# Machine Learning Engineer Nanodegree

## Capstone Project Report

Gennady Lungu
April 24, 2019

## I. Definition: Kaggle Quora Insincere Questions Classification

### Project Overview

The project is a Kaggle competition for binary classification of Quora questions by sincerity (sincere or insincere). The competition is already closed, although it is very recent: it was closed on Feb 14, 2019. Yet, it is still possible to apply late submissions and get a score against a test dataset.

The competition is initiated by Quora. Quora is a platform that empowers people to learn from each other. On Quora, people can ask questions and connect with others who contribute unique insights and quality answers. Since Quora's main idea is people learning from each other by asking questions and providing answers, and since virtually any question can be asked, some posts may present a problem. It is those posts that camouflage as a question yet aim at promoting a pre-conceived point of view or seek to slander a group of people without any intent of asking a question or learn something.

Therefore, the task is to classify a question as sincere or insincere (0 or 1) taking question text as input. Next section states the problem in more detail.

### Problem Statement

Given a question (interrogative sentence) classify it as sincere or insincere. By insincere we will mean questions that are “founded upon false premises, or that intend to make a statement rather than look for helpful answers”. Some characteristics that can signify that a question is insincere:

- Has a non-neutral tone
    - Has an exaggerated tone to underscore a point about a group of people
    - Is rhetorical and meant to imply a statement about a group of people
- Is disparaging or inflammatory
    - Suggests a discriminatory idea against a protected class of people, or seeks confirmation of a stereotype
    - Makes disparaging attacks/insults against a specific person or group of people
    - Based on an outlandish premise about a group of people
    - Disparages against a characteristic that is not fixable and not measurable
- Isn't grounded in reality
    - Based on false information, or contains absurd assumptions
- Uses sexual content (incest, bestiality, pedophilia) for shock value, and not to seek genuine answers

A dataset of over 1,300,000 questions marked as insincere or sincere is provided by Quora. This training data includes the question that was asked, and whether it was identified as insincere (target = 1). The ground-truth labels contain some amount of noise: they are not guaranteed to be perfect.

The task is to train a model on this dataset, and then apply this model to a test set to see how well it generalizes to questions it has never seen before.

## Datasets and Inputs

The following files are provided as inputs.

- **train.csv** - the training set
- **test.csv** - the test set
- **enbeddings** – the word embeddings (word to vector encodings that seek to encode closely related words as closely located vectors) from four well-known sources that can be used along with the dataset when building or training the model. These are as follows:
    - GoogleNews-vectors-negative300 - https://code.google.com/archive/p/word2vec/
    - glove.840B.300d - https://nlp.stanford.edu/projects/glove/
    - paragram_300_sl999 - https://cogcomp.org/page/resource_view/106
    - wiki-news-300d-1M - https://fasttext.cc/docs/en/english-vectors.html

The training set itself contains the following data fields:

- **qid** - unique question identifier
- **question_text** - Quora question text
- **target** - a question labeled "insincere" has a value of `1`, otherwise `0`

## Metrics

Submissions are evaluated on F1 Score between the predicted and the observed targets on a test dataset. Since training data is imbalanced (there are significantly more sincere labels than insincere), both recall and precision have to be taken into account, and F1 score is widely used for this task. Another benefit of F1 score is that it tends to be much lower if one of the characteristics (precision or recall) has low value, as compared to arithmetic average for instance.

During the competition, participants were able to see their scores calculated on a 15% portion of the test set (called public score), and after the competition ended it was once calculated on the rest 85% of the test set (called private score), and final standings were based on this private score. This was to avoid overfitting the public score leaderboard and selecting the model that generalizes best.

Since the competition is closed, now submissions get both public and private scores calculated, therefore we will simply target the best private score.

It is also important to note that competition rules limit the time of the whole training process to 2 hours on GPU, and do not allow use of any other input except the given training data and 4 embedding files described above.

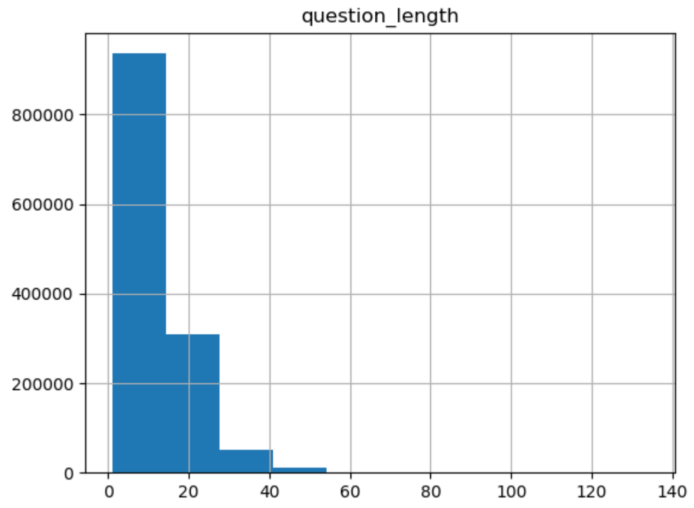
# II. Analysis

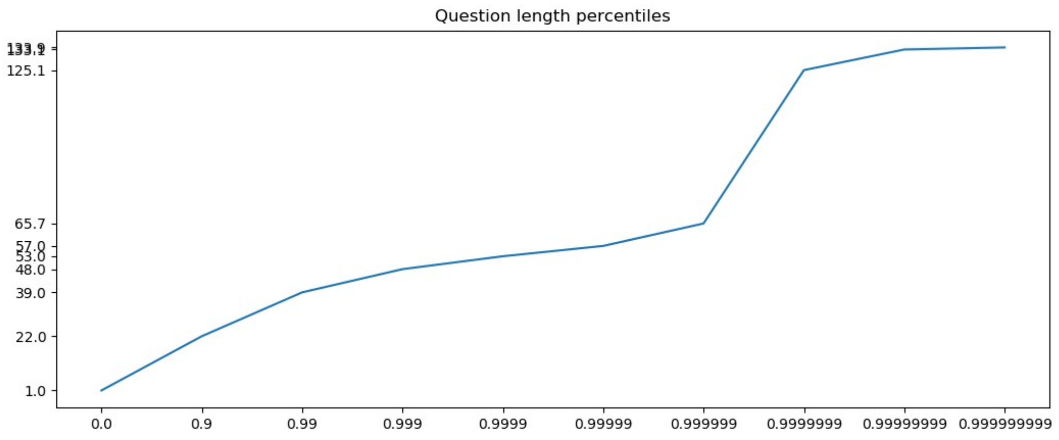
## Data Exploration

Our input data is text, and we are going to apply word vectorization, therefore the following characteristics would be of interest:

- **Number of words in question text** – this will be our sequence length in recurrent network, the correct choice of this value will affect our model. Too low – lose valuable data information, too high – increase computation cost and time.

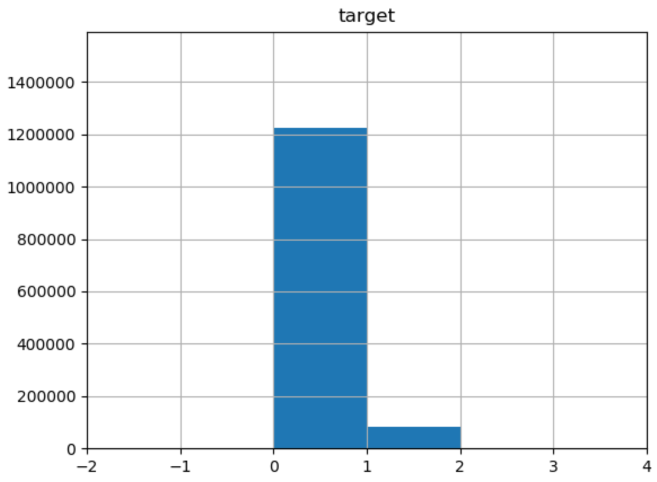
As the plots below show, most of the questions have length less than 60 words. To be more precise, 0.000001 or 0.0001% of training examples have length over 66 words. This actually corresponds to only one question that has length 134. It is 'sincere' and we can truncate it to 66 words.





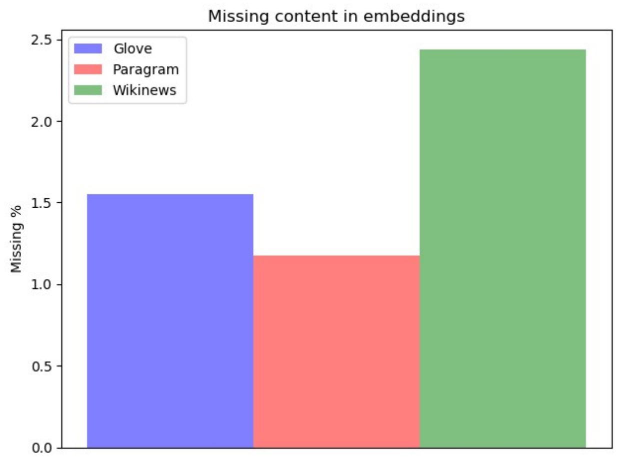
- **Percentage of 1-labeled training examples to total examples** – this will show how imbalanced our data is and possibly, provide some hints for training.

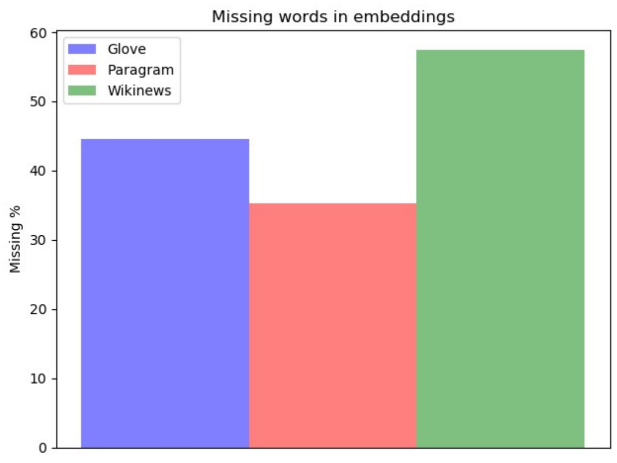
This value turns out to be 6.19%. The data is very imbalanced, but this is of course explainable – most questions asked on Quora actually are sincere. With such strong imbalance, we cannot make it balanced by throwing away a portion of sincere questions, since we will throw away very large number of examples. Rather, we can have a hint that we're interested in accuracy larger than 0.9381, since this accuracy is achieved by a totally useless model predicting always 0. The model start learning something useful when achieving a larger accuracy.



- **Embeddings' missing words**. Since we're using embeddings, another important characteristic is how many words used in questions are missing in the embeddings. Such words would be substituted with some predefined values, therefore it is important to know what percentage it is, and actually look at them to see if we can reduce it.

Here we use three embeddings – 'Glove', 'Paragram' and 'Wikinews' – and plot percentages of missing words (by counting just words and counting each of their occurences).





We can see that 35% to 57% of words are missing in different embeddings, which corresponds to 1.17% to 2.43% of missing content (calculated by counting all words occurences). So, the missing words are rare, yet 2% seems to still be a large number. If we look at top 25 of the missing words and their occurrences we can immediately get some preprocessing ideas:

```
what's :  13432
i'm :  13261
isn't :  3597
```

```
i've :  2819
i’m :  2575
you've :  2457
don’t :  2248
aren't :  2131
what’s :  1818
won't :  1728
trump's :  1589
they're :  1239
haven't :  1126
shouldn't :  1085
he's :  1082
it’s :  1078
can’t :  1040
wouldn't :  949
quorans :  853
who's :  849
doesn’t :  814
today's :  801
someone's :  789
there's :  787
```

So, here are the preprocessing ideas:

- Embeddings do not know about some common verb contractions (when used with pronouns), e.g. i've, wouldn't, can't etc. We can replace them with their full versions without losing much of the original meaning: i've = I have, wouldn't = would not, can't = cannot, etc
- Embeddings do not know about possessive nouns when used with 's, e.g. today's, someone's etc. These ones can be replaced with the word they denote possession for, e.g. today's → today, someone's → someone, etc. This does transform the text, but in most cases retains the original meaning, e.g. “Who is the leader of today's Iran?” Can be replaced with “Who is the leader of today Iran?” May not be totally perfect English, but the meaning is preserved and it is better to replace “today's” with embedding vector which will give it the context meaning of “today” rather then ignore the word “today's” altogether. This should help RNN make more sense out of this question.
- Thirdly, before applying previous two rules, we can unify the use of apostrophe symbol, because as we can see some use ', others - ’, yet others - `, etc.

More on preprocessing rules in the corresponding 'Methodology' sub-section below.

## Algorithms and Techniques

### Recurrent Neural Networks

Text_classification is a very common class of Machine Learning problems, and is applied to such tasks as: Email_filtering, Readability Assessment, and Sentiment_analysis. The most widely used Machine Learning models for this task is Recurrent Neural Networks (RNNs), because they not only capture each word separately, but also some temporal sequence dynamic by using internal state (memory). When processing input sequence, the output from processing one element is fed into the same network (or, sometimes, only some recurrent layers of the network) alongside next element in the sequence, thus making use of whatever was learned while processing the sequence so far. A sequenced N-featured input then gets an additional dimension which is the sequence length.

The two most well-known and widely used RNN layers are Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

RNNs can also be unidirectional and bi-directional. In bi-directional RNNs sequences are passed through the network not only start-to-finish, but also finish-to-start in order to capture both past and future sequence contexts.

For our task we are going to use RNN with bi-directional LSTM layers. We will use a simple architecture consisting of two RNN layers each with 64 hidden features, and then one Dense layer with a sigmoid activation to output the number between 0 and 1, representing probability of a question text being insincere (0 – totally sincere, 1 – totally insincere).

We will also set pre-trained first layer called Embedding Layer, which will make use of the provided embeddings, described in the next section.

### Embeddings

Word embeddings are word to vector encodings that seek to encode closely related words as closely located vectors (in terms of vector distance). “Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, explainable knowledge base method, and explicit representation in terms of the context in which words appear” (Wikipedia).

Usually it takes a very large text corpus to get good embeddings, and 4 such well-known public embeddings are provided as input in this competition.

Also, embeddings shall be used as a pretrained first layer of our RNN as described here: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/. This technique uses Keras' Tokenizer API, where words are assigned numbers according to their frequencies, and all word sequences are encoded as sequences of these numbers. The Embedding layer then transforms each sequence of numbers into sequence of vectors according to the embeddings.

### Ensemble

We also use the technique of combining several trained models known as 'ensemble'. In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

We train several models (different approaches of achieving several models are described later) and use the 'voting' techniques to combine their results (in our case, we average results from individual models). The models should be simple enough to be able to train a number of them within 2 hours of GPU usage limit.

## Benchmark

In the benchmark model, we used so called word n-grams (in the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech). For example, in “What is your last name?” we have the following 3-grams: “what is your”, “is your last” and “your last name”. The 3-grams (n=3) were used in the benchmark model.

For each 3-gram we calculate the number and, more importantly, the frequency of its occurrence in ‘sincere’ and ‘insincere’ questions. Those 3-grams whose frequency in insincere questions is at least one order of magnitude (i.e. at least 10 times) higher than in sincere ones, will be considered “target” 3-grams. Finally, we check each test question on occurrences of target 3-grams, and if more than one target 3-gram is contained within a question, we shall predict it as ‘insincere’, and ‘sincere’ otherwise.

The benchmark model above achieved F1 score of 0.49038 on the private set (view on kaggle):

Quora 3-Gram Benchmark (version 5/5) 0.49038 0.48258
2 days ago by Gena
Kernel renamed

This provides us with some starting point value, although judging from the competition leaderboard scores this value is not high at all and should be easy to beat.

We can set ourselves additional goal by considering a hypothetical model with a precision of 0.8 (when predicting insincerity the model is 80% right) and recall of 0.6 (out of all insincere questions the model finds 60% of them) – this is quite good model, and it's F1 score is 0.6857. Since the score of 3-gram model proved easily beatable with RNN, we will try to beat this score as well.

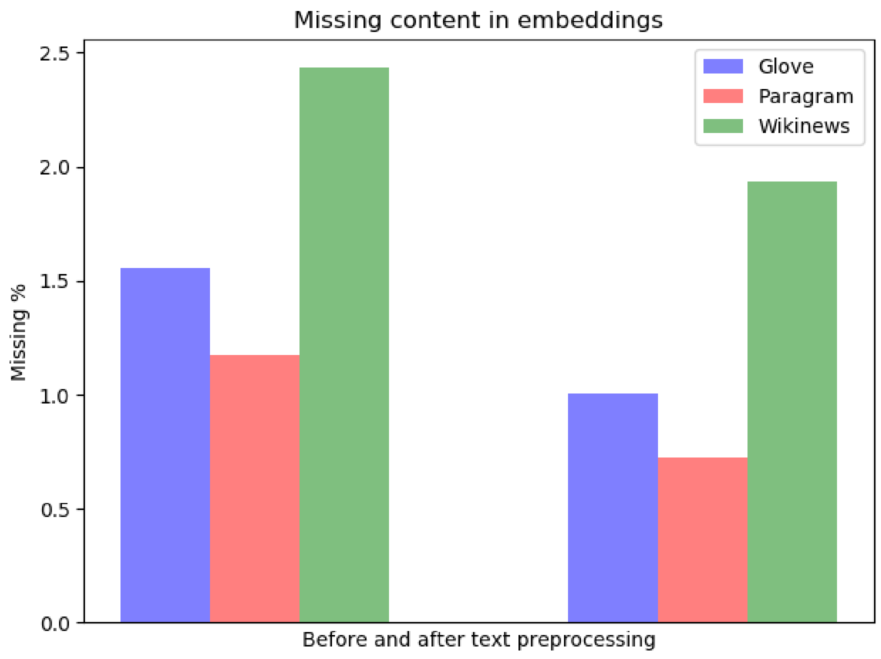
## III. Methodology

### Data Preprocessing

As discussed in the section on Data Exploration, we have got some ideas on data preprocessing:

- Replace different apostrophe's symbols with a single one: '
- Replace noun + modal verb contractions with their full versions:  i've = I have, wouldn't = would not, can't = cannot etc
- Replace possessive construct using 's with the word itself which in most cases should retain the meaning:  today's = today, company's = company etc

Below is bar plots showing missing content results (ratio of sums of missing and all word occurrences) before and after applying text preprocessing.



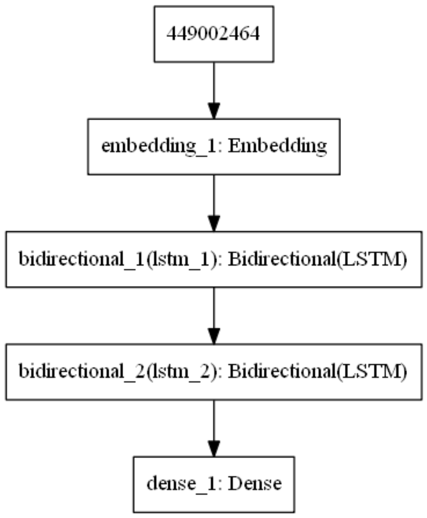
For all embeddings we were able to reduce missing content by approximately 0.5-0.6%. This improved performance by approx. 0.03 in the F1 score. It should be noted that only few most occurring missing words were replaced.

### Implementation

- **Embeddings** -  Implemented first Keras' Tokenizer API. Method fit_on_texts() prepares word_index that maps words to their numbers according to their frequencies. Dictionary size of 120,000 was used, higher dictionary sizes did not have influence on performance. Mappings from word number to corresponding embedding vector are then prepared using word_index and embeddings, and used to initialize Keras' Embedding layer. This technique is also described here: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/. texts_to_sequences() method is used before feeding sequences into the network.

- **RNN Architecture**

Keras Sequential Model is used: Embedding → LSTM → LSTM → Dense, as shown in the model plot:


449002464
embedding_1: Embedding
bidirectional_1(lstm_1): Bidirectional(LSTM)
bidirectional_2(lstm_2): Bidirectional(LSTM)
dense_1: Dense

- **Early Stopping –** During the training process the number of epochs to train was determined dynamically by measuring F1 score on the validation set (5% of the training set, or approx 65,000 examples). The training was stopped after there was no F1 score improvement after k epochs. Values of k of 1, 2 and 3 were tried. Best results were achieved with k=3.
- **Ensemble Voting –** N=9 Models were trained using slightly different architectures (every even used LSTM, odd – GRU) and different embeddings (Glove, Paragram, Wiki, then repeat). Predictions of each of those models were then averaged and used for final result. Training of only 9 models fit into the limit of 2 hours.
- **Threshold** – Usage of common 0.5 threshold did not prove to be optimal by several experiment runs, so 0.33 was used. The imbalance of data (much more 0 labels then 1) actually hints that a weaker than 0.5 signal should be enough to classify question as 1. Threshold of 0.33 provided much better results, but not enough experiments were made to actually prove that it was optimal in this configuration, yet it is not far from optimal.

## Refinement

Actually, all of the areas of implementation described above were refined during the process of improvement, and previous section contains only the description of the final implementation. Here is how some of these areas were improved (the most critical for performance are described):

- **Embeddings –** first embeddings were used to manually encode word sequences, but using Keras Tokenizer API and making Embeddings part of the network as first Embedding Layer speed up the process (allowing more models to train) and provided better results after fine-tuning dictionary size, and max sequence length
- **RNN Architecture** – more complex architectures were tried by adding more LSTM/GRU or

Dense layers, but this only slowed the training process without huge gain in the score. It was decided to stick to this simple RNN architecture of 3 layers in order to train more models for the ensemble. This decision was strongly influenced by this public kernel: https://www.kaggle.com/mihaskalic/lstm-is-all-you-need-well-maybe-embeddings-also

- **Early Stopping** – First used fixed number of epochs (5) was used that seemed to be a good number judging from a few experimental runs. Later, early stopping was added with copying and saving the model at exactly the epoch before validation F1 score began to go down. But, stopping after 3 epochs without improvement on the best F1 score proved to be the best strategy, as the seemed to keep learning even though F1 score started to decline from the best value, it may have started to increase after that (usually without beating the best value, but still growing). This increased number of epochs to 7-9 epochs, but showed better performance of the ensemble. It is very probable, that previous best F1 score was only local maximum for some of the models, so potential room for improvement.

- **Ensemble** – first only one glove embedding was used. Next, all embeddings were averaged as argued in these papers http://aclweb.org/anthology/N18-2031, https://arxiv.org/pdf/1804.07983.pdf (present an argument for averaging as a valid meta-embedding technique, and found experimental performance to be close to, or in some cases better than that of concatenation, with the additional benefit of reduced dimensionality) and argued in this public kernel: https://www.kaggle.com/shujian/single-rnn-with-4-folds-clr. Later, even better performance was obtained by training different models on different embeddings (without averaging), e.g. an ensemble of 9 models where each of the 3 embeddings were used in 3 models each, performed slightly better than ensemble of 9 models trained with same averaged embeddings.

So, basically, the process started with a single model, manual embedding encoding (not as RNN Embedding layer), fixed number of epochs, and a single embedding. It then evolve into ensemble of 10 RNNs (with Embedding Layer) each using one of the different embeddings, and early stopping while training.

## IV. Results

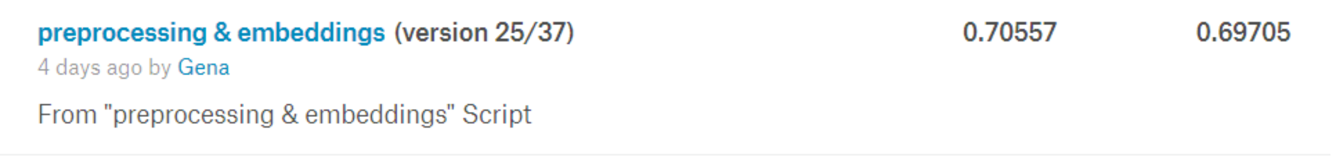
### Model Evaluation and Validation

During the competition, participants would see only their public score and then after a deadline final private score was computed only once based on the private test set. Final standings were determined based on this private score. Therefore, public test set served as sort of validation set and participants chose their best models based on it. The private test set served as actual test score to provide the model's final evaluation.

Since competition is closed and both public and private scores are output immediately upon submitting, strictly speaking, both public and private test sets have become validation sets. And we do not have a real test set to provide the final score. Since we also do not have other source of labeled data, the right way would have been to set aside a portion of the trained data as test set and calculate final score on it. But this is a bit problematic because this would be outside the regular kaggle competition process and not possible to evaluate against competition's leaderboard.

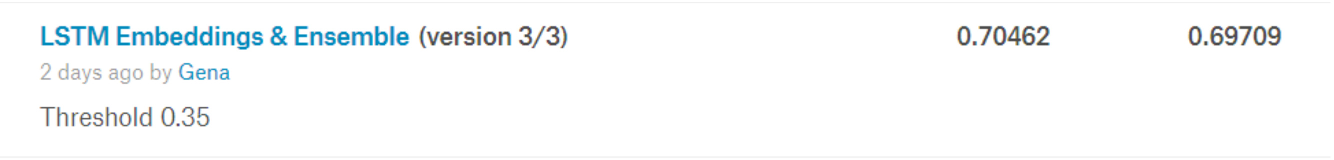
Therefore, it was decided to stick to kaggle process all the way, and simply select the best model based on the private score.

The best private score we came up with is 0.70557:

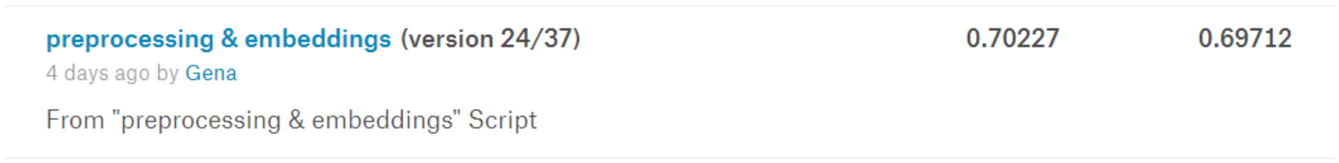

preprocessing & embeddings (version 25/37) 0.70557 0.69705
4 days ago by Gena
From "preprocessing & embeddings" Script

This score is clearly higher than our benchmarks, and is actually in the top 4% on private scores of the competition leaderboard. Yet, such comparison is not plausible, because competition private score leaderboard was calculated once after participants submitted their models.

In order to make more plausible comparison with the competition leaderboard, we can take some of the best public scores we achieved (as we would have during competition) and see the private score. In this case we have a higher public score (0.69709) but slightly lower private score 0.70462 (view on kaggle), which puts us in the top 5%:


LSTM Embeddings & Ensemble (version 3/3) 0.70462 0.69709
2 days ago by Gena
Threshold 0.35

Yet, it has to be mentioned, that the absolute best public score we achieved was 0.69712, but it corresponds to an even lower private score of 0.70227, which is in the top 37%. This emphasizes the fact that we may have began to overfit the (much smaller) public set at the cost of losing some ability to generalize. In the real competition, not only the public score should have been taken into account, but also an average validation score of the trained models (each model had its own validation set during training):


preprocessing & embeddings (version 24/37) 0.70227 0.69712
4 days ago by Gena
From "preprocessing & embeddings" Script

But, in any of those models, we beat the score of a (hypothetical) model with 80% precision and 60% recall, which can be considered as a good result based on our expectations.

# V. Conclusion

## Free-Form Visualization

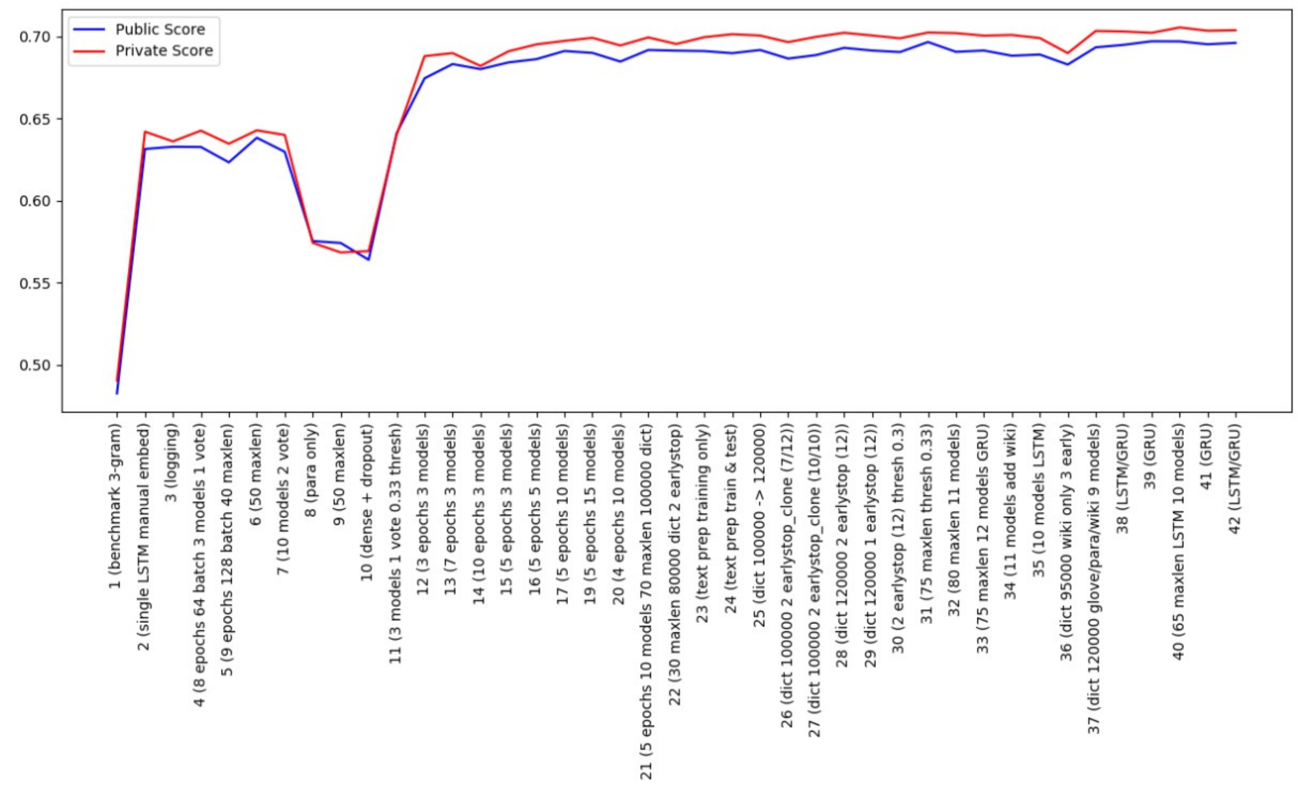
The most interesting and challenging in this project for me was the process of model improvement. There are so many hyper-parameters to change and sometimes several of them are changed at a time so that later it is hard to tell which of them actually led to a better result. Besides, when one parameter is found to be optimal, as soon as it changes other parameters may not be optimal anymore, and have to be re-evaluated.

Some visualizations of the process are found in the next 'Reflection' section.

## Reflection

During the course of working on this project over 40 commits (i.e. versions) were made to kaggle, so seems like a good idea to overview all of them and try to extract the main points and ideas that actually led to improvements. Such review can also identify points for further improvements.

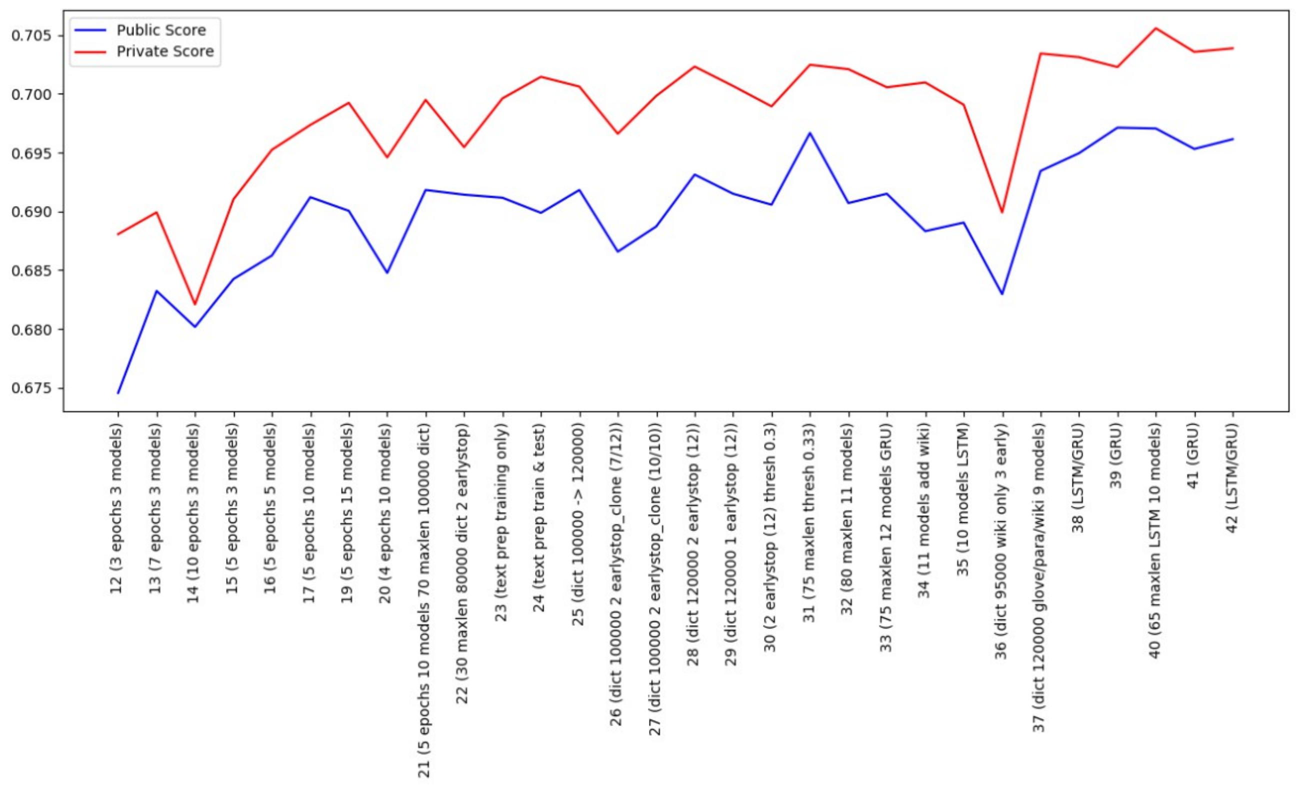
Here is the overall picture (most important improvements emphasized below):



11: Changed threshold from 0.5 to 0.33

12: Started using Embedding Layer, ensemble of 3 models and fixed number of epochs. Embedding Layer used average of the 2 embeddings (glove and paragram).

Steps 12 to 42 of the process are zoomed into below:



23-24: Introduced text pre-processing.

28: We're training as many models as we can in 2 hours, and using all of them in the ensemble. Also, stopped cloning best scoring model during training. Just use the actual model after early stopping, not the one before 'patience' epochs.

31: Maxlen increased from 30 to 75. Longer sequences is better, but still did not realize it is enough to set it to 65.

37: Using 3 embeddings separately (not averaging) on different models, can train only 9 models to fit time limit. Keep using early stopping patience of 3.

40: Finally text analysis done, realized no need to set maxlen higher than 65. This helped save time for training 1 extra model (10 total now).

## Improvement

One of the aspects that could be improved is the threshold value. Only a few tests were made to indicate that 0.33 is a good value for threshold. Yet, such tests were not made on the final model and it is possible that it is not optimal for that particular model.

Secondly, GoogleNew embeddings set was not used at all, because it is somewhat different from other embeddings, e.g. it misses some of the common words like 'to', and works with numbers differently. So, preprocessing data for this embedding and using it in some of the RNN models in the ensemble may add additional value.

Still another big area of improvement that could make a difference is other network architectures. More LSTM or Dense layers could be added. Besides, modern techniques such as using Attention in RNNs could be tried as well (see for example Recurrent Attention Unit, and Recurrent Attention Network on Memory for Aspect Sentiment Analysis, among others).

We could see from the competition's leaderboard that F1 score of over 0.7132 exist (as compared to our 0.7055). The change of 0.008 is significant for this project, so there is definitely room for improvement.