Machine Learning Engineer Nanodegree

Capstone Project

Kevin Casado April 25th, 2017

I. Definition

Project Overview

Natural Language Processing(NLP) has gotten a lot of attention lately with the improvements in deep learning. The visible impacts are in products such as googles translational application or in the resurgence of chat bots. One area that has also been generating a lot of interest is through duplicate question detection. The idea is to be able to tell if two questions are asking the same thing. Current implementations of doing so primarily use extensive feature engineering and an ensemble of machine learning models. However nobody has been able to give a definitive deep learning method that solves this problem.

Problem Statement

The problem I am going to try and solve is being able to detect whether two questions are asking the same question. Given question 1 and question 2 my machine learning model will output either a 0 (Not duplicate) or a 1 (duplicate). I will attempt to solve this using deep learning and to be more specific using variations of Recurrent Neural Networks. (https://en.wikipedia.org/wiki/ Recurrent_neural_network) Quora has realeased a dataset that contains over 400 thousand questions pairs with nearly half being duplicate pairs.

Metrics

For this project I will be using the F1 score (https://en.wikipedia.org/wiki/F1_score) to calculate the effectiveness of my model. I am choosing the F1 score because it is a well known way of incorporating not only the accuracy of the model but also the ability to have good recall and precision. I will be using the first formula without adjusting beta and simply using two as the multiplier.
The formula for F1 score is: (2 * (precision* recall))/(precision + recall). The reason for choosing this formula is that in previous experience as well as in this dataset whenever dealing with imbalanced classes the F1 score does a good job of showing the models ability to predict while also including the imbalance. As discussed below the dataset is imbalanced towards questions that are not duplicate. Explicitly analyzing recall, this is the ability to predict the situation in whether two questions are duplicates. For instance if I had a data set where it was 80% of not duplicates I could simply predict that all questions are not duplicates and I would receive 80% accuracy where as my recall would be 0%.

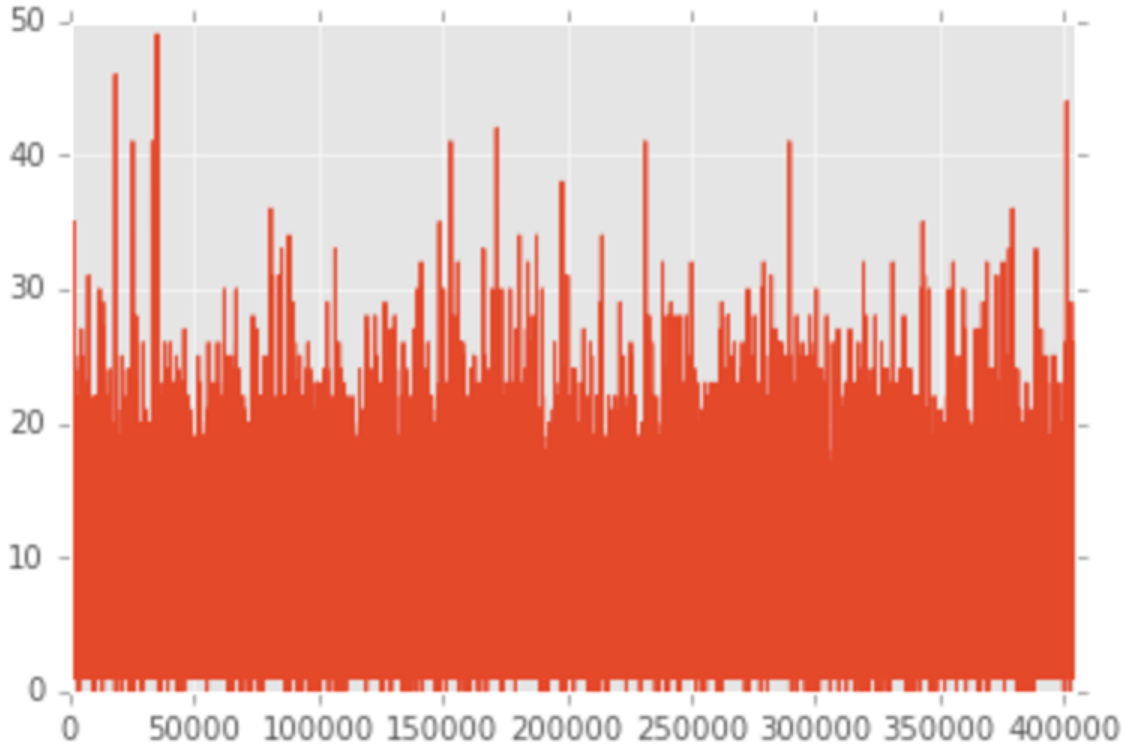
II. Analysis

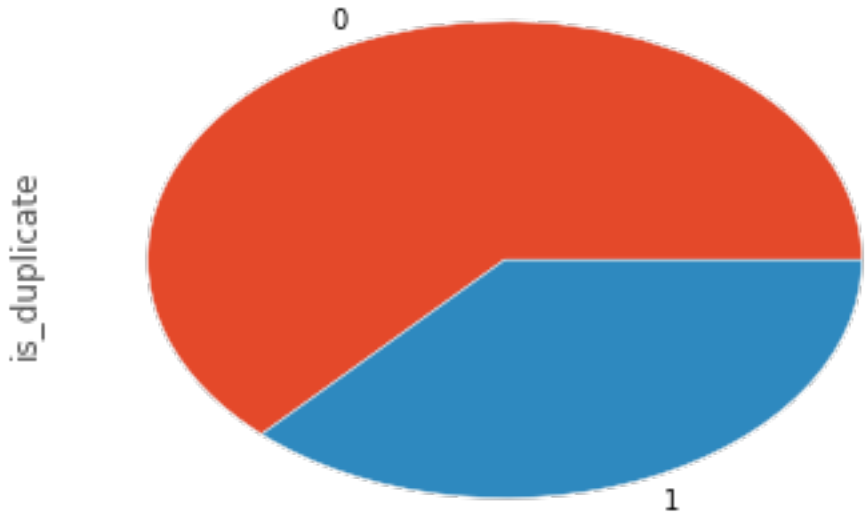
Data Exploration

The dataset that I will be using is the data provided by quora in (https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs). In total it has 400 thousand question pairs with 63% being non duplicates. When not removing any words the average length is 11 words and the maximum is 248 with a standard deviation of 5. Without removing any words the total vocabulary size is 110 thousand. For some reason when first reading in the dataset there was one row with Nan values so that was removed in the beginning. There also were some instances of brackets or braces that lead me to also remove those.
This is an example of how the data looks before cleaning:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

Exploratory Visualization





Attached are two graphs, one represents the word counts for the questions, this shows that most fall under the 30 count mark and that very few are above 40. This was used to determine the sequence length that I would like to work with.
The other associated chart is a pie chart showing the class imbalance present in the data set. In personal experience this is typically the issue faced in the industry. Whether it be churn, or specific signals that are not as common. However this can be a tricky balance as you want training data to be a good representation of existing situations and typically the existing situation is imbalanced.

Algorithms and Techniques

For NLP recurrent neural networks have shown to be beneficial because they are able to remember previous inputs. This helps in instances where the first word of a sentence plays a role on what is being said in the last word. A simple example can be seen where you say "I am not happy". In this instance if you are using a simple bag of words method you would not be able to grasp the fact that not came before happy and thus negated the meaning. When it comes to Recurrent neural networks there are two basic types of cells, Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) I will try using both but expect LSTM's to generate better results as they have shown to typically give better results.
As a general overview a LSTM has a basic structure of an RNN however the cell implementation is different. In a vanilla RNN the previous cell has a full input to the next cell. When this was first implemented many people experienced what was called a vanishing gradient problem. To counteract this one of the new implementations was a LSTM where there are various activations to determine what gets input to the next cell. In total there are four separate gates: xt, it, ft, and ht. This helps solve the vanishing gradient problem and also allows the model to have a “selective” memory towards what it decides to use.

I will also be using an embedding layer containing containing pre-trained word2vec embeddings. Word2vec has shown to be very useful as many existing semantic classification examples rely simply on word2vec embeddings. The Glove implementation is an unsupervised method that was used to convert the sparse information that is NLP towards a dense and closely related space.
As well as fully connected layers at the end before a sigmoid activation. Since most feature engineering methods use things like euclidean distance or cosine distance I will incorporate this into the representation given after the RNN. I will also try using attention to see if this generates better results as it has shown to help in many other situations.

Since before LSTM’s convolutional networks achieved state of the art for text classification I will see if feeding in the results of convolution will help the LSTM better recognize patterns. Convolutional networks have long been the best solution for computer vision as they are able to recognize features and recognize specific patterns.
So I will try a very basic LSTM siamese style approach as well as a very deep approach involving many different styles.

Benchmark:

Since the dataset has more items that are not duplicate the baseline would be anything above 63% accuracy. This is because when looking at the value counts of is_duplicate 63% are zero. However the values I will be shooting for are the ones that quora has published receiving in there own internal work. https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning The best f1 score that they get is 88%. One thing to note after looking through the

kaggle discussions (https://www.kaggle.com/c/quora-question-pairs). There was a common issue with overfitting as well as the training set being not representative of the testing set. In many of my tests I noticed that the training set overfit quickly and did not represent my validation set very well even with adding a lot of dropout.

III. Methodology

Data Preprocessing

In NLP there are many wayts to preprocess data so I will discuss some of the approaches I will try. Stemming and Lemmatization (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html) is a way to reduce the total vocabulary of the corpus that you are working with. One of the issues when dealing with NLP is how sparse the data is. In a image you can think of the possible matrix as a L x W matrix where the image has a resolution of L x W. Each value within that matrix can then be typically from 0 - 255 if you are dealing with the typical RGB values. However in NLP there are a few things to consider, The obvious consideration is whether you are looking at words and what you are denoting to be a sequence length. This can translate easily to the example of an image. However in the example of NLP it is not a simple 0-255, since words are so sparse you then are looking at a possible V features where V is the amount of unique words in the corpus. Typically this is in the hundreds of thousands or millions.

Another way of decreasing the vocabulary of the corpus is to remove all of the stop words (http://searchmicroservices.techtarget.com/definition/stop-word) these are very common words that have shown to have very little impact on the meaning of a sentence. After doing some analysis on the length of the sentence statements I was able to see that using sentence lengths of 30 encompasses a large majority of the words used in each sentence.

Word2Vec (https://en.wikipedia.org/wiki/Word2vec) embedding is currently the best way to try and attack the sparsity that is present in NLP. I will use the Glove pretrained word vectors (https://nlp.stanford.edu/projects/glove/) to initialize the values for each words and then I will train an embedding layer on top of this. I will explain the embedding layer in the implementation portion.

Implementation

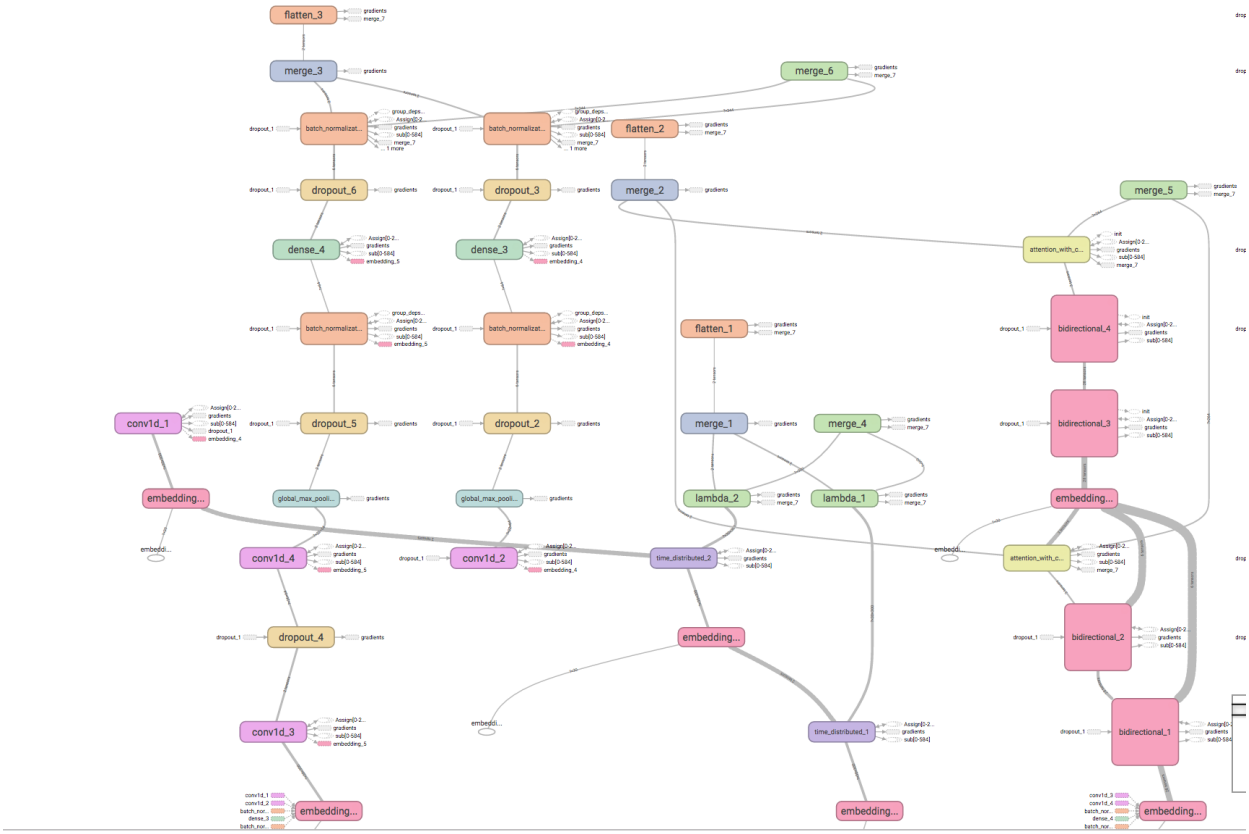
As discussed above I used all of the preprocessing methods before implementing the model. For the model I used keras as it allowed me to easily modify and run many instances without worrying about graph re instantiation or other various issues. I was still using the tensorflow backend and used tensorflow for some of the custom implementations. For the model the final result followed the illustration below. Some of the things I played around with were adding the cosine distance and euclidean distance between the two LSTM layers as well as using deeper representations with up to 3 LSTM layers all getting gradually smaller.

For the basic structure of the graph I used an embedding layer which both questions used and then fed this into two separate LSTM layers for each question. After the LSTM representation this was fed into a concatenated layer of the two representations and then fed into 2 fully connected layers with dropout and batch normalization before a sigmoid activation. I used batch normalization (https://arxiv.org/abs/1502.03167) because it has proven to help models converge faster and receive more accurate results. Dropout was used in order to try and prevent over fitting for the model. I tried to play with using different loss functions such as documented in this paper (http://www.mit.edu/~jonasm/info/MuellerThyagarajan_AAAI16.pdf) however in the end the loss that worked best was the binary cross entropy and the best optimization was the Nesterov Adam optimization(http://cs229.stanford.edu/proj2015/054_report.pdf).

As stated above I also tried adding convolutions to the network and settled on a very deep model as shown below. In this model it considers all of the major approaches people have taken in the past for text classification. There is the straight forward time series approach off of embeddings, shown in the middle, the convolutional neural network approach shown to the left and the LSTM structure shown to the right with attention added.

For the LSTM i made it bi directional as well as two layers deep. i made it bi directional because this allows the model to use the context of the statement going forwards as well as backwards. The way a bi directional simply by having nodes that go both ways. This allows the first word to also know about the last word.

Attention has shown to give improvements in allowing models to better analyze which words are more important than others. Although this seems to give better results when dealing with long texts as opposed to short ones I still used it here and found some small improvements.



After going through these layers I also included cosine distance and euclidean distance since we are trying to test for similarity.

Refinement

As discussed above on way that I really tried to refine was to create a bigger network and have more approaches incorporated into one as a type of ensemble approach. For trying to tweak hyper parameters I used random values and trained overnight to try and come up with the best values for dropout, and dense, lstm layer units. In the end I was able to come up with around a 20-30% dropout rate and having the LSTM and Dense layers around 300. This seemed to make much more of a difference the deeper I got with LSTM layers. For doing the testing I used Keras callback functions heavily such as early stopping and model checkpoint to iterate and save through model variations.

IV. Results

Model Evaluation and Validation

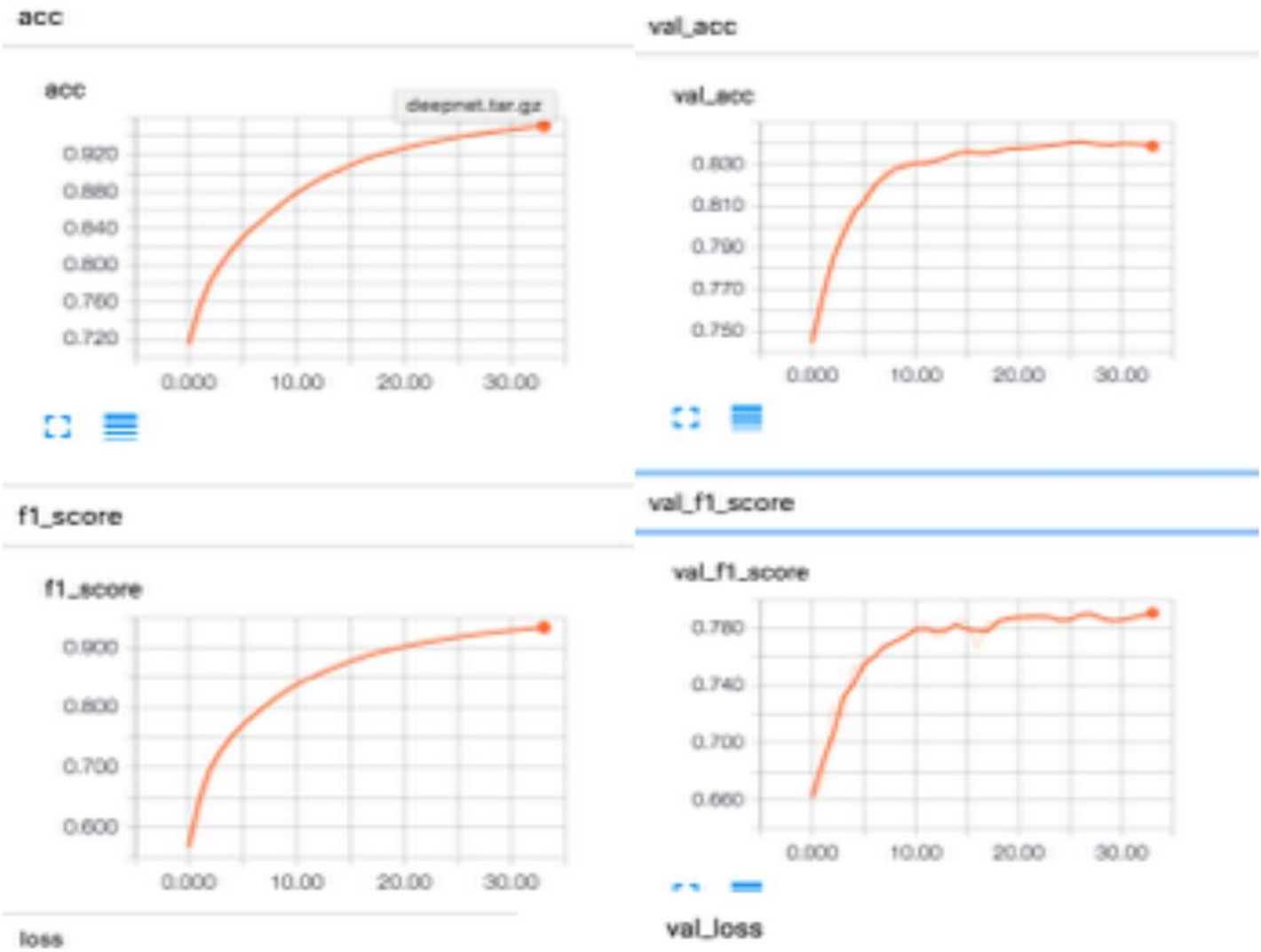
The model I chose at the end was the deepest network that I attached the diagram for. For validation data I held out 10% which was randomly selected and not used for training at all. Unfortunately there was no way to measure the f1 score on the testing dataset provided by Kaggle as the only metric that they provide is the log loss and that is independent of the f1 score. As discussed above many people felt that the training data was not representative on the testing data provided by Kaggle so I am not sure if it would perform the same but I would not expect a drastic drop off.

Justification

The final results receive an accuracy score of 84% and a f1 score of 79% for the validation date. This is higher than the benchmark I initially created. It does not hit the mark that quora was claiming but they do not state if their results are done on the same dataset. Perhaps with more data this model could do a better job of predicting. This solution does a good job of predicting whether questions are duplicates or not but I do not think quora would want to implement a solution that only had a 84% accuracy. One thing to note is that this is only slightly better then the results I received on the basic LSTM model however took 4X the amount of time to train. This would mean to infer results this would also probably cost significantly more resources. For my GTX1070 it took around 500s for one epoch compared to 90s for the simple one layer LSTM. Because of this it would probably be a better result to use the simple one layer LSTM in production.

V. Conclusion

Free-Form Visualization



This shows the training metrics during training as well as the associated value metrics. (This was gotten from tensorboard of the most successful model run) One thing to note is that the value log loss increases significantly to . 540 after reaching a minimum. This also shows the class imbalance experienced in the training and testing set. One way to solve this would be to associate class weights with the given label to mirror the class imbalance however this did not give as good of results for the f1 score. As you can also see most of the metrics flattened out after 10-15 epochs and after had slight fluctuations. This shows that the model did converge relatively quickly and using more epochs would not of improved the model significantly.

Reflection

When working on this project there were some clear processes that took overly long. For refinement of the model this by far took the longest. As making the model deeper and deeper

required longer and longer training time. Typically I would work for twenty minutes before I went to bed and had the model run over night to try and get results for the next day. This clearly illustrates why there is such a big push towards TPU's by google and trying to push the state of the art as far as hardware is concerned. I was using a GTX 1070 which is considered mid tier as far as current GPU's go and training on a GTX 1080 also did not improve training time that much. The part that was actually the most enjoyable was cleaning and munging the data. Typically this is the part that engineers dislike the most but since the feedback comes much faster then training a model I found it much more enjoyable. However this might change when you move more into big data with things like spark or h2o. Also using spacy was a lot easier than using nltk or gensim. In the past I have used a combination of gensim and nltk modules for python but spacy was significantly faster as well as much simpler to use. I don't think this solution is good enough for a production environment but it does show the capability of what deep learning can do in this area. For it to be in a production environment it would need to be tweaked further and contain more feature engineering to be more accurate.

Improvement

I think there could of been more feature engineering. The only real feature engineering done in my solution was using word vectors as opposed to raw text data and using cosine similarity and euclidean distance. Otherwise the purpose of deep learning typically is that the model will take care of the feature engineering. I think someone with domain expertise would better be able to construct features that would provide better signals for the model. I think that this solution is a simple benchmark that can easily be implemented and tested against for others to improve on.