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 infomation but also relevant results which best match customers'need.

The relevance between search/product pair can be 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

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.

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 crowdsourced 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.

To tackle this problem, I will extract effective features from the available data, namely feature extraction.

The available raw data consists of

- search_term
- product_title
- relevance

then I will learn from the features and predict the relevance between search queries and products. Then I try different methods using the features extracted and use GridsearchCV to choose best parameters for the models

Metrics

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

II. Analysis

Data Exploration

code/step01-raw-data-analysis.py

- Training data columns

```
Index([u'id', u'product_uid', u'product_title', u'search_term', u'relevance'], dtype='object
```

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

```
COUNT(UNIQ(search_term) = 11795
```

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

Exploratory Visualization

code/step01-raw-data-analysis.py

```
    count    74067.000000
    mean         2.381634
    std          0.533984
    min          1.000000
```

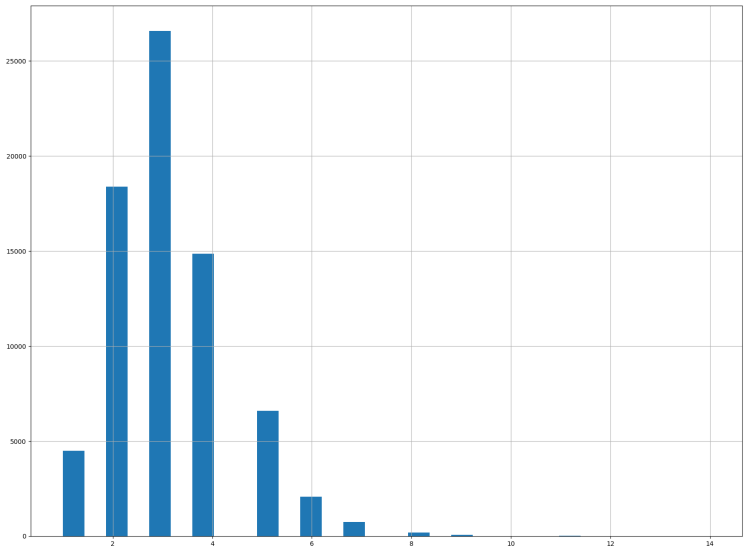
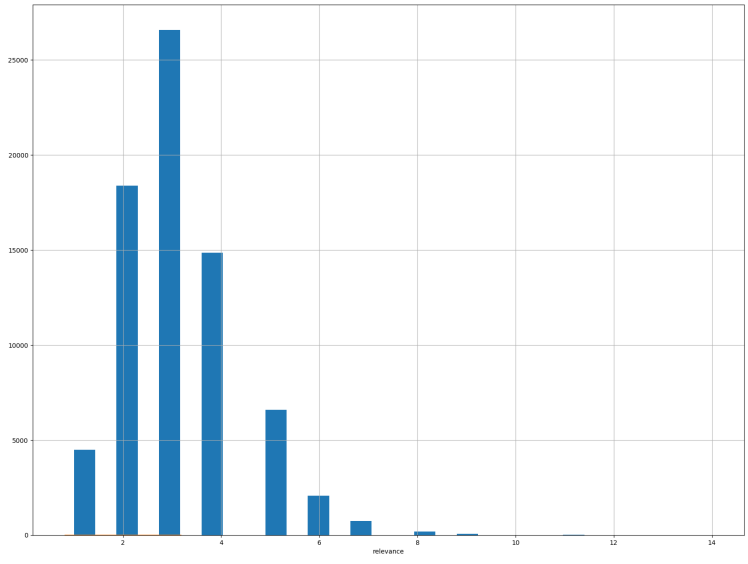
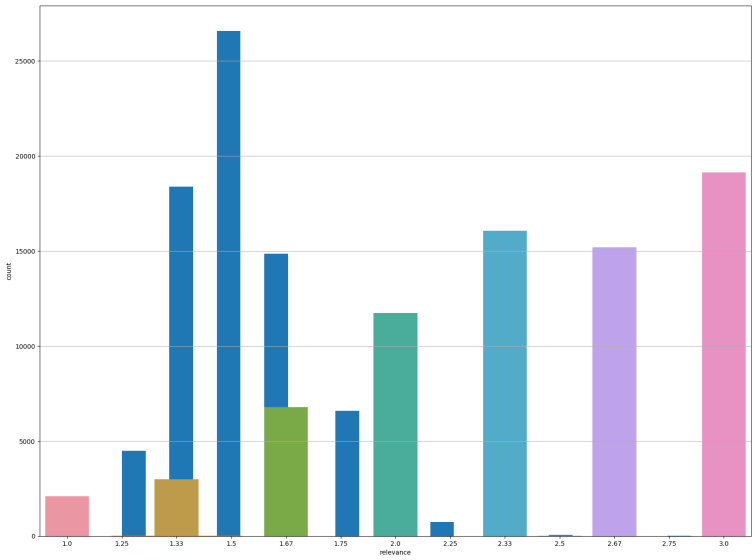


Figure 1: 01.search-term-histogram.png

```
25%          2.000000
50%          2.330000
75%          3.000000
max          3.000000
```



distplot



counterplot

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, `lower case`, `standardize_units`, `PorterStemmer`

- **FUNC extractNumberFeatures**

• 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 • 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 • 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
```

- pairplot
- relevance-vs-productid-pointplot

Implementation

Train baseline model

code/step02-feature-engineering.py

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

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

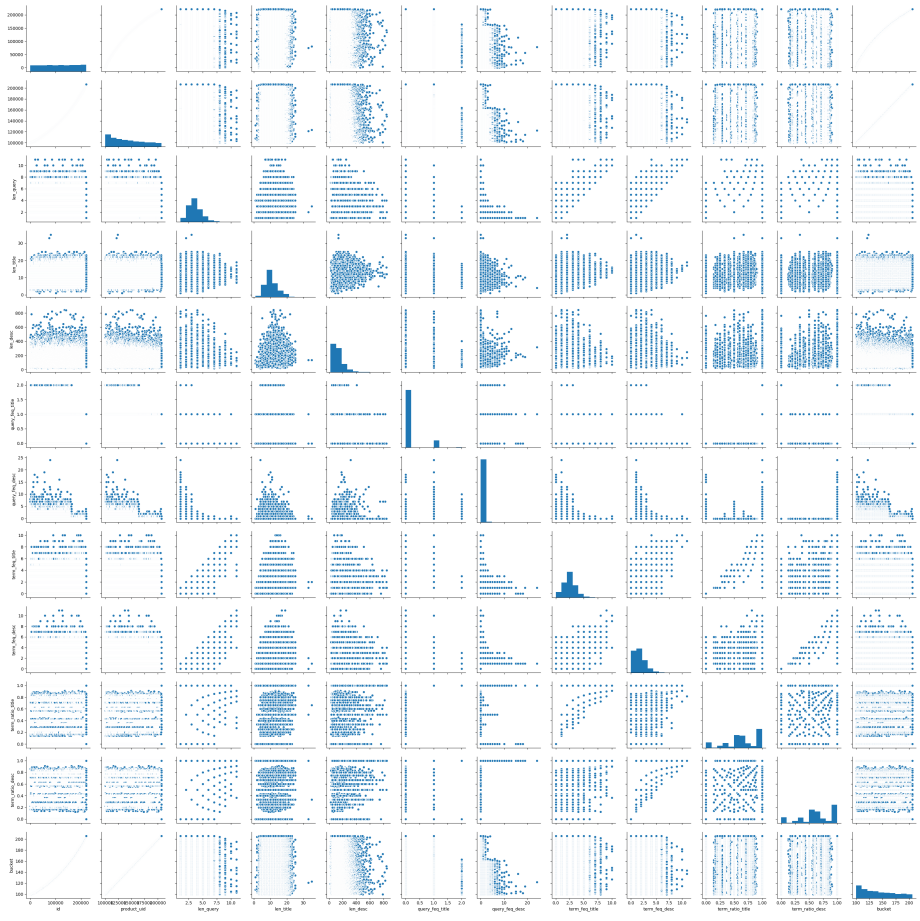


Figure 2: 02.pairplot.png

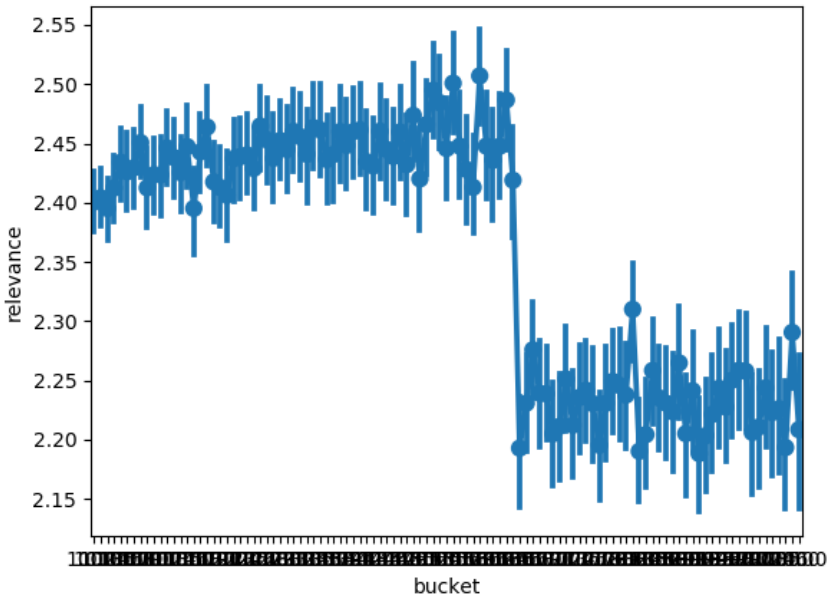


Figure 3: 02.relevance-productid-pointplot.png

Train out-of-box models

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

given the result above, I decide to further improve GradientBoostingRegressor model

Refinement

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

Apply GridsearchCV method to 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.

```
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
```

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

In this project, the data source are all text features. The critical challenge is the feature engineering. Specially, is how to select/generate numberic features out of text features. I applied generic NLP method to derive some most obvious features.

Then I trained the lineal regression as base line model. then I trained few other models from sklearn package and then I tuned it using GridSearchCV to overcome overfiting to 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.

In the future, more teniques can be used to extract more features. For example, I can try other approaches to expand the query. Also, I may explore further how to ensemble different results together to get more improvements on the final results.