Project 5 Summary - Identifying similar intent in a pair of questions

Jason Salazer-Adams

Overview

I wanted to explore the NLP topic of semantic similarity between questions. Specifically, whether or not two questions have the same intent or not. Training a classification model to detect whether or not two questions have the same intent can lend itself to different applications. For example, the model could be used to easily lookup answers to previously asked questions. I took this example and developed a proof of concept Slack bot to demonstrate how my trained model could search previously answered questions, and responded with similar answers.

Methodology

First, I needed a data set to train a classification model to detect whether or not a pair of questions had the same intent. I utilized a data set provided by Quora and hosted on Kaggle. The data set contained 400,000 pairs of questions which were manually labeled as having the same intent or not. The questions are all real questions posted on Quora with a diverse set of topics, and spelling and grammar mistakes.

Next, I cleaned and lemmatized the questions using the spaCy model, `en_core_web_lg`, and engineered features. The full details of the features can be found in the Feature Engineering section.

Next, I trained Random Forrest and Gradient Boosted models. The Gradient Boosted model was the `xgboost` implementation. I hyper tuned both the best Random Forrest and Gradient Boosted model, and ultimately selected the Gradient Boosted model based on average validation AUC in 3-fold cross validation.

Finally, I utilized a question and answer data set from MS MARCO as the basis for all known questions and answers in the Slack bot. The Slack bot was developed utilizing `Flask` to handle the HTTP requests from from the Slack API, and `ngrok` to expose the Flask server to the Slack API server.

Feature Engineering

I engineered two sets of features. One set focused on the pair of questions, and the other set focused on each individual question.

N-gram similarity

The n-gram similarity feature is a set comparison similarity of the n-grams from the pair of questions. I considered single, bi, and tri grams. The feature is calculated as,

2 · (|S1| / |S1 ∩ S2| + |S2| / |S1 ∩ S2|)^−1

Where S1 is the set of n-grams from the first question, and S2 is the set of n-grams from the second question.

Distance features

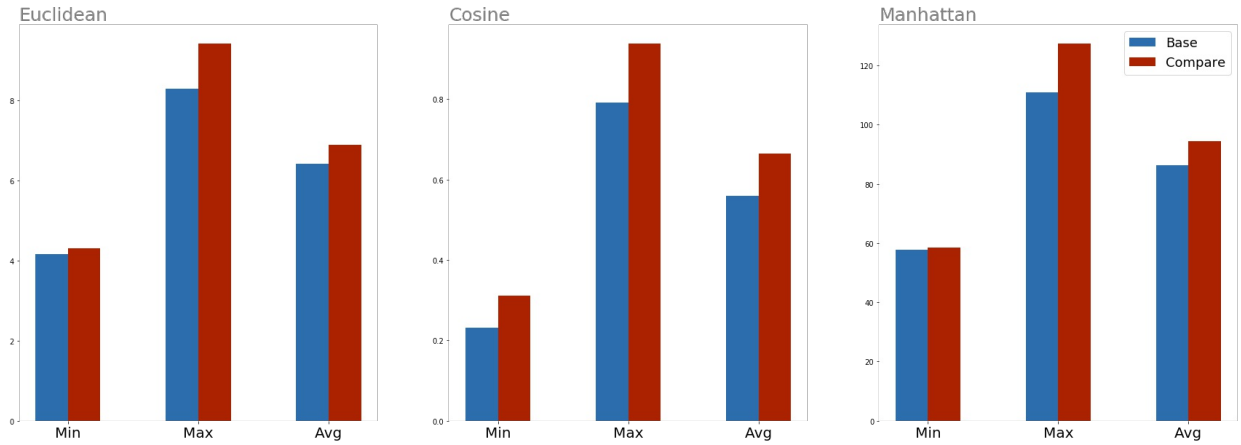
I utilized GloVe word embeddings provided by the spaCy `en_core_web_lg` model to calculate distance features for each question. The minimum, maximum, and average distance between all words in a question were calculated. The set of features was calculated on the individual question, rather than the pair of questions.

Example

Let’s consider these two questions.

1. How do I learn machine learning?
2. How do I become an astronaut?

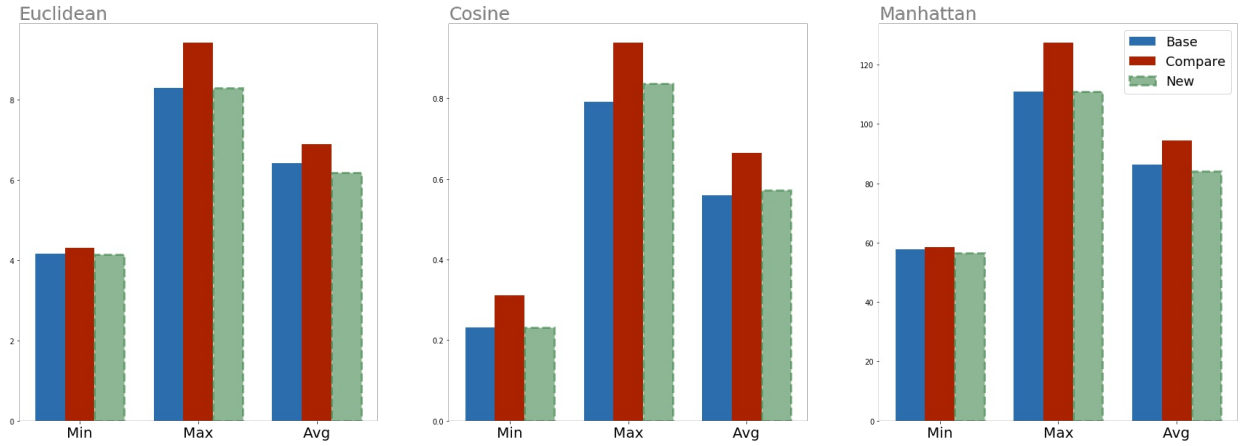
Each word in each question can be mapped to a word embedding, and then distances calculated. Euclidean, cosine, and manhattan distances were calculated, and the comparison of the two questions is below.



Now we can add a third question, unseen by the model, and see how a classification model can start to learn semantic similarities between two questions. The third question is,

3. What steps should I follow to learn machine learning?

The updated graph is now below,



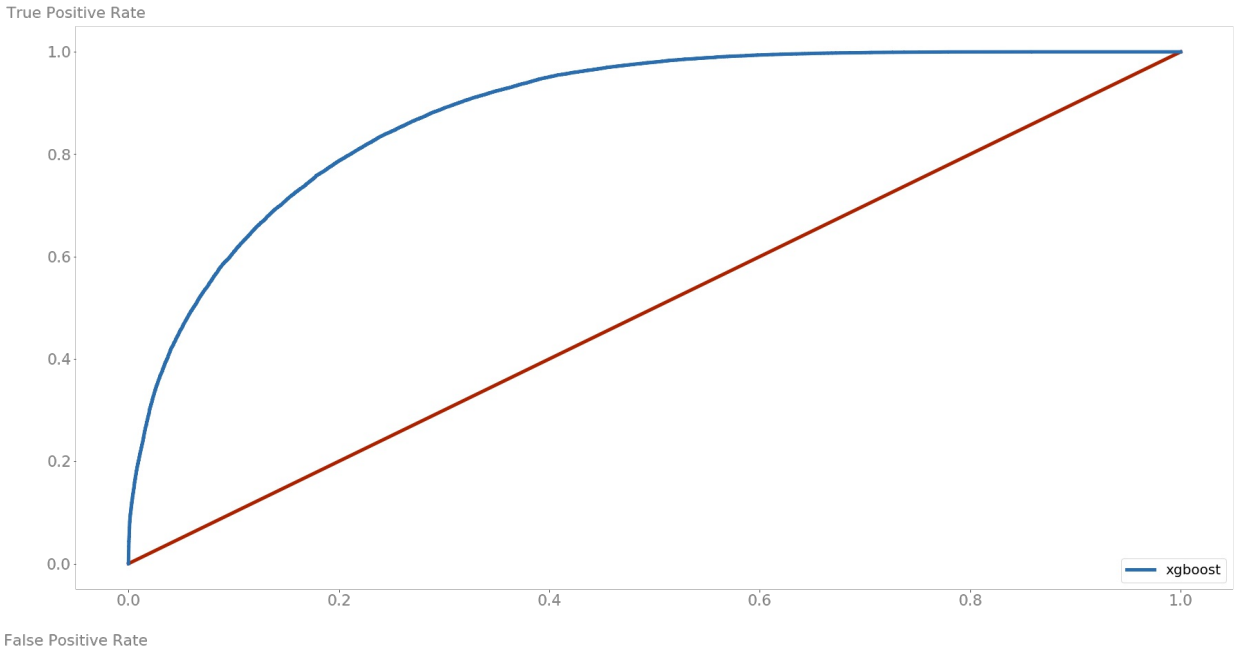
Model Selection and Results

I trained many models with different features. A summary of those models can be seen below.

	avg_auc	avg_f1
xgb_hypertuned	0.884651	0.729187
xgb_hypertuned_dup_features	0.873554	0.713308
rf_feat_eng_model_lemma_clean	0.868202	0.705774
ensemble_rf_xgb	0.863334	0.702935

xgb_feat_eng_incl_nums	0.851957	0.69159
feat_eng_model_lemma_clean	0.846923	0.684044
feat_eng_model_lemma_fix	0.822197	0.642201
feat_eng_model	0.82107	0.640434
ensemble_rf_xgb_cos_sim	0.819987	0.636128
lstm_Bidrectional	0.80354	0.632158
lstm_dropout_50	0.802315	0.638877
lstm_LEMMA_dropout_20_lstm_layer_DO_20	0.801938	0.624349
lstm_mvp	0.801019	0.637183
cos_sim_tfidf_model	0.800271	0.59901
lstm_LEMMA_SW_dropout_20_lstm_layer_DO_20	0.794318	0.622797
lstm_dropout_50_lstm_layer_DO_50	0.78585	0.61836
lstm_CNN	0.776909	0.607373
lstm_dropout50_dense50_BatchNorm	0.772344	0.573654
cos_sim_model	0.746769	0.53212
mvp (tf-idf, nmf(5), xgboost)	0.740593	0.487325
mvp (+ lemma)	0.738037	0.485464

Ultimately, Random Forrest and Gradient Boosted models with the features described within this document performed the best, and both were hyper-tuned. I hyper-tuned utilizing `BayesSearchCV` from `skopt`, and utilized validation AUC to select the `xgboost` labeled as `xgb_hypertuned_dup_features` in the above table. The ROC curve of the held-out test data is below, and the model scored an 0.88 AUC on the test data, thus a little better than the avg validation AUC.



JasonBot (Q&A Slack bot)

I applied the trained model to a question and answer Slack bot application. The underlying assumption for the application is that some historical questions have already been asked and answered. This data originated from MS MARCO. The Slack bot was deployed on the Metis Community Channel and the `/q` command was enabled to receive any question and respond with the top 3 answers. In addition to the answers, the similar question and confidence level are returned. A demo of the bot is below or here

Future Work

Here are some areas where my model can be enhanced further,

1. Utilize different word embeddings to see if different embeddings could enhance the distance feature set.
2. Apply a Siamese network architecture utilizing LSTM to classify whether or not two Quora questions have the same intent.
3. Change the Slack bot paradigm from a question and answer lookup to text generation utilizing the Wikipedia corpus.

Tools

I utilized the following tools for my analysis.

- Python
  - Data storage/manipulation: pandas, matplotlib
  - Feature Engineering: spaCy, numpy, nltk
  - Modeling: sklearn, xgboost, skopt, AWS
  - App Dev: Flask, ngrok, Slack API
- Keynote - presnetation
- Kaggle / Quora - training data
- MS MARCO - question and answer data

What I would do differently?

I felt really efficient as I was able to get a MVP within 3 days of my initial proposal. I analyzed errors from the MVP and plotted a course of which features to enhance in each iteration. This allowed me to quickly get a high AUC classification model within a week. I was then able to spend time learning Keras to see if a neural net could perform better, and also build out the Slack Bot application. I wish I could have gotten an ensemble model or a neural network to work as I had hoped. However, I am happy with my interpretable model, and look forward to working on neural network as a future enhancement to this project.