Machine Learning Engineer Nanodegree

Capstone Project

December 31st, 2019

I. Definition

Project Overview

I found this challenge from Kraggle. https://www.kaggle.com/c/home-depot-product-search-relevance

Customer can come to homedepot.com to find and buy the latest products and to get timely solutions to their home improvement needs. It is critical for HomeDepot not only to provide accurate product information but also relevant results which best match customers'need.

Search relevancy is an implicit measure Home Depot uses to gauge how quickly they can get customers to the right products. Currently, human raters evaluate the impact of potential changes to their search algorithms, which is a slow and subjective process. By removing or minimizing human input in search relevance evaluation, Home Depot hopes to increase the number of iterations their team can perform on the current search algorithms.

Search relevance is a challenging problem to solve as part of search engine system for any ecommerce company. More importantly, search relevance plays critical important role to convert the view/browser of customer activity to generate final sale. Some of papers have more details on how search relevance works.

For example, this one amazon search, presents few challenges on improve search relevance, including how to match product type and search queries and how to find specific vertical against query. Another one is here eBay search, which talks about in details on how information retrieval technologies are applied to get to the best search relevance.

The relevance between search/product pair are evaluated by human raters. However, human rating is a slow and subjective process. There are almost infinite search/product pairs so it is impossible for human raters to give a relevance score to all possible search/product pairs.

A better way is to build a model based on search/product pair and its relevance score and use it to predict the relevance

The core technologies in the domain of search relevance comes down to how to find the closeness in terms of simlarity between query and item textual info and the model needs to optimized to match and evaluated/tested by human raters's judgement.

Problem Statement

The data set contains a number of products and real customer search terms from Home Depot's website.

To create the ground truth labels, Home Depot has crowd-sourced the search/product pairs to multiple human raters.

The relevance is a number between 1 (not relevant) to 3 (highly relevant). Each pair was evaluated by at least three human raters. The provided relevance scores are the average value of the ratings. The relevance score is between 1 (not relevant) to 3 (perfect match). A score of 2 represents partially or somewhat relevant.

The available raw data consists of

- search_term
- product_title
- relevance

1. As the raw data are all text information, it can not be used to train the model. So the first step is to map text information, i.e search terms and product title to numbers. That's called text vectorization in NLP world. A simple and effective model for thinking about text documents in machine learning is called the Bag-of-Words Model, or BoW. In brief, it throws away all of the order information in the words and focuses on the occurrence of words in a document. For example, how many terms of search query are found in product title.
2. Pre-processing or transforming text into something an algorithm can digest it a complicated process. Before analyzing a text document, a consistence processing steps need to apply to both search query and product title. Tokenization, stop-words elimination and stemming are the most widely used pre-processing methods. With the context of this specific problem, some other procedures are also needed, for example, normalize units/size info, etc.
3. Now explore the features, understand the features, target variable and get some basic observations.
4. Train a baseline model using Linear Regression. Gauge the performance. Then train several other algorithms and gauge the performance.
5. Tune the ensemble model and use GridSearchCV to find best parameters for the models and do final evaluate and see if there are more ways to improve it.

Metrics

The quality of the model is evaluated using root mean squared error (RMSE), as it is suggested by original problem

$$\mathrm{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

Figure 1: RMSE

The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and it is the most important criterion for fit if the main purpose of the model is prediction. Besides, it is largely used for numerical predictions, which is what this problem fit.

Choosing the Right Metric implies RMSE is the default metric of many models for prediction because loss function defined in terms of RMSE is smoothly differentiable and makes it easier to perform mathematical operations.

In short, we will evaluate model on the root mean squared error, as it is a reasonable pick.

II. Analysis

Data Exploration

code/step01-raw-data-analysis.py

- Train dataset has total 74067 instances

There are 5 columns in train set, named as 'id', 'product_uid', u'product_title', u'search_term', u'relevance' where `relevance` is relevance of pair search_term / product per human rater

- Checking into `search_term`, the size of search query

`COUNT(UNIQ(search_term) = 11795`

and len of tokens in each search term

```
mean          3.159207      std           1.262096      min           1.000000
25%           2.000000      50%           3.000000      75%           4.000000
max          14.000000
```

- Chekcing into the `relevance`,

```
count    74067.000000   mean          2.381634   std           0.533984
min          1.000000   25%           2.000000   50%           2.330000
75%          3.000000   max           3.000000
```

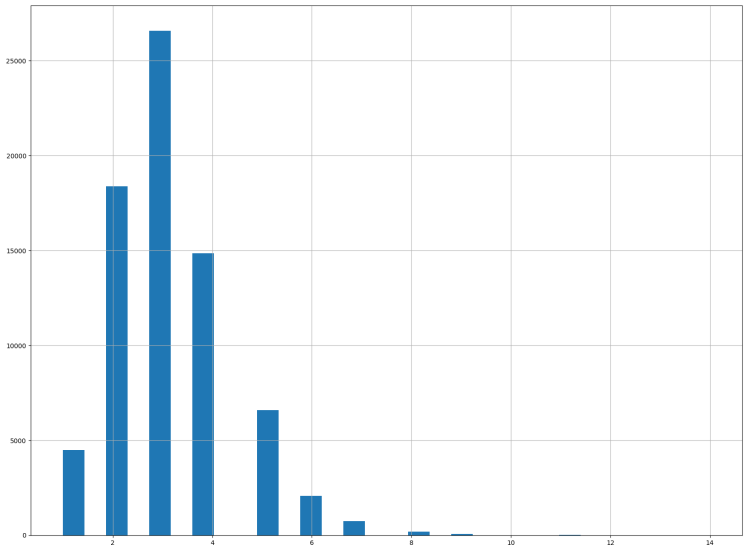
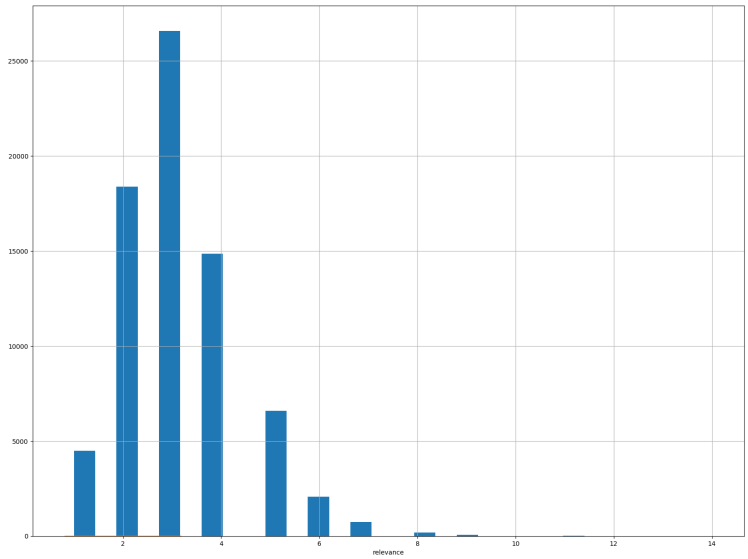


Figure 2: 01.search-term-histogram.png

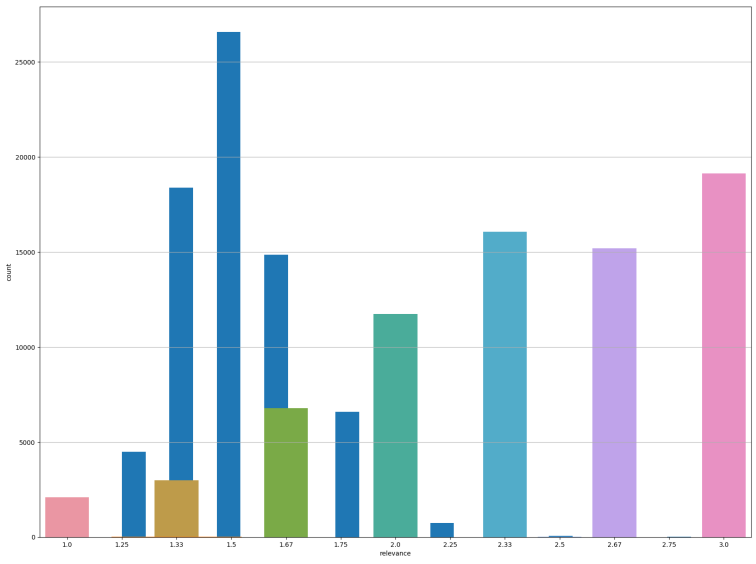
the search term mostly are short queries, as mean=2

Exploratory Visualization

code/step01-raw-data-analysis.py



distplot



counterplot

The relevance is a number between 1 to 3.

- `1 (not relevant)`
- `2 (mildly relevant)`
- `3 (highly relevant).`

Per the above diagram, the query and products are mostly relevance per countplot

There are more details to explore after feature pre-processing...

Algorithms and Techniques

I will use various NLP techniques to generate new numeric features out of original text data. Then I will train SKLearn regressors, optimized with grid search.

Programing language - python and `Scikit-learn` toolkit

Benchmark Model

- Use simple naive model as baseline model - `Linear Regression`
- Per Kraggle leader board, the top one is `0.43192` of RMSE. I will upload my model to compare and hope to achieve good ranking

III. Methodology

Data Preprocessing

code/step02-feature-engineering.py

- **FUNC preprocessingTextFeatures**

run uniform normalization process against three text fields

`search_term`, `product_title`, `product_description`

The process consists three basic steps, - `Tokenization` - `standardize_units` - `PorterStemmer`

- **FUNC extractNumberFeatures** Feature extraction is the most important part and most challenge for this problem.

I adopted the common/basic technique of NLP and information retrival

- Counting features • len_query : length of the search term • len_title: length of the title of each product • len_desc: length of the description of each product
- Common words in search terms and the text information about the product • query_feq_title: common words between search terms and title of the product • query_feq_desc: common words between search terms and description of the product

- Statistical features
  • term_ratio_title: term_feq_title / len_query • term_ratio_desc: term_feq_desc / len_query

Numberic Feature Exploration

code/step03-data-exploration.py

- corrmatrix

```
relevance           1.000000
term_ratio_title    0.352815
term_ratio_desc     0.285134
term_feq_title      0.215921
query_feq_title     0.170965
term_feq_desc       0.161321
query_feq_desc      0.086555
len_desc            0.040001
len_title          -0.019840
len_query          -0.073189
product_uid        -0.130656
```

- pairplot

Correlation coefficients are always values between -1 and 1, where -1 shows a perfect, linear negative correlation, and 1 shows a perfect, linear positive correlation

per above plot, the relative highly positve corelated features are - `term_ratio_title` - `term_ratio_desc` - `term_feq_title` - `query_feq_title`

even though the corelation overall is not strong.

On the other hand, `product_uid` shows negative corelated, which is a little surpise. It has to do with how product_uid is chosen internal to home depot

- relevance-vs-productid-pointplot

This diagram implies some correlation between productid distribution and search relevance

Algorithms/Models

This is a regression problem essentially. In other words, we need fit the training data on a learning model and give predictions on the test data. I plan to explore models as below

- Linear Regression model, or ordinary least squares OLS It is a type of linear least squares method for estimating the unknown parameters in a

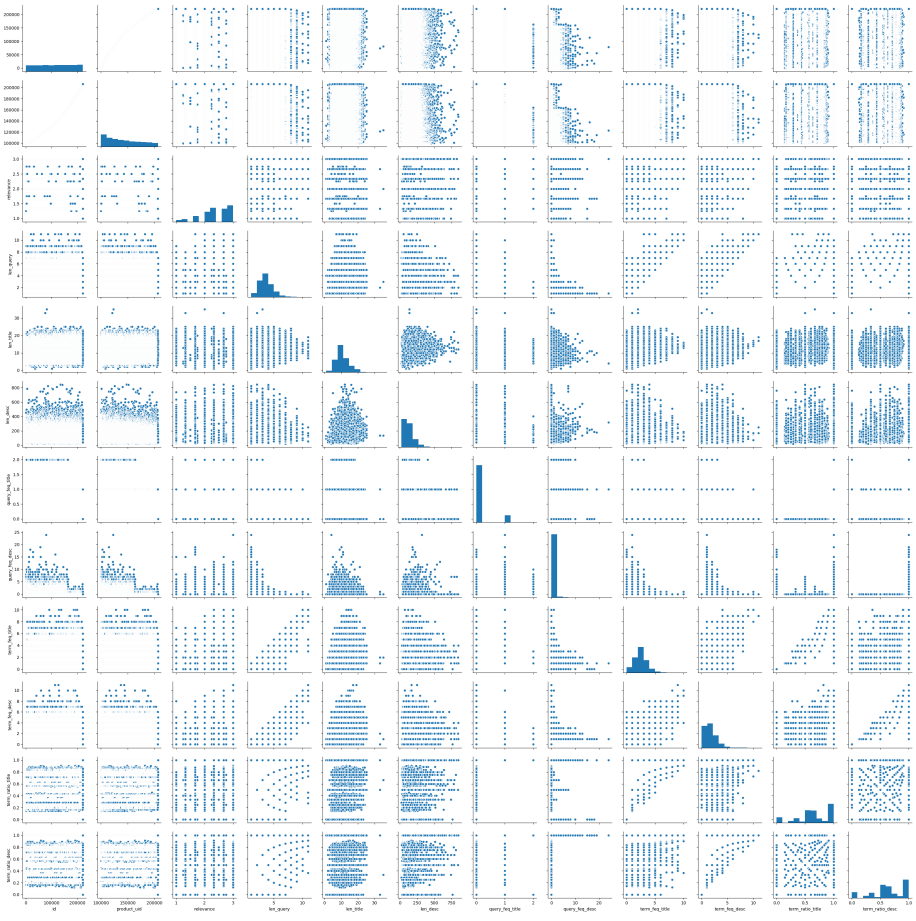


Figure 3: 02.pairplot.png

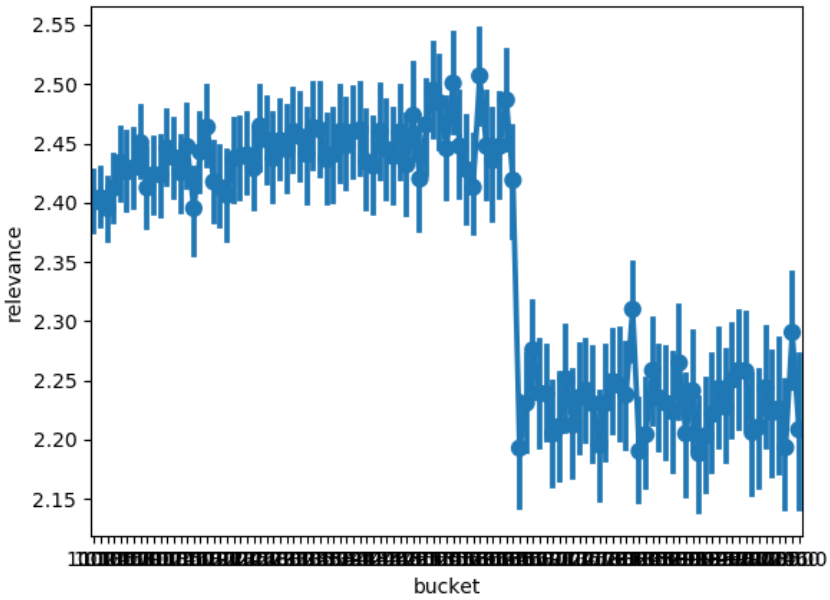


Figure 4: 02.relevance-productid-pointplot.png

linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by the principle of least squares: minimizing the sum of the squares of the differences between the observed dependent variable (values of the variable being predicted) in the given dataset and those predicted by the linear function. At its heart, linear regression is about finding the coefficients of a line that best match the given data. The biggest pros to using linear regression are its simplicity. On the other hand, if nonlinear relationships exist (and cannot be accounted for by combining features), predictions will be poor.

- Decision Tree Model

Decision trees are one of the most popular algorithms used in this particular domain, i.e. search relevance modeling.

Again, refer to thse two papers on real-world implementation, amazon search, and eBay search,

The output of a decision tree can be easily interpreted (by humans) as rules.

Besides, few other pros are also attractive and make it popular,

```
- it will address non-linearity, compared with baseline model. Great at learning complex, hi
- simplicity, easy coding
- fast computation for large dataset.
```

The idea of the Decision Tree is to divide the data into smaller datasets based on a certain feature value until the target variables all fall under one category. While the human brain decides to pick the "splitting feature" based on the experience (i.e. the cloudy sky), a computer splits the dataset based on the maximum information gain.

Though, decision trees can be prone to major overfitting. There are a set of parameters it can be tuned to overcome overfitting. But in reality, ensemble techniques is ususally applied on top of decision tree model.

- Bagging/Boosting ensemble methods

The idea of `ensemble methods` is that a set of weak learners are combined to create a strong learner that obtains better performance than a single one.

Ensemble methods combine several decision tree models to produce better predictive performance than a single decision tree model. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model.

1. Bagging, or bootstrap then aggregate process. For example, `RandomForestRegressor`. Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset. Bootstrap involves random sampling of small subset of data from the dataset.

2. Boosting. Boosting refers to a group of algorithms that utilize weighted averages to make weak learners into stronger learners. For example, here is the real implementation as part of sklearn `GradientBoostingRegressor`

I will dig into more details on the hyper-parameters of the model, which consists of more information we need to understand to best utilize the machine learning technique.

Implementation

Training logic in brief

1. Abstract all command function blocks into python module as code/regressionutil.py

- `get_data_tuple()`: load training from '/data.csv.gz', run train/test split per constant `PCT_TEST_DATA_SIZE`(i.e .3 or 30%) and return data tuple as `X_train, X_test, y_train, y_test`
- `calculate_metrics(y_true, y_pred)`: calculate RMSE metrics given inputs of y_true and y_pred
- `train_classifier(clf, X_train, y_train)`: basically invokes `clf.fit(X_train, y_train)`
- `predict_labels(clf, features)` : given a set of features data, predicts the label for each row

2. Each training process will run the same sequence of function blocks

- train_classifier(clf, X_train, y_train)
- predict_labels(clf, X_train) and calculate_metrics(y_train, train_pred)
- predict_labels(clf, X_test) and calculate_metrics(y_test, test_pred)

Train baseline model

code/step02-feature-engineering.py

Rule of thumb, starts with `simple`. Beginning with the simple case, Single Variable Linear Regression is a technique used to model the relationship between a single input independent variable (feature variable) and an output dependent variable using a linear model i.e a line. Linear regression is simple to understand which can be very valuable for business decisions.

Training sklearn's `linear_model.LinearRegression()` with all training data, it yields the following results

```
trainSet RMSE=0.488282
testSet  RMSE=0.488518
```

The result looks relatively good given we fed into large amount of training data.

But it seems that the model is too simple with bias generated.

Train out-of-box models

Given the nature of this particular problem, the next model is chosen by me is `DecisionTree Regressor`.

code/step05-train-model-outofbox.py

Classifier	score - test	score - train	size	time - predict	time - train
LinearRegression	0.4888	0.4949	5000	0.0014	0.0030
LinearRegression	0.4888	0.4921	10000	0.0017	0.0015
LinearRegression	0.4885	0.4882	74067	0.0006	0.0040
DecisionTreeRegressor	0.6941	0.0115	5000	0.0047	0.0016
DecisionTreeRegressor	0.6822	0.0320	10000	0.0048	0.0025
DecisionTreeRegressor	0.6840	0.0667	74067	0.0070	0.0190
GradientBoostingRegressor	0.4843	0.4674	5000	0.0356	0.0100
GradientBoostingRegressor	0.4827	0.4728	10000	0.0249	0.0111
GradientBoostingRegressor	0.4800	0.4780	74067	0.0315	0.0918
RandomForestRegressor	0.5243	0.22441	5000	0.0314	0.0106
RandomForestRegressor	0.5230	0.22385	10000	0.0370	0.0181
RandomForestRegressor	0.5216	0.22546	74067	0.0558	0.1391

Please note:

- RandomForestRegressor with bagging is another method besides `Grandient boosting` to try for optimal model performance.
- Gradient boosting is a machine learning technique for regression and classification problems, which produces a model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
- Gradient boosting is typically used with decision trees of a fixed size as base learners, namely gradient boosting trees. It's a generalization of the tree ensembles and can prevent overfitting effectively.

The above result shows `DecisionTreeRegressor` without tuning manifests high overfitting, whereas `RandomForestRegressor` and especially `GradientBoostingRegressor` performs much better even with default hyper-parameters

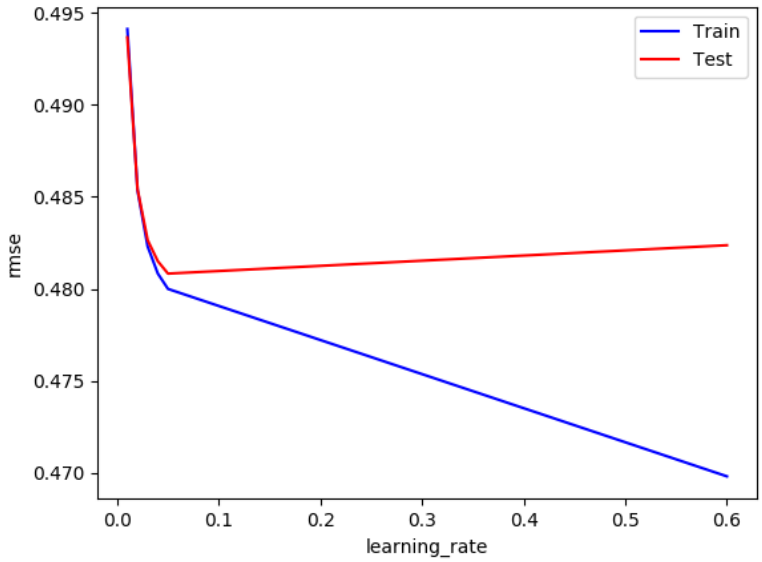
Given the result above, I decide to further improve `GradientBoostingRegressor` model.

Refinement

code/step07-train-model-refinement-gb.py

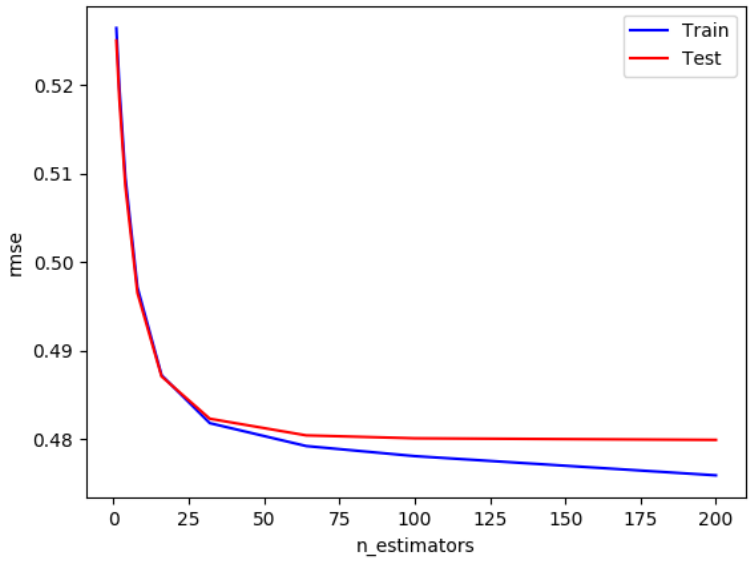
I tuned the parameters by following this article

- first, take a look at the performance per single parameter choices, refer to `def tuningOnsingleParameter` for detailed implementation
- learning_rate : learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. per



result,
it is about `~0.1`

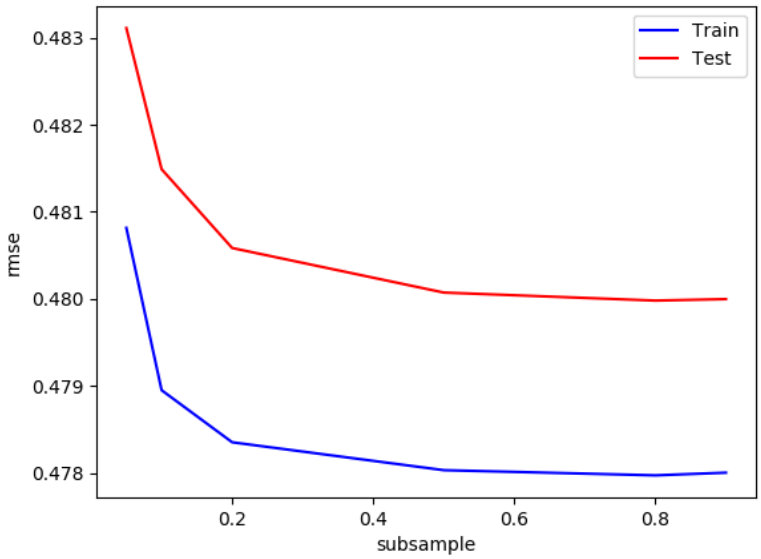
- n_estimators represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably, therefore we do a parameter search to find the sweet spot. per tuning result,



,

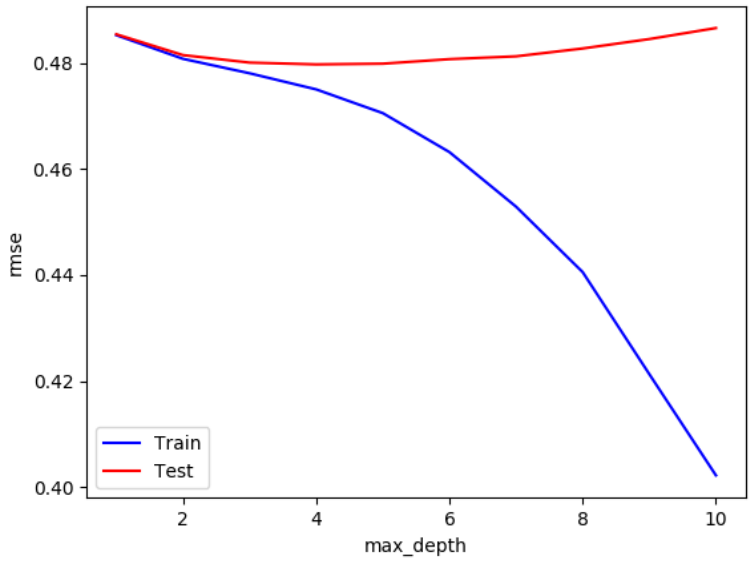
it is about between 25 to 50

- subsample : float, optional (default=1.0) The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias.



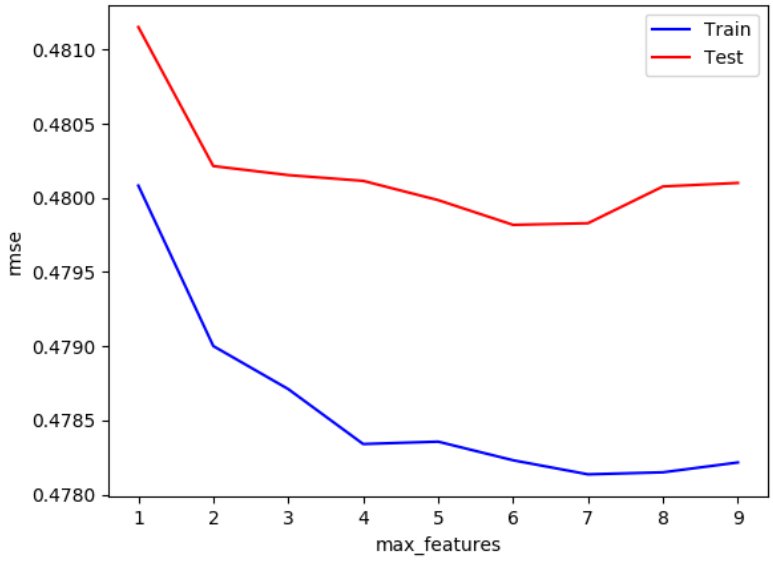
per tuning result, ,
it is about `~0.8`

- max_depth : integer, optional (default=3) maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables. This indicates how deep the built tree can be. The deeper the tree, the more splits it has and it captures more information



per tuning result, ,
it is actually around the default value, i.e. = 3

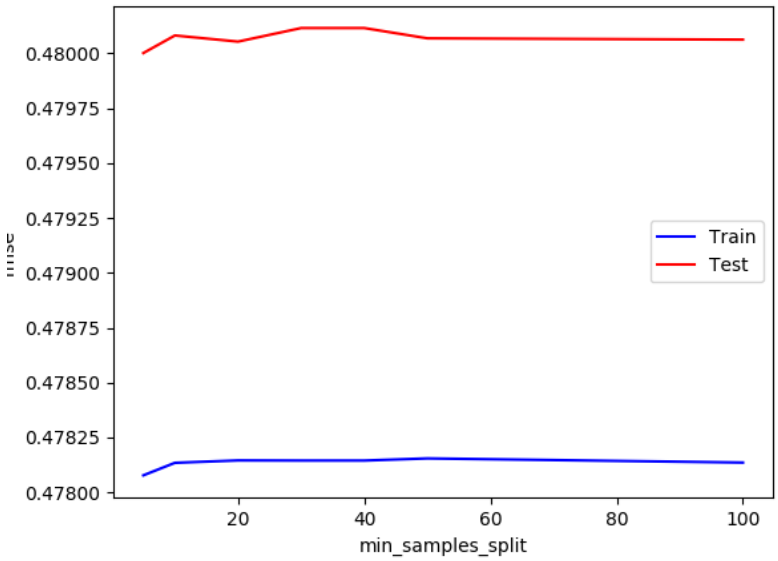
- max_features : The number of features to consider when looking for the



best split per tuning result, ,
it is actually around 6~7 features

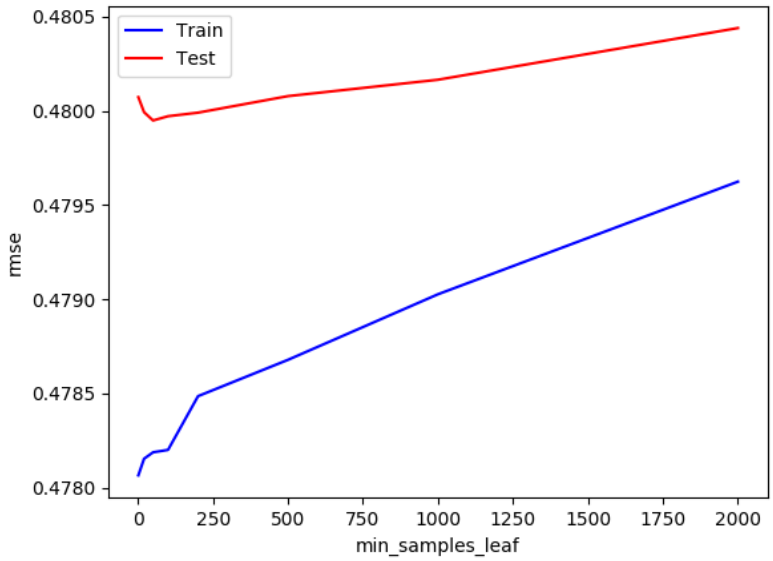
- min_samples_split: it represents the minimum number of samples required

to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, the tree becomes more constrained as it has to consider more samples at each node.



per tuning result, ,
the curve is pretty flat. will go with default value

- min_samples_leaf: the minimum number of samples required to be at a leaf



node. per tuning result, it is actually around 100 ,

- Now, did few attempts by selecting parameter options and apply GridsearchCV method

```
param_grid = {
    'n_estimators': [15, 45, 70],
    'max_features': [4, 6, 8],
    'max_depth': [6, 8],
    'learning_rate': [0.1],
    'min_samples_leaf': [50],
    'subsample': [0.8]
}

param_grid = {
    'n_estimators': [40, 45, 50],
    'max_features': [4, 6, 8, 10],
    'max_depth': [6, 8],
    'learning_rate': [0.1],
    'min_samples_leaf': [50],
    'subsample': [0.8]
}

param_grid = {
    'n_estimators': [40, 45, 50],
    'max_features': [3, 4, 5, 6],
```

```
    'max_depth': [4, 5, 6, 7, 8],
    'learning_rate': [0.1],
    'min_samples_leaf': [50],
    'subsample': [0.8]
}
```

finally yield the best result as

```
    trainSet score=0.473896
    testSet score=0.479311
    Feature Importances
    {'learning_rate': 0.1, 'min_samples_leaf': 50,
    'n_estimators': 45, 'subsample': 0.8, 'max_features': 4, 'max_depth': 6}
    Best CV Score:
    -0.4798894154650474
```

`Now both scoring on trainSet and testSet are consistently good.`

IV. Results

Model Evaluation and Validation

code/step07-train-model-refinement-gb.py

In `main` method, I run the evaluation using the after-tuning parameters and run **FUNC `def afterTuningParameter()`** .

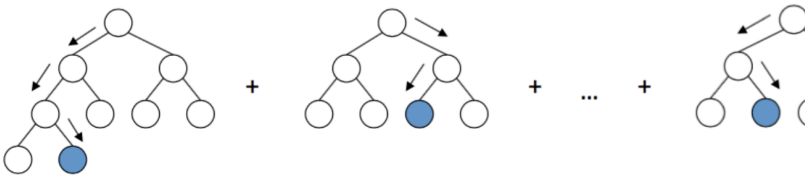
Did a `cross_val_score(clf, X, y, cv=11, scoring=rmse_scorer)` (refer to ** FUNCE ** `def evalulateModel()`) to get the score across fold of 11 as below,

- score list per each fold is
  `[-0.48950211 -0.47874718 -0.47798169 -0.47218829 -0.47783923 -0.47377536 -0.47287921 -0.47623568 -0.48764387 -0.48442463 -0.49399407]`
- DescribeResult(nobs=11, minmax=(-0.4939940667918515, -0.47218828844078503), mean=-0.4804737551026021, variance=5.36617971646771e-05, skewness=-0.5910596043262953, kurtosis=-0.9594787666425146)

score overall is stable.

```
Regression training, dur=1.85773301125
trainSet score=0.473896, dur=0.0938727855682
testSet score=0.479311, dur=0.0370998382568
                  importance
product_uid         0.214745
term_ratio_title    0.198089
len_desc            0.141595
term_ratio_desc     0.135161
```

```
len_title           0.104772
term_feq_title      0.079104
len_query           0.073408
term_feq_desc       0.026005
query_feq_title     0.013829
query_feq_desc      0.013292
```



The mode can be visualized as per image

with optimal parameter set as

```
{'learning_rate': 0.1, 'min_samples_leaf': 50,      'n_estimators':
45, 'subsample': 0.8, 'max_features': 4, 'max_depth': 6}
```

the model consists of 45 weak learner per `n_estimators`. Each learner is a decision tree. The 45 week learner comes together to form a strong learner, thus increasing the accuracy of the model.

Justification

- **product_uid** : just as the above diagram `relevance-vs-productid-pointplot` in section **Numberic Feature Exploration**. the product_uid does play important factor on relevance.
- **term_ratio_title** : it does look like that the more search-term is found in title, the more relevance.

V. Conclusion

This is very interesting as well as challenging project to me. To be able to accomplish this project, I need to prepare my knowledge beyond specific in machine learning technique. I learned a lot on how normal information retrieval does the work and what are the NLP technique for. I also studies papers from other ecommerce company besides homedepot and see how search relevance is improved/implemented.

On top of knowledge preparation, I pretty much have idea on each steps of works. For example, I can see it would make sense and good fit to choose decision tree as the base model.

This project is unique such that the raw data source are all text features. And since those are all text features, it can not be learned directly. The critical challenge is the feature engineering. Feature engineering for this project consists of two stages, feature pre-processing and feature extraction or word embedding process. There are many actions that can be taken in this stage like case conversion, tokenization, lemmatization, selection of variables, word spell correction. I picked up the there major processes

Then I trained the lineal regression as base line model for its simplicity. Then I trained a decision tree model so to be able to fit on non-lineality. Then I used Gradient Boosting on top of decision tree model and GridSearchCV to overcome overfiting and achieve best model performance.

Improvement

I've submitted attempts on the Kaggle website. The best result is 0.4830(Ranked at 1000) vs the best of 0.43. It's obvious that more work can be done to improve my final results.

- Data pre-processing/feature extraction if time-permitting, I would try to do more work in this area.

a. spell-correction
b. number normalization. for example 1 vs one, it could be normalized to just `1`
c. synonyms or synoyms per product category
d. stop words
e. generate features of n-grams
f. furtherly, employs `named entity recognition` to enrich the query In this domain, there could endless of work to be done.

- Word embedding by deep learning method
  For this domain, https://www.quora.com/How-is-GloVe-different-from-word2vec, two major techinques including Glove or word2Vec can be used to generate feature vectors

- Different regression algorithm to ensemble I could try to combine decision tree model and deep learning model together by ensembling method to get more improvements on the final results.