2016
Kaggle Competition - Predict the relevance of search results on homedepot.com

D'Souza, Ajay
ajaydsouza@gatech.edu

May 23, 2017

Abstract

The goal of this Kaggle Competition is to develop an algorithm that matches the results of the manual rating score for determining the relevance of a search string to a given product

Contents

List of Figures

List of Tables

1 Introduction

- `www.kaggle.com` has a an open competition for **Home Depot Product Search Relevance** at `https://www.kaggle.com/c/home-depot-product-search-relevance`. At Home Depot humans rate the relevancy of a search string to a product by assigning an rating score to a tuple of $\langle product, searchstring\rangle$. The on-line search algorithm at Home Depot is evaluated and tuned to use this manual rating score. Manual rating however is a slow process and will not scale. Through this competition Home Depot is seeking to find an algorithm than can learn from the manual rating process and mimic it.

- Home depot has provided a training data set that has a tuple of $\langle product, searchstring\rangle$ along with a rating score for each tuple. The rating score is in the range of $1 \ldots 3$, with 1 indicating that the product is irrelevant to the search string and 3 indicating that the search string is fully relevant to the product. We need to train a model to be able to predict this rating score for a tuple of $\langle product, searchstring\rangle$.

- A test data set comprising of tuples of $\langle product, searchstring\rangle$ is provided. The model needs to make predictions for this test set. The test results set are to be submitted and will be evaluated using RMSE (Root Mean Square Error).

- It should be noted that the goal is not to search for the best product for a search string, but instead to develop a model which predicts a rating score that matches the manual rating score for a tuple of $\langle product, searchstring\rangle$.

2 Problem Definition

2.1 Source of Data

- The data for the competition is provided on Kaggle at `https://www.kaggle.com/c/home-depot-product-search-relevance/data`. The data comprises of training data, test data, a catalog of product descriptions, product attributes, instructions for manual rating. The table (1) lists the data files provided along with a description as mentioned on `https://www.kaggle.com`

- Table (2) provides a detailed description of the contents of the data provided in the training dataset, product description and product attributes files.

train.csv	The training set, contains products, searches, and relevance scores
test.csv	The test set, contains products and searches. You must predict the relevance for these pairs
product_descriptions.csv	Contains a text description of each product. You may join this table to the training or test set via the product_uid
attributes.csv	Provides extended information about a subset of the products (typically representing detailed technical specifications). Not every product will have attributes.
sample_submission.csv	A file showing the correct submission format
relevance_instructions.docx	The instructions provided to human raters

Table 1: Data for - **Home Depot Product Search Relevance** Competition

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

Table 2: Data description - **Home Depot Product Search Relevance** Competition

3 Data Cleanup

The following data clean up was performed

1. There are 13 distinct values for the response variable $relevance$ in the training set. Figure (1) is a bar plot of the count for each of the 13 values of the response variable in the training data set. As can be seen over 99.9% of the values are taken by seven values of the response variable. The remaining six account for less than $.01\%$ of the rows in the training data. This data was considered as out-lier data and rows for this data where cleaned up from the training data set. The retained seven values are also well rounded numbers. This indicates that most of the rating scores might have been actually be the scores of a single rater rather than a average of multiple raters.

2. The text and training data provided is in English. All figures, numeric data, escape sequences, Unicode characters and any other special characters where converted to text or replaced by empty space from the test and training data.

3. Text data is further cleaned to remove stop words, words less than 2 characters. All text data is then converted to the same lower case.

4. All text in the test and training dataset for the fields search string, product title,

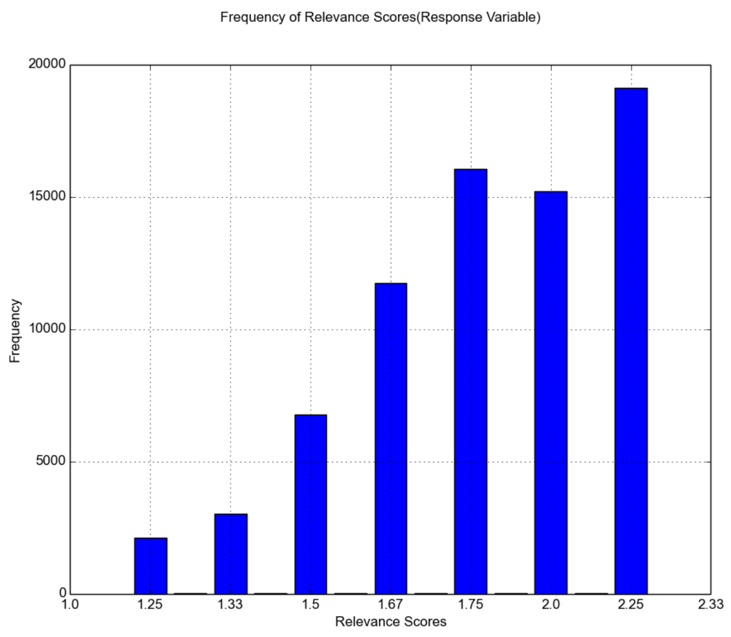


Figure 1: Frequency of Relevance Score Values (Response)

product description and product attributes is stemmed. Porter stemmer implementation in python is used for stemming.

4 Exploratory Data Analysis and Generating the Feature Vector

Once we have the cleaned and stemmed data we need to represent each row into a feature vector of numeric or categorical data. We can use this feature vector representation for fitting a learning model. An exploration of the text strings in attributes shows that the string *mfg brand name* is the most common n-gram for a bullet point in attributes. Its attribute value is taken to be the brand and is extracted and added as a column for each record.
For the predictors we have 4 bodies of cleaned stemmed text data for each record.

1. The Product Title

2. The Product Description

3. The Product Attributes

4. Brand
5. The Search String

4.1 Generation of Feature Vectors

The following exploratory data plots will help to engineer some of the features. The plots plot the various possible metrics for the data versus the relevance values. Since we have 13 distinct relevance values in the training data, each point in the plot will represent an average of that metric across all records that have that relevance value. Records will be grouped by their by their relevance score and their metric value will be averaged for that group to get a single value of that metric for a given relevance score to be used for these plots.

- Figure (2) is a word count plot. It plots the word count in each of the fields of product title, product description and search string averaged for each of the 13 values of the response variable. The word count does not appear to show much of an explicit correlation to the relevance score except for product description for which there is a slight increase in relevance scores as the words in product description increases.

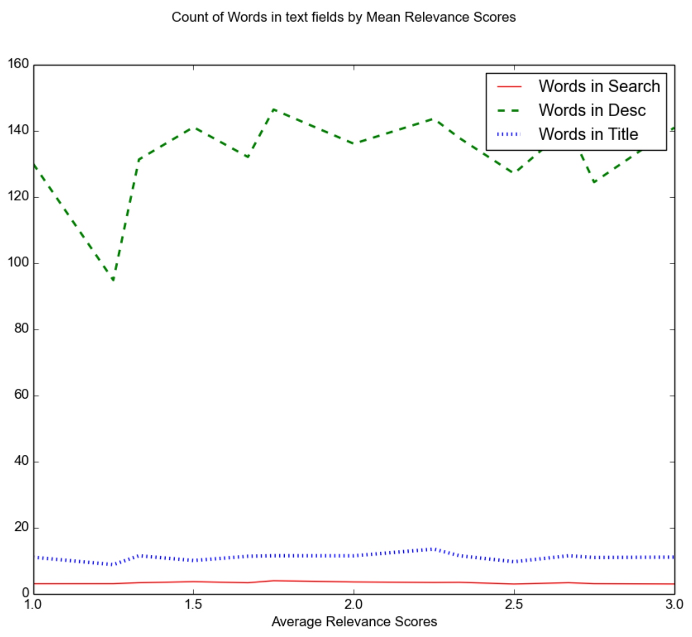


Figure 2: Count of words in different text fields Vs Mean Relevance Score

- Figure (3) plots the average count of matching n-grams between the search string in a record with each of the fields of product title, product description, product brand in that record. Again the average is taken across the records for each of the 13 values of the response variable. As can be seen in the plot, a slight increase in relevance scores as the count of matching n-grams between search and title, description and brand increases.

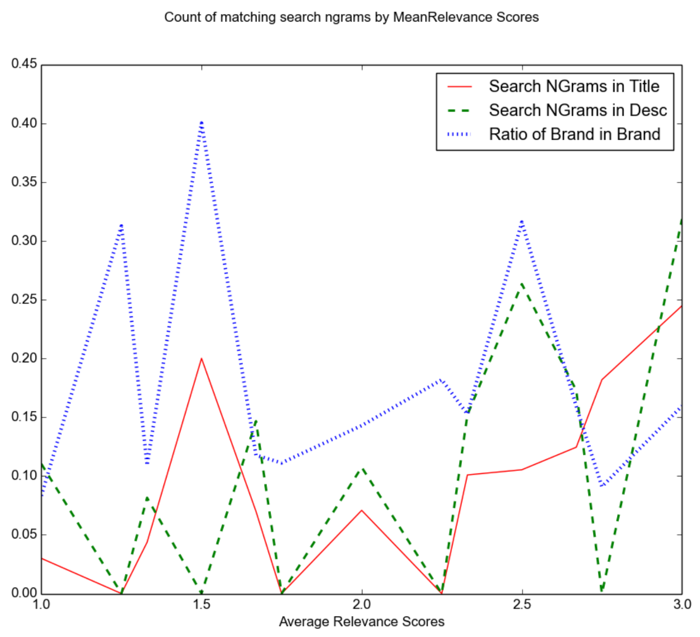


Figure 3: Count of matching n-grams between search and text fields by Relevance Values

- Figure (4) is a more normalized representation of the matching word count between search and the text in title, description and brand. The ratio of the count of matching words to the total number of words in the search string is plotted. The values grouped by relevance and are averaged for each of the 13 distinct relevance values. There is a pronounced increase in relevance score as the ratio of the word match increases for the title and description fields.

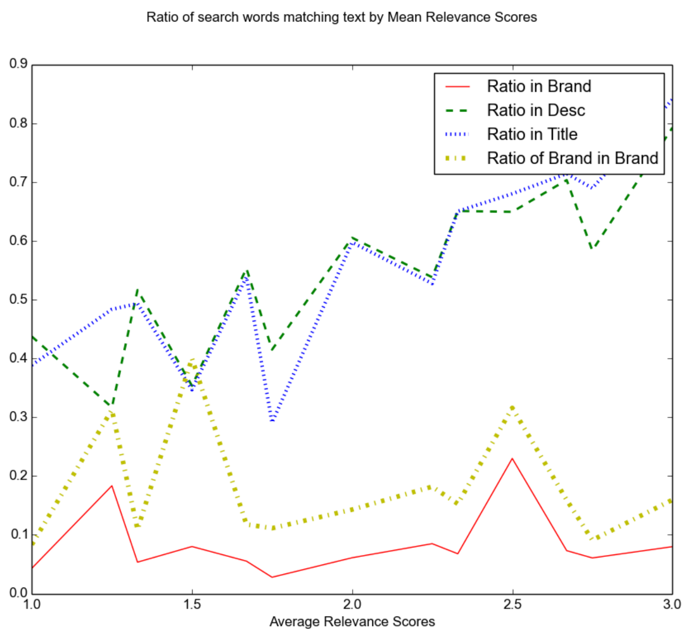


Figure 4: Ratio of word count matching with search to the total words in search by Relevance

- Figure (5) plots the Jaccard Coefficient between the words in search string to each of the text fields of product title, product description and brand. The values are grouped and averaged by relevance score values. The Jaccard coefficient for the product title has a strong positive correlation to the relevance score. The Jaccard coefficient for brand and description with search appears to fairly neutral to slightly positive in its correlation to the relevance score values.

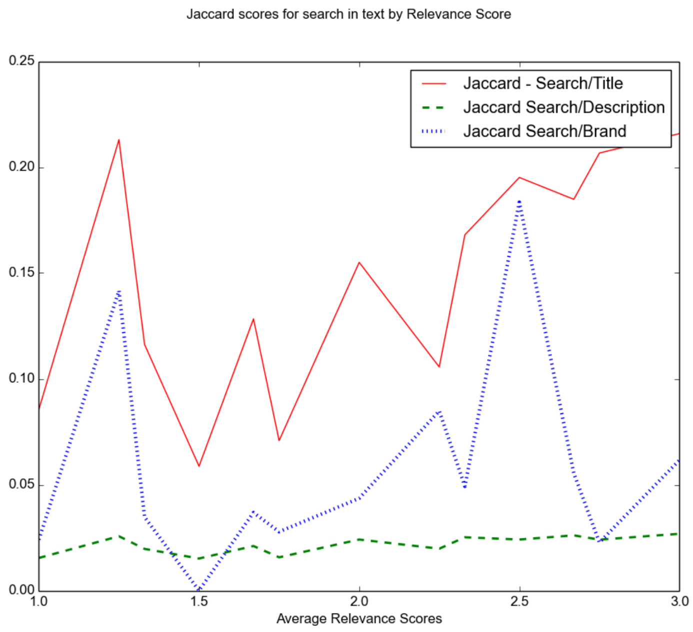


Figure 5: Jaccard Coefficient between search and text fields by Relevance

- Figure (6) scatter plots count of a product id versus the average relevance score for that product id. Product ids are first grouped and counted and then the mean of their relevance is taken for this plot. The plot shows that as the product is seen in more records in the training data, its average relevance value across the records also increases. Product ids with a low count have a much wider spread of their relevance values, while those with a higher count have a narrower band of relevance values.

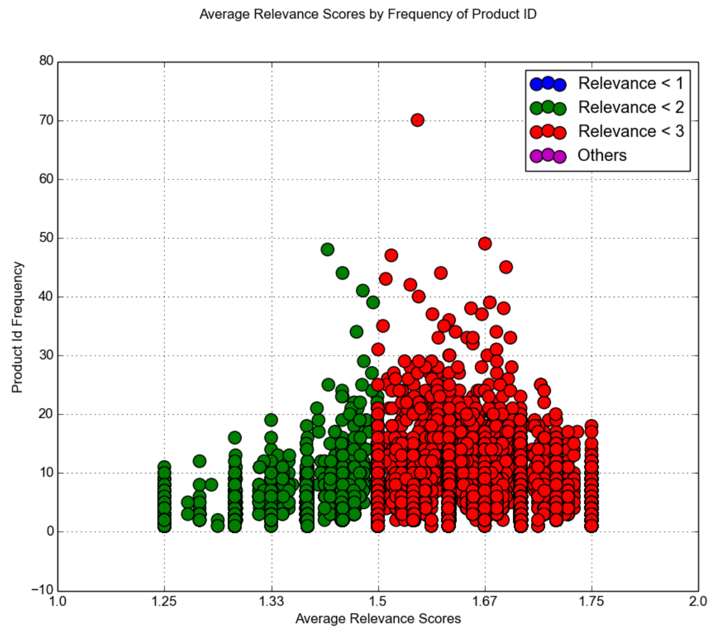


Figure 6: Scatter Plot of the count of product id Vs Mean Relevance

- Figure (7) is a plot of the word count of the words in title and description matching the last word in search. As discussed earlier the records are grouped by their relevance values and an average of this metric value is taken across the records with the same relevance for this plot. There is a very strong and positive correlation between the count of matching last search words to the relevance score as can be seen in this plot.

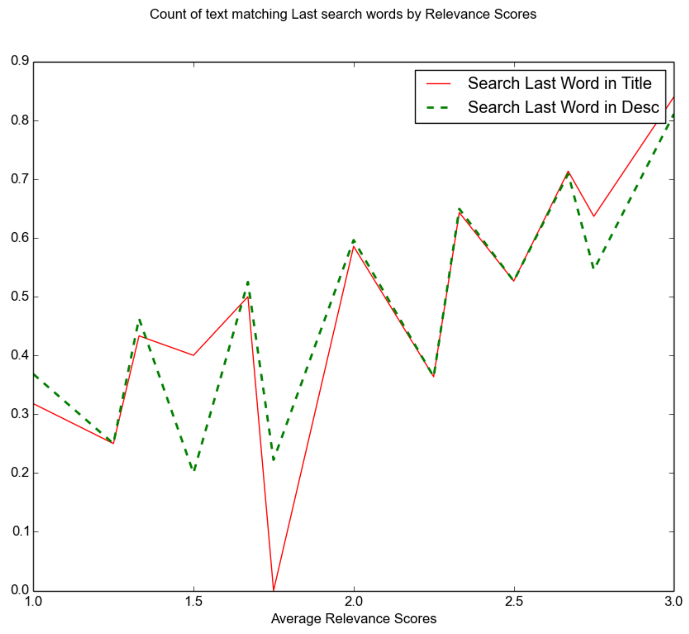


Figure 7: Count of matching words in text fields to last the word in search Vs Relevance

- Figure (8) is a plot of the count of all the matching segments between text in search to the text in title. The count is grouped and averaged by relevance values. The correlation is neutral with a slight increase for the mid range of relevance scores. It does not appear to be a simple linear correlation.

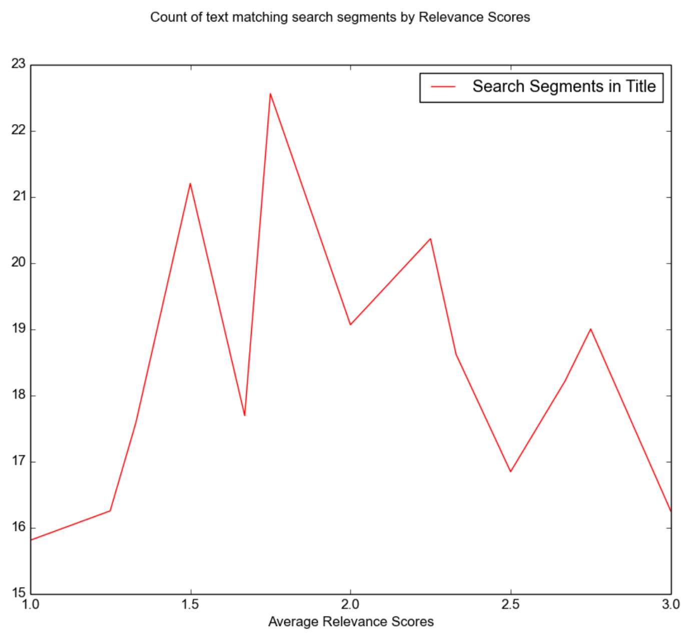


Figure 8: Count of matching segments between search and product title by Relevance

- From the rating guidelines document a search word matching brand is mentioned as an important criterion for a high rating score. As discussed earlier, brand is extracted from the *mfg brand name* attribute value as a text string. From these brand names we build a list of unique brand names and identify each unique brand name by a numeric brand id. We engineer a feature for the brand by populating back the brand id for the brand name string in the record.

- Thus based on the discussion so far the following 22 features will be engineered for each record.

Feature	Description
len_of_query	Count of words in search string
len_of_title	Count of words in title
len_of_description	Count of words in description
len_of_brand	Count of the words in brand
query_in_description	Count of the number of matches of all the ngrams in search to description
query_in_title	Count of the number of matches of all the ngrams in search to title
query_last_word_in_title	Count of the number of matches between the last word in search to words in title
query_last_word_in_description	Count of the number of matches between the last word in search to words in description
word_in_title	Count of the common words between search and title
ratio_title	Ratio of the count of the common words between search and title to the count of the words in search
word_in_description	Count of the common words between search and description
ratio_description	Ratio of the count of the common words between search and description to the count of the words in search
word_in_brand	Count of the common words between search and brand
ratio_brand	Ratio of the count of the common words between search and brand to the count of the words in search
search_term_feature	Count of the segments in search that math the segments in product title
jaccard_search_title	Jaccard Coefficient between words in search to words in title
jaccard_search_description	Jaccard Coefficient between words in search to words in description
jaccard_search_brand	Jaccard Coefficient between words in search to words in brand
pid_count	Count of the number of training records having this product id
brand_feature	Numeric ID for each unique brans
product_uid	Product id
id	id for each record

Table 3: Feature Vector instrumented for each record

4.1.1 SVD and tf-idf

- The search feature vectors engineered thus far are simple word and segment matches between the stemmed search string and the text fields in title, description and brand. However these search metrics while being helpful also tend to capture the noise inherent in the text data. A better approach to searching for text fields would be to be able to project each of these text fields as a vector into a reduced C dimensional space where the noise is filtered out and the variance between them is accentuated. One way to do this is by building a $tf - idf$ (Term Frequency and Inverse Document Frequency) matrix for each text field. This matrix counts the normalized frequency of n-grams in the document factored by how unique a n-gram is for a given document.

- The n-gram size is a tunable parameter that needs to be chosen

- We can perform a dimensionality reduction on this tf-idf matrix using SVD (Singular Value Decomposition). This lower C dimension space is the one which maximizes variance and reduces its noise. This offers a qualitative better space to search in.

- For each of the text fields of product description, title, search string and brand we will have a $tf - idf$ matrix reduced to the first C dimensions and appended to the feature vector for each record.

- The number of dimensions C is a tunable parameter that needs to be chosen.
- With this we have a engineered feature vector of size $22 + 4N$. This feature vector is now ready for fitting a model on.

5 Proposed Method

5.1 Choice of Prediction Models

- With a engineered feature vector for the train and test data, we can begin exploring fitting the different models for learning data from this data.
- The engineered features as explained in section (4.1), are generated from the same body of text and try to express a different dimension of the data. So obviously there is a strong correlation between these engineered features. Thus the rating score depends largely on the interaction between the various engineered features for tuples of $\langle product, search\_string \rangle$, rather than on the independent presence or absence of particular features in the feature vector.
- Decision trees by design handle the interaction in the predictors as each subsequent split depends on the previous splits made. So in comparison to regression models where we have to specifically add the interaction terms to capture this interaction, decision trees are very well suited for learning from a dataset based on interaction like this.
- So the model of first choice for learning from this data set is the decision tree
- However a single decision tree model built on a training set performs poorly on variance and will tend to overfit the training data, and so might perform poorly on the test data set.
- A way to take advantage of the intrinsic benefits of decision trees while ultimately seeking to reduce variance of the fitted model is to use the following ensemble variants which are built on decision trees, an ensemble of independent trees built with bootstrapping, an additive ensemble of trees built with boosting, an additive ensemble of trees built with boosting and regularization.
- For an ensemble of independent trees based bootstrapping we will use the Random Forest model, for the additive boosting version we will use the Gradient Boosted Regression Trees and for the additive boosting with regularization we will use the

Extreme Gradient Boosted Trees. The details for each of the models with the bias variance trade off in them are discussed below.

- For a bench mark comparison we will also try to fit a regularized linear model of type LASSO as well as the Support Vector Regression Model.

5.1.1 Random Forests Regressors

- An ensemble of decision trees are trained independently in parallel on bootstrapped samples of the training data. The regression results of the trees in the ensemble are averaged to get the prediction for te ensemble
- Randomness is introduced in the bootstrap random sample of training data for each tree, a different random subset of features looked at by each tree.
- The characteristics of the trees like its depth, leaves, min data to split etc. are configurable parameters which must be chosen. The number of trees in the ensemble is also a parameter to be tuned.
- The prediction of the ensemble models is the average the results of the constituent classifiers.
- While the randomness in samples and the set of features increases the bias, the averaging of the trees tends to reduce the variance.
- The advantage with this model is that it is faster to develop as independent trees can be trained in parallel. Random Forest Regressor trees are very good at reducing variance with a trade off in bias.

5.1.2 Gradient Boosted Regression Trees

- The Gradient Boosted Regression Trees are a boosting additive model of decision trees. The loss function that will be used in this case is the default least squares loss function. The model seeks to add the best tree at each stage that reduces the objective loss function. So in that sense this model tends to reduce bias and has good predictive power.
- The characteristics of the trees like its depth, leaves, min data to split etc. are configurable parameters which must be chosen. The number of trees in the ensemble is also a parameter to be tuned.

- Variance can be reduced by introducing bootstrapping by specifying a subsample to be used for training each tree, variance could be further reduced by choosing a random subset of features to be used by each tree.

- Regularization can be introduced to the model by specifying a learning rate which shrinks the contribution of each step. The learning rate is a tunable parameter that needs to be chosen.

- Boosted regression trees are slower to train as trees need to be built in succession.

5.1.3 Extreme Gradient Boosted Regression Trees

- While Gradient Boosted Regression Trees through its tunable parameters provides a good handle for balancing bias and variance, It still does not provide a way to penalize for model complexity and overfitting. Extreme Gradient Boosted Regression Trees seeks to address that by adding a penalizing terms for the complexity of the tree at each step to the objective function that needs to be minimized. This regularization parameter γ is a tunable parameter that must be chosen.

- While the tree complexity penalty which tends to penalize overfitting might make Extreme Gradient Boosted Regression Trees have a higher training error, it will benefit its performance on the test data.

5.1.4 Linear Regression Models - Lasso

Linear regression models need to have explicit interaction terms defined in order to capture the interaction between predictors in the feature vector. So for a dataset like this ,where the results depend on the interaction of predictors, it is hard for regression models to outperform a specialized decision tree model like random forests and boosting. However we will try to fit a Linear Regression Model with a regularization of the L1 norm, the Lasso Linear Regression Model.

5.1.5 Other Models - SVM

In addition to the two models mentioned above we will also try the Support Vector Machine Regressor model.

5.2 Error Rate - RMSE

The error rate between the predicted relevance values and the actual relevance values are measured as root mean squared error (RMSE)

5.3 Splitting into Training and Test Data

For the Kaggle competition, Kaggle has a hidden test set against which the test RMSE error is computed for a submitted model. The performance of the models is rated based on this test RMSE error. However for this project we will divide the training data into a project training set and project test set. The model performance will be evaluated on this hidden project training set as well.

5.4 10 Fold Cross Validation for choosing the Optimum Model Parameters

As discussed in section (4.1.1) and (5.1) there are several parameters that need to be tuned for getting the best fit for each of the models discussed. The number of n-grams in tf-idf, the number of dimensions C in SVD, the number of trees in the ensemble models, the tree depth, the number of features, the learning rate, the regularization parameter for model complexity γ, the shrinkage factor α for Lasso are some of the parameters that need to be chosen.

1. The optimum value of these parameters will be chosen using ten fold cross validation on the training set

2. Once the optimum parameter values are chosen for each model, the model with those optimum parameters will then be retrained on the whole project training set

3. The Project test results will be computed on the held back project test set

4. Next each of these models will be trained on the full competition training data set provided by Kaggle

5. This final model trained on the whole competition training set will then be used to generate the test results for the competition. These test results will be submitted to the Kaggle competition

5.5 Choosing the Best Model by Bootstrapping

As discussed in section (5.4) we are training a Random Forest, a Gradient Boosted Regressor, a Extreme Gradient Boosted Regressor and a Lasso linear model with their optimum hyper parameters chosen by cross validation. From these we would need to choose the best model for generating the test results for submission to the competition.

1. The best performing model will be evaluated using bootstrapping on the training set.
2. A $50 : 50$ split will be used for training and testing during bootstrapping
3. A 50 iteration bootstrapping will be performed
4. A statistical test (T-test, Wilcox test) on the bootstrapping test results, will be used to choose the best performing model

5.6 Tools Used

The project is implemented in Python V2.0. Table (4) lists the primary python modules used for this project.

Python Module	Description
pandas	DataFrames for storing, filtering , processing and grouping train and test data
numpy	For n dimensional arrays and math computation
sklearn	For machine learning libraries for training models and cross validation
nltk	For nltk segmentation and stemming
xgboost	For the implementation of the Extreme Gradient Boost Regressor
scipy	For T-Test and Wilcox test and implementation
matlabplotlib	For creating the plots

Table 4: Python Modules Used

From section (5.4) we need to perform cross validation to choose a set of optimum parameters for each model. We perform this cross validation using the GridSearchCV class available in python. GridSearchCV will perform a n-fold cross validation for each of the possible combination of parameters values to be tried for the model and choose the set of model parameters which give the best result for the scoring function provided. This scoring function in this case is the RMSE.
Table (5) lists the perl implementation of the machine learning models that are used in this project.

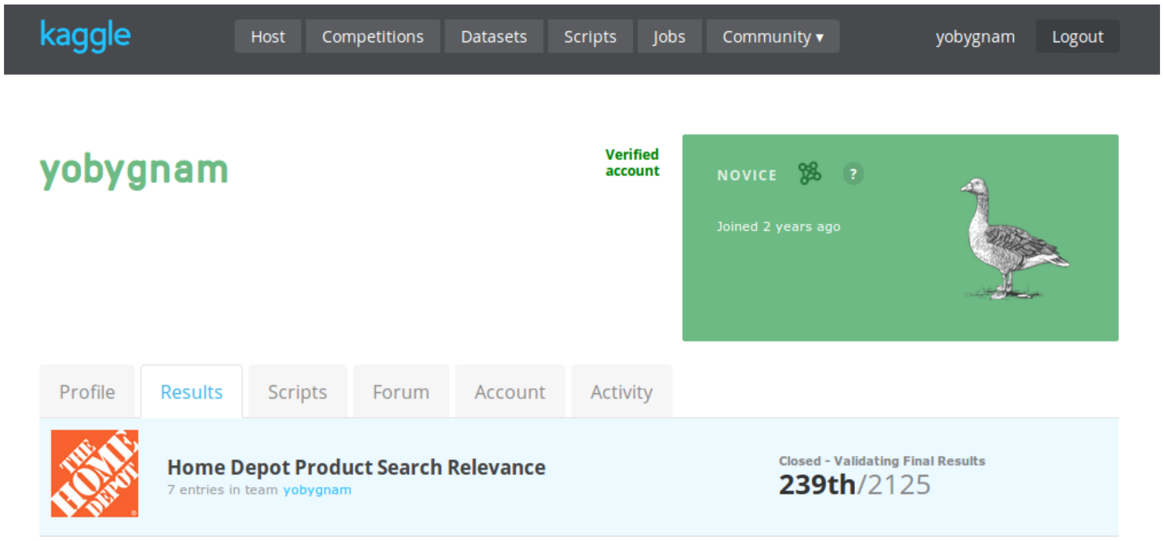
Python Machine Learning Model	Description
GridSearchCV	Perform N fold cross validation to choose best paramaters for a model
TruncatedSVD	Dimensionality reduction using truncated SVD of the tf-idf matrix
TfidfVectorizer	Convert text to matrix of TF-IDF features
LassoCV	Fit a Lasso Model with Cross Validation to choose α
RandomForestRegressor	Implement the Random Forest Regression model
GradientBoostingRegressor	Implement the model for Gradient Boosting for Regression
XGBRegressor	Implement the Extreme Gradient Boost Regressor model

Table 5: Python Regression Model Implementations Used

6 Analysis and Results

6.1 Competition Summary - Kaggle Home Depot Product Relevance Search

- Extreme Gradient Boosting Regressor ws the model chosen for submission. The optimum values of parameters for this model are in table (6)
- My submission to the Kaggle competition was ranked at 239th/2125. The best RMSE score on the test set for my submission was 0.46710. The winning team in the competition had a RMSE score of 0.43192. There where a total of 35,000 submissions made for this competition. This project made 7 submissions. Figures (9) and (10) are screen shots of the results from Kaggle. Results can also be seen at the following URL `https://www.kaggle.com/c/home-depot-product-search-relevance/leaderboard`



yobygnam

Verified account

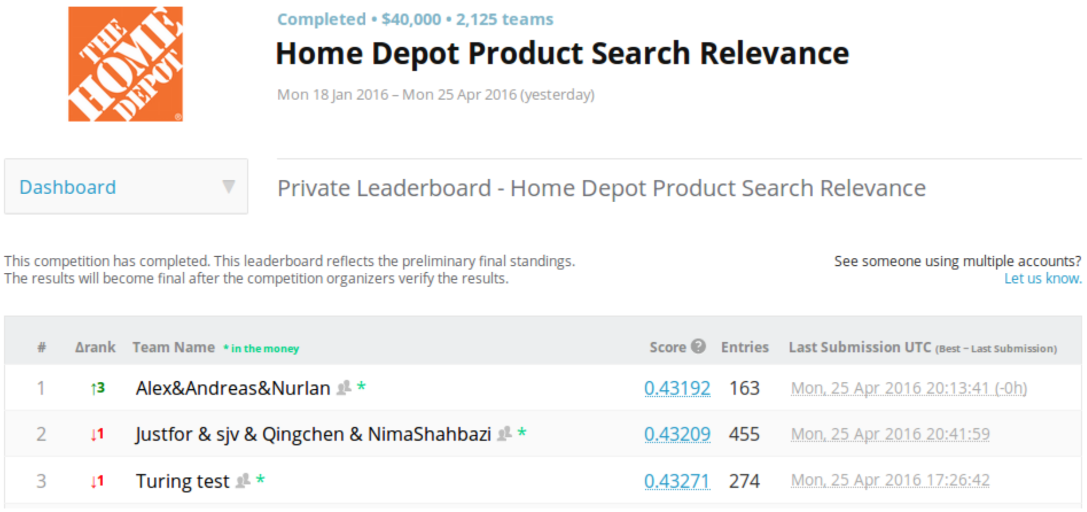
NOVICE ?

Joined 2 years ago

Home Depot Product Search Relevance
7 entries in team yobygnam

Closed - Validating Final Results
239th/2125

Figure 9: Rank in Kaggle Home Depot Product Relevance Search Competition for my submission


Completed • $40,000 • 2,125 teams

Home Depot Product Search Relevance

Mon 18 Jan 2016 – Mon 25 Apr 2016 (yesterday)

Private Leaderboard - Home Depot Product Search Relevance

This competition has completed. This leaderboard reflects the preliminary final standings.
The results will become final after the competition organizers verify the results.

See someone using multiple accounts?
Let us know.

#	Δrank	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	↑3	Alex&Andreas&Nurlan *	0.43192	163	Mon, 25 Apr 2016 20:13:41 (-0h)
2	↓1	Justfor & sjv & Qingchen & NimaShahbazi *	0.43209	455	Mon, 25 Apr 2016 20:41:59
3	↓1	Turing test *	0.43271	274	Mon, 25 Apr 2016 17:26:42

Figure 10: Top Rank results in Kaggle Home Depot Product Relevance Search Competition

6.2 Results for Project test set

- We created a hold out test set from the training data provide from Kaggle for this project. The RMSE error on that data set was 0.4672668171284915
- The optimum values of parameters chosen by 10 fold cross validation for Extreme Gradient Boosting Regressor are in table (6)

	Extreme Gradient Boosting Regressor	Gradient Boosting Regressor	Lasso	Random Forest Regressor
Optimum Parameters Chosen by 10-Fold Cross Validation	estimators 600 learning rate 0.01 gamma 0 max_depth 5 subsample all features all	estimators 400 learning rate 0.05 max_leaf_nodes 5 features all subsample all	alpha 0.001	estimators 200 max_leaf_nodes 4 features all sample all

Table 6: Optimum model Parameters chosen by 10-Fold Cross Validation

- Table (7) tabulates the test error rate for predictions performed with the optimum parameters for each model on the hold out project test set. The Extreme Gradient Boosting has the best performance , followed by Gradient Boosting Regressor, Random Forest Regressor and finally Lasso.

	Extreme Gradient Boosting Regressor	Random Forest Regressor	Gradient Boosting Regressor	Lasso
Project Test Error Rate	0.4672668	0.472557	0.468457	0.477327

Table 7: Test Error Rate for the models on the Project hold out test set

6.3 Top Features

- Figure (11), (12) and (13) list the top 20 features as seen by the Random Forest Regressor, Gradient Boosting Regressor and the Extreme Gradient Boosting Regressor respectively. The models seem to concur that the useful features are those engineered for word and segments in search string matching with those in the title. Both boosting models find the reduced dimensions of the tf-idf matrix for search string segments matching those in title take most of the top 20 places. So these plots suggest that search segment and title are mostly what relevance score is based on in this data set.

  .

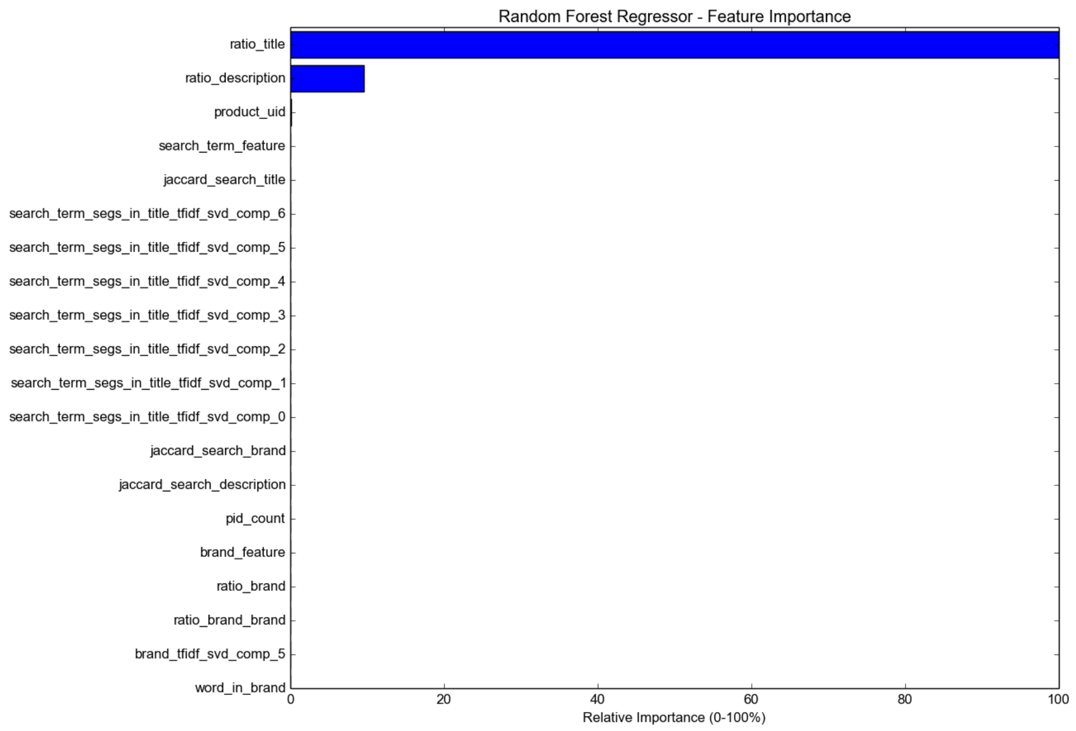


Figure 11: Top features from the Random Forest Regressor

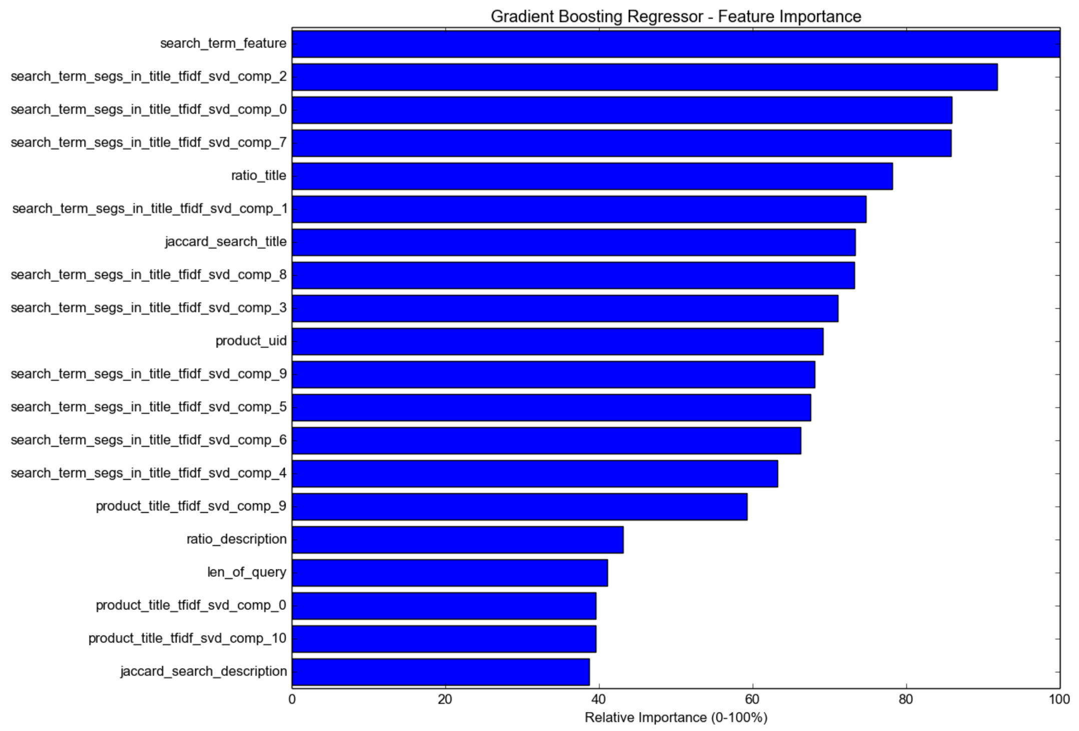


Figure 12: Top 20 features from the Gradient Boosting Regressor

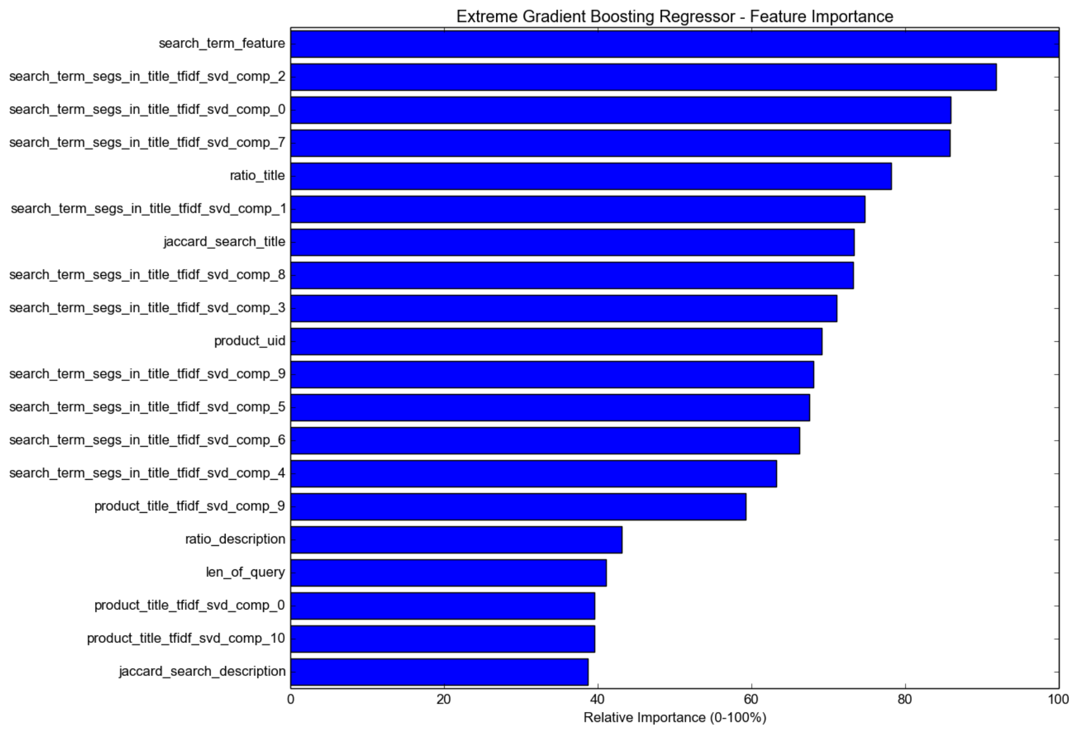


Figure 13: Top 20 features from the Extreme Gradient Boosting Regressor

6.4 Cross Validation Plots - Extreme Gradient Boosting Regressor

- Figure (14) is a 3D plot of the CV results for Extreme Gradient Boosting Regressor. RMSE is plotted against n_estimators/ and γ for different learning rates. The tree_depth is held constant at the prior chosen optimum value of 5. A low learning rate a large number of estimators seem to provide the optimum combination of parameters.

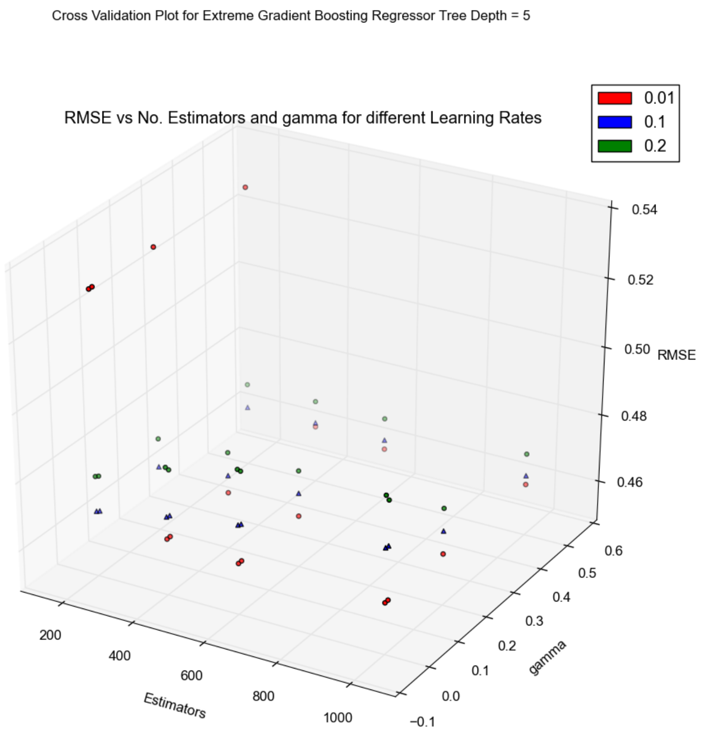


Figure 14: Cross Validation Results for n_estimators,gamma,learning rate for Extreme Gradient Boosting Regressor

- Figure (15) and (16) is a plot the training and test RMSE as the number of estimators is increased for both the Gradient Boosting Regressor and the Extreme Gradient Boosting Regressor. The other parameters for the models are set to their optimum values as given in table (6). Plots show that training error and test error rates form an elbow at around 200 estimators, further increase in the number of estimators shows very little reduction in test error rate. However for the Gradient Boosting

Regressor the training error rate does decline as the number of estimators is further increased, indicating that Gradient Boosting Regressor since it does not have a penalty on the model complexity might have a tendency to overfit in comparison with the Extreme Gradient Boosting Regressor. n_estimators/tree_depth/learning rate

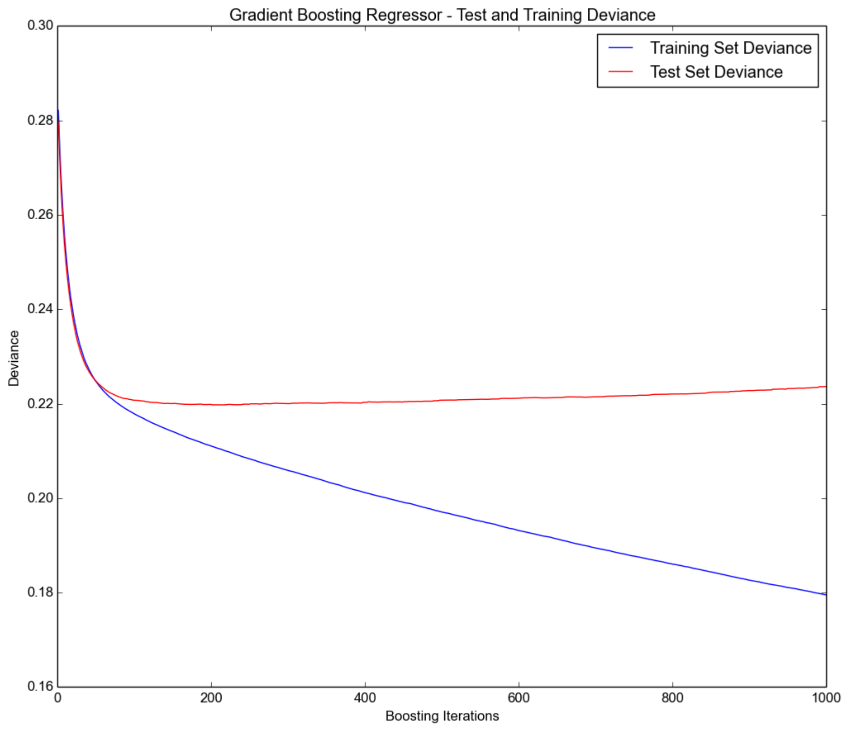


Figure 15: Training and Test error rate Vs n_estimators for Gradient Boosting Regressor

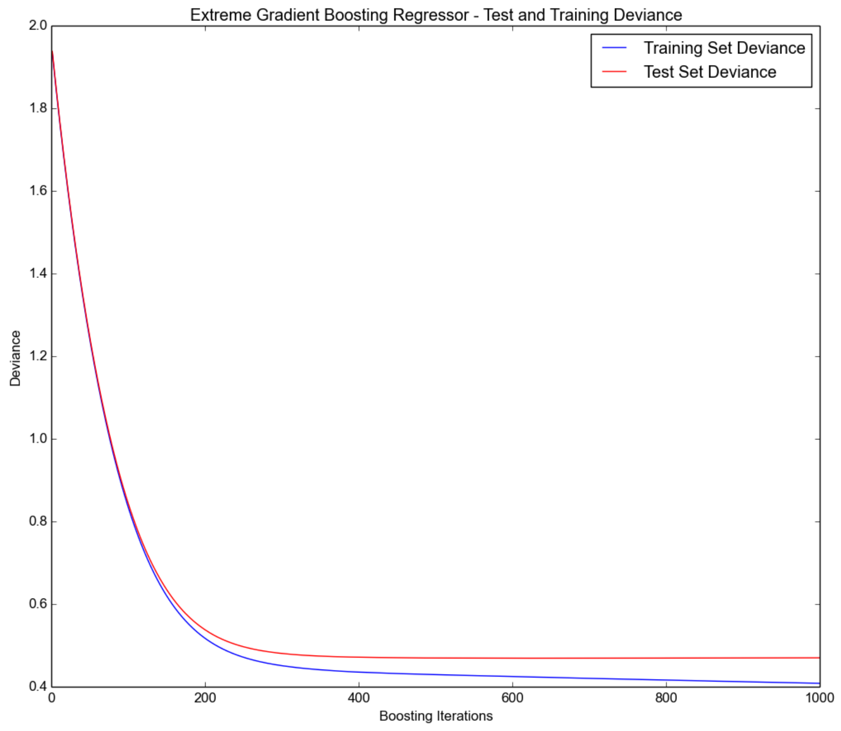


Figure 16: Training and Test error rate Vs n_estimators for Extreme Gradient Boosting Regressor

6.5 Choosing the best model - Bootstrapping Results

- As discussed earlier the best model for submission will be chosen based on bootstrapping results. We performed a 30 iteration bootstrapping with a random 40% of data chosen with replacement as the test set and the remaining as the training set. Figure (17) is the box plot of the bootstrapping results. The Extreme Gradient Boosting Regressor performed the best on the test set,

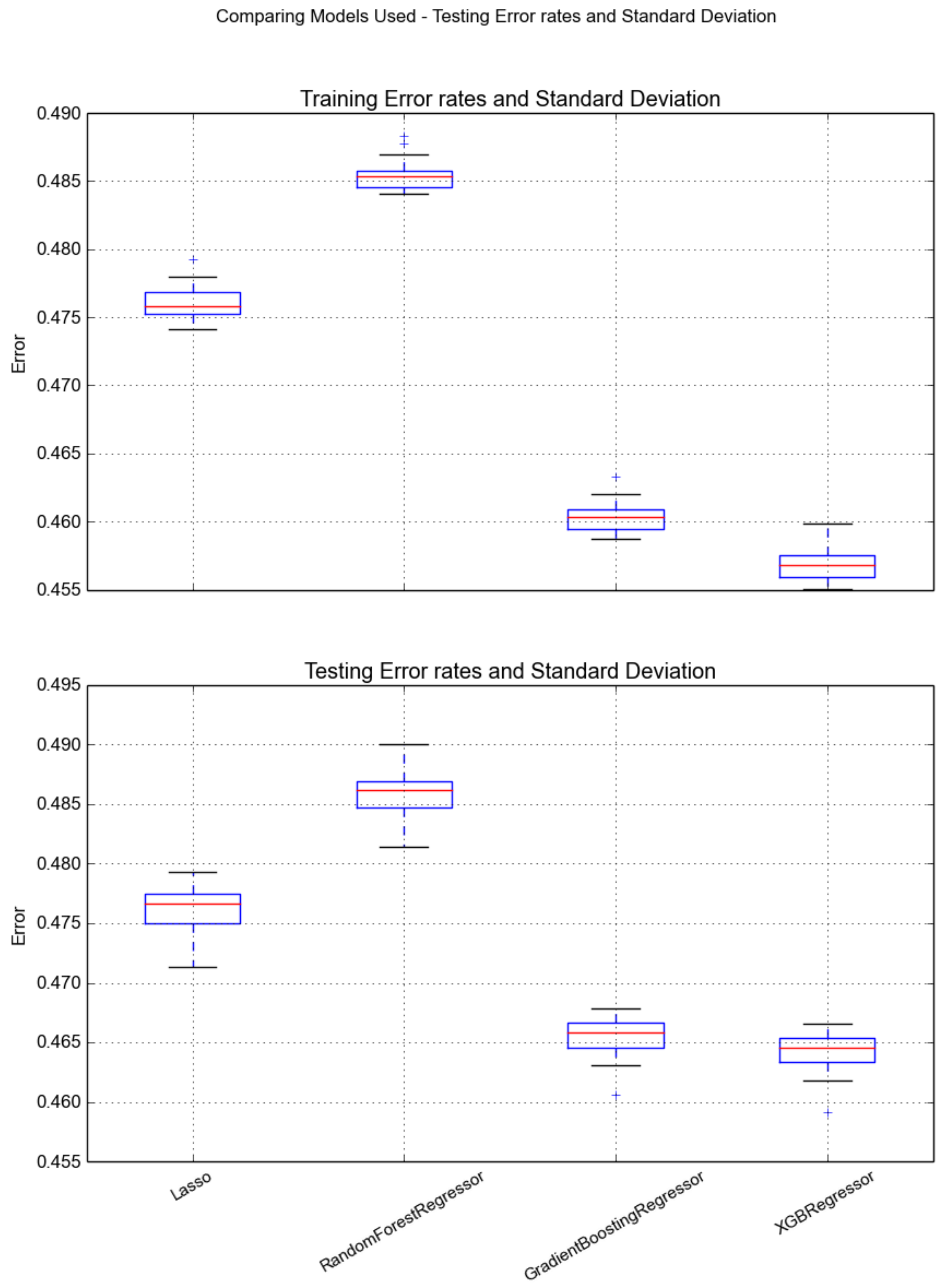


Figure 17: Boxplot of B=300 Bootstraping results

6.5.1 T-test and Wilcox Test for choosing the best model

- We performed a statistical T-test as well a a Wilcox test to confirm that the Extreme Gradient Boosting Regressor results was the best model based on the bootstrapping test set results.

- Table (8) tabulates the T-test and Wilcox 2 sample paired test results between Extreme Gradient Boost Regressor and every other method. The $P - Values$ are very low for significance level of $\alpha = 5\%$. This leads us to to reject the NULL hypothesis at the significance level of $\alpha = 5\%$ and strongly conclude that Extreme Gradient Boost Regressor is the best model trained for this dataset.

	RandomForestRegressor	GradientBoostingRegressor	Lasso
T-Test	1.0701507e-47	0.0074	3.12418e-34
Wilcox Test	1.7343976e-06	1.734397e-06	1.734397e-06

Table 8: T-test and Wilcox Test comparing Error rate of Extreme Gradient Boost Regressor with other Regression models used

7 Conclusions

- Understanding the domain, exploring and analyzing the data are the most important steps for engineering good features to train a model on. In this competition there where no ready features provided, the input was text data and engineering features was left to ones ingenuity and imagination.

- A good and well instrumented feature matrix, that is a faithful representation of the data is most crucial for performing any learning on the data

- While various training models have different assumptions, they will still do a reasonably decent job on a very well engineered feature vector, but the reverse is not true. Even the best and most appropriate model will do a poor job of learning on poor feature vector that does not represent the data very well

- Given a well engineered feature vector and an appropriate model, one still needs to perform a exhaustive search for the most optimum model parameters. This process is time and resource intensive and is best done in parallel on a distributed system. A single CPU PC is not a great tool for this purpose and will not permit an exhaustive search with cross validation on even a medium sized data set

- As expected training error rate is lower than test error rate for all models.
- Beyond a certain threshold of ensembles the models will over fit the training data unless penalized.
- For this competition Cross Validation and Bootstrapping are the primary tools used to train and evaluate the models and seem to be the default methods to do so.
- Ensemble models with boosting which have a penalty for model complexity in their objective function are a fairly robust learning model as they have the mechanism to best balance bias variance tradeoff right into the objective function
- Boosting methods since they add learners in series are slower than pure ensemble methods which can take advantage of parallel processing to train and independent learners. However boosting methods are better on bias while non boosted ensembles trade bias for variance.

7.1 Lessons from the Project

- From the point of view of the competition the top 10 submissions as well as most of the top 200 ranks where taken by teams of more than 1. Competing as a team in competitions like this gives an advantage for trying new ideas and working on them in parallel
- One needs to go deeper with engineering features and go wider in searching for optimum parameters when fitting models on this trained data.
- An exhaustive search for optimum parameters on a grid of parameter value ranges using a ten fold cross validation is a slow and time consuming process. One needs to have an efficient strategy on perform this computation as it is crucial for fine training the model for a competition like this. Parallel and distributed computing on a cloud environment is the best way to perform this computation both for speed and efficiency and precision of the project.

8 Appendix

1. The python source code used for performing the data analysis and for engineering the feature vectors for the this project is in file $ajdsouza31 - hd\_rf\_featurize.py$, which is submitted on $T - Square$ along with this report.

2. The python source code for training the models, choosing the best model and for generating the test results for submission to the Kaggle competition is in file $ajdsouza31 - hd\_rf.py$, which is submitted on $T - Square$ along with this report.

3. The following is the URL for the Kaggle Home Depot Product Search Relevance competition. The test and training data sets are available at this link `https://www.kaggle.com/c/home-depot-product-search-relevance`

9 References

[1] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *LATEX: Elements of Statistical Learning*, Elements of Statistical Learning Ed. 2, 2009.

[Friedman(2001)] J.H. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[Friedman et al.(2000)Friedman, Hastie, and Tibshirani] J.H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion). *The Annals of Statistics*, 28:337–407, 2000.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830

[Zhang and Yu(2005)] T. Zhang and B. Yu. Boosting with early stopping: convergence and consistency. *The Annals of Statistics*, 33:1538–1579, 2005.