Capstone Project - Machine Learning Engineer Nanodegree - Crowdflower Search Results Relevance

Arvin Dwarka

September 10th, 2016

Definition

Project Overview

We, humans, are adept communicators compared to other lifeforms on this planet. Through the act of communication we can share what's on our mind, our desires, our needs and our pains. We naturally do this verbally, but as literacy have become table stakes in the modern world, we have incorporated the written form of communication in the daily lives. We've embraced the latter has been so much that we are inundated with content causing the need to search for everything. Historically that was done by sifting through a catalogue at a library and recently, and digitally, that was done by using a search algorithm on a web browser.

So many of our favorite daily activities are mediated by proprietary search algorithms. Whether you're trying to find a stream of that reality TV show on cat herding or shopping an eCommerce site for a new set of Japanese sushi knives, the relevance of search results is often responsible for your (un)happiness. Currently, small online businesses have no good way of evaluating the performance of their search algorithms, making it difficult for them to provide an exceptional customer experience.

The goal of this project is to create a model that can be used to measure the relevance of search results. Given the queries and resulting product descriptions from leading eCommerce sites, this project tries to evaluate the accuracy of search algorithms.To evaluate search relevancy, CrowdFlower has had their crowd evaluate searches from a handful of eCommerce websites and launched a machine learning competition on Kaggle.

A total of 261 search terms were generated, and CrowdFlower put together a list of products and their corresponding search terms. Each rater in the crowd was asked to give a product search term a score of 1, 2, 3, 4, with 4 indicating the item completely satisfies the search query, and 1 indicating the item doesn't match the search term.

The challenge is to predict the relevance score given the product description and product title. To ensure that the algorithm is robust enough to handle any noisy HTML snippets in the wild real world, the data provided in the product description field is raw and contains information that is irrelevant to the product. Finally, the end algorithm would be computationally advanced enough to process the written text and assign relevance. This will involve Natural Language Processing (NLP) that will be explored and discussed throughout this project.

Problem Statement

This project is looking to solve the problem of search results' relevance to users so that it can be used as a measure of performance for search algorithms. As mentioned above, a total of 261 search terms were generated and ranked by a human rater on a scale of 1 to 4 where 4 indicates the item completely satisfies the search query, and 1 indicates the item doesn't match the search term. This makes up the the training dataset used for fitting supervised machine learning models to. The latter model is then used on an unlabeled dataset, called the test data, where a predicted score of 1-4 is assigned. The aim here is to accurately score the test data and submit that dataset to the kaggle competition. While the competition has passed its deadline and is inactive, test data predictions can still be submitted for benchmarking on the leaderboard and a true evaluation of the performance of the algorithms developed based on the quadratic weighted kappa. The latter will be discussed in detail in a subsequent section.

After exploring the dataset and understanding its makeup and structure, it became apparent that this was an NLP problem. Using the given text, some form of meaning needed to be derived from the queries and product titles and description that would allow the inference of their relevance. Numerous public attempts were made on this Kaggle competition from most basic to the most advance (see winner's interview). I decided to base my work on my understanding from researching NLP. This meant that the solution here would have considerable room for improvement, but one that I can speak to comfortably.

The approach taken in this project is as follows: - pre-process the data to remove noise in the form of html tags, remove all characters that are not alphanumeric, and stop words - reduce words to with root/base form, called stemming - attempt feature engineering where various combinations of queries, product titles and product descriptions are tried out - transform the data using a Term Frequency Inverse Document Frequency (TF-IDF) vectorizer - fit the various models to the vectorized data - evaluate the performance of each model using cross validation splits of the training dataset on the quadratic weighted kappa metric - tune the models if needed with grid search cross validation technique with various parameter settings - perform predictions on the test dataset using tuned models - ensemble the models predictions to get a final score - submit the test dataset to the Kaggle competition for benchmarking and evaluation

Metrics

The metric used is quadratic weighted kappa, which measures the agreement between two ratings. This metric typically varies from 0 (random agreement between raters) to 1 (complete agreement between raters). In the event that there is less agreement between the scorers than expected by chance, the metric may go below 0. The quadratic weighted kappa is calculated between the scores assigned by the human rater and the predicted scores, which make it a good measure for qualitative and categorical items. It is generally thought to be a more robust measure than simple percent agreement calculation, since it takes into account the agreement occurring by chance. The choice of this metric is further bolstered by the fact that it is also the chosen metric for the Kaggle competition.

The results have four possible ratings: 1,2,3,4. Each search record is characterized by a tuple (ea,eb), which corresponds to its scores by Rater A (human) and Rater B (predicted). The quadratic weighted kappa is calculated as follows:

First, an N x N histogram matrix O is constructed, such that Oi,j corresponds to the number of search records that received a rating i by A and a rating j by B. An N-by-N matrix of weights, w, is calculated based on the difference between raters' scores:

$$w_{i,j} = \frac{(i-j)^2}{(N-1)^2}$$

An N-by-N histogram matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater's histogram vector of ratings, normalized such that E and O have the same sum.

From these three matrices, the quadratic weighted kappa is calculated as:

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}$$

Analysis

Data Exploration

A good place to start exploring, is by looking at the head of the dataset (a sample is not highlighted here due to the length texts). We can easily pick out some abnormalities in the dataset that would need pre-processing, such as special characters `itâ€™ll`, `dÃ©cor` and `\n`.

The following are some snippets of noisy text that would require cleaning:

```
WTGR1011\nFeatures\nNickel base, 60,000 average hours, acrylic resin bulb
material\nChristmas light bulb\nSteady dimmable replacement lamps\nNickel bases
prevent corrosion in sockets\nWattage: 0.96 Watts\nVoltage: 130 Volts\nDimmable:
Yes\nLight Source: LED\nBulb Shape Type: Candle\n\nColor Amber\nBulb Color: Amber
```

```
'ITEM#: 13308316\nProtect your passport and stay organized with this sleek, low
profile folio. This folio features one clear pocket for your passport and five
additional pockets for business, credit or travel cards.'
```

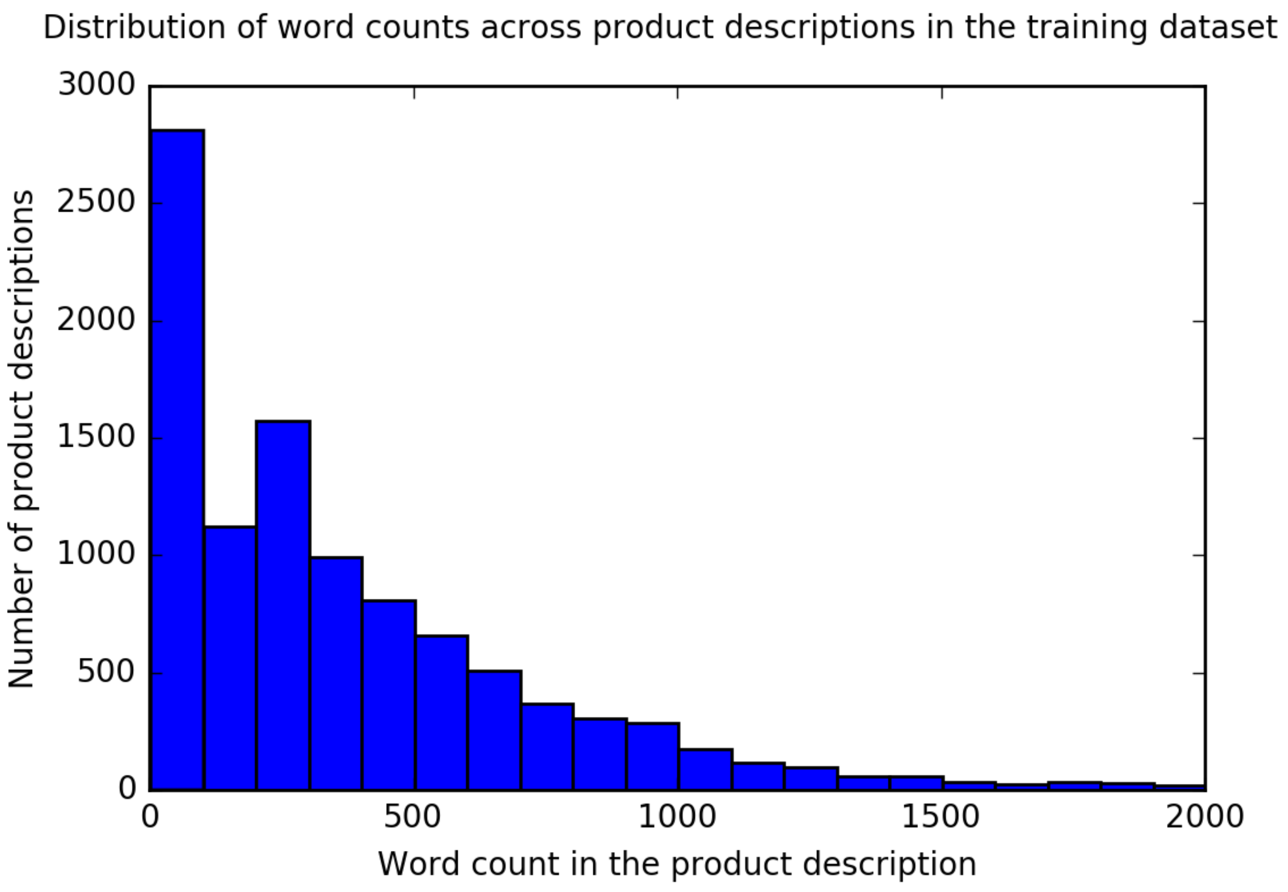
```
'Linksys Smart Wi-Fi Router with wireless AC technology\n2.4/5 GHZ dual band
wireless\nCompatible with 802.11a/b/g/n/ac WiFi\nSecurity protocols supported\n3
external antennae\n4 Ethernet ports\nCompatible with Windows 7 and 8'
```

As observed, there are a fair bit of new line separators (`\n`) for HTML styling that would need to be addressed. Secondly, puctuations and special characters would need to be removed. Lastly, and understandably, there are multiple variations of words, such as print, printing, printable, and prints that should be grouped together - this can be addressed by stemming (discussed below).

Diving into the dataset some more, we can observe that the product descriptions are not available for all. Out of the 10158 records, 2444 had missing values (NaN), which makes up 24% of the records. This represent a significant imbalance in the dataset. To overcome this, I will look into clearing out the NaN values and then concatenating the query, product titles and product descriptions prior to vectorizing to enrich the corpus as much as possible.

The target, `median_relevance` had a mean of 3.309805 and a standard deviation of 0.980666. The score of 4 indicates the item completely satisfies the search query. Since the mean is so high, it makes sense that the metric used in this project is not a precision score. That would have enabled algorithms to overfit.

Exploratory Visualization



The figure about shows the word count distribution of product descriptions in the training dataset. The graph is skewed to the right with a fairly long tail. This is particular interesting as I would have assumed that there would have been some strict word restrictions on a product webpage. It could potentially complicate things during the vectorizing phase as it might blow up the vector size. To mitigate this, I might either truncate the vector size or reduce the words feeding into the vectorizer. I have a preference for the former as the latter leads to loss in information.

Algorithms and Techniques

I decided to try out ensembling for this project. Ensembling is a general term for combining many classifiers by averaging or voting. It is a form of meta learning in that it focuses on how to merge results of arbitrary underlying classifiers. For the models themselves, I decided to use Support Vector Machine, Gaussian Naive Bayes, Stochastic Gradient Descent.

The ensembling technique was chosen as it allowed me to work with multiple algorithms and combine them such that the result was greater than each individual one alone. I used a weighted model average scoring method that compared each model's performance on the quadratic weighted kappa scale. A random set of weights were used to vary the output of the tree models and the average kappa score measured. The weights that produced the highest kappa score was used on the test prediction datasets to produce the final ensemble submission.

I hypothesized that the distribution of words relative to their search relevance would have fairly formed clusters. Given this assumption, Support Vector Machine (SVM) was a logical choice as it is discriminative classifier defined by its separating hyperplane. But it alone was rather ineffective as the vectorized training data greatly inflated the training space complexity. Therefore, I decided to use a Pipeline method. It is essentially used to chain a set of steps together: dimensionality reduction, standardization and SVM. Linear dimensionality was reduced by means of truncated singular value decomposition (SVD) to 400 components. Then, the features were standardized using a Standard Scaler to give a normally distributed shape. This fed into the SVM algorithm nicely.

Gaussian Naive Bayes (NB) are commonly used for spam detection, which made it an appropriate model for this purpose too. NB works by computing the probability of a word being related to an outcome. In the case of spam, messages containing the word "counterfeit" have a much higher chance of being unsolicited, therefore leading to a higher probability value of it being spam. Similarly in the search relevance, words would have a higher probability value if they are related to the body of text.

The last model is an estimator that implements regularized linear models with stochastic gradient descent (SGD) learning. It is a simple yet very efficient approach to discriminative learning of linear classifiers, similar to SVMs. SGD has also been applied heavily in NLP, which made it one to consider for this project. The fact that it was a fast and efficient algorithm and that it supports multi-class classification by combining multiple binary classifiers in a “one versus all” (OVA) scheme solidified its choosing.

The input training data will preprocessed and vectorized, and then being handled by the SVM pipeline, NB and SGD models. Using cross validation techniques on the training data, five splits will be made and used to test the models' fit on the quadratic weighted kappa score. After the models are tuned, they will then be fitted with the whole training dataset and used to predict the test dataset. The output of the models will be ensembled using a weighted average approach to produce the final submission.

Benchmark

The benchmark for this project is taken directly from that of the Kaggle competition's python benchmark code score of 0.34125 on the quadratic weighted kappa metric. This benchmark was provided by the competition's organizers who released a started code that lead to the score of 0.34125. While it's tempting to try and measure up to the winner's 0.72189 score, I will instead compare my model's performance solely on the competition's benchmark score of 0.34125.

Methodology

Data Preprocessing

There were some main issues with the dataset. Namely, lots of NaNs particularly in product descriptions, prevalence of html noise, special characters, and a large word distribution. Therefore, there was a need for some significant preprocessing.

To begin, I filled all NaN values with an empty string. Then I proceeded to parse the text to remove all the unrelated characters, such as '\n'. I used the BeautifulSoup library with lxml parser. This went through each data columns and removed non-relevant characters and outputted UTF-8 text.

Then it was time to remove special characters, such as punctuations, that give meaning to human readers but create a lot of interference in statistical analysis techniques employed in this project. To do this, I used a regular expression parser that removed all non-alphanumeric values.

This made it easy to start stemming the words to their root form. A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argues" reduce to the stem "argu". This is important as it reduces the vector space considerably and allows the algorithm to better retrieve information. I used the Snowball Stemmer, which is an improvement of the classical [Porter Stemmer](https://tartarus.org/martin/PorterStemmer/.

Lastly, I also removed stop words from the corpus. Stop words are words that appear very frequently in the English language and tend to drown out statistical analysis of other less frequent but more meaningful words. The following are some examples of stop words removed:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her',
'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no',
'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain',
'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn',
'mustn', 'needn', 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn']
```

The idea is to penalize the total count of a word in a document by the number of its appearance in all of the documents. The higher this number the less valuable the word is – it contains less information that can identify the document. In the extreme case, where the word appears in large fraction of the documents, usually it is even better to completely eliminate the count. This is known as stop words, or corpus specific stop words.

Once the data was cleaned thoroughly, I performed some simple feature engineering. I combined the query, product title and description together. The reason why I chose this is that I plan on vectorizing the data. That is, counting the occurrence of words in the given corpus. Two extremes are to count them as many time as they appear, or just count them once and ignore the other appearance of the word. But a more sensible approach is to use a Term Frequency Inverse Document Frequency (TF-IDF) function approach. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Implementation

There were three algorithms used as described above: SVM Pipeline, NB, and SCD. They all took in vectorized data for fitting and prediction. To empirically evaluate each model, I used a five fold cross validation technique on the training dataset. This basically splits the training dataset into five parts, each with its own training and testing set with features and labels. Cross validation is crucial to avoid overfitting the data. But performing five validation using the quadratic weighted kappa scorer, each algorithm was assigned an average score that could be compared to the benchmark as a measure of performance.

SVM was the most sensitive to the large number of features after vectorizing due to its space complexity. This is why a pipeline method was used where the number of features were truncated than standardized before fitting to the model. Truncating the vector space into a denser one made it easier for the SVM model to process the data.

NB and SCG were comparatively much easier to work with. Since they both had simpler space complexities, they had a relatively quick training time despite the large number of features.

The cross validated outputs were used in the ensembling phase to determine the weights to be assigned to each model. The weight vectors were generated from a combination of the following five numbers: 0, 0.1, 0.25, 0.5, 0.2, 1.0. All possible combination were attempted on cross validated output. The max weights were selected by tuning for the maximum quadratic weighted kappa score. This max weight was then applied respectively to the three models' predictions of the test dataset. Once the values were rounded, they were submitted to the Kaggle leaderboard for final evaluation. The models were tuned to optimize for higher kappa score, and with an aim to outperform the benchmark score.

The following is an implementation steps carried out in this project:

1. load datasets in a pandas dataframe
2. parse the query, product title and product description columns using
   - the BeautifulSoup library to remove html tags
   - a regular expression to remove non-alphanumeric characters
   - the Snowball stemmer library to reduce words to their base form
   - the Stopwords library to remove high frequency stop words
3. concatenate query, product title and product description together as the feature set
4. assign median relevance as the target
5. split the training dataset using the Stratified K-fold function for model cross validation (not for performing predictions on the test dataset)
6. vectorize the feature set using TD-IDF vectorizer with 1 to 4 ngrams
7. fit the models with the vectorized data
8. predict median relevance target
9. compute quadratic weighted kappa scores
10. tune the model parameters using a Grid Search Cross Validation method if needed, and repeat steps 5 to 8 unti a satisfactory score is received
11. update model parameters for final predictions of the test dataset
12. vectorize the entire training features together and fit to the models
13. predict the median relevance from the test dataset features
14. determine weightages to be used on the three models by optimizing for the higest quadratic weighted kappa score
15. use the max weights to ensemble the three models' output together
16. submit to the Kaggle competition to evaluate performance against the benchmark

Refinement

There were two main parts that were refined for the final version: the models, and the vectorizer.

The models were tuned using a Grid Search Cross Validation technique that exhaustively searches over specified parameter values. A dictionary object would be inputted, full of various parameter combination, to the Grid Search function that would test out all possible combinations. It would optimize for the highest quadratic weighted kappa score. An example of this where the NB model is tuned for the alpha value is available in the code base, but commented out to preserve computing efficiency. Notable parameters were capping the number of components to 400 on TruncatedSVD function, using a 0.0015 alpha value on the NB model, and using a modified huber loss function on the SGD model.

The following are the models and their parameters that were tuned:

- SVM Pipeline:

  - `TruncatedSVD`:
    - `n_components` (number of components)
  - `StandardScaler`:
    - `with_std` (scale the data to unit variance)
  - `SVC`:
    - `C` (Penalty parameter C of the error term)
    - `kernel` (kernel type to be used in the algorithm)
    - `gamma` (kernel coefficient)
- NB:
  - `alpha` (additive (Laplace/Lidstone) smoothing parameter)
- SGD:
  - `loss` (the loss function to be used)
  - `n_iter` (the number of passed over the training data performed)
  - `shuffle` (whether or not the training data should be shuffled after each epoch)

These are the results before and after tuning the models:

Model	Before tuning	After tuning
SVM Pipeline	0.310	0.437
NB	0.127	0.413
SGD	0.322	0.412

The TF-IDF vectorizer was refined mainly for the minimum document frequency (min_df), and the ngram range. The min_df is important when building the vocabulary of vectors such that it sets a cut-off point on terms that have a document frequency lower than that specified. It was found, after much trial, that a default value of 1 actually resulted in the best performance. The other parameter tuned was the n-gram, which is a contiguous sequence of n items from a given sequence of text or speech. The n_gram range specifies the lower and upper boundary of the range of n-values for different n-grams to be extracted. The best value was (1,4) that produced a sense of continuation in the data by preserving some distance relations of words.

Results

Model Evaluation and Validation

The aim of the models is to accurately predict the search relevance by measure of beating the competition's benchmark score of 0.34125. To arrive at the final ensembled model, each individual algorithm was tuned using a Grid Search Cross Validation technique to find its best parameters. Then they were further evaluated by performing a five fold Cross Validation fit and predict on the training dataset.

The following are the quadratic weighted kappa scores for each of the final models along with their tuned parameters.

```
Pipeline(steps=[
    ('svd', TruncatedSVD(algorithm='randomized', n_components=400, n_iter=5,
       random_state=None, tol=0.0)),
    ('scl', StandardScaler(copy=True, with_mean=True, with_std=True)),
    ('svm', SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
      max_iter=-1, probability=False, random_state=None, shrinking=True,
      tol=0.001, verbose=False))])

Kfold score 1: 0.40373079476
Kfold score 2: 0.459261263657
Kfold score 3: 0.430236862101
Kfold score 4: 0.454702295811
Kfold score 5: 0.438576122171

Average score: 0.4373014677
```

```
MultinomialNB(alpha=0.0015, class_prior=None, fit_prior=True)

Kfold score 1: 0.399317084143
Kfold score 2: 0.423828888216
Kfold score 3: 0.412011399384
Kfold score 4: 0.442722924395
Kfold score 5: 0.387523476607

Average score: 0.413080754549
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='modified_huber', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=0, shuffle=True, verbose=0,
       warm_start=False)

Kfold score 1: 0.412482658164
Kfold score 2: 0.414299975701
Kfold score 3: 0.402699915436
Kfold score 4: 0.433199606389
Kfold score 5: 0.397759184077

Average score: 0.412088267953
```

The scores of all three models had a fairly low variance, which indicated that the algorithms were generalizing the dataset well and it robust to unseen data. The average scores outperformed the benchmark score of 0.34125, which is very much inline with expectations.

The ensembling technique used was a weighted average score one, where a multitude of weights of the three models were attempted for a max quadratic weighted kappa score. The following are the respective weights for the SVM Pipeline, NB and SGD models, and their corresponding score:

```
Best set of weights: (0.25, 0.25, 0.5)
Corresponding score: 0.463531102751
```

The final ensembled model had a score of 0.48710 from the Kaggle leaderboard.

Justification

The final ensembled predictions gave a score of 0.48710, which is higher that the benchmark score of 0.34125. This represents a 43% increase in performance from the benchmark. While the final score is still a far cry away from the winning entry of 0.72189, it still significantly outperforms the set benchmark and thus meets the goal of predicting the search relevance to a higher degree.

Conclusion

Free-Form Visualization

A good way to familiarize oneself with a metric is to try it out on a sandbox example and to tinker with it slightly. The snippet below is the output of this test to see how the quadratic weighted kappa score would react to slightly different predictions in comparison to more familiar metrics.

```
ONE EXTREME ERROR
Ground truth:   [1 2 3 1 4 4 4 4 4 4]
Predicted   :   [4 2 3 1 4 4 4 4 4 4]
MAE         :   0.3
Accuracy    :   0.9
Kappa       :   0.656488549618

FIVE SMALL ERRORS
Ground truth:   [1 2 3 1 4 4 4 4 4 4]
Predicted   :   [1 2 3 1 4 3 3 3 3 2]
MAE         :   0.6
Accuracy    :   0.5
Kappa       :   0.703703703704

KAPPA CHANGES WHEN DISTRIBUTION CHANGES
Ground truth:   [1 1 3 1 4 4 4 4 4 4]
Predicted   :   [1 1 3 1 4 3 3 3 3 2]
MAE         :   0.6
Accuracy    :   0.5
Kappa       :   0.75
```

The biggest takeaway was that a single big mistaken prediction can damage the score more than 50% small mistaken predictions. The winner of the competition, Chenglong Chen, showed on the blog that he had an understanding of the instability of the metric too. He used this knowledge to increase the number of bagged ensemble models and actively tried to combat this instability. The understanding of the metric and using the ensemble technique gave the winner an edge. This heavily influenced my decision to try out an ensemble of algorithms in this project.

Reflection

The goal of this project was to predict search relevance of test data after training a supervised algorithm. My approach had three main parts: pre-processing, modeling and ensembling.

The pre-processing portion included cleaning the dataset of NaN values, removing html tags and special characters, parsing the dataset of non-alphanumeric characters, reducing words to their base form through stemming, and removing stop words. This was an interesting part as I learned how to handle noisy, real-world data. I found myself spending a lot more time here than expected as I explored the dataset and found unexpected characters and words. Though, it was fun learning how to elegantly parse through text and output a workable corpus.

The modeling piece was fairly straight forward to me, save for the vectorizer. Here the cleaned data was vectorized using the TD-IDF vectorizer with ngrams of 1-4. This converted the text data to numerical data, and also kept some of the spacial information intact with the ngrams. After that, an SVM Pipeline was setup where the feature set was truncated, standardized, and then fitted with an SVM model. The other two models used were NB and SGD. The three chosen models were tuned extensively using a Grid Search Cross Validation technique that optimized for the best quadratic weighted kappa score across an array of given parameters. The best parameters were updated in the model and five-fold cross validation training and testing were performed to test the generalizability of the algorithms and their performances.

The ensembling technique was completely novel to me, and was a complex but fun process to learn. The biggest hurdle was understanding how to deal with the data structures present to compute the values needed. After much research and trial, using the cross validated datasets of the three models, a tuple of three weights were selected that gave the max quadratic weighted kappa score. The selection process was done by iterating through all possible combinations of a set of floats between 0 and 1, and computing the kappa score. The best weights had the highest corresponding kappa score of 0.463531102751. The predictions of the SVM Pipeline, NB and SGD were weighted 0.25, 0.25 and 0.5, respectively. That ensembled prediction was then rounded before submitted for final evaluation on the Kaggle leaderboard. It received a score of 0.48710, which is 43% higher that the benchmark score of 0.34125. As such, it can be used to test search relevance of queries.

Improvement

The biggest area of improvement is the feature engineering. In this project, I vectorized the whole corpus and modeled off it directly. Looking through some of the Kaggle post on the competition and the write up from the winner, it appears that the correlation or distance between query and product title/description is a strong predictor of search relevance. Jaccard coefficient and Dice discance can be used as distance metric measures, which could be used to group similar samples together. This would allow a better performance on clustering algorithms like SVM.

A notable mention that could have improved the output here would be to perform spelling correction and synonym replacements on the text data. Lastly, there was much praise given to XGBoost and neural network algorithms as they then to score well in this competition. Perhaps the most telling is the winner's comments on XGBoost, "...the best single model we have obtained during the competition was an XGBoost model with linear booster... score 0.70768." They were not attempted here, but it does appear that their proper implementation could very much out-perform the best score obtained.