G9: Home Depot Product Search Relevance

Rohan Sadale, Akshay Kulkarni, Utkarsh Kajaria and Gautham Sunder

1. Introduction

To enhance customer experience and engagement on online platforms it is imperative that the search results are highly relevant to the search terms. The objective of this project is to improve shopping experience of Home Depot customers by developing a model that can accurately predict the relevance of search results. The dataset is obtained from Kaggle and consists of description of every product available in Home Depot's catalog, product attributes, search term/s and relevance score (a real number between 1.0 to 3.0) of the search terms to products.

The original problem as presented on Kaggle is a regression task where the objective is to predict a relevance score between a search term and the given product. In addition to regression problem, we tried to approach this task as a classification problem (Two-class and three-class classification). Our motivation for treating this as a classification method was to leverage more intuitive evaluation methods such as accuracy, confusion matrix, f-score over RMSE in a typical regression problem. For two-class classification, we used a threshold of 2.0 i.e any relevance score above 2.0 was considered as relevant and anything below as irrelevant. For three class classification, we divided scores into 3 categories low(1.0, 1.67], medium(1.67, 2.5] and high(2.5, 3.0] scores. Our approach this problem consists of four main parts: text cleaning, feature engineering, model building and evaluation.

2. Methodology

2.1 Text Cleaning

The dataset primarily being textual in nature calls for extensive text cleaning. The methods used for this are as follows:

- Removal of stopwords
- Stemming and Tokenizing
- Spell correction of search terms (Google spell check)
- *Lower-Upper-Case splitter:* The product description often contained concatenated terms, of which, the individual terms were otherwise unrelated. This could be due to the fact that while creating the product description file, all the lines in a description file were concatenated without adding spaces. example-*viewDurable*.
- *Letter-Letter splitter:* Words that appeared to be joined by hyphen and slashes were seperated by spaces in between them eg. *Cleaner/Conditioner => Cleaner Conditioner*
- *Html and Special character cleaner:* Special characters and HTML tags were removed as they didn't add any value to the learning process.
- *Digit-Letter splitter:* - Numerical figures containing comma-separators were converted into just numbers, eg. convert 48-Light to 48 Light and convert 1x1x1 to 1 x 1 x 1.
- *Digit-Comma-Digit merger:* eg. Convert 1,000,000 to 1000000.
- *Number-Digit Mapper:* eg. one => 1; two => 2.
- *Unit Converter:* pounds|pound|lbs|lb|lb. => lb; wattage|watts|watt => watt
- Convert all the alphabets to lowercase

2.2 Feature Engineering

Post the completion of text preprocessing, we focused on identifying meaningful attributes from the product description and title that potentially played a role in ascertaining a product's relevance to a given search query. Feature engineering involved developing basic descriptive features, distance features, word2vec embedding features, intersect count and position features. This led to construction of 26 features. The metrics used for building these features are described below.

- *Jaccard Distance* between search term and product title (and product description).
- *Edit Distance* between search term and product title (and product description).
- *First and Last ngram features:* Weights were added to the n-grams of search terms based on the order of their occurrence in product title. We gave more weight to the first and lst n-gram of the search query. We used answer to the following questions while framing these features:
  - How many word ngram in product title closely matches with first ngram of search term?
  - How many word ngram in product title closely matches with last ngram of search term?
- *Intersect count Features:* Count of word ngram of search term that closely matches any word ngram of product title
- *Intersect position Features:* For each word ngram of search term it's position in the product title was computed and the average position over all search term n-grams was recorded.
- *Cooccurrence Count:* Count of closely matching word ngram pairs between search term and product title
- *Word2Vec/Doc2Vec Average Similarity:* We computed the average similarity between word n-gram of the search term and that of the product title using word2vec
- *Word2Vec/Doc2Vec Centroid RMSE:* We also computed the RMSE between search term and the product title using word2vec
- *Longest Match:* Longest string match length between search term and product title
- *TF-IDF Scores:* For each word ngram in search term, we computed its TF-IDF score and then took the average of those scores. We used the product description and title as our training corpus for building the document term matrix.
- Asymmetric features that are non contextual such as presence of product dimensions (*width, height and weight*)

The feature values were normalized to account for the varying lengths of product title and description where ever appropriate. We also consider two words to be matching (closely matching) if their relative edit distance is less than 0.2 in order to account for misspelt characters in a word, eg. *conditionar (which probably should be conditioner)*. In addition to this, we consider uni-gram and bi-gram representation of term while calculating distances.

2.3 Model Building

We used the following algorithms to train our models.

2.3.1 Linear Models

- **Naive Bayes:** Naive Bayes is a generative model characterized by a strong assumption of the independence of the features it operates on. The probability of a feature given the class is based on the joint probability of the feature and the class. The aim is to choose the class which maximizes $\prod_{i=1}^{m} P(f_i|c)$ where f_i represents the current feature and c is the current class being evaluated.

- **Logistic Regression:** We used Logistic Regression with no regularization. Logistic Regression selects a parameter θ which maximizes the likelihood function -

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{(y^{(i)})}(1-h_\theta(x^{(i)}))^{1-(y^{(i)})} \qquad where \qquad h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

2.3.2 Non-Linear Models

- **Random Forests:** Random forests uses an ensemble of decision trees to make classification/regression models. The trees are built from a training dataset and can be used to make predictions on the test dataset. Random Forests grows many classification/regression trees. To predict a new information from an input vector, pass the input vector down each of the trees in the forest. Each tree gives a prediction. For classification, the forest chooses the class having the most votes (over all the trees in the forest). For regression, the forest uses averaging over all trees to improve the predictive accuracy.

- **AdaBoost:** It is an algorithm for constructing a *strong classifier* as linear combination of simple *weak* classifiers. Boosting proceed in rounds.

  - Initially all data points are assigned equal weights.
  - Sample points from the training dataset with probability of each point being sampled proportional to its weight and build classifier on those sampled points.
  - Choose weight for the classifier. $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
  - Update weight of all original data points. $w_{t+1}(i) = \frac{w_t(i) exp(-\alpha_t y_i G_t(x_i))}{Z_t}$
  - Final Output $g(x) = sign[\sum_{t=1}^{T} \alpha_t G_t(x)] = sign[G(x)]$

- **Neural Networks:** We trained a 5 layered neural network with one input layer, 3 hidden layers and 1 output layer. The network weights were initialized to a small random number generated from a uniform distribution. We set n+4, n+2 and n+4 nodes for the 3 hidden layers where n is the number of features in the dataset. RELU activation function was used on the nodes in hidden layers. The output layer used sigmoid activation (for binary classification) and sparse_categorical_crossentropy activation (for multi-class classification). Optimization was done using Adam Algorithm.

- **XGBoost:** It is an implementation of Gradient Boosting algorithm. In this algorithm instead of simply optimizing the loss function, we also incorporate model complexity into the loss function. More specifically, the loss function that is optimized is shown below. λ and γ are parameters and w is the vector of scores on the leaves.

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \qquad where \qquad \Omega(f) = \gamma T + 0.5\lambda \sum_{j=1}^{T} w_j^2$$

2.4 Experimental Setup

2.4.1 Parameter Tuning

The learning algorithms have few parameters that tuned to maximize the performance on the dataset. We used GridSearch over possible set of parameters to obtain best set of parameters. In GridSearch, for a set of parameters, cross validation is performed to obtain performance. Eventually we choose those parameters for which our metric (performance) is optimized. While doing cross validation we have ensured that class ratio is maintained in training and testing data. Following parameters were tuned for different models:

- Logistic Regression: {λ = [0.001, 0.01, 0.1, 0, 1, 10, 1000]}
- Random Forests: {num_estimators=[300, 500], criterion=[*gini, entropy*], max_features=[*auto, log2, sqrt*]}
- AdaBoost: {num_estimators:[300, 500], criterion=[*gini, entropy*], max_features=[*auto, log2, sqrt*]}
- Neural Network: {num_hidden_layers=[1,2,3], activation=[*relu, sigmoid*]}
- Xgboost: {n_estimators : [100,150,200], max_depth : [3,5,7,9]}

2.4.2 Model Evaluation

2.4.2.1 Methods

- **K-fold Cross Validation:** We used 10 fold validation to measure the performance of our model. In this method, the dataset is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. By this method we get more accurate estimate of performance of our models.
- **F-measures:** Since, number of data points belonging to each class is not same, we needed some metric to judge how well our classifier does for a particular class. F1- scores are good way to measure performance of classifier on each class as it takes into account precision and recall. To measure measure F-score we divided our data into 2 parts i.e 80% for training and 20% for testing.
- **Learning curve:** For generating learning curve, we split data K-times into training and testing set. Subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size and the test set will be computed. Afterwards, the scores will be averaged over all k runs for each training subset size. By this method, we can gauge how well our model is learning as we feed more data to our model.

2.4.2.2 Results

- **10-fold Cross-validation (Classification)**

	2-class Accuracy	2-class Std Deviation	3-class Accuracy	3-class Std Deviation
by Random (Benchmark)	0.67	-	0.44	-
Naive Bayes	0.645	0.018	0.438	0.009
Logistic Regression	0.685	0.008	0.523	0.024
RandomForest	0.738	0.007	0.55	0.006
Adaboost	0.698	0.012	0.528	0.01
Xgboost	0.72	0.01	0.532	0.008
Neural Networks	0.725	0.005	0.484	0.016

- **10-fold Cross-validation (Regression)**

	CrossValidation RMSE	Kaggle RMSE (actual test-data)
SVM	0.287	0.526
Random Forests	0.225	0.491
Xgboost	0.231	0.476

- **F-measure results**

	2-class Class-1	2-class Class-2	3-class Class-1	3-class Class-2	3-class Class-3
Logistic Regression	0.341	0.802	0.061	0.462	0.656
RandomForest	0.468	0.783	0.296	0.47	0.641
Adaboost	0.541	0.705	0.386	0.403	0.609
Neural Networks	0.522	0.749	0.345	0.453	0.615

- **Learning curve** The learning curve results (for 2-class classification) are shown in Appendix (Figure-1). We have also plotted detailed learning curve results for Random Forest in Appendix (Figure-2)

3. Conclusions and Future Work

In this project, we set out to train models on the Home Depot dataset and used established evaluation criterion (RMSE, accuracy, f1-score) to arrive at more relevant products for user queries. The models that we evaluated did give us some encouraging results. We saw that when treated as a classification problem, the models that we trained (AdaBoost, Random Forests and 5-layer Neural Network) all yielded error rates lower than that of a random classifier. Out of these, the Random Forests model, when tuned to its optimum parameters surpassed the others in terms of error rate. On the other hand, when treated as a regression problem and using 10-fold cross-validation RMSE as our metric, we found that Random Forests outperformed SVM and Xgboost. However when evaluated on the actual test data (i.e. Kaggle leaderboard), we found XgBoost performed better than Random Forests put us on the top 25 percentile of the leaderboard.

It is clear however, that there is scope for us to explore newer ideas to enhance the models. So far, our feature engineering led to 26 attributes. We feel that this set of attributes might not be exhaustive and further exploration into similar problems in the domain of text mining will help us in constructing more useful features. In addition, for our ensemble methods, it will be interesting to find out if model stacking makes any difference when used in place of traditional methods for combining the classifiers. We hope that using these ideas will help us gain greater insight into the problem and perhaps, higher accuracy.

4. References

- Data Source: https://www.kaggle.com/c/home-depot-product-search-relevance/data
- Kaggle Forums: https://www.kaggle.com/c/home-depot-product-search-relevance/forums
- Gensim python package for using word2Vec https://radimrehurek.com/gensim/models/word2vec.html
- AdaBoost Classifier http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- RandomForest Classifier http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Learning Curve http://scikit-learn.org/stable/modules/generated/sklearn.learning_curve.learning_curve.html#sklearn.learning_curve

- Word2Vec https://en.wikipedia.org/wiki/Word2vec
- Source Code: https://github.com/rohansadale/Kaggle-HomeDepot
- XGBoost: http://xgboost.readthedocs.io/en/latest/
- Keras Python Package for Neural Networks:- https://keras.io/
- Machine Learning library from Sklearn: http://scikit-learn.org/stable/

5. Appendix

Learning Curves

- Learning curve for All Classifiers

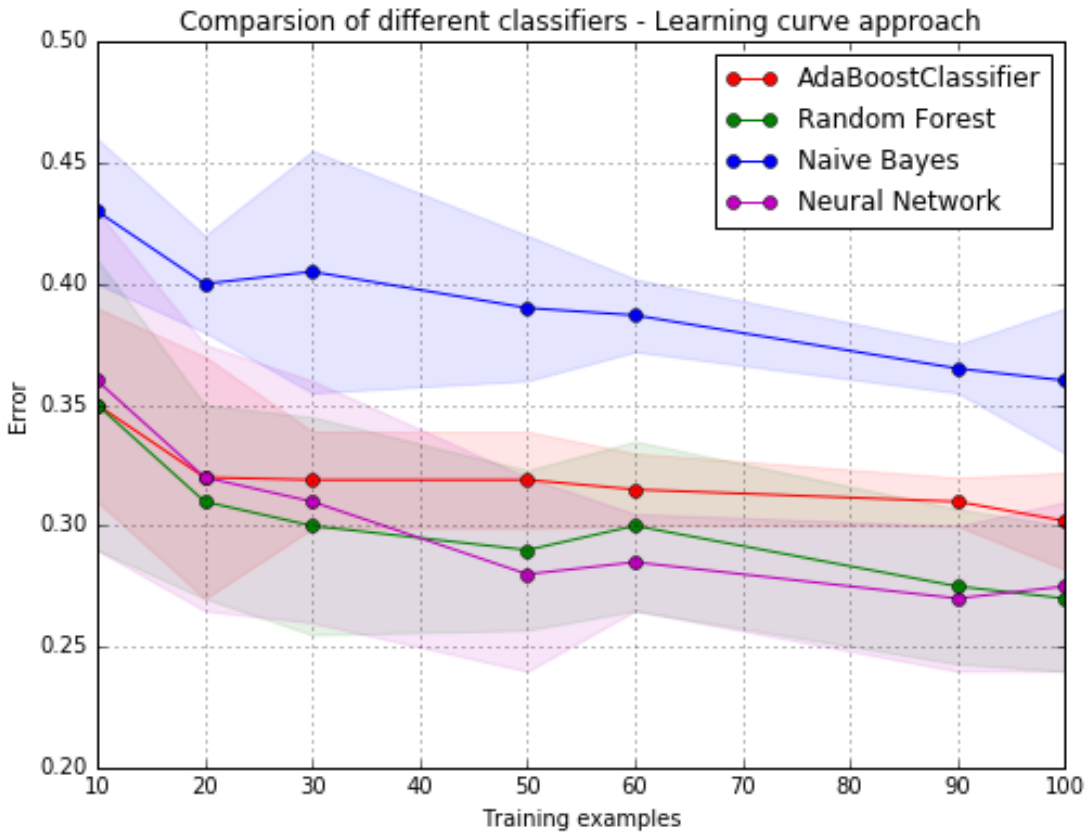

Figure 1: Learning Curve Results

- Learning curve for RandomForest

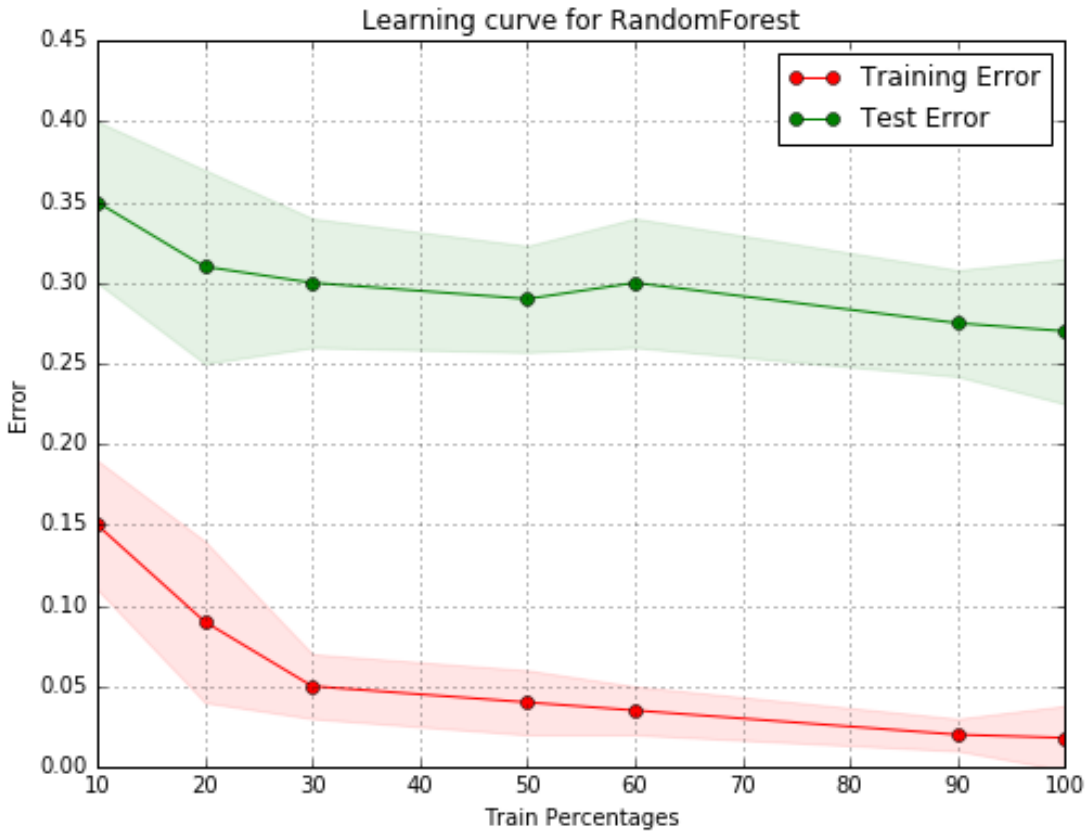

Figure 2: RandomForest Learning Curve