semicolon, question marks and exclamation mark. Since the comments inside the dataset are collected online and internet users often disregard the formal grammar when commenting, the above symbol split strategy may sometimes fail to split one comment into correct sentences. Therefore the inputs of the sentence-level model may contain more noise than other two RNN models and result in its relatively low performance.

	validation		Test	
	F1	*AUC*	*F1*	*AUC*
	Traditional Models			
lexicon features	0.6359	0.8437	0.6476	0.7215
lexical featues	0.7252	0.9101	0.7043	0.8233
syntatic features	0.5097	0.6753	0.5212	0.6325
full features	0.7444	0.9191	0.7383	0.8447
winner's model				0.8425
	RNN Models			
baseline	0.7552	0.9215	0.7647	0.8396
word-level	**0.7650**	**0.9250**	**0.7739**	**0.8516**
sentence-level	0.7469	0.9220	0.7583	0.8365

Table 4.1: RNN Results

One interesting phenomenon in the experiment result is that although the AUCs of the RNN models are close to that of the best traditional model (full features model), their F1 scores greatly exceed the full features model. To investigate the reason behind, a ROC curve of all three RNN models and the full features model are plotted in Figure 4.17, the dots in each curve represent the True Positive Rate (TPR)

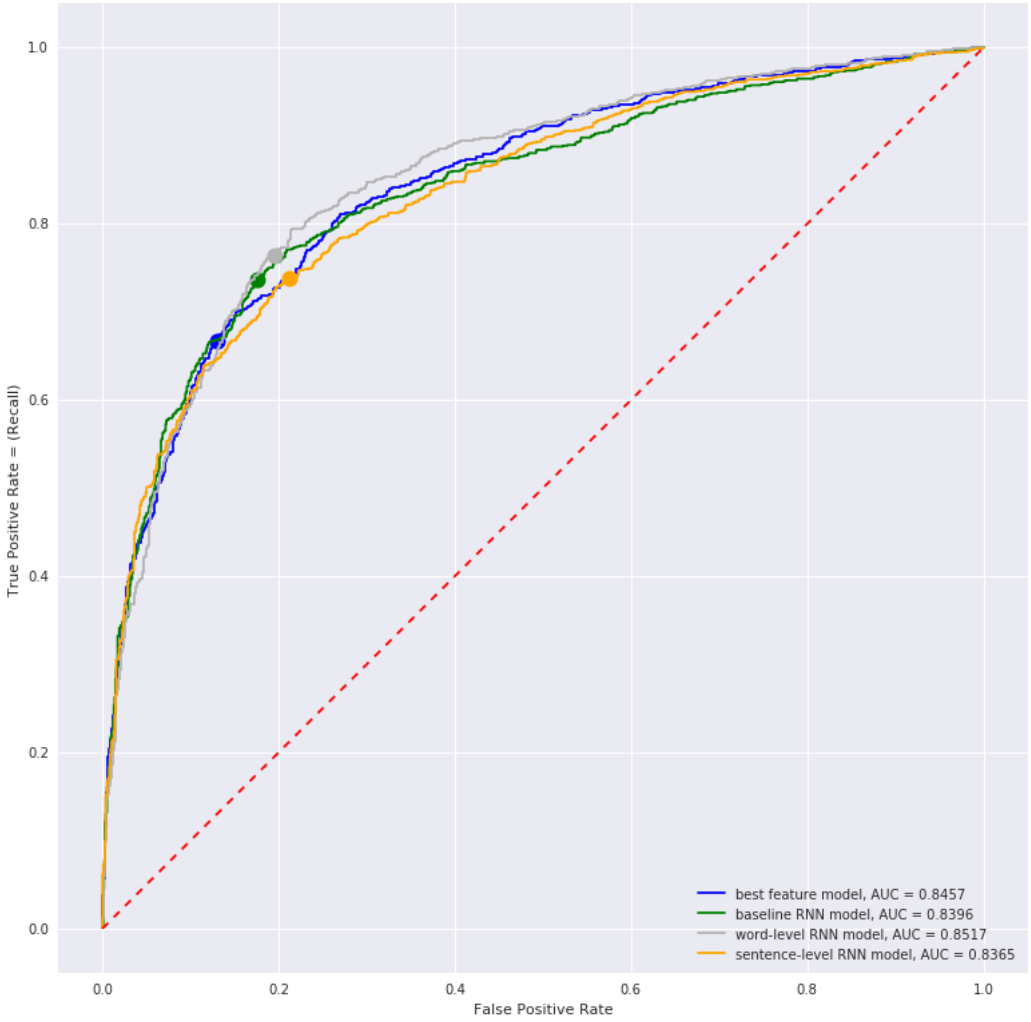


Figure 4.17: Receiver Operating Characteristic of models

and False Positive Rate (FPR) of the predicted test set of each model. It can be seen that although the ROCs and AUCs of four models are relatively close to each other, the TPRs of the test predictions from RNN models are much higher than that of the traditional model. Since TPR is equivalent to recall:

$$TPR = \frac{\textit{True Positive}}{\textit{True Positive+False Negative}} = Recall$$

and

$$F1 = 2\frac{precison*recall}{precison+recall}$$

the low TPR (=Recall) of traditional model test prediction might be the reason of its low F1 score, which indicates that the model is more likely to misclassify the insulting comments as normal comments, therefore if the objective of an insulting comment detection system is to filter as many insulting comments as possible, the RNN models might be a better choice compared to the traditional models.

Chapter 5

Ensemble

Model ensembling is a useful technique to improve accuracy on machine learning tasks. Figure 5.1 illustrates the general structure of model ensembling: first creates a series of base models, and then uses one ensemble strategy to combine all generated base models.

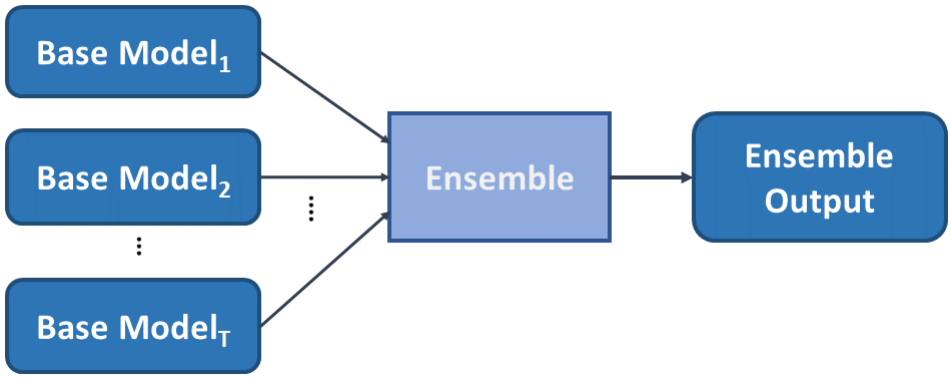


Figure 5.1: Ensemble Structure

The performance of ensemble depends on the base models' accuracies and diversity. Considering a simple example: in a binary classification task, we have three classifiers h_1, h_2, h_3 and their performance on three test samples t_1, t_2, t_3 is show in Table 5.1, where ✓/✗ indicates the classification on the corresponding test sample is right or wrong. The performance of ensemble depends on the majority decision. In Figure 5.1 (a), each base classifier has 66% accuracy and predictions of h_1, h_2, h_3 on three test samples are different, the ensemble increases the accuracy to 100%; in Figure 5.1 (b), each base classifier has still 66% accuracy but the predictions are identical, the ensemble has no effect this time; in Figure 5.1 (c), each base classifier has only 33% accuracy and the predictions are different, however this time the ensemble decreases the accuracy to 0%. This simple example shows that in order to get a good ensemble result, base models needs to be both accurate and diversified [32].

	t_1	t_2	t_3
h_1	✓	✓	✗
h_2	✗	✓	✓
h_3	✓	✗	✓
ensmeble	✓	✓	✓

(a) ensemble has positive effect

	t_1	t_2	t_3
h_1	✓	✓	✗
h_2	✓	✓	✗
h_3	✓	✓	✗
ensmeble	✓	✓	✗

(b) ensemble has no effect

	t_1	t_2	t_3
h_1	✓	✗	✗
h_2	✗	✓	✗
h_3	✗	✗	✓
ensmeble	✗	✗	✗

(c) ensemble has negative effect

Table 5.1: The performance of ensemble using different base models

In this project, I choose the best traditional model (full features model) and all three RNN models as the base models, and use simple average, weighted average and stacking as the ensemble strategies to further improve the prediction accuracy.

5.1 Simple Average Ensemble

Simple average ensemble is one of the simplest ensemble strategies. Assuming there exist T base models $h_1, h_2, ..., h_T$, and the output of h_i on input x is $h_i(x)$, the final output $H(x)$ of a simple average ensemble is:

$$H(x) = \frac{1}{T}\sum_{i}^{T} h_i(x)$$

The results of the simple average ensemble are shown in Table 5.2. Simple average ensemble is first implemented using the word-level RNN and another model from the rest three models as base models. The similarity here is measured using cosine similarity between the predicted probabilities of two base models, and the size of similarity is inversely proportional to the base models' diversity. The simple average ensemble result is consistent with the theory. Simple average ensemble using word-level + full feature as base models gives the highest diversity and the best result. Its test AUC even exceeds that of the simple average ensemble using all four models.

		Validation		**Test**	
Base Models	*Similarity*	*F1*	*AUC*	*F1*	*AUC*
word-level + baseline	0.9296	0.7646	0.9335	0.7699	0.8601
word-level + sentence-level	0.9296	0.7583	0.9312	0.7786	0.8586
word-level + full feature	**0.8938**	**0.7837**	**0.9422**	0.7797	**0.8706**
All four models	/	0.7712	0.9422	**0.7814**	0.8695

Table 5.2: Simple Average Ensemble Results

5.2 Weighted Average Ensemble

Weighted average ensemble is another average ensemble strategy. For same base models $h_1, h_2, ..., h_T$ and input x, the final output $H(x)$ of a weighted average ensemble is:

$$H(x) = \sum_{i}^{T} w_i h_i(x)$$

where w_i is the weighted of base model h_i, and $w_i \geq 0$, $\sum_i^T w_i = 1$.

The weights here is computed using the algorithm introduced in 'using selection ensemble from libraries of models' [33], the procedure of the algorithm is similar to the forward feature selection, except now we select models instead of features:

- Starting from the empty ensemble
- Adding in the ensemble a model from the base models which maximise the performance on the validation set.
- Repeating the above step for a fixed number of iterations.

In each iteration, the new base model $h_c(x)$ is added into the ensemble with ratio 1:9, so that:

$$H_{new}(x) = 0.1 * h_c(x) + 0.9 * H_{old}(x)$$

The number of iterations is set to 50. Two weighted average ensembles are implemented using F1 score and AUC as the measurements of performance respectively when selecting the best added base model in each iteration. The results are shown in Table 5.3, using F1 as the measurement of performance achieves better result. Overall the weighted mean ensemble outperforms the simple mean ensemble.

	Validation		**Test**	
Performance Measurement	*F1*	*AUC*	*F1*	*AUC*
F1	**0.7906**	**0.9438**	**0.7849**	**0.8731**
AUC	0.7748	0.9438	0.7776	0.8724

Table 5.3: Weighted Average Ensemble

5.3 Stacking

Stacking is a strong ensemble strategy, it can achieve outstanding performance when the size of training data is relatively large. Stacking uses the predictions from base models as features to construct a second-layer model. The second-layer model's training data usually comes from the prediction of base models on the original unused training samples to prevent over-fitting.

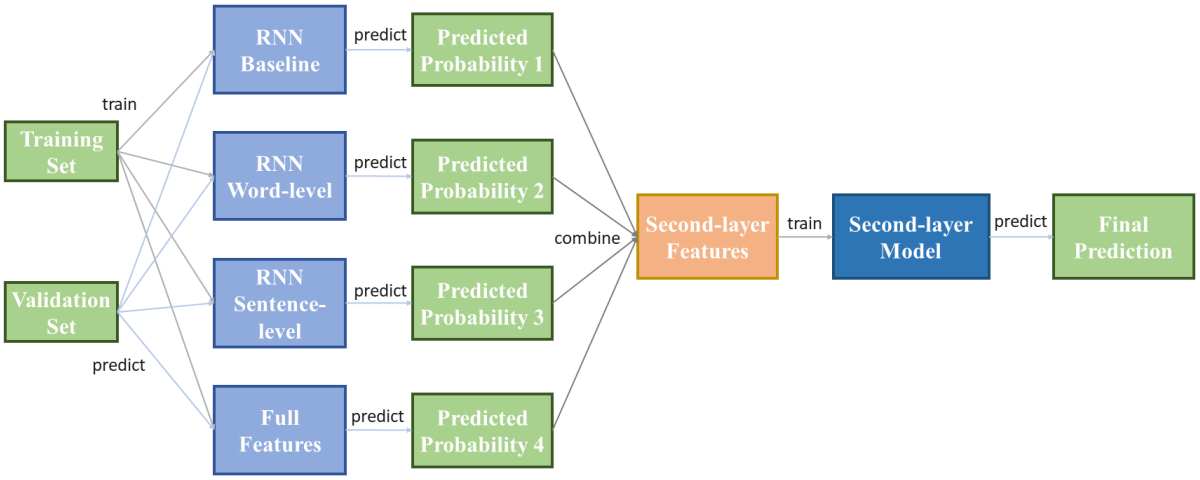


Figure 5.2: Stacking Structure

Here due to the computational resource limitation, I only use the predictions of base models on the validation set ((size = 659) as the training data of the second-layer model. Logistic regression (l_2 regularisation with cost = 2 and balanced weighted loss), Ridge regression (no intercept and cost = 10) and ElasticNet regression (no intercept, cost = 10, l_1 ratio = 0.5) are used as the second layer models. Ridge regression is the linear model which uses linear least squares with l_2 regularisation, while ElasticNet regression is the linear regression with combined l_1 and l_2 regularisation. Figure 5.2 illustrates the structure of the stacking ensemble used in this project.

The performance is shown in Table 5.4. Since the validation set is used as the second layer training set, the performance of stacking on the validation set is not comparable with other results thus is not included in the table. Using Ridge regression as the second-layer model gives the best test F1 while ElasticNet regression achieves the best overall performance. The test F1 of all three stacking ensembles are higher than that of the weighted ensemble, indicating the potential of stacking ensemble on improving the accuracy on the insulting comment detection.

	Test	
Second-layer Model	*F1*	*AUC*
Logistic regression	0.7794	0.8701
Ridge regression	**0.7913**	0.8691
ElasticNet regression	0.7823	**0.8727**

Table 5.4: Stacking Results

Chapter 6

Conclusion and Discussion

In this project I explore the offensive comment detection problem using a dataset from Kaggle. Both traditional feature-based models and Recurrent Neural Network models have been implemented. The traditional models consist of three single-type-feature models (lexicon-based, lexical and syntactic) and one full features model which contains all three types of features and some other supplementary features. The RNN models consist of the baseline LSTM model, the bidirectional word-level model with weighted sum pooling and the hierarchical sentence-level model.

The performances of all models constructed in this project, together with the winner's approach from the Kaggle competition, have been compared. It turns out that the best traditional model (full features model) and RNN model (word-level model) outperform the winner's model on the test AUC (0.8447/0.8516 vs 0.8425), and overall the RNN models achieves better F1 score than the traditional models.

Simple average, weighted average and stacking ensembles are also used to further improve the prediction accuracy. Stacking using Ridge regression as the second-layer model achieves the best result on test F1, while the weighted average ensemble achieves the best result on the test AUC which outperforms the winner's model by 3.06%.

All codes can be found in *https://github.com/zcakzwa/UCL_GraduateProject_InsultingCommentDetection/*. The final result is shown in Figure 6.1, it suggests that:

- The winner's approach in the Kaggle competition still has room of improvement, since the traditional model constructed in this project, without any ensemble method, achieves better AUC on the test set than the winner's model.
- The success of recurrent neural network on sentimental analysis can be transferred into offensive comment detection and achieve decent results.
- Ensemble methods especially stacking can improve the accuracy of offensive comment detection. Stacking method achieves decent result using only 10% of the training set to compute the second-layer features, indicating its tremendous potential on further improving the performance.

	validation		Test	
	F1	*AUC*	*F1*	*AUC*
Discriminative Models				
lexicon features	0.6359	0.8437	0.6476	0.7215
lexical featues	0.7252	0.9101	0.7043	0.8233
syntatic features	0.5097	0.6753	0.5212	0.6325
all features	0.7444	0.9191	0.7376	0.8457
winner's model	/	/	/	0.8425
Generative Models				
baseline	0.7552	0.9215	0.7629	0.8409
word-level	0.7650	0.9250	0.7739	0.8516
sentence-level	0.7469	0.9220	0.7583	0.8365
Ensemble				
Simple Average Ensemble				
word-level + baseline	0.7646	0.9335	0.7699	0.8601
word-level + sentence-level	0.7583	0.9312	0.7786	0.8586
word-level + full features	0.7837	0.9422	0.7797	0.8706
All four models	0.7712	0.9422	0.7814	0.8695
Weighted Average Ensemble				
F1 as performance measurement	**0.7906**	**0.9438**	0.7849	**0.8731**
AUC as performance measurement	0.7748	0.9438	0.7776	0.8724
Stacking				
Logistic regression	/	/	0.7794	0.8701
Ridge regression	/	/	**0.7913**	0.8691
ElasticNet regression	/	/	0.7823	0.8727

Table 6.1: Final Result

In the following section, I would like to discuss several limitations of this project related to both the dataset and the models, and possible further work.

6.1 Limitations

6.1.1 Dataset

During the experiment, it has been found that the dataset used in this project has problems of inconsistency and subjective labelling (section 2.1.3 and 3.5) which restrict the performance of models. In addition there are only 6594 training data in the Kaggle dataset. Due to the large number of parameters, the required number of the training data for deep neural network is relatively large, thus the small number of the training data here might further restrict the performance of recurrent neural networks.

6.1.2 Test Set

The performance of models in this project is determined by their F1 and AUC on the test set, mainly to compare my models with the models from the Kaggle competition. However there exists a large gap between models' performance on the validation and test sets, which indicates the inconsistency between the validation and test data. Also the class distribution of the test set is balanced, which is obviously

inconsistent with the ratio of normal and offensive comments in reality. Considering the above potential issues, only using the test set to evaluate the models' performance might not be appropriate.

6.1.3 Uninterpretability of RNN

Although the word-level RNN model outperforms the traditional feature-based models, its uninterpretable property is still a serious limitation. In tradition models, we can obtain a direct idea on how the model works by viewing the parameter coefficients of each features in a linear model, or feature importance in a tree-based model. However the features in recurrent neural networks are automatically generated, although I use some visualisation method to indirectly visualise part of the network's components (section 4.4), the direct mechanism of the network is still kind of a black box. This uninterpretability might raise the difficulty to apply RNN models on real-world applications, since the it is hard to convince users to choose a model if we can not explain how it works.

6.1.4 Hyperparameter Tuning of RNN

There exist many hyperparameters in the recurrent neural networks, including the word vector size, sentence LSTM hidden size, comment LSTM hidden size, attention size, dropout rate, activation functions in different layers, fully connected layer size in classifier, etc. Due to the time and computational resource limitations, only a small set of hyperparameter combinations have been tested. Therefore the current selection of hyperparameters may be still far away from its optimum, and the RNN models have not achieved their best performance yet.

6.2 Further Works

6.2.1 Different Datasets

Since there exists evidence that the dataset used in this project has several problems, one potential further work is using different datasets to train the offensive detection models. In 2017, Davidson et al. [34] publish a new dataset which contains 24,802 tweets labelled as hate speech, offensive language, or normal. Each tweet within this dataset was labelled by three or more people, and the final label assigned to each tweet is decided by majority decision. Further exploration on offensive comment detection can be implemented using a dataset like this, and the performances of constructed models can be compared using the experiment results from the papers.

6.2.2 Different Tasks

One of the advantage of recurrent neural network models is its strong transferring ability on different tasks. As long as the inputs are text information consist of words, we can use the same neural network structure on different tasks by simply modifying its classifier part. Considering the decent performance of RNN models in offensive comment detection, it would be interesting to explore its performance on different tasks.

6.2.3 Different Neural Network Structures

In this project I only explore the performance of recurrent neural network models on the offensive comment detection task. In recent years, other artificial neural networks especially the convolutional neural network (CNN) also achieve remarkable results on text classification tasks. In fact, in a machine learning competition I participated this year, the best single model approach is achieved using CNN model. Considering the power of CNN on text classification, it is worthwhile to explore the performance of CNN models on offensive comment detection.

6.2.4 Different Ensemble Strategies

As discussed, due to the time and computational resource limitations, in stacking ensemble only 10% of the training set are used to compute the second-layer features, which greatly restricts the stacking performance. Many stacking strategies use k-fold cross validation to compute the predicted probabilities on k validation sets, so that the combination of all validation sets can cover the full original training set and no information in the training set is missed. Using k-fold cross validation strategy might further improve the accuracy of offensive comment detection.

Bibliography

[1] Jacqui Cheng. Report: 80 percent of blogs contain "offensive" content. https://arstechnica.com/information-technology/2007/04/report-80-percent-of-blogs-contain-offensive-content/, 2007.

[2] E. Spertus. Smokey: Automatic recognition of hostile messages. *Innovative Applications of Artificial Intelligence*, page 1058–1065, 1997.

[3] D. Adamson P. Gianfortoni and C. Rose. Modeling of stylistic variation in social media with stretchy patterns. *Proceedings of the First Workshop on Algorithms and Resources for Modelling of Dialects and Language Varieties*, pages 49–59, 2011.

[4] D. Davidov O. Tsur and A. Rappoport. a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. *the Fourth International AAAI Conference on Weblogs and Social Media*, page 162–169, 2010.

[5] Ling Wang Jason I. Hong Carolyn P. Rose Guang Xiang, Bin Fan. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1980–1984.

[6] S Zhu H Xu Y Chen, Y Zhou. Detecting offensive language in social media to protect adolescent online safety. *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE.

[7] H. Mao J. Bollen and A. Pepe. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. *ICWSM, Barcelona, Spain*, 2011.

[8] O. Tsur D. Davidov and A. Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 241–249, 2010.

[9] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[10] Junbo Zhao Xiang Zhang and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, pages 649–657, 2015.

[11] Richard Socher Kai Sheng Tai and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[12] Bing Qin Duyu Tang and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. *Empirical Methods in Natural Languag Processing*, pages 422–1432, 2015.

[13] Chris Dyer Xiaodong He Alex Smola Eduard Hovy Zichao Yang1, Diyi Yang1. Hierarchical attention networks for document classification. *HLT-NAACL*, pages 1480–1489, 2015.

[14] Andreas Mueller. What do you use in detecting insults in social commentary. https://www.kaggle.com/c/detecting-insults-in-social-commentary/discussion/2744, 2012.

[15] tuzzeg. Kaggle competetion: Detecting insults. https://github.com/tuzzeg/detect_insults/blob/master/.

[16] Andrei Olariu. My first kaggle competition (and how i ranked 3rd). http://webmining.olariu.org/my-first-kaggle-competition-and-how-i-ranked/.

[17] Andreas Mueller. Recap of my first kaggle competition: Detecting insults in social commentary. http://peekaboo-vision.blogspot.co.uk/2012/09/recap-of-my-first-kaggle-competition.html.

[18] Andrea Esuli and Fabrizio Sebastiani. sentiwordnet. http://sentiwordnet.isti.cnr.it/.

[19] Google. Bad words, swear words, offensive words, profanities banned by google, 2012. https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/.

[20] Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation, Vol. 28 Issue: 1*, pages 11–21, 1972.

[21] Wikipedia. Langugae model. https://en.wikipedia.org/wiki/Language_model.

[22] Stanley F.Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318, 1998.

[23] Wikipedia. Receiver operating characteristic. https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve.

[24] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural computation Volume 9, Issue 8*, pages 1735–1780, 1997.

[25] Patrice Simard Yoshua Bengio and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks Volume 5, Issue 2*, pages 157–166, 1994.

[26] Pascal Vincent Yoshua Bengio, Rejean Ducharme and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, page 1137–1155, 2003.

[27] Greg Corrado Jeffrey Dean Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[28] Christopher D. Manning Jeffrey Pennington, Richard Socher. Glove: Global vectors for word representation. *Empirical Methods in Natural Language Processing*, 14, 2014.

[29] Ryan Kiros Aaron Courville Ruslan Salakhutdinov Richard Zemel Kelvin Xu, Jimmy Ba and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning*, pages 2048–2057, 2015.

[30] Kyunghyun Cho Dzmitry Bahdanau and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[31] G.E. Van der Maaten, L.J.P.; Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, pages 2579–2605, 9 Nov 2008.

[32] Zhihua Zhou. *Machine Learning*, chapter 8, pages 171–173. Tsinghua University Publisher, 1 edition, 1 2016. ISBN 978-7-302-42328-7.

[33] Geoff Crew Alex Ksikes Rich Caruana, Alexandru Niculenscu-Mizil. Ensemble selection from libraries of models. *Proceedings of the twenty-first international conference on Machine learning.*, page 18, 2004. ACM.

[34] Michael Macy Thomas Davidson, Dana Warmsley and Ingmar Weber. Automated hate speech detection and the problem of offensive language. *arXiv preprint arXiv:1703.04009*, 2017.