

# Detecting Insincere Questions

Recognizing provocative and toxic content to improve online conversations

Theodoros Konstantinidis  
Computer Science Department  
Aristotle University of Thessaloniki  
Thessaloniki, Macedonia, Greece  
theodorosk@csd.auth.gr

George Georgiou  
Computer Science Department  
Aristotle University of Thessaloniki  
Thessaloniki, Macedonia, Greece  
georgiougk@csd.auth.gr

Panagiotis Papaemmanouil  
Computer Science Department  
Aristotle University of Thessaloniki  
Thessaloniki, Macedonia, Greece  
ppapaemm@csd.auth.gr

## ABSTRACT

For this study, inspiration was drawn from the Kaggle community. The Kaggle competition chosen is the famous Quora Insincere Questions Classification, held in 2018 with more than 4000 teams participating. Quora's challenge was to be able to maintain a respectful and inclusive community by barring any mischievous content (racism, trolling, toxicity, etc.) without the need for 24/7 human resource availability.

The project's goal is to develop machine learning models that allow for automatic content flagging, thus leading to their easier or automatic removal. The data used were provided by Quora, a sample of more than 1.3 million questions in the English language already flagged for insincerity.

Feature engineering is performed by using classic text mining methods, embeddings, and exploratory data analysis (EDA). In the current scope, various binary classification Machine Learning algorithms within the *scikit-learn* and *Keras ML frameworks* are examined.

Evaluation of the implementation(s) will be done based on the **F1-score** between predicted and observed targets in the test dataset.

## KEYWORDS

Natural Language Processing, Text mining, Kaggle competition, Quora, Machine Learning, Neural Networks, Toxic content, Python, scikit-learn, Keras

## Github repository

<https://github.com/papaemman/NLP-AUTH>

## 1 Introduction

### 1.1 Quora

In the Wikipedia article about Quora, it is stated that: "*Quora is a question-and-answer website where questions are asked, answered, followed and edited by users*". Questions can be edited and edit to answers can be suggested by multiple users, thus resulting in a collective knowledge-sharing platform. Since its launch in June 2010, it has grown to be visited by 590 million unique people every month. With such a high number of users, monitoring for toxic content is required, however, it is humanly impossible to do so. That is where machine learning jumps in.

### 1.2 Toxic content

Toxicity is an ever-spreading phenomenon, emerging in online communities where a screen offers partial anonymity. Content labeled as toxic includes but is not limited to, general trolling, racism, hate speech, and sexism. In the early days of Internet culture, people created false personas to integrate into online communities and ultimately derail group conversations. These people were described by the well-known term "Internet trolls" or just "trolls". However, their behavior evolved from just mischievous trickery to actual harassment, sometimes even reaching the levels of all-out verbal warfare.

The amount of information exchanged has risen so sharply over the last few years that, even proportionately, the rate of toxic content being posted has followed suit, resulting in a constant, uncontrollable surge of toxicity on the Internet. This phenomenon can no longer be dealt with by human moderators, making the need for systems capable of automatic recognition of such content greatly present.

### 1.3 Dataset

The dataset used was provided by Quora, through the "Quora Insincere Question Classification" Kaggle competition page. It consists of three CSV files and a ZIP

archive. The latter contains pre-trained embeddings, while the three CSV files correspond to train and test data and sample submission for the competition. The test data and sample submission CSV files were omitted at the training stage as non-exploitable.

The ZIP archive's pre-trained embeddings are derived from Google News, Stanford University's Glove Project, University of Pennsylvania's Cognitive Computation Group, and Wikinews. Each one contains 300-dimensional vectors corresponding to words. The train data CSV file of 1.306.122 rows contains three columns: "*qid*" as the unique question identifier, "*question\_text*" as the actual text being processed for information, and "*target*" as the value flagging a question as insincere/toxic.

## 2 Related Work

Classifying questions, and content in general, as insincere is a task that pertains to a number of different definitions according to context. As such the range of related tasks and topics on which similar work applies is rather large. Some of these include the classification of toxic comments, recognizing hate speech, cyberbullying, and online harassment as well as identifying provocative and troll content.

The works presented, in a non-exhaustive manner, in this chapter, are indicative of research that mainly targets recognizing troll content and toxic online comments rather than outright hate speech and/or cyberbullying, as we believe this task is more closely related to our own.

The [Aken, Betty Van, et al] address the problem of toxic comment multiclass classification on 2 datasets, one provided by Google Jigsaw on Kaggle's Toxic Comment Classification Challenge and a second one consisting of annotated Twitter data. The authors apply a number of methods ranging from Logistic Regression to bidirectional LSTMs and GRUs, also utilizing Glove and FastText embeddings, and find that an ensemble of all the methods, weighted accordingly, achieves the best results. This attempt also presents an in-depth error analysis, recognizing the most common challenges such as doubtful labels, quotations, irony, etc.

In [Luis Gerardo Mojica de la Vega et al.] the authors attempt a comprehensive trolling categorization based on Intention, Disclosure, Interpretation, and Response. Logistic regression is used as a classifier, using n-gram features and Glove embeddings. For the research purposes, an

annotated dataset derived from Reddit comments was created and made publicly available.

In [Mihaylov, Todor, and Preslav Nakov] two distinct binary classification problems are addressed. Based on data crawled from Bulgarian community forums the authors try to determine both whether a comment can be constituted as "troll" as well as determine whether it was created by a paid troll, a mentioned troll, or a regular user. The research utilizes a number of different features, most interestingly including metadata on the comment (such as the time of posting) and some rudimentary sentiment analysis, and then evaluates how different features contribute to the results.

As annotated toxic content datasets often exhibit data scarcity as well as imbalance across samples, in [Juuti, Mika, et al.] data augmentation attempts are made to remedy that. Beginning from Kaggle's Toxic Comment Classification Challenge dataset, the initial data are processed via various methods such as substitutions from knowledge bases, substitution from neighboring embeddings (i.e Glove, BPEmb), majority class addition, and GPT-2 conditional sample generation. The results we evaluated using character and word-based Logistic Regression as well as a word-based CNN and the pre-trained uncased BERT. The authors find that data augmentation can lead to comparable results to highly complex models, while significantly reducing computational intensity. The optimal results were obtained through GPT-2 sample generation.

More loosely related work is presented at [Miao, Lin, et al] where the focus is on recognizing troll content in a bilingual (English and Russian) setting. Machine Translation along with text features and deep learning methods such as CNNs and RNNs are utilized for this purpose. The results point towards bilingual learning as opposed to either monolingual or cross-lingual, for better results.

The use of different Deep Neural Networks for toxic comment detection is examined in [D'Sa, Ashwin Geet, et al.]. Binary classification is performed on the Wikipedia Detox Corpus, with some data preprocessing, mainly for computational reasons. A CNN, a bi-LSTM, and a bi-GRU network were used alongside BERT fine-tuning. Various embeddings were also used ranging from one-hot approaches to Mikolov's and FastText embeddings. Optimal results were obtained via fine-tuning the BERT model that also proved to be the most robust to word appending attacks, regarding robustness.


An established capsule network for the classification of 2 different datasets (Kaggle's Toxic Comment Classification Challenge and TRAC) was used in [Srivastava, Saurabh, et al.]. The proposed model consists of 4 layers, the word

### 3.1.2 Word Clouds

Generating word clouds for each class, after of course removing irrelevant tokens such as stop words, can help more clearly illustrate the dominant tokens in each class.

[illegible]

The term “India” appears in both word clouds, most probably denoting the bulk of Quora’s audience. The term “think” is also present in both clouds, which is to be expected for a question answering service, in all cases. Other than that, the dichotomy in the content is rather obvious to a human observer.

[illegible]

A bar chart titled 'Count' on the y-axis and 'Class' on the x-axis. The y-axis has a multiplier of  $10^6$  at the top. The x-axis has two categories: 0 and 1. The bar for Class 0 is blue and reaches a value of approximately 1.25 on the y-axis. The bar for Class 1 is orange and reaches a value of approximately 0.1 on the y-axis.

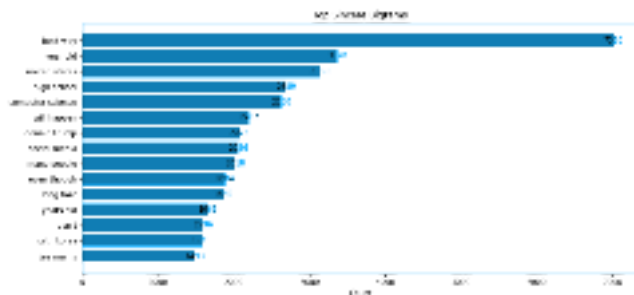
Class	Count
0	1.25
1	0.1

The sincere questions word cloud presents fairly expected themes, like “best”, “use”, “difference”, “work” etc most likely linked with questions about alternatives and options that regularly appear in question answering services.

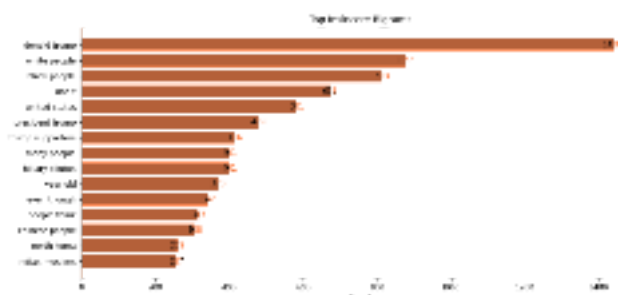
Insincere questions heavily involve politics with terms like “Trump”, “American” and “Liberal”. Themes that may be linked to racist or misogynistic remarks like “Muslim”, “China”, “women” and the dominant term “people” that is most likely involved in bigrams such as “black people”, “white people” etc.

### 3.1.3 Dominant unigrams & bigrams

In an additional attempt to better illustrate the topics most apparent in each of the dataset classes, the top unigrams and bigrams of each class were extracted. Observing the dominant bigrams of each class, can perhaps more accurately than the word clouds convey the key points of most questions.



Some demonstrative examples are “white people”, “black people”, “chinese people” in the insincere class. The innocuous word “people” that appeared in the word cloud, is now much clearer that was used in context to race, being indicative of the toxic content in the dataset questions.



## 3.2 Feature Engineering

Before extracting relevant features from the dataset, some basic preprocessing was performed, mostly using the Stanford NLTK. The question text was tokenized with word\_tokenize and punctuation was removed from each question. Stopwords were also removed, based on the ‘English’ stopwords provided by NLTK. Finally, using the PorterStemmer all remaining tokens were stemmed. This final transformation was kept separately as a feature column since stemmed words can lose information and are not always desired. Finally, we apply a spell correction algorithm to deal with misspelled words.

### Text Preprocessing steps

1. lower Case
2. nltk Tokenize
3. strip Punctuation
4. remove Stopwords
5. stem with PorterStemmer
6. spelling Correction

#### 3.2.1 Spelling Correction

In order to deal with misspelled words, we apply a custom spelling correction algorithm, based on the fasttext pre-trained word embeddings dataset.

#### Algorithm:

**Step 1:** Use words from fasttext as vocabulary

**Step 2:** From the given dataset (i.e. Quora’s questions) find words outside the vocabulary

**Step 3:** Generate possible spelling corrections for the unknown words, finding minimum edit distance words from fasttext vocabulary

- All edits that are one edit away from the word
- All edits that are two edits away from the word

**Step 4:** Replace with the most probable spelling correction for the word. Note: Use the fasttext ordering to infer the probability of each word.

This method was extremely helpful because was used to correct the misspelled words, but at the same time increased the coverage between our vocabulary and the pre-trained word embeddings vocabulary at 100%.

#### 3.2.2 Classic Feature Engineering

The classic features that can be extracted from a text concern metrics such as the number of words, characters, and sentences found within the corpus, and average values, e.g. average word length. The current study’s scope includes extracting the following features from each question:

1. Number of words
2. Number of unique words (based on the previous)
3. Number of characters
4. Number of characters after removing punctuation and spaces (serves as a variable for average word length but can be used for other features as well)
5. Number of stopwords (uses the number of words)
6. Number of punctuation marks
7. Number of uppercase words

8. Number of title case words
9. Number of sentences
10. Average word length (also uses a number of words)

The script for feature extraction is written in Python and makes use of the **pandas**, **re**, **nltk**, **string**, and **time** libraries, the latter being used for measuring the script's performance. The number of words and unique words uses the same regular expression code, with the slight difference that, for the number of unique words, the code is wrapped in an NLTK frequency distribution function, the length of the resulting dictionary being returned as a result. Counting characters is a trivial issue, solved by acquiring the length of each question, unlike counting a question's characters without taking into account spaces and punctuation, which requires the usage of a "translation table", which removes said characters beforehand.

Stopwords are words that are filtered out before or after processing natural language data. Usually referring to the most common words in a language, but there is no universal list of stopwords. For the scope of this study, however, the NLTK list of stopwords is used for filtering question texts. The number of stopwords in a question occurs by subtracting the number of words after filtering out any stopwords found in the stopwords list from the total number of words, as counted above.

Likewise, the number of punctuation marks was calculated by counting a question's length and subtracting the length of a "translated" version of the same text but using a different "translation table" which does not remove spaces.

Uppercase and title case words are counted by splitting a sentence into individual words and checking for each one if it is an uppercase or title case word.

For the number of sentences, the sentence tokenization feature of NLTK was used.

The average word length for each sentence is a matter of dividing the number of letter characters by the number of words. Within the given dataset, there was one question with zero characters, which resulted in a *division by zero* runtime error, mitigated by using a ternary conditional operator, which, if there are no words found, returns zero.

Running on a Windows 10 computer with an octa-core AMD Ryzen 7 4800H processor and 8GB of 3200MHz DDR4 RAM, the average execution time is about 120 seconds, from loading the libraries to extracting the initial dataset with the additional columns appended.

Apart from using hardware capable of high-speed processing, the performance was also achieved with the usage of lambda expressions for each action on the dataset

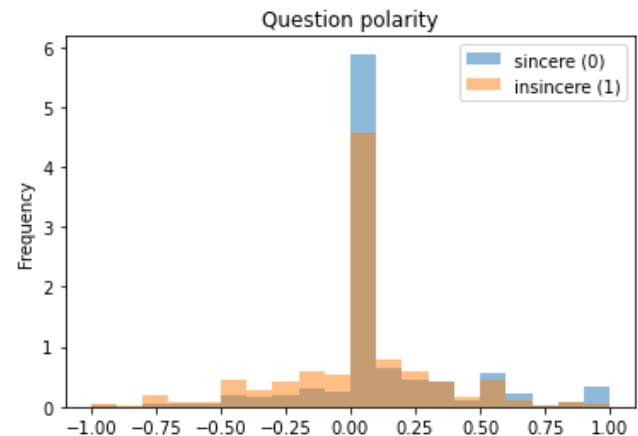
and careful implementation of the functions within the expressions. Some variables were also stored in memory instead of being calculated in every run of the lambda function, resulting in faster access to data (e.g. number of characters without punctuation or spaces, "translation tables") and use of data already calculated and appended to the dataset was also made (number of words).

### 3.2.3 Extended Feature Engineering

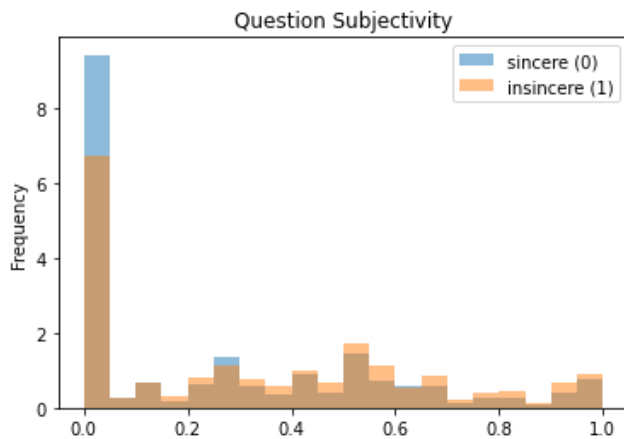
After establishing the base, classic features; some less common features were extracted from the dataset in an attempt to better distinguish the two target classes.

Utilizing the [TextBlob](#) library, which provides a pre-trained pattern analyzer we calculated the **polarity** and **subjectivity** of question text for each class.

**Polarity** returns as a float in the  $[-1.0, 1.0]$  interval denoting the provided text as negatively or positively polarizing. A value of 0 denotes a neutral polarity. As can be observed in the corresponding figure, while the target classes are not very distinguished, insincere questions demonstrate a higher tendency to have a negative polarity, with the opposite being true for sincere ones.

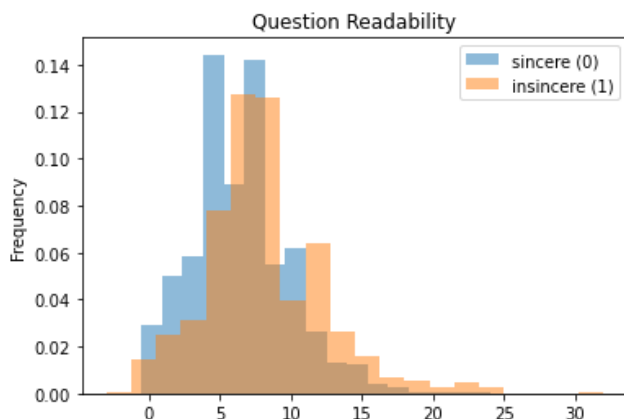


**Subjectivity** as a float in the  $[0, 1]$  interval denotes text that is more objective close to values of 0 and more subjective close to values of 1. The subjectivity of the dataset questions is less easy to distinguish between classes, although in a small margin insincere questions do tend to be more subjective.



Another feature that could prove helpful in separating the target classes is the **readability** of the question text. The library [textstat](#) offers a number of different readability indexes that roughly translate to the required reading ability, in terms of a school class, in order to understand the given text. Some examples of these include, but are not limited to, the SMOG Index, the Automated Readability Index, or the Linear Write Formula.

We have utilized the library-provided ensemble method that combines and averages the different readability metrics. Values represent the needed academic level to understand the scored text. As such, higher readability scores translate to poor readability and text that is, in general, more difficult to read. As seen in the relevant graph insincere questions tend to be less readable, perhaps due to a more convoluted meaning and target



### 3.3 Vector Semantics and Embeddings

To apply any Machine Learning method for our Toxic Identification NLP task, the first step we need to perform is to derive features based on the dataset, which consists of questions in natural language. In the previous paragraph, we describe several methods to extract features from raw text based on simple heuristics and traditional text mining approaches. In this paragraph we will focus on a different strategy, to achieve this vector representation, namely **Vector Semantics**.

Before we dive into technical details, it is wise to describe the intuition behind the vector representation of words. The objective for such a task is to create a numerical vector that will achieve to capture the meaning of the word. Therefore we need a definition of the "meaning of the word". The philosopher Ludwig Wittgenstein states that **"the meaning of a word is its use in the Language"**.

Based on this powerful idea, Linguists define a word by its environment or distribution in language use. **A word's distribution is the set of contexts in which it occurs, the neighboring words or grammatical environments**. The idea is that two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning, therefore these methods are based on cooccurrence calculations.

There are 2 general methods to create Vector Semantics. The first method results in very long and sparse vectors (**TF-IDF**), while the second one constructs short and dense vectors (**word embeddings**). For this project experiments were conducted based on both methods, therefore we will describe them both. However, we focused more on word2vec methods because 4 pre-trained models were available for competition participants, from the organizers.

#### Notes

- Vectors for representing words are generally called **embeddings** because the word is embedded in a particular vector space.
- For this task, any reference to a "document" implies a distinct Quora's question.

#### 3.3.1 Term Frequency - Inverse Document Frequency (TF-IDF)

This is the simplest method to create Vector Semantics for words and often acts as a baseline for more sophisticated techniques.

The intuition behind this method is to create similar vectors for words appearing in the same context frequently i.e. in the same questions, but at the same time to reduce noise for very frequent words that appear almost in every question.



**TF-IDF** is a product of 2 terms, as the name implies:

- **TF = count(t,d)**: term frequency – frequency of the word  $t$  in document  $d$ .
- **IDF**: inverse document frequency.

**IDF** is defined using the fraction  $N/df$  where  $N$  is the total number of documents in the collection, and  $df$  is the number of documents in which term  $t$  occurs.

The TF-IDF vectors' length is equal to that of the **vocabulary**  $|V|$  and, as can be understood, they are sparse, i.e. they have few non-zero values.

### 3.3.2 Word embeddings

Even though the TF-IDF method achieves a reasonable vector representation of words, the constructed vectors are long and sparse. This kind of feature vectors is not optimal for Machine Learning methods, because they force the classifier to learn too many parameters (weights) and often lead to overfitting.

To avoid this, researchers came up with a different set of approaches, called **word2vec**.

Some popular word2vec methods are **skip-gram with negative sampling (SGNS)**, **GloVe (Global Vectors for word representation)**, **fasttext**, etc.

The intuition of word2vec is that instead of counting how often each word  $w$  occurs near each other, we'll instead train a classifier on a binary prediction task: "**Is word  $w$  likely to show up near another word  $q$ ?**". We don't care about this prediction task; instead, we'll take the learned classifier weights as the word embeddings. This modeling allows us to use running text to create training examples.

The dimensionality of dense vector models ranges between 50-1000 and dimensions are harder to interpret. Word2vec algorithms (e.g. skip-gram) are a popular and efficient way to compute dense embeddings.

The available pre-trained models are described below:

- **GoogleNews-vectors-negative300**  
<https://code.google.com/archive/p/word2vec/>

This tool provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing vector representations of words.

Skip-gram trains a logistic regression classifier to compute the probability that two words are 'likely to occur nearby in a text'. This probability is computed from the dot product between the embeddings for the two words. Skip-gram uses

stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.

The process is as follows:

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the regression weights as the embeddings.

The intuition of the skip-gram model is to base this probability on **similarity**: a word is likely to occur near the target if its embedding is similar to the target embedding.

- **glove.840B.300d**  
<https://nlp.stanford.edu/projects/glove/>

**GloVe** is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

- **paragram\_300\_sl999**  
[https://cogcomp.org/page/resource\\_view/106](https://cogcomp.org/page/resource_view/106)

Paragram embeddings associated with the paper "*From Paraphrase Database to Compositional Paraphrase Model and Back*".

- **wiki-news-300d-1M**  
<https://fasttext.cc/docs/en/english-vectors.html>

1 million word vectors trained on Wikipedia 2017, UMBC web base corpus, and statmt.org news dataset (16B tokens). **Fasttext** computes word embeddings by summing embeddings of the bag of character  $n$ -grams that make up a word.

### 3.3.3 Similarity measures

Cosine similarity is a measure applied to vector semantics vectors presented earlier to quantify the "similarity" between different words.

It's defined as  $\frac{a \cdot b}{|a| |b|} = \cos(a, b)$

The  $\cos(a, b)$  is the dot product of the vector representation of the 2 words, normalized by their length.

## 4 Machine Learning methods and Evaluation

Multiple approaches were tested for the solution to the classification problem. These range from traditional machine learning models, in conjunction with the different feature sets, to complex deep neural network architectures such as biLSTMs and GRUs.

### 4.1 Evaluation Methods

In order to evaluate each approach, we utilized 3-fold cross-validation, splitting our train dataset into 3 equal parts and dividing into 2 parts of training and 1 validation. The validation set was, of course, shuffled each time and the metrics were averaged amongst each result. Given that the test set was unlabeled and thus proved difficult to check against, we only utilized it with our best performing, from the cross-validation phase, models.

All experiments were carried out on hardware specified on the corresponding table.

#### Available Hardware - Machine Specifications

Model name	Asus TUF Gaming A15
OS	Ubuntu 18.04.5 LTS
Processor	AMD Ryzen 7 4800H CPU @4.20GHz (8 cores, 16 threads)
GPU	NVIDIA GeForce RTX 2060 6GB
Memory	40 GB

As the dataset is heavily imbalanced, accuracy would prove to be an ill-chosen metric. Due to the accuracy paradox, a completely naive classifier that always predicts the sincere class would falsely yield a high accuracy score.

The F1 score, the harmonic mean of precision and recall is a suitable metric for models classifying imbalanced binary data and thus was the metric of choice.

### 4.2 Classical ML methods

We trained and evaluated 4 different traditional machine learning models. An **L2-regularized Logistic Regression**, a **Naive Bayes classifier**, a **Random Forest**, and an **XGBoost classifier**. The models were used as implemented in the *scikit-learn* and *xgboost* python libraries.

We trained our models with 3 different feature sets. The first one was the hand-engineered features described in earlier chapters. Small adjustments were made to accommodate model restrictions, i.e. text polarity was re-centered around the value of 1, rather than 0, as the Naive Bayes Classifier does not accept negative values. Next, the models were trained using the TF-IDF vector representation of the textual data. Finally, the question text as vectorized by the embeddings vectors was used. Due to the dimensionality of the embeddings, to effectively train our models, the embedding values were averaged for each question text, thus creating a 1x300 matrix for each question (row). This had immediate negative results on the performance of the models, which will be discussed.

### 4.3 Neural Networks

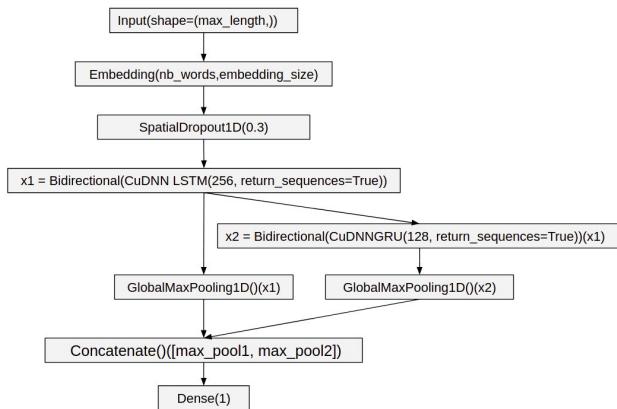
Apart from classical Machine Learning Methods, we also experiment with Neural Network models. We use the **Keras** library with TensorFlow as the backend to build and train our models.

At first, we attempted to tackle the problem using naive architecture, using an embedding layer, dropout, and some dense layers. After some round of experimentation, it became clear that we need to leverage more powerful and advanced layers in order for our model to be able to capture the sequential manner of the text data.

Our final architecture consists of an embedding layer, dropout, bi-LSTM, bi-GRU, Max pooling, and a Dense Layer.



## Detecting Insincere Questions



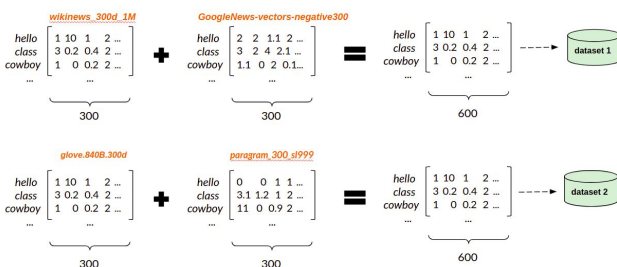
About the **Embedding layer**:

### 4.4 Final Solution

A major challenge for this competition was to find the optimal way to combine multiple pre-trained word embeddings models, in order to achieve the best F1-score.

After extensive experimentation, we manage to find a solution, concatenating the word vectors by 2 and using the newly created 600-dim word vectors for training. Therefore, we train 2 different Neural Networks models, using the 2 new datasets, each one consisting of combinations of 2 pre-trained embedding models.

A visual explanation of this process presented in the image below.

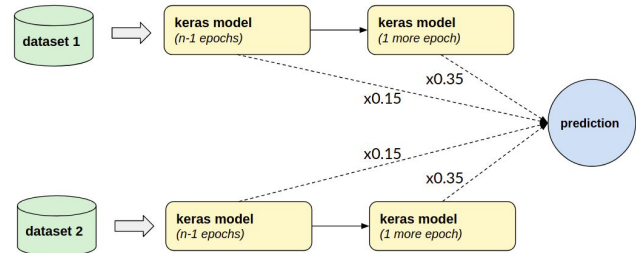


The final solution was a weighted sum of the predictions of the two models.

Another trick used here is that we derive predictions during 2 different stages of the training. At first, we let the model train for ***n-1 epochs*** and get the first prediction, and after that, we continue the training for ***1 more epoch*** and get the second prediction. The intuition behind this trick is to take

Data and Web Science MSc Program, January, 2020,  
Thessaloniki, Greece

into consideration a possible “under fitted” prediction and an “overfitted” prediction, for the final prediction.



The final set of optimal hyperparameters is:

Hyperparameter	Value
Optimizer	adam
Loss	Binary cross-entropy
Metrics	Accuracy
Activation	sigmoid
max_length	55
embeddin_size	600
learning_rate	0.001
batch_size	512
num_epochs	4

## 5 Results and Conclusion

Below can be seen the resulting scores from the 3-fold cross-validation executed on the algorithms.

### Typical Performance:

Evaluation Metric - f1 score

1.	Baseline ML models + Handcrafted Features	~ 0.20
2.	Baseline ML models + Embeddings (mean)	~ 0.45
3.	Baseline ML models + TF-IDF	~ 0.55
4.	Simple NN architecture	~ 0.60
5.	Final NN architecture + all embeddings (Competition winning model)	~ 0.70

Pre-trained embeddings (low dimensional and dense vectors) worked better than TF-IDF vectors (high dimensional and sparse). This is also supported by other studies, albeit with no mathematical evidence explaining the observed phenomenon. Handcrafted features are out of the question.

However, baseline Machine Learning models (i.e., non-Neural Network models) do not score well with the pre-trained word embedding vectors, since performing a calculation of the mean vector of all words in a question results in the loss of information, hence the low scores seen in the evaluation.

The TensorFlow Keras model achieves a good score, using the Embedding, LSTM, and GRU layers since it is by design specialized to work with embeddings. Even so, it is important that there be extensive coverage of the dataset by the pre-trained word embeddings. In this case, there were words spelled incorrectly or not found in the pre-trained embeddings' dictionary, a problem mitigated by a method of spell correction.

## Github repository

<https://github.com/papaemman/NLP-AUTH>

## REFERENCES

- Aken, Betty Van, et al. "Challenges for Toxic Comment Classification: An In-Depth Error Analysis." *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, 2018, doi:10.18653/v1/w18-5105.
- D'Sa, Ashwin Geet, et al. "Towards Non-Toxic Landscapes: Automatic Toxic Comment Detection Using DNN." *ArXiv.org*, 16 Sept. 2020, [arxiv.org/abs/1911.08395](https://arxiv.org/abs/1911.08395).
- Juuti, Mika, et al. "A Little Goes a Long Way: Improving Toxic Language Classification despite Data Scarcity." *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, doi:10.18653/v1/2020.findings-emnlp.269.
- Miao, Lin, et al. "Detecting Troll Tweets in a Bilingual Corpus." *ACL Anthology*, [www.aclweb.org/anthology/2020.lrec-1.766/](http://www.aclweb.org/anthology/2020.lrec-1.766/).
- Mihaylov, Todor, and Preslav Nakov. "Hunting for Troll Comments in News Community Forums." *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, doi:10.18653/v1/p16-2065.
- Srivastava, Saurabh, et al. "Identifying Aggression and Toxicity in Comments Using Capsule Network." *ACL Anthology*, [www.aclweb.org/anthology/W18-4412/](http://www.aclweb.org/anthology/W18-4412/).