**Kaggle Competition – NLP**

**Quora Insincere Questions Classification**

**Detect Toxic Content to Improve Online Conversations**

**Authors:**

**Aleix Casellas Comas**

**Rubén Barco Terrones**

**Andreu Masdeu Ninot**

**Pablo Lázaro Terrones**

**Marco Gani Remane**

**Master on Fundamental Principles of Data Science**

**Natural Language Processing**

**23rd of April 2019**

**Methodology**

First of all, we have used the Pandas library from Python to read and prepare the data of the files *“train.csv”* and *“test.csv”.* We have divided the train dataset into two subsets, one for training and one for validation. To make this division we have used the ‘model\_selection’ function from ‘sklearn’ with a validation size of 20%. The validation set is the one we use to check the performance of the methods that we are proposing.

We have mainly used three different ways to encode the document: *CountVectorizer*, *TfidfVectorizer* and *word2vec*. They are already implemented in some libraries of Python, so we have just been modifying them and changing their hyperparameters.

For *CountVectorizer* and *TfidfVectorizer* we have been trying the following alternatives for the feature vector creation:

* **Lemmatization**: we have used the *WordNetLemmatizer*
* **Stemming**: we have used the *PorterStemmer*
* No additional preprocessing

In all these three cases, we have been trying as analyser different *N-grams* ratios. In addition, we have tried to train the models using and not using *stop-words*. We have been trying as well some preprocessing by hand. We will see in the section of Experiments the results obtained.

After obtaining the feature vectors, we have used the function from *sklearn* to create a Pipeline with the vectorizer, the feature selector and the classifier used. Then we have done Cross Validation using a randomized search for the model selection instead of grid search, a bit different depending on the classifier used (different hyperparameters). We have chosen randomized search because the results in parameter settings is quite similar, while the run time for randomized search is drastically lower. We have then used the functions *.fit* and *.predict* to train the model and get the accuracy and f1 score.

The thing that we commonly do in all the methods is the features selection. We use the function from *sklearn* *SelectKbest*, taking as score function chi-squared. We select the value of k, the number of top features to select, doing Cross Validation over the range between the minimum and maximum number of features available.

The classifiers we have evaluated using the different procedures explained before are:

* Logistic Regression
* XGBoost
* Multinomial Naïve Bayes

For these classifiers, we have done Cross Validation in different hyperparameters:

* Logistic Regression: Inverse of regularization strength C
* XGBoost: Learning rate
* Multinomial Naïve Bayes: Additive smoothing parameter alpha

In the notebook you can see how we have implemented all the previously explained methods.

For the *word2vec* experiments, we have implemented the ‘*word2vec’* function from the *‘gensim.models’* library. In general, in these experiments we have used a very simple preprocessing of the data: just splitting and converting the text to lowercase. One reason of this procedure is that we have seen that there are some questions in the dataset written in other languages such as Russian, Arabian or Chinese. So, if the *word2vec* does not have these kinds of words in the construction of the vocabulary, when we use the regressor (i.e. the Logistic Regressor) we are going to obtain a lot of NaN values. In these experiments we have tried manually some changes in the parameters of the *word2vec* definition:

* Number of **epochs**: 10 and 20 epochs
* Number of **features**: 200, 300 and 350 features
* **Context window** length: 3, 5 and 7 words for the context window

We have additionally tried two different approach from the ones seen in class, using embeddings and GRU or LSTM network. For these cases, we have used some special preprocessing in order to have all the embeddings correctly from the scratch. First, we have erased all the punctuations (because we have strange symbols) and replaced all the contractions in specific way for obtaining the best embeddings. Then, we have lowered all the text because in most cases it is advisable to do it if you are going to use embeddings. After preprocessing and fixing the parameters for the embeddings, we have tokenized all words and padded the sequences to ensure that all sequences have the same length. For the GRU, we have made some shuffle in order to improve the results. For the tokenization and the model specification, we have used keras. In the experiments section we will talk about the architectures of this two methods and the results we obtained.

**Experiments**

In this section we present the results obtained with all the different methodologies used, that we have explained previously. We have done cross validation for the hyperparameters, but still has been quite a manual process to choose which classifier or vectorizer to use, as well as the parameters of it. The following table present these results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy (%) | | | F1-Score | |
|  | Feature Extractor | Feature Selection | CV | Regression Algorithm | | Tr. | Te. | Tr. | | Te. |
| 1 | CountVect (no-param) | SelectKBest (chi2) | Random | Logistic Regression | | 94.98 | 94.94 | 0.436 | | 0.431 |
| 2 | CountVect  (Stemmer) | SelectKBest (chi2) | Random | Logistic Regression | | 95.42 | 95.24 | 0.531 | | 0.516 |
| 3 | CountVect  (Lemmatizer) | SelectKBest (chi2) | Random | Logistic Regression | | 95.46 | 95.25 | 0.535 | | 0.51 |
| 4 | CountVect  (Lemmatizer+  stop\_words) | SelectKBest (chi2) | Random | Logistic Regression | | 95.17 | 95.01 | 0.484 | | 0.466 |
| 5 | TfidfVect  (no-param) | SelectKBest (chi2) | Random | Logistic Regression | | 94.92 | 94.87 | 0.418 | | 0.411 |
| 6 | TfidfVect  (Stemmer) | SelectKBest (chi2) | Random | Logistic Regression | | 94.99 | 94.92 | 0.438 | | 0.431 |
| 7 | TfidfVect  (Lemmatizer) | SelectKBest (chi2) | Random | Logistic Regression | | 94.98 | 94.91 | 0.433 | | 0.437 |
| 8 | CountVect  (ngram(1,2)) | SelectKBest (chi2) | Random | Logistic Regression | | 96.49 | 95.55 | 0.654 | | 0.551 |
| 9 | CountVect(ngran(1,3)) | SelectKBest (chi2) | Random | Logistic Regression | | 97.26 | 95.53 | 0.741 | | 0.543 |
| 10 | CountVect(Lemmatizer) | SelectKBest (chi2) | Random | XGBoost | | 94.48 | 94.44 | 0.26 | | 0.26 |
| 11 | CountVect(ngram(1,2)) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 95.39 | 93.97 | 0.699 | | 0.571 |
| 12 | CountVect(ngram(1,2),stop\_words) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 97.48 | 94.70 | 0.815 | | 0.531 |
| 13 | CountVect(ngran(1,3)) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 97.7 | 94.76 | 0.8319 | | 0.563 |
| 14 | CountVect(Lematizer,ngram(2,3)) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 97.65 | 94.81 | 0.83 | | 0.513 |
| 15 | CountVect(Lematizer,ngram(2,3),stop\_words) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 99.31 | 94.57 | 0.943 | | 0.358 |
| 16 | CountVect(Lematizer,ngram(1,2)) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 94.39 | 93.13 | 0.662 | | 0.566 |
| 17 | TdifVectorizer(ngram(1,2)) | SelectKBest (chi2) | Random | Multinomial Naïve Bayes | | 94.86 | 93.36 | 0.684 | | 0.569 |

|  |  |
| --- | --- |
| Accuracy (%) | F1-Score |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Preprocessing | Feature Extractor | Nº epochs | Nº features | Algorithm | Tr. | Te. | Tr. | Te. |
| 18 | Lower | Word2Vec | 10 | 300 | Logistic Regression | 94.70 | 94.70 | 0.432 | 0.428 |
| 19 | Lower | Word2Vec | 20 | 300 | Logistic Regression | 94.71 | 94.67 | 0.434 | 0.431 |
| 20 | Lower | Word2Vec | 20 | 200 | Logistic Regression | 94.66 | 94.63 | 0.426 | 0.422 |
| 21 | Lower | Word2Vec | 20 | 250 | Logistic Regression | 94.74 | 94.69 | 0.439 | 0.434 |
| 22 | Lower | Word2Vec | 20 | 300 | Logistic Regression | 94.66 | 94.65 | 0.425 | 0.424 |
| 23 | Lower | Word2Vec | 20 | 300 | Logistic Regression | 94.72 | 94.69 | 0.438 | 0.435 |
| 24 | Capital letters and stop words | Keras Tokenizer + padding | 2 | - | Embedding + LSTM | 95.55 | 95.53 | 0.671 | 0.629 |
| 25 | Lower, contractions and punctuations | - | 2 | - | Embedding + Bidirectional GRU | 96.05 | 95.71 | 0.745 | 0.648 |

**Discussion of the results**

In the previous two tables, we have the 25 pipelines we have tested, and the results obtained. The best three results (regarding f1 score in test) are highlighted in yellow.

If we compare all the pipelines done with *CountVectorizer* and *TdifVectorizer*, we see that we get our best results when we use *CountVectorizer* with *ngram=(1,2)* and a Multinomial Naïve Bayes as a classifier. Adding *stop\_words=’english’* doesn’t improve the result. Even more, it decreases the f1 score. We see this behaviour in other cases that we have put this parameter. If we increase the *ngram,* for example to (1,3), we get better results in training, but worst in test; a sign that we are overfitting. A similar thing occurs if we increase the *ngram range* to (2,3). In general, *CountVectorizer* outperforms *TdifVectorizer*, no matter the parameters or variables we use. Adding a lemmatizer or a stemmer provides better results than without any parameters, but when we also add *ngram range*, the score obtained decreases or increases insignificantly, and it takes much longer to train. We see that we get pretty poor f1 score when we use XGBoost; that’s is why we have done just one trial with it.

On the other hand, we can see that using word2vec does not improve neither the results. Trying different number of features, epochs and context window lengths, all the results are very similar (the variation is between 0.42 and 0.44 in terms of F1-score) and they are lower than the rest of experiments. One way to improve the results of using word2vec may be combining the feature vector of word2vec with other type of feature vector in order to have more information about each of the questions.

The results obtained using the previous feature extractors weren’t bad, but neither satisfactory. That is why we tried different approaches. Finally, we see that deep learning approaches yield the best results. These are highly complex models that require lots of data to correctly generalize. Fortunately, the Quora dataset is big enough to let the GRU and the LSTM correctly learn the features to classify properly the data. Both models have an embedding layer first, and jointly they learn features from the tokens, unlike the *CountVectorizer* or *Tfidf* approaches. We use in both methods a bidirectional RNN to preserve information from the past and the future. Then, we applied a Global Max Pooling, something that is advisable when using RNN for a NLP problem to reduce dimensionalty. We introduce dropout to avoid overfitting and Batch Normalization in the case of LSTM. Finally, we create as output a final dense layer and we compile the model in both cases with Adam optimizer. Given the results, these features learning approach seems like a better option for this specific task. In the case of Bidirectional GRU, we checked the results in Kaggle with test data and we obtained a F1-score of 0.636, quite good taking into account that we have not used any ensemble technique or a more complex architecture to improve the model.

