CSC 575 Final Project (Kaggle) -- Winter 2019

Learning to Rank Home Depot Product Search Relevance

March 18, 2019

Super Prediction
Yizi(Winnie) Huang YiziwinnieH@hotmail.com
Joshua Smith jsmit301@mail.depaul.edu
Wenyi (Alex) Yan alexyan21@hotmail.com

Kaggle Rankings

Public Score: 0.47453 [First Place in Competition]
Private Score: 0.47336 [First Place in Competition]

Abstract

Home Depot is an American home improvement supplies retailing company that sells tools, construction products, and services[1]. To improve the online shopping experience of their customers, we intend to develop models that can better predict the relevance score of search results so that customers can get exactly what they want through the search engine.
During this process, we applied information retrieval knowledge, calculating different similarities, machine learning knowledge, domain knowledge, feature engineering, and the creation of ensemble models. Ultimately, the best model was obtained using a Gradient Boosting Regressor which achieved a RMSE of 0.47453 (Public - 30% of test set) and RMSE of 0.47336 (Private - 70% of test set), ranking first on Kaggle (March 17, 2019 submission).

Data Overview

Four datasets are used in this project:

- train_new.csv
- test_new.csv
- product_descriptions_new.csv
- attributes_new.csv

Those four datasets include features which are listed as below.
id - a unique Id field which represents a (search_term, product_uid) pair
product_uid - an id for the products
product_title - the product title
product_description - the text description of the product (may contain HTML content)
search_term - the search query
relevance - the average of the relevance ratings for a given id
name - an attribute name
value - the attribute's value

Goal

[1] https://en.wikipedia.org/wiki/The_Home_Depot

The goal for this project is to predict a relevance score for the provided combinations of search terms and products. Our target is the lowest possible RMSE for our submission on Kaggle.

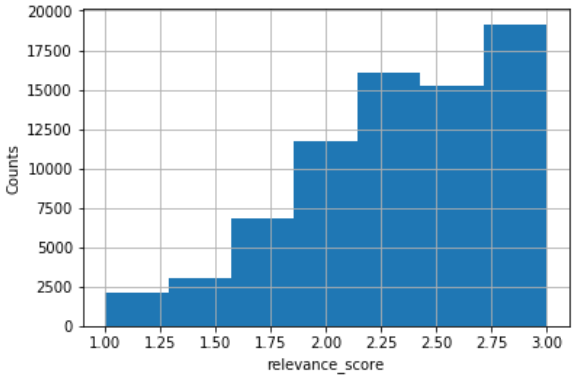
1. Exploratory Analysis

1.1 Dataset overview

Training set (train_new.csv):

- 74067 rows with 5 columns
- The graph below highlights the unbalanced distribution of target values (relevance scores)

```
Data columns (total 5 columns):
id                74067 non-null int64
product_uid       74067 non-null int64
product_title     74067 non-null object
search_term       74067 non-null object
relevance         74067 non-null float64
dtypes: float64(1), int64(2), object(2)
```



Testing set (test_new.csv):

- 112067 rows with 4 columns

```
Data columns (total 4 columns):
id                112067 non-null int64
product_uid       112067 non-null int64
product_title     112067 non-null object
search_term       112067 non-null object
dtypes: int64(2), object(2)
```

Product description set (product_descriptions_new.csv):

- 124059 rows with 2 columns

```
Data columns (total 2 columns):
product_uid            124059 non-null int64
product_description    124059 non-null object
dtypes: int64(1), object(1)
```

Product attributes set (attributes_new.csv):

2044803 rows with 3 columns and very few missing values

```
RangeIndex: 2044803 entries, 0 to 2044802
Data columns (total 3 columns):
product_uid    float64
name           object
value          object
dtypes: float64(1), object(2)
```

```
MFG Brand Name          86250
Bullet02                86248
Bullet03                86226
Bullet04                86174
Bullet01                85940
Product Width (in.)     61137
Bullet05                60529
Product Height (in.)    54698
Product Depth (in.)     53652
Product Weight (lb.)    45175
```

1.2 Word map

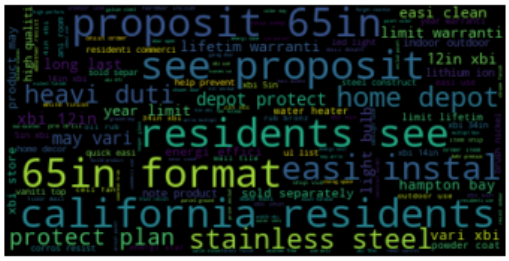
Please Note: These word maps were generated after text preprocessing which is outlined in a subsequent section.

To better understand the text in two datasets, we made word maps for their terms in query ('search_term') and product information (including 'product_description', 'brand' and 'product_title') respectively:

Training set

Terms in Query:

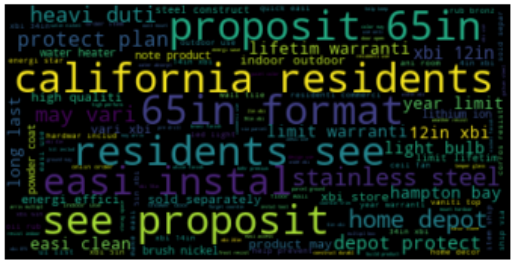
Terms in Product information:



Testing set

Terms in Query:

Terms in Product information:



We can see keywords of query and product information in both datasets are similar.

For query, it can be readily seen that users primarily searched for items based on their functions (e.g ceiling fan, water heater) and textures/colors(e.g white, wood). Specially, we can see that there is a big part of customers search for Hampton Bay which is a wholly owned subsidiary of Home Depot. This highlights that this self-owned brand is performing well (in terms of product sales) and is a great asset of the Home Depot organization. In relation to information retrieval, this shows the importance of branding in customer queries.
For product information, the website mainly describes the size (e.g: 65in format, 12in), special notes (e.g: california residents proposition, protection plan, lifetime warranty) and product features (e.g easy clean).

2. Data Processing

Text processing includes spell checking, punctuation removal, stop words removal, tokenization, stemming, lemmatization and ect. Some examples of text preprocessing are shown as below. However, some of them may not included in a given model as they produced a worse result or negligible difference. As an example, stemming and lemmatization are both word normalization and as such, may not produce changes in the final results given their similarity.

2.1 Spelling Correction

The spelling correction was achieved using Google Spell Check[2] and forms the first step of model creation. A static corpus of spelling corrections was utilized to improve performance in repeated calculations, rather than manually querying Google each time.

An example of the corpus contents:
spell_check_dict={'steele stake': 'steel stake','gas mowe': 'gas mower', 'metal plate cover gcfi': 'metal plate cover gfci','lawn sprkinler': 'lawn sprinkler',.....}

2.2 String Cleaning through Regular Expression

Regular expression processing is done using the RE package[3]. The string cleaning can be separated into four distinct categories:

2.2.1. General processing:

Split words; lowercase; and remove html character 'nbsp'.

2.2.2 Characters processing:

Replace some special characters into white space, remove slashes; replace various spellings of 'by' into 'xbi'.
for examples:

```
s = s.replace("$"," ")
s = re.sub(r"([a-z])( *)\.( *)([a-z])", r"\1 \4", s)
s = s.replace(" x "," xbi ")
```

2.2.3. Standardize Representation

This primarily targets unit representations as equivalent identifiers should be collected under a common term to ensure they are treated as one type. As an example, gallon(s) should be standardized to gal. A few more examples from our code are shown below:

```
s = re.sub(r"(?<=[0-9])[\ ]*pound[s]*(?=\ |$|\.)", "-lb", s)
s = re.sub(r"(?<=[0-9])[\ ]*lb[s]*(?=\ |$|\.)", "-lb ", s)
s = re.sub(r"(?<=[0-9])[\ ]*gallon[s]*(?=\ |$|\.)", "-gal ", s)
```

[2] https://www.kaggle.com/steubk/fixing-typos/comments
[3] https://www.tutorialspoint.com/python/python_reg_expressions.htm

2.3 Remove stop words

"Stop words" are the most common words in a language like "the", "a", "on", "is", "all". These words do not carry important meaning and are usually removed from texts. This is achieved through the TFIDF vectorizer as indicated by the stop_words parameter.

2.4 Tokenization

Tokenization is the process of splitting the given text into smaller pieces called tokens. Words, numbers, punctuation marks, and others can be considered as tokens. This is inherently done via the string splitting which occurs throughout the data processing steps.

2.4 Stemming

Stemming is used to prepare text, words, and documents for further processing. In our project, one "English" stemmer *PorterStemmer()* and one "non-English" stemmer *SnowballStemmers()* which supports a myriad of languages used. The final model utilizes the Snowball stemmer given its improved performance (resulting RMSE).

2.6 Lemmatization

The lemmatization is also applied in the text preprocessing. Because the aim of lemmatization, like stemming, is to reduce inflectional forms to a common base form. Again, as outlined above, considerations were made to ensure the similarity in normalization was not an issue.

	Before lemmatization	After lemmatization
1	Not only do angles make joints stronger, they ...	not angle make joint stronger ...
2	BEHR Premium Textured DECKOVER is an innovativ...	behr premium textured deckover innovativ ...
3	Update your bathroom with the Delta Vero Single...	update bathroom delta vero single …

3. Data Features

The key to project is mostly preprocessing and feature engineering as the primary data is text.
We made three different data frames in feature engineering for model generation. These vary in their focus on different aspects of the data. The data frames are outlined as follows:

3.1 Option 1: Create new features based on different measurements of similarity

The processed text features generate 7 numeric variables. Additionally, one numeric variable is obtained from original training set.

3.1.1 Functions

def str_stemmer(s) only takes one string, applies porter stemmer and lower case into text, and returns stemmed text. To generate numeric variables, all text features I used stemmed text as an input.
def str_common_word(str1, str2) takes two string arguments st1 and str2 and returns the number of shared words.
def similarity (str1, str2) takes two string arguments str1 and str2 and returns the similarity between those two strings using Jaccard distance

def cosine_sim(str1, str2) takes two string arguments str1 and str2 and returns the cosine similarity between those two strings considering tf_idf between those two strings.

3.1.2 Features

Information-related features;

'len_of_query' : Then length of query. The longer of “search_term” (query), the higher possibility of match the description of searched item.

'word_in_title' : The number of shared words in “search_term” and “prodcuct_title”.

'word_in_description' : The number of shared word in “search_term” and “product_description”. The number also be generated by function str_common_word.

Similarity-related features:

'jd_search_title' : The Jaccard similarity between “search_term” and “product_description”

'jd_search_description': The Jaccard similarity between “search_term” and “prodcuct_title”.

'cosine_search_title': The cosine similarity between “search_term” and “product_description”

'cosine_search_description': The cosine similarity between “search_term” and “product_description”

'relevance': gained from training set

3.2 Option 2: Create new features based on information carried by queries and product information

3.2.1 Functions

def str_stemmer(s): Utilized to stem terms found in searches and product information.

def str_common_word(str1, str2): Used to calculate the number of common words between two strings

3.2.2 Features

Excepting including some common information-related features (i.e 'len_of_query', 'word_in_title', 'word_in_description'), we create other 12 new features instead of similarity-related features(11 numerical features and 1 text features which is 'brand').:

Extract brand names:

From 'attributes_new.csv' as extra info of products. and create:

'brand': name of brand

'len_of_brand': The length of brand name

'brand_in_search': The number of shared words in 'brand' and 'search_term'(query)

Information combination features:

'product_info': combine 'search_term', 'product_title' and 'product_description'(for more features generation but would be removed later)

'attr': combine 'search_term' and 'brand'(for more features generation but would be removed later)

'prod_desc_merge': merge 'product_description' and 'brand' into a new feature 'prod_desc_merge' to deal with the Null value in these two features

Ratio information features:

'ratio_brand': The ratio of the number of shared words in 'brands names' and 'search_term'('brand_in_search') taking in length of brand name('len_of_brand').

'ratio_title': The ratio of the number of shared words in “search_term” and “prodcuct_title”('word_in_title') taking in length of query('len_of_query')

'ratio_description': The ratio of the number of shared word in “search_term” and “product_description” ('word_in_description') taking in length of query('len_of_query')

Matching exploration features:
Discover more relationships among the first and last word of query and product title and description

'first_word_title_match': if the first word of query matches 'product_title'(binary)
'first_word_description_match': if the first word of query matches'product_description'(binary)
'last_word_title_match': if the last word of query matches 'product_title'(binary)
'last_word_description_match': if the last word of query matches 'product_description'(binary)

3.3 Option 3: Create new features based on information provided by queries, product information (titles, descriptions, and product attributes)

This option relies on domain knowledge for the generation of further features to append to Option 2's feature set. Most notably, the attributes as a whole are included in this option with the respective ratio/frequency features (same type as other values like the product description). Domain knowledge was applied to select the two additional key features of common units and common colors. These were specifically chosen given that this dataset focuses on products that are in the home improvement sector (a common description of Home Depot's selection). Units become especially important when searching for specific items that rely on varying units to differentiate between types. An example of this can be found in the different types of "screws" that are offered. Screws with metric units (i.e. mm) are vastly different than those with imperial units (i.e. inch).

3.3.1 Functions

def str_stemmer(s): Utilized to stem terms found in searches and product information.
def str_common_word(str1, str2): Used to calculate the number of common words between two strings
def str_count_common_units(dataframe): Used to calculate the number of common unit terms between the query and the product_info
def str_count_same_colors(dataframe): Used to calculate the number of common color terms between the query and the product_info

3.3.2 Features

Excepting including some common information-related features(i.e 'len_of_query', 'word_in_title', 'word_in_description'), we create other 12 new features instead of similarity-related features(11 numerical features and 1 text features which is 'brand').:
Extract brand names:
From 'attributes_new.csv' as extra info of products. and create:

'brand': name of brand

'len_of_brand': The length of brand name

'brand_in_search': The number of shared words in 'brand' and 'search_term'(query)

Information combination features:

'product_info': combine 'search_term', 'product_title', 'product_description', 'attribute' (this is an intermediary feature which is used for later calculations)

'attr': combine 'search_term' and 'brand'(for more features generation but would be removed later)

'prod_desc_merge': merge 'product_description', 'brand', and 'attribute' into a new feature (allows for null values to be properly accounted for)

Further Textual Information Features:

'len_of_query': The length of the query string (search_terms)

'len_of_attribute': The length of the attribute string (the collection of names and values from the attribute CSV)

'word_in_title': The number of words which occur in both the query and in the product's title

'word_in_description': The number of words which occur in both the query and in the product's description (query + title + description)

'word_in_attributes': The number of words which occur in both the query and in the product's attributes (name + value)

Ratio information features:

'ratio_brand': The ratio of the number of shared words in 'brands names' and 'search_term'('brand_in_search') taking in length of brand name('len_of_brand').

'ratio_title': The ratio of the number of shared words in “search_term” and “prodcuct_title”('word_in_title') taking in length of query('len_of_query')

'ratio_description': The ratio of the number of shared words in “search_term” and “product_description” ('word_in_description') taking in length of query('len_of_query')

'ratio_attributes': The ratio of the number of common words in the attribute ('word_in_attributes') over the length of the query ('len_of_query')

Matching exploration features:

Discover more relationships among the first and last word of query and product title and description

'first_word_title_match': if the first word of query matches 'product_title'(binary)

'first_word_description_match': if the first word of query matches'product_description'(binary)

'last_word_title_match': if the last word of query matches 'product_title'(binary)

'last_word_description_match': if the last word of query matches 'product_description'(binary)

Features informed through domain knowledge:

'num_common_units_measurement': The number of common unit identifiers (i.e. “in.” and “cu.ft.”) between the product information (title, description, and attributes) and the query

'num_common_colors': The number of common color strings (i.e. “blue”) between the product information (title, description, and attributes) and the query

3.4 Features overview

In the Random Forest model, we use the data frame made by the Option 1 feature method. The variable named product_info was created by concatenating 'product_title', 'search_term', and 'product_description'. This is used to obtain numeric values by applying the functions shown above in Option 1. After that, all text columns are dropped, and numeric variables are kept to generate the model.

Random Forests model (Using option 1 method)

Initial	Training	id	product_uid	product_title	search_term	relevance	
	Test					-	
	product_des	-	product_uid	product_des	-	-	
Final	df_all	id	product_uid	~~product_title~~	~~search_term~~	relevance	~~product_des~~
			~~product_info~~	word_in_title	word_in_des	jd_search_title	jd_search_des
			cosine_search_title	cosine_search_des			

In the Gradient Boosting model and Xgboost model, we use the data frame made by the Option 2 feature method. The variable named 'product_info' is created by concatenating 'product_title', 'search_term', and 'product_description' and the variable named 'attr' is created by concatenating 'search_term' and 'brand'. Again, this is used to assist in the creation of numeric values by applying the functions outlined above (under Option 2). After that, all text columns are dropped, and numeric variables are kept to create the model.

Gradient Boosting model and Xgboost model (Using option 2 Method)

Initial	Training	id	product_uid	product_title	search_term	relevance	
	Test					-	
	product_des	-	product_uid	product_des	-	-	
	attributes	-	product_uid	name	value		
Final	df_all	id	product_uid	product_title	search_term	relevance	len_of_query
			len_of_brand	word_in_title	word_in_des	brand_in_search	ratio_brand
			ratio_title	ratio_description	last_word_title_match	last_word_description_match	first_word_title_match
			first_word_description_match	prod_desc_merge	~~product_des~~	~~name~~	~~value~~

For the final Gradient Boosting model, the dataframe (as shown below) from Option 3 was utilized. The variable named 'product_info' was created by concatenating 'product_title', 'search_term', 'product_description', 'product_attributes'. The variable named 'attr' was created by concatenating

'search_term' and 'brand' to assist with calculating the numeric values as previously outlined. By concatenating, a single map could be applied with an internal split to obtain the proper feature. After that, all text features are dropped, leaving only the numeric variables for model generation.

Gradient Boosting model and Xgboost model (Using Option 3 Method)

Initial	Training	id	product_uid	product_title	search_term	relevance	
	Test					-	
	product_des	-	product_uid	product_des	-	-	
	attributes	-	product_uid	name	value		
Final	df_all	id	product_uid	product_title	search_term	relevance	len_of_query
			len_of_brand	word_in_title	word_in_des	brand_in_search	ratio_brand
			ratio_title	ratio_description	last_word_title_match	last_word_description_match	first_word_title_match
			first_word_description_match	prod_desc_merge	word_in_atributes	ratio_attributes	len_of_attribute
			num_common_units_measurement	num_common_colors	product_attributes		

4. Calculating TF-IDF Performing Feature Reduction (For the data frames built based on Option 2 and Option 3)

We use a tf-idf vectorizer to vectorize the words and then perform feature reductions using Truncated SVD (a.k.a. LSA) on each text feature ['product_title', 'search_term', 'product_description', 'brand', (and 'product_attributes' for Option 3)].
In particular, to deal with the respective Null in 'product_description' and 'brand', we merge them into a new feature 'prod_desc_merge'.

4.1 Query expansion: Singular Value Decomposition (SVD) and Latent Semantic Analysis (LSA)
Truncated SVD[4] (in Sklearn package) performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with *scipy.sparse* matrices efficiently.
In particular, Truncated SVD works on term count/tf-idf matrices as returned by the vectorizers in *sklearn.feature_extraction.text*. In that context, it is known as Latent Semantic Analysis (LSA)[5] which is a technique in Natural Language Processing that can analyze relationships between a set of documents and the terms they contain. This produces a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis). A matrix containing word counts per paragraph (rows represent unique words and columns represent each paragraph) is constructed from a large piece of text and a mathematical technique called Singular Value

[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
[5] https://en.wikipedia.org/wiki/Latent_semantic_analysis

Decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns.
We utilize LSA to do query expansion, because it can group words together based on their meaning in “search_term”, “product_item”, and “product_ description” (and “product_attributes” for Option 3) resulting the changing of similarity compared with the original model. Query expansion can help us overcome that the vocabulary that searchers use is often not the same as the one by which the information has been indexed. The query is expanded using terms which have a similar meaning or bear some relation to those in the query, increasing the chances of matching words in relevant documents.

5. Model architecture

5.1 Random Forests
The linear regression model is built as benchmark first. In order to improve the performance, the Random Forest method is applied because it fits decision trees on various sub-samples of the training dataset and uses averaging to improve the predictive accuracy and control over-fitting. The *RandomForestRegressor()* is used at first. After that, the bagging is applied after *RandomForestRegressor()* which includes multiple random trees, each considering a different subset of variables and dataset.

5.2 Gradient Boosting Regressor
We also tried the Gradient Boosting regressor. Gradient Boosting builds an additive model in a forward stage-wise fashion. It allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

5.3 Xgboost
XGBoost is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms. It's main goal is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate for large scale tree boosting.

6. Parameter settings

6.1 Random Forests Regressor
The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid. GridSearchCV implements a “fit” and a “score” method as the basis of determining the best resulting model. The GridSearchCV is used twice in our approach. The first instance is used for selection of optimal parameters for RandomForestRegressor(). The parameter grid is set with "n_estimators":[10,15,20,25],"max_depth":[2,4,6,8,10]. The best score and best parameters are generated by setting n_estimators= 20 and max_depth =8. The second time the GridSearchCV is used for select best parameters for bagging with RandomForestRegressor() including best parameters generated previously. The parameter grid is set with "n_estimators":[10,15,20,25,30,35,40,45,50],"
"max_samples":[0.1,0.2,0.3]. The best score and best parameters are generated by setting n_estimators=25, and max_samples =0.3. The best RMSE we got for this model is 0.48100.

6.2 Gradient Boosting Regressor
The parameters tuning for gradient boosting is also by grid search using cross-validation score.

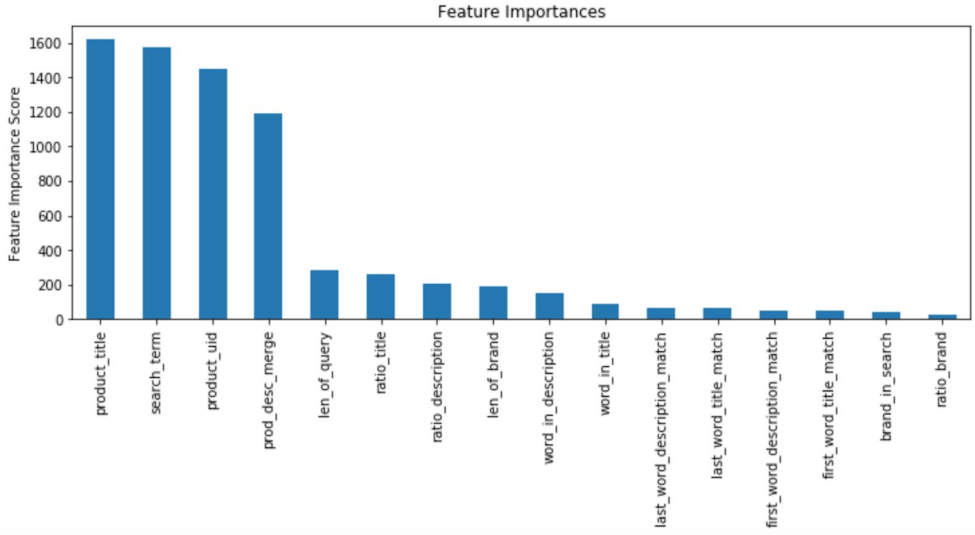
Two main parameters are taken into consideration: 'n_estimators' , which is to adjust the number of regression estimators, and 'max_depth', which is to adjust the maximum depth of the individual regression estimators. Given that gradient boosting is fairly robust in terms of overfitting correction, a large number of estimators typically results in better performance.
The parameter grid is set with 'n_estimators' : np.array([int(e) for e in np.linspace(30, 40, 11)]), 'max_depth': np.array([int(e) for e in np.linspace(2, 10, 5)]), and finally the best score and best parameters are generated by setting *'n_estimators'=6* and *'max_depth'=36*. The best RMSE we got for this model is 0.47336 (obtained via Option 3 features).

6.3 Xgboost

The parameters tuning for gradient boosting is also by grid search using cross-validation score.
Two main parameters are taken into consideration: 'n_estimators' , which is to adjust the number of regression estimators, and 'max_depth', which is maximum depth of a tree (increasing this value will make the model more complex and more likely to overfit).
The parameter grid is set with *'n_estimators' : np.array([int(e) for e in np.linspace(30, 40, 11)]*, *'max_depth': np.array([int(e) for e in np.linspace(2, 10, 9)]*, and finally the best score and best parameters are generated by setting *'n_estimators'=8* and *'max_depth'=40*. The best RMSE we got for this model is 0.47879.

6.3.1 Feature Importance

A benefit of using gradient boosting is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance scores for each attribute.
Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function.
The feature importances are then averaged across all of the the decision trees within the model.
We used Xgboost's in-built functions `get_booster()` and `get_fscore()` to get the feature importance and visualized it as below:



From the visualization above, we can see that the most important features are 'product_title', 'search_term', 'product_uid' and 'prod_desc_merge', since we use LSA on these columns(expect

'product_uid'), we assume that query expansion have a big effect on the models built based on the option-2 dataset.

7. Final model

The final model was created with the Gradient Boosting Regressor and achieved a RMSE of 0.47336, ranking first on the private leaderboard on March 17, 2019.

The best model is: gbr1 = GradientBoostRegressor(n_estimators=36, max_depth=6).

For the reason why gradient boost beat random forest, we think is *Gradient Boosting: GBT* builds trees one at a time, where each new tree helps to correct errors made by previously trained tree, while RFs train each tree independently, using a random sample of the data, and can't correct previous errors.

For the reason why gradient boost beat Xgboost, we believe the parameters of Xgboost were not tuned very well during testing. From a mechanism perspective, both xgboost and gbm follows the principle of gradient boosting. There is however, the difference in modeling details. Specifically, xgboost uses a more regularized model formalization to control over-fitting, which should give it better performance, and thus this is one area where our project can be improved.

8. Other related information

The report provided by a Kaggle competition team named *Turing Test* allows us to think about the uniform spelling in a straightforward manner. This report encourages us to verify that numerous words are spelled differently in the dataset: mail box and mailbox, fibre and fiber, chain saw and chainsaw, aluminium and alu-minum. The spelling of such words must be uniform throughout all text columns and thus this should be included in the text preprocessing.

For the query expansion portion, we had tried to apply wordNet in the model, however, the running time was very long and froze local machines for extended periods of time. Thus, we only choose LSA to handle this process. We read this paper[6] that compares LSA, wordNet and PMI-IR on semantic similarity measurements.

9. Journey of development to get to the final model and parameters

To avoid multicollinearity problem and improve the robustness of the models, we decided to use ensemble model instead of regular linear regression model.

The first model we considered for building was random forest. This model we created without using the attribute dataset because it was not properly processed at that time. At the beginning of the project, we started with rough text processing like the removal of stop words and stemming. Those text processing methodologies were applied into text columns by using map(). After that, there are three numeric variables generate from preprocessed columns. Those three numeric variables are 'len_of_query', 'word_in_title', 'word_in_description'. In our first model, those three numeric variables and 'relevance', which comes from original dataset, were included in the random forest. The RMSE was very large at the

[6] http://ix.cs.uoregon.edu/~hornof/downloads/CHI05_Semantics.pdf

time. Four more numeric variables were generated with extra text preprocessing work (for example, remove punctuation, lemmatization, and etc). The more variables which were included, the better the performance exhibited. Last but not least, the bagging and grid search was utilized to build the model to obtain further improved performance. The details of the last random forest model and parameters can be found in the previous Parameter Settings section.

After random forest, we started to consider some other ensemble models. More importantly, we made a greater effort on feature engineering in the following models, including extracting additional useful information such as 'brand', combining product information, calculating ratio information, and exploring matching features (additional details were listed in the feature engineering section)

After adding many new features, the second model we considered to build is Gradient Boosting Regressor, we picked the parameters and their ranges based on previous random forest model since they are all need set many estimators and their depths. The result of the first trial was a RMSE of 0.47879 and is better than the best score we got from random forest.

The third model we considered to build is Xgboost Regressor, we picked the parameters and their ranges based on previous Gradient Boosting model since they are really similar and Xgboost and be regarded as ‘Regularized Gradient Boost’. However, the RMSE we got is worse than that of Gradient Boost.

10. Final results

For the final results, we obtained first place in both the public leaderboard and the private leaderboard with a RSME=0.47453 and RSME=0.47336 respectively.

We really excited about this result as it highlights the efforts taken over the project development. At the same time, we know that we can do better in some aspects such as optimization of parameters of Xgboost and Gradient Boost, building models based on features built in Option 1/Option 2/Option 3 and so on. We believe we can do better on this project in future.

11. Comments and reflection on project overall

This Kaggle competition is very interesting. Almost all variables are text which is very different from the previous project we completed. A large portion of our time was spent on text preprocessing and feature engineering. We found even that some text processing could reduce the performance and thus represents how difficult of a problem this can be to solve. Even though we have gotten some weak features based on feature importance score, we would rather keep them in our models given that they could still contribute to your final ensemble. What's more, it was good to keep working out new ideas, even if we were falling behind on the public leaderboard at first. To sum up, the project is helpful to go through what we have learned from the information retrieval class and provided a great exercise to exhibit the knowledge gained through the course.

12. Comments on member contributions

It should be noted that this report was compiled from information obtained by all group members. Option 1 and its respective model were outlined/built by Winnie. Option 2 and its respective model were outlined/built by Alex. Option 3 and its respective model were outlined/built by Josh. Given that some of

the work outlined in this report was duplicated, we have collected the findings under common sections to ensure coherency and brevity.

References

[1] https://github.com/ChenglongChen/Kaggle_CrowdFlower.
[2] https://github.com/geffy/kaggle-crowdflower.
[3] http://blog.kaggle.com/2015/07/22/crowdflower-winners-interview-3rd-place-team-quartet
[4] https://www.kaggle.com/steubk/home-depot-product-search-relevance/ fixing-typos.
[5] http://norvig.com/spell-correct.html.
[6] https://www.kaggle.com/c/dato-native/forums/t/16626/
beat-the-benchmark-0-90388-with-simple-model.
[7] https://www.kaggle.com/c/home-depot-product-search-relevance/forums/
t/19463/what-s-the-lb-shakeup-potential/116689#post116689.
[8] https://github.com/manasRK/glove-gensim.
[9]http://billy-inn.github.io/papers/cmput690.pdf
[10]https://www.machinelearningplus.com/nlp/gensim-tutorial/
[11]https://link.springer.com/chapter/10.1007/978-3-319-26850-7_33