A Comparison of Approaches for Insincere Questions Classification

Marvin Thiele
Hasso-Plattner-Institut
Potsdam, Germany
marvin.thiele@student.hpi.de

***Abstract*—The problem of toxic content detection is shifting more and more into the public sphere. Often, this content is present in a textual format. In this report, we want to focus specifically on the distinction between toxic and non-toxic questions. Solutions to this real-world problem will be evaluated against a dataset published by Quora, a website focused around asking and answering questions. Moreover, this dataset is part of a challene on the big-data competition platfrom Kaggle.**

This report describes techniques and method on all important steps of natural language processing pipeline such as preprocessing, vectorization and classification. The proposed pipelines show a performance in the range of the current state-of-the-art solutions using traditional machine learning approaches as well as deep-learning approaches.

I. Introduction

Many websites which offering user-driven content creation suffer from the problem having to deal with toxic and divisive content. Detecting and filtering this content is difficult, since the line between a good and bad is thin and often blurry. In this report, we want to especially focus on user-generated textual questions.

Quora is a platform where users can ask questions and have them answered by the community. The company offered a challenge on the big data competition website Kaggle to detect toxic and divisive questions. The challenge is notated with a price pool of 25.000$ and has currently more than 3400 competing teams, which can consist of up to 4 people.

The problem of text classification can be broken down into a few general steps. At first, the data must be cleaned and preprocessed. After that, it needs to be transformed into vectors. This step allows us to translate the text into a numerical format. The vectorized data is then used to train a machine learning algorithm, which learns the distinction between benign and toxic questions.

In this report, we will present multiple ways to perform each of these steps and discuss their advantages and disadvantages. Moreover, we will provide a detailed evaluation of these methods. Besides an extensive discussion of the preprocessing and the vectorization, the report also proposes several machine learning classifiers to solve the problem of insincere question detection.

II. Related Work

Question Classification is a problem which has been widely researched in literature. There have been approaches made using a variety of different classifiers. While some reports achieve state of the art performance using Support Vector Machines [1], other rely on neural network based architectures by employing Convolutional Neural Networks (CNN). [2] Similar problems like toxic comment detection have also been successfully solved by using more complex network structures like CapsuleNets. [3] In addition to these methods, the usage of, especially bidirectional, Long-Short Term Memory Networks (LSTM) has also to be noted. [4]

III. The Kaggle Environment

Kaggle is a platform where organizations can post challenges to the data-science community. These competitions are often rewarded by price pools ranging from 10.000$ to 100.000$, creating a strong incentive to develop a solution for the problem. After the competition is over, the organizations can then take these solutions to solve their specific issue.

Since the Kaggle platform strives for a fair competition, both hardware and data wise, the usage of so called ”Kernels” is necessary to submit a solution to the competition. A Kaggle kernel is very similar to a Jupyter Notebook, but can also be displayed as a Script. The developers are also granted processing resources such as CPU's and GPU's to run their algorithms. There are some restrictions when it comes to runtime, but these limits are high enough to no be disturbing. Another advantage of the system is that the developer is not required to download the, at times large, datasets.

The usage of these resources also means that this project was not limited by computing resources, which leads to fact that some code parts potentially take very long when not executed on the GPU version of TensorFlow with a high-end datacenter-grade graphics card.

IV. The data

To work on the challenge, Quora offers the competitors two different data files. There is a labled training data file with 1.306.122 labled data points and a non-labeled test dataset with 56.370 data points. The test dataset is used to calculate a score for the leaderboard, acting as a hidden test.

A data point consists of a question id, the question text and a label. Moreover, the dataset is highly unbalanced, since it contains 93.81% non-toxic questions. A toxic question is labled with the class 1. The training data has an average

question length of 12.80 words with a median of 11 words, which indicates that the questions are rather short in nature.

The solutions from the competing teams are ranked using the F1-score, which is often used in the natural language processing literature, since it concentrates on the, often in the minority, positive class. This means that when the dataset is strongly imbalanced, we still get a good performance metric. [5] Other measure like accuracy would give a wrong impression, since one can reach a very high accuracy without detecting a single toxic question.

In addition to the data itself, the challenge also gave access to several publicly available word embeddings such as Word2Vec, Paragram-300, GloVe & FastText. [6]–[9].

V. Cleaning the text

Common text cleaning methods are misspell correction, stop-word removal, removal of punctuation & symbols, stemming, removal of rare & very common words and lemmatization. Within our efforts, we have used several of these methods in order to improve the quality of the input text. At first, we will transform the sentence into lower-case, since it makes it makes a difference at the vectorization step later. After that, we will remove punctuation, special characters and symbols. This allows for a more unified spelling (unify ”don’t” and ”dont”) and to split words which are connected using symbols such as ”&” or ”/”. The next step is to remove common misspellings from the data. Instead of using a library for this task, we chose to only correct the most common misspellings and to transform the most common words which are not covered by the later explained word embedding. We came to this solution after evaluating the very long run-time of external misspelling correction functions.

We implemented just a subset of the most used preprocessing methods. This decision was mostly based on the later use of word embeddings. A word embedding is trained on a large set of text data, which means that is covers a large amount of words. For example, the word embedding GloVe covers both ”dont” and ”don’t”. The vector of both these words will be very close, which is a major difference to more simple techniques such as TF-IDF. Word embeddings give words a semantic, rather than just transforming them based on their frequency in the text like TF-IDF. This is the reason that we don’t need to transform ”writing” into it’s word stem ”write” by using stemming, since both version are represented in the embedding and have a close vector distance. We found that common word removal such as ”the”, ”what”, ”to” even decreased the performance of classification.

Approaches which need to be evaluated further include for example sentiment analysis, which calculates how subjective or how positive or negative a certain text has.

VI. Vectorization

Once the data is cleaned, the questions need to be converted into vectors. This step is necessary, since the machine learning classifiers can not natively work with textual data. There are several popular techniques to achieve this task. These range from simple approaches such as text tokenization & TF-IDF to complex pre-trained word embeddings like GloVe.

A. Tokenization

Tokenization can be described as the one of the most simple ways to convert a sentence in a vector. It works by assigning each unique word an identifying integer. The most common words have the lowest indexes in this format, which means that for example ”the” has the index 1, while rare words such as ”tokenization” have a high index (82151). This sorting allows to specify a cut-off, which means that rare words are not included in the tokenization. A tokenized sentence is then simply an array of word-indexes. This method is rarely used alone, but it used as a necessary step for using word-embeddings.

B. TF-IDF

TF-IDF stands for Term Frequency - Inverse Document Frequency. It works by ”calculat[ing] values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in.” [10]. Should a word with a high TF-IDF value appear in a document, it indicates that the word and the document are highly related. A practical example is that the word ”and” has a low TF-IDF value, since it appears in almost every document. When a word is rare, but used often in a document, it has a very high TF-IDF value, since these documents can most likely be described by that word. This method has been very popular in natural language language processing since the creation of IDF in 1972 by Karen Spärck Jones.

C. Word Embeddings

Word embeddings are an interesting technique used to transform words into a dense vector. While these can be learned on the text itself, the most common usage involves pre-trained word embeddings. These have the speciality that similar words semantically have a close vector distance to each other. For example, ”king” and ”queen” are closer related to each other than ”king” and ”tree”. Most word embeddings transform the input word into a 300-dimensional vector. This causes the embeddings themselves to be often very large. For example, GloVe takes 5,55 GB of space while Wiki FastText used 2,2 GB of storage.

Another interesting way of using word embeddings is unifying multiple of such embeddings to a so called meta-embedding. These can be created by simply taking the average of the different embeddings. Related work in these areas show that meta-embeddings can be beneficial in some use cases. [11]

When using word embeddings, one has many ways of transforming a sentence into a vector. While neural network architectures often transform every word into a vector and represent the sentence as a series of word-embedding vectors, one can also average all word-vectors form a sentence to use classifiers which do not usually work with time-series like data (here: SVM). [12]

VII. Overview over Classifiers

When the data was preprocessed and transformed using vectorization, we can input the resulting dataset into one of several classifiers. In this section we can the explain the general ideas behind the used classifiers and why they are relevant for natural language processing.

A. SVM

A SVM is a support vector machine, which is often used due to its good performance in many applications . On a simplified level, a SVM works by transforming the data using different kernels and then trying to linearly separate it. We chose this classifiers since it has a fast run time and represents the more classical machine learning algorithms.

We have implemented two approaches using a SVM classifier. The first one, "TF-IDF SVM", uses TF-IDF while the second classifier "MEMB SVM" uses mean-embedding vectorization (MEMB).

B. CNN

Convolutional Neural Networks (CNN) are usually used for image classification. They work by running many filters over an image which compress several pixels into one without majorly reducing the image size. Each of these learned filters creates a separate image, meaning that each convolution layer creates many images. To compress the many resulting images, the network uses so called max-pooling layers to compress the created images into a smaller size. This allows the network to detect bigger and bigger features of the images with the number of layers, since the original image information gets compressed into fewer and fewer pixels. In the end, all these images get concatenated and flattened, creating a large data array which is then evaluated by a dense layer with an activation function to get a classification.

One can also use these networks for classifying textual data. Conceptionally, we could image the network first detecting single words and then abstracting more complex structures such as sentences from the single words over the course of the convolution & max-pooling layers.

The proposed CNN first encodes the input data with the help of the GloVe embedding. Then the data will be given to three parallel convolution & max-pooling layers with different filter sizes. After this step, the many vectors created by the filters will be concatenated using an concatenation layer. To get a classification result, two dense, fully connected layers will provide a classification into toxic and non-toxic questions.

C. LSTM

Recurrent neural networks (RNN) are often used for time-series tasks. The one main point which differentiates RNN from normal feed-forward neural networks is that they introduce a memory. This allows that the first element from the sequence is modifying the reaction of the network to the next element in the sequence. An advancement of this idea is used in Long Short Term Memory (LSTM) networks. Another advantage of RNN-like networks is that they do not require a fixed input length like many other neural networks.

The proposed LSTM has a similar architecture to the CNN proposed in the section before. First, we have an embedding layer followed by parallel LSTM units. After that, the result will be concatenated again and handed into fully connected layers to get a classification result.

D. Ensemble Learning and CapsuleNet

To give our classifiers a strong competition, we also evaluated two approaches takes from the Kaggle community. Both approaches use a meta-embedding in combination with different classifiers. The first of both is using an ensemble formed from various CNN & LSTM classifiers, while the second one is using Capsules in combination with Gated Recurrent Units (GRU), an architecture partly developed by the famous machine learning researcher Geoffrey Hinton. Both approaches are well researched and widely used. [13] [14]

VIII. Results

To evaluate the approaches and algorithms presented in this report, we want to validate the effectiveness of our methods in several different areas.

A. Choice of Word Embeddings

Method	Rank Run 1	Rank Run 2	Rank Run 3	Mean Rank
GloVe	1	4	1	2.00
Fast	7	6	7	6.66
Para	6	3	6	5.00
GF	5	7	3	5.00
GP	4	1	5	3.33
FP	2	5	4	3.66
ALL	3	2	2	2.33

The challenge offers many different word embeddings. In addition to that there is also the possibility of combining multiple embeddings to create a meta embedding. To find a suitable embedding choice, we evaluated each embedding and every possible combination against the data. For the combined embeddings, the first letter of both is used as a name ("GP" means GloVe and Para). To generate the results, we took a CNN classifier and ranked the classification results by F1-score. The raw scores of the different classifier are very close together. The average standard deviation over the three runs is 0.0026, meaning that all embeddings have a very similar performance.

We can see that the GloVe embedding and all embeddings together show the best performance. Since taking all embeddings takes more loading time, we decided to go with GloVe as our embedding of choice.

B. Effect of Preprocessing on Word Coverage

Method	Unique Words	Covered Words	Coverage Percentage
Native	216044	121087	56.05%
Preprocessed	207559	124195	59.84%

In this table we can see the effect of the proposed preprocessing on the word coverage. The preprocessing reduced the total number of unique words, which means that the classifiers is exposed to less unique words. Moreover, we can also see that we can increase the amount of words which are covered by the embedding. The decrease of unique words means that we removed a lot of uniquely concatenated word combinations such as "university/school". The increase of covered words means that through this splitting of words, we discovered words which were not present in the dataset without being concatenated.

Both of these effects should be beneficial to the classification performance, since there are less words to learn in total and we can get more information from the sentences, since the embedding covers more words.

C. Analysis of Non-covered words

One may object that a coverage of about 59% is just slightly above half, but if we sort the words present in the dataset by occurrence, we can see that only 96287 words have more than one occurrence in the training set.

```
('demonetisation', 68),
('bhakts', 46),
('chsl', 41),
('josaa', 29),
('bahubali', 26),
('adhaar', 23),
('rohingya', 20),
('clickbait', 20),
('iisc', 19),
('demonitisation', 18),
('schizoids', 18),
('fortnite', 16),
('unacademy', 16)
```

When we loot at the words which are not covered by the GloVe embedding, we can see that they have a low occurrence count as well as being words which are not easily understood ("chsl") or region specific (India: "bahubali", "adhaar", "bhakts"). Modern words which established itself in the last few years such as "brexit", "cryptocurrencies" or "fortnite" are also often not covered. In total, the analysis of the not-covered words indicates that the word embedding is covering the most important words from the dataset.

D. Effect of Preprocessing on Classification

Method	Avergae F1	Std. Dev	Mean Rank
Native	0.6641	0.0059	1.7273
Preprocessed	0.6658	0.0064	1.2727

The consequences of using preprocessing before using a word embedding are not very clear. We can only see a very slight increase in classification performance when using a CNN classifier. We evaluated the performance over 11 runs and averaged the F1-score and calculated the standard deviation. Current research "shows that there is a high variance in the results depending on the preprocessing choice (±2.4% on average for the best performing model)". [15] Moreover, the report states that "word embeddings trained on multiword-grouped corpora [here: Word2Vec] perform surprisingly well when applied to simple tokenized datasets". This result is similar to our experience, where preprocessing doesn't majorly affect the performance of classification when using word-embeddings. The SVM in combination with TF-IDF also seems robust against the usage preprocessing.

E. Classification Results

To compare the different classifiers we have tested them against the same training-test split multiple times. It's important to note that the neural net classifiers perform differently based on training time, batch size and hyper-parameter tuning. We tried to keep the effects of those factors to a minimum to make the comparison fair.

Since the computation time of certain approaches is very long, we only performed three runs for each classifier. We chose the F1-score as the main performance metric, since the competition also uses this metric. Moreover, several other metrics such as accuracy are inaccurate due to the high class inbalance. To give a better overview, we also provided the average F1-score as well as the standard deviation.

Classifier	F1 Run 1	F1 Run 2	F1 Run 3	Mean	StdDev.
MEMB SVM	0.3676	0.4174	0.3658	0.3836	0.0293
TF-IDF SVM	0.4477	0.4535	0.4470	0.4494	0.0036
LSTM	0.6653	0.6704	0.6686	0.6681	0.0026
CNN	0.6640	0.6756	0.6697	0.6698	0.0058
Capsule GRU	0.6827	0.6830	0.6819	0.6825	0.0006
Ensemble	0.6910	0.6931	0.6916	0.6919	0.0011

Here we can see the different generated by the different classifiers on the same training-test split. To give an easy overview, we also included the mean F1-score as well as the standard deviation. The deep learning classifiers outperform the traditional approaches (SVM) by a large margin, while being closely grouped together.

When looking at the results, we can see that the proposed methods perform worse than approaches proposed by the community (Capsule GRU and Ensemble). To put this into perspective, we have to remind ourselves that these solutions were developed by top researchers & data engineers from all over the world. Most of these people have multiple Kaggle competitions under their belts and a strong background in data-science. This emphasizes that solutions close to these classifiers can still be considered a success.

To give these numbers some context, we can see that the current world record on the challenge is at an F1-score of 0.711 (accessed: 14.01.2019), meaning that the result presented in this report are competitive in nature. The best solution of more than 95% of the competitors have an F1-score of less than 0.70. It can also be noted that the results presented here were created using a training-test split, which usually gives slightly higher results than the real testing set.

IX. Conclusion

We see that the problem of binary question classification can be solved adequately by using state-of-the-art machine learning techniques.

While some approaches like the SVM delivered under-average performance, we can clearly see that the deep learning approaches really shine when dealing with such a large and diverse dataset. The usage of word-embeddings has established itself as a de-facto standard in today's natural language processing world. This can also be verified by the presented results. The preprocessing performed under our own expectations, but still allowed the word embedding to cover more of the input data.

When looking at the results, we can also see that the window for improvement is large in this subarea of natural language processing. Even an F1-score of 0.71 will not be sufficient to automatically solve the problem of toxic textual content in an unsupervised way.

References

[1] Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 26–32, New York, NY, USA, 2003. ACM.

[2] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[3] Saurabh Srivastava, Prerna Khurana, and Vartika Tewari. Identifying aggression and toxicity in comments using capsule network. In *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, pages 98–105, 2018.

[4] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *CoRR*, abs/1611.06639, 2016.

[5] Wikipedia. F1 score.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[7] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015.

[8] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[9] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[10] Juan Ramos. Using tf-idf to determine word relevance in document queries. 01 2003.

[11] Joshua Coates and Danushka Bollegala. Frustratingly easy meta-embedding - computing meta-embeddings by averaging source word embeddings. *CoRR*, abs/1804.05262, 2018.

[12] Nadbor Drozd. Text classification with word2vec.

[13] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

[14] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[15] José Camacho-Collados and Mohammad Taher Pilehvar. On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. *CoRR*, abs/1707.01780, 2017.