Capstone Project

Hazem Halawi

Machine Learning Engineer Nanodegree

December 15, 2018

Definition

Project Overview

Social media platforms, online communities sharing knowledge, and many other public websites on Internet have become increasingly known for many problems related to the users' unethical comments, cyberbullying, etc.

And since manually supervising comments and discussions can be cumbersome regarding the large volume of comments, companies often have to ask employees to take time away from their regular work to sift through comments or taking other measures. This clearly emphasises the need to automate the process of comments classification.

In this project I have trained a machine learning model that is capable of classifying unethical comments using hundreds of thousands of 'Quora' labelled comments.

This project is proposed by 'Quora' as a 'Kaggle' competition.

Problem Statement

The goal is to handle non-ethical comments, in order to improve online conversation on Quora. This is done by identifying them and flagging them as insincere question. An insincere question is defined as a question intended to make a statement rather than look for helpful answers.

The process for solving such problem would be to use a supervised machine learning algorithm (many of them were tested in this project but the final model was a convolutional neural network) that will learn the patterns from the labelled comments.

The final model is supposed to, given a random comment, be able to classify it as ethical (good) or non-ethical (insincere).

Metrics

The F1 score is used for this project to evaluate the performance of the models. While accuracy is a good metric, it will not show a real indication of how the model is performing. Especially since the labels, as we will see in the next section, are highly imbalanced. The model could be saying everything is insincere and still get a high accuracy.

F1 score considers both the precision and the recall of the test to compute the score: precision is the number of correct positive results divided by the number of all positive results returned by the model, and recall is the number of correct positive results divided by the number of all samples that should have been identified as positive. The F1 score is the harmonic average of the precision and recall, where the best score is 1 and worst is 0.

Formula: 2*((precision*recall)/(precision+recall)).

Analysis

Data Exploration

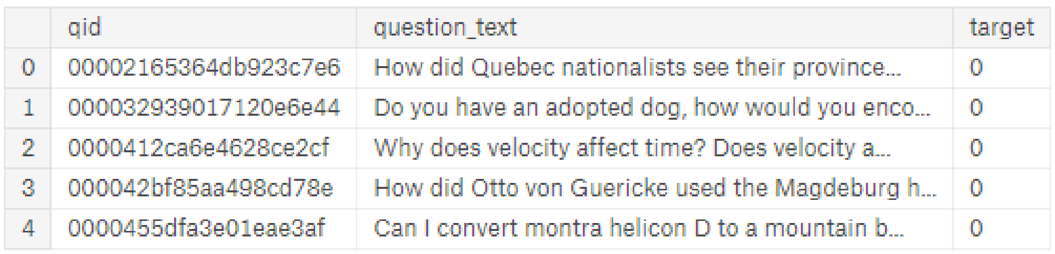
The data set used for this project is provided by Quora, and could be found on 'Kaggle' in the data section of the competition: "https://www.kaggle.com/c/quora-insincere-questions-classification/data".

It consists of two csv files:

- The first, train.csv, has a total of 1306122 rows each containing an id, comment (String) and label.
- And test.csv of 56370 rows, each containing an id and a comment.

The labels for each comment are '0' for insincere comment and '1' otherwise. The ground-truth labels contain some amount of noise; they are not guaranteed to be perfect.

Fig. 1 shows a sample of the data:



	qid	question_text	target
0	00002165364db923c7e6	How did Quebec nationalists see their province...	0
1	000032939017120e6e44	Do you have an adopted dog, how would you enco...	0
2	0000412ca6e4628ce2cf	Why does velocity affect time? Does velocity a...	0
3	000042bf85aa498cd78e	How did Otto von Guericke used the Magdeburg h...	0
4	0000455dfa3e01eae3af	Can I convert montra helicon D to a mountain b...	0

Fig. 1

In addition to this training and testing data, word embedding files (.txt) are used in the final model of this project. This file represents a collection of words with their corresponding embed vector, i.e., the number of features that represent similarities between words.

Note: These vectors were obtained by training a neural network (using one-hot encoded vectors as inputs), and taking the weights between the input and first hidden layer of the network after training. The embedding matrix here has a size of 2,196,017 by 300. This means that it contains 2,196,017 words, each having a vector of size 300.

Exploratory Visualization

Fig. 2 shows the number of comments for each label (0 - 1).

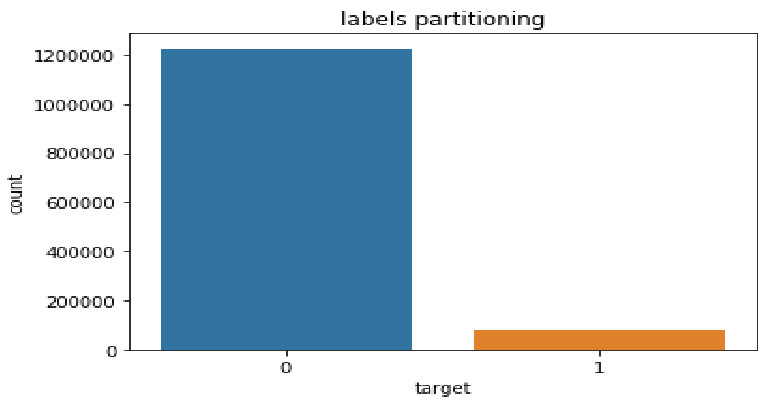


Fig. 2

As we can see, the comments with target '0' are with far greater number than the other label. For the training set, we have exactly 1225312 comments that are labelled '0' and 80810 as '1', which represents 94% and 6% of the total dataset respectively.

And this is a clear reason why we cannot use accuracy as a metric as mentioned before.

We can also check the comments column, by checking their length.

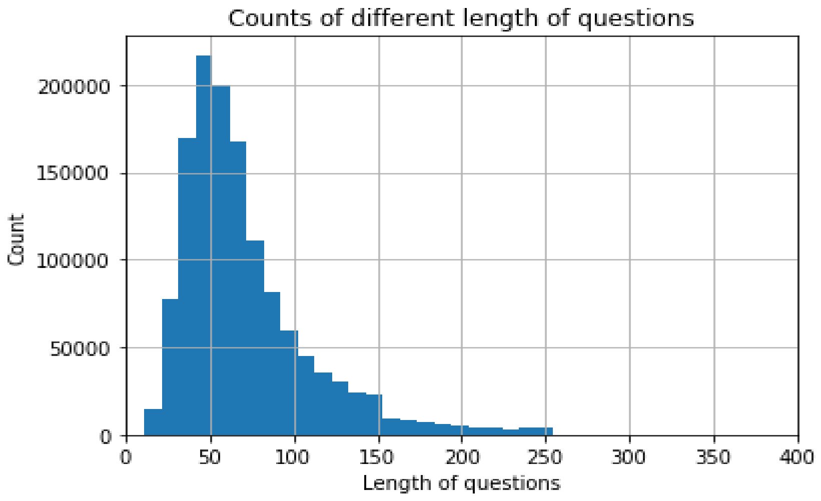


Fig. 3

As we can see in the Fig. 3, most of the comments have a length of around fifty characters, and the minority have more than 150 characters.
Also none of columns had 'null' values.

Algorithms and Techniques

The dataset provided is labelled, which means we are trying to solve a supervised learning problem that requires a supervised learning algorithm/classifier.

The final classifier used to solve it is a convolutional neural network. Although CNNs are extensively used for image classification, it can also perform nicely when using it for natural language processing problems, especially when, like our case, we have a large amount of data.

This large data is composed of a set of questions (words), and so before feeding it to the network:

- Every word is label encoded, which means that every word is represented by a particular number.
- An embedding matrix is created, combining the embed vectors from the word embedding files, and a subset of the words that are present in

the dataset's vocabulary. This matrix is used as weights of the first layer of the CNN.

This transformation is necessary because the CNN requires the input to be a vector of continuous values. If the input and output are of binary values (not raw words), the network will be able to perform and learn better.

To optimize the result, some parameters could be tuned:

- Text pre-processing:
    - Number of words to consider from all the vocabulary (the label encoded vocabulary).
    - Maximum number of words for each question.
- CNN architecture :
    - Number of layers
    - Layers type (convolutional, fully connected…) (note: it is best that the last layer is a fully connected layer with one or two nodes, because it's a binary label classification).
    - Type of Activation function for each layer (sigmoid, tanh, Relu…)
- Model training parameters :
    - Loss function and optimizer
    - Batch size
    - Number of epochs
    - Learning rate
    - Best threshold (when converting test prediction values ranging from 0 to 1 to binary values)
    - Dropout probability
    - Filter size

Benchmark

As a benchmark, the supervised learning algorithm is switched from a CNN to a Gaussian Naïve Bayes. All the pre-processing steps were tested. However, all pre-processing steps preceding the label encoding step decreased the model score. So this final model was a simple Gaussian Naïve Bayes (no parameters to tune) with features label encoded and one-hot encoded. This model needs only a couple of minutes to be executed. The best result obtained by this model is an f1 score of 0.217.

Methodology

Data Pre-processing

The first step of pre-processing was to split the data into features and labels separately.

The following text processing methods were attempted but not all of them were used:

- Adding space between words and punctuation. The problem was that, the ones that were not spaced, were considered by the label encoder as a new word in the vocabulary (ex: engineer and engineer? Are two separate words in the vocabulary).
- Removing punctuation.
- Besides removing punctuation, some words that will not help in the classification problem (they are found in most questions of both classes) must be removed. So, in this step, these 'stopwords' were removed.
- Replacing all number with the string 'num'. This last step was done because the total vocabulary of numbers alone was too large and most of them are non-meaningful.
- The rest of the words (the ones that are neither stopwords nor numbers) are Stemmed using 'SnowballStemmer'. This step also helps reducing vocabulary by merging words with the same root.

The next step was to split every question into separate words in order to be fed to the label encoder which encodes words and not phrases.

- All the words from all the questions are then label encoded by fitting on the vocabulary and then transforming every question separately. (Of course before this is done all the questions are concatenated, from both training and testing datasets, into one string that contains the whole vocabulary of words before encoding). This step is necessary because the embedding layer requires the input data to be integer encoded. In this step, the max length of words is set to 70 for all questions.

Now that we have a well pre-processed set of words contained in the tokenizer's 'word_index', we can create the embedding matrix mentioned earlier. This matrix was created as follow:

- First the maximum number of words to be considered is set (In the final model, 50,000 words were selected for learning)
- The embedding matrix will contain the vectors of these words selected from the embedding file if they exist, and empty vectors if not, both of size 300.
- So the final embedding matrix size will be 50,000 by 300. This step reduces the dimensionality space, especially when comparing with one-hot encoding.

The final step of pre-processing before training was to split the features and labels into training and validation sets.

Implementation

After all the pre-processing steps, the training set is now ready to be fed to the CNN classifier.

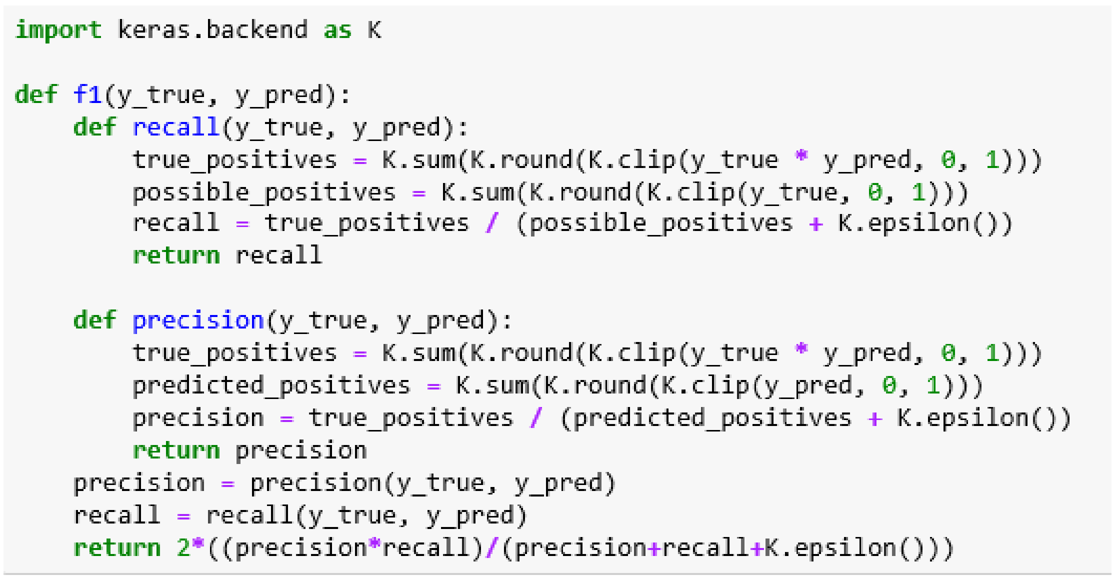
First, the architecture of the network is built as follow:

- Input layer equal to the max length set (70).
- Embedding layer. For this layer, at least three arguments must be specified: the size of the vocabulary (50,000), the size of the embedding vector (300), and the input length already set to 70. In addition, since the embedding vectors are pre-trained weights representation, this layer's weights are set equal to the embedding matrix weights. These weights are also set to 'non-trainable'.
- The embedding layer contains 70 vectors of 300 dimensions each, one for each word. We could flatten this to a 21000 element vector to pass it to the dense output layer, or we could add more convolutional layers before doing so. (Of course it will no longer be a 21000 element vector when we flatten it but a vector with a much smaller size).
- Following the embedding layer are three 1D convolutional layers. The reason why these layers are 1D and not 2D is because we are working with text data and not images.
- A dropout is added after every convolutional layer with a probability of 10%.
- A fully connected layer with 128 nodes with a 'tanh' activation function.

- And the last layer is a fully connected layer with one node (binary classification) and a 'sigmoid' activation function.

The loss function, optimizer and evaluation metric defined for this network are 'binary_crossentropy', 'adam' and F1-score respectively.

Note that the F1-score is a function defined by the code in Fig. 4:



```python
import keras.backend as K

def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Fig. 4

The total number of parameters in this model is 16,240,705, 15,000,000 of them are non-trainable (embedding layer parameters).

The following step is to train the model. The model is trained with the training set of features and labels, with a number of epochs equal to 10 and a batch size of 100.

After that the model is evaluated using the validation set and F1-score.
Note that this validation set is obtained in a previous step from the training set. The testing set provided by Quora is not labelled, but could be tested on, by submitting the model in a Kaggle Kernel. The true labels are not shown, but the F1-score is returned after submission.

Refinement

The first classification algorithm attempted was the benchmark model already discussed in a previous section, using a Gaussian naïve Bayes Classifier.

The next model was a deep neural network. For this classifier, the only text pre-processing that enhanced the model was stemming, the rest of them decreased the accuracy. Integer encoding and one-hot encoding is then applied on the features and their corresponding targets, in order to be fed to the network. The network architecture is a simple two dense layer with 128 and 2 layers respectively. After tuning the number of epochs and using different portions of the dataset the results were the following:

- The best number of epochs before the model starts overfitting is 8.
- The model's performance increases as we add more data until we reach 75% of the dataset where it starts to drop.
- The highest score this model reached was 0.35, which is an improvement over the benchmark model.
- The execution time was much greater than the naïve Bayes model where the first took one to two hours while the second took couple of minutes.

The final classifier attempted is a Convolution neural network. First, all punctuation are removed. The rest of text pre-processing decreased performance. And, like previous models, questions are label encoded. Next, instead of one hot encoding, embedding technique is applied to reduce the dimensionality of the network's input. Concerning the CNN architecture and training the following changes improved the performance:

- Adding up to three convolutional layers. Adding More Conv. Layers decreased performance
- Adding a 128-nodes fully connected layer before the last fully connected layer
- Adding a dropout with 10% probability
- Changing the convolutional filter size to 2
- Adding early stop approach to avoid overfitting from too many epochs

Results

Model Evaluation and Validation

The final model that I chose is the convolutional neural network model, where as we saw performed much better than the Gaussian naïve Bayes and the deep neural network. This model can be described by following steps:

- Removing punctuation
- Label encoding
- Adding embedding technique
- Training the model with a CNN architecture (detailed in earlier section).
- Evaluating the model with the F2 score

The final model parameters which had the best performance:

- 50,000 Maximum features
- 70 maximum number of word per question
- 128 filters of size 2*2 for each convolutional layer
- Around 8 epochs and a batch size of 2048

The final f1 score for this model is a maximum of 0.629 on the validation set and 0.648 on the test set (dataset size equal to 56370) which is a good one especially since the dataset tested on has an uneven class distribution. This means that this model's predictions have a low percentage of 'False Positives' and 'False Negatives' and a high percentage of 'True Positives'. Thus we can conclude that this model can make prediction reliably.

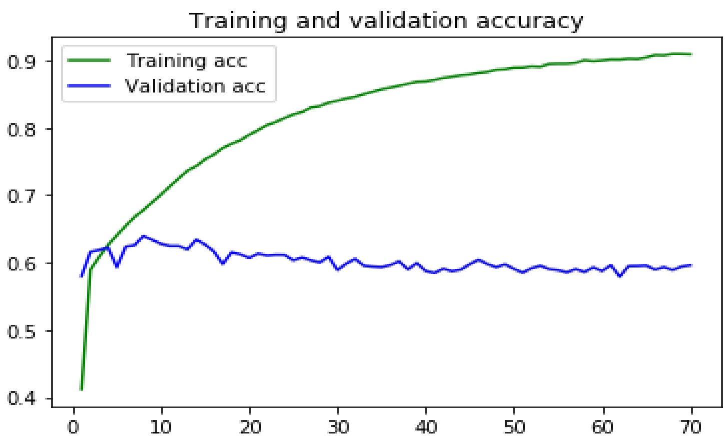
Justification

The benchmark model best score was 0.217, while the CNN scored 0.648. Both took around ten minute to train on the data. It is clear that the CNN is a much better classifier than the benchmark and also than the Deep Neural Network mentioned earlier which scored 0.35 and took more than hour to train. The robustness of this model is also discussed in the previous section, where the model is capable of classifying reliably on unseen data with uneven class distribution.

Conclusion

Free-Form Visualization

The CNN is a classifier with many tunable parameters. One of them is the number of epochs. And while more epochs can decrease the error and enhance the performance, too many epochs may cause the model to overfit the training data. Overfitting can be detected and visualized by plotting the training and validation score (or error) along with their corresponding number of epochs.



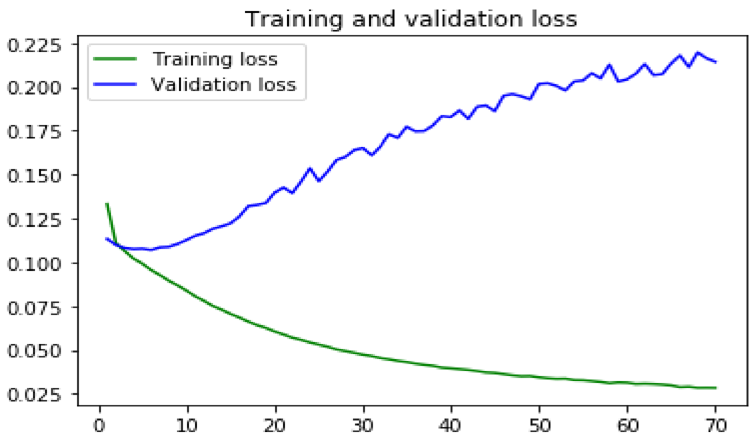


Fig. 5

We can clearly see from the Fig. 5 how the validation score starts to decrease while the training data is increasing as the model is trained with more epochs. This is a clear sign of overfitting. This issue however was handled by adding 'early stop' approach, where it will stop training as soon as the model starts overfitting, in this case, around epoch 8.

Reflection

This project could be summarized as follow:

- A dataset is provided by 'Quora' containing questions from this site that should be classified as insincere or not
- The dataset is first discovered and pre-processed
- A benchmark model is created to be compared with the final model
- An embedding matrix is created with the intent of reducing the training input
- The CNN classifier is trained, using both embedding and pre-processed data, with many epochs
- The robustness of the model is then tested using F1-score

The hardest step I faced in this project was the embedding phase, simply because it was new to me. But I am actually glad that I discovered the embedding method, for it was an essential part of building a good reliable model that reduced the dimensionality and training time. Plus this method will be useful when working on future projects.

The final model did perform quite well, and could be used to help classifying insincere questions.

Improvement

Many modifications could be made on the final model that might achieve better results:

- Exploring the dataset more and removing noisy words. Actually the whole data vocabulary is immense and could contain various words that would not come to mind unless discovered.
- Expanding the embedding matrix by using additional pre-trained embedding files.
- Replacing the CNN with a recurrent neural network