STATS 201B: Statistical Modeling and Learning

Home Depot Product Search Relevance

Rebecca Correia
rebecca.correia@cs.ucla.edu
204587944

Salil Kanetkar
salil@cs.ucla.edu
704557096

Abstract

With an increasing need to improve the user experience, e-commerce industries like Home Depot are taking measures to analyze the relevance of the product results that are retrieved for every search query entered on their website. This paper revolves around exploring and processing the data provided by Home Depot, extracting useful features from the rich data, applying different machine learning algorithms to devise efficient solutions to the problem and analyzing how each of the approaches trade-off against the other.

1 Introduction

In the past few decades the e-commerce industry has been expanding tremendously. There have been provisions for users to buy whatever they need just by the click of a mouse or tap of the screen. Home Depot is one such online marketplace that allows its users to scan through and purchase the latest home improvement products ranging from home appliances to kitchen remodeling, bathroom decorating ideas to patio furniture and so on. However, in order to thrive amongst the ever-growing competition in the industry, providing a smooth user experience is of utmost priority. As a result, it is necessary to meet the customer expectations of obtaining accurate results to the queries typed in by them as quickly as possible.

This project is based on an ongoing data science challenge on Kaggle posted by Home Depot. The goal of this project is to develop such a model that could accurately predict the relevance of search results, thereby helping them improve their customer's shopping experience. Home Depot makes use of the concept of search relevancy as a measure of how quickly their customers are able to find the products that they need. As of now, the search algorithms that are being used by Home Depot are analyzed by some designated human raters. However, evaluating the impact of different potential changes on the algorithm manually is a pretty daunting and time-consuming task. Through this project, Home Depot aims at decreasing the human input in the evaluation of the search relevancy, in order to make substantial increments in the number of iterations that can be performed by their team on the search algorithms being used at present.

In this paper, we have aimed at giving an overview of the different approaches that were adopted by us in order to derive a solution to the problem given in three main sections. In the first

section, we discuss about the initial steps of our project that involved a thorough exploration of the datasets provided to us, followed by the pre-processing of the textual data. The next section deals with extracting various relevant features from the dataset and using the standardized data to fit different non-linear regression models. Finally, in the last section we examine the results obtained by applying our approach and analyze how each of the methods trade-off against each other.

2 Dataset Exploration and Pre-processing

2.1 Description of the dataset

The data that has been provided by the hosts of this competition is spread across three main datasets: train, product descriptions and attributes. The train data consists of approximately 74k observations. The other two datasets are provided to us in order to give us detailed information about each of the Home Depot products.

The training set contains the products as well as the corresponding search terms that were used to retrieve the product results. Additionally, the training set also provides us with the relevance scores for each product-search pair. These scores are obtained by averaging the relevance ratings provided by three human raters manually and fall in a continuous range of 1 to 3 with "1" signifying least relevance and "3" signifying highest relevance. For the purpose of validating our model, we decided to split our train set into 70:30 ratio, with 70% data being used to fit the model and the remaining 30% data being used as the test data. Our task is then to predict the relevance scores for the product-search pairs for the test data. Product description and attributes dataset contains useful information such as textual description and technical specifications of the products.

2.2 Data exploration

As an initial step in the project, we decided to perform some analysis on the dataset provided to us in order to gain deeper insights into how the data is structured and discover some useful patterns from the same. Since there were more number of fields in the train and attributes dataset, we decided to explore them individually.

The target variable that has been provided in the train dataset is the 'relevance score'. We observed that there were total 13 unique relevance scores distributed across the dataset. Their frequencies can be observed in the form of a bar-chart as given below. As we can notice from the graph, the relevance score "3" appears in the highest number of observations followed by "2.33" and "2.67" respectively. It should be noted here that some of the relevance scores such as $[1.25, 1.5, 1.75, 2.25, 2.5, 2.75]$ were found to have extremely low frequency count and so are not visible within the scope of the bar-chart.

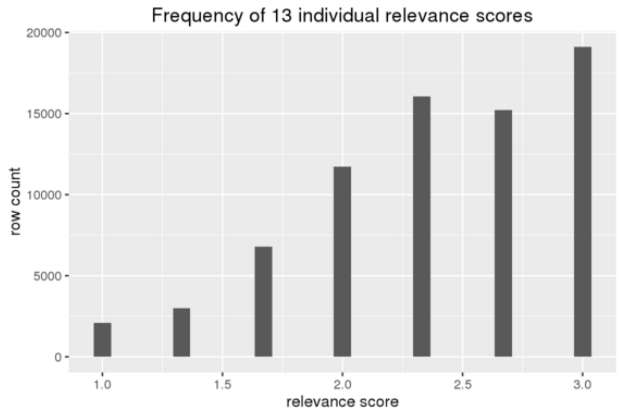

Figure 1: Frequency of relevance scores

We then tried to identify the relationship between the training and the attributes dataset and noticed that some products did not have their corresponding attributes listed in the attributes dataset. Moreover, it was also interesting that the presence of the attributes for any given product had an impact over its relevance score. We can observe this behaviour from the graph given below that shows the relevance trends of products who have their attributes listed and those that do not.

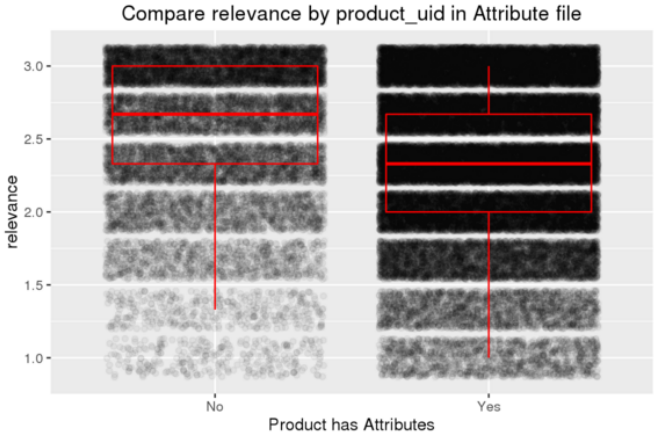

Figure 2: Relevance of products with and without attributes

2.3 Pre-processing

After exploring the dataset, we decided to make three main modifications to the training dataset in order to accommodate useful information from the product descriptions and attributes datasets. Firstly, we appended the product descriptions for each product based on its unique identifier, product id. Secondly, we extracted the brand name for each product from the attributes dataset and appended this information to the training set. Lastly, we added a new variable to the dataset

called as "product info" that was a combination of the search query, product title and product descriptions. However, after making these modifications, we observed that there were a number of missing values in the dataset owing to under-representation of the data that was newly added to the training set, for example, there were some products that did not have any brand information related to it. Since we wanted to optimize the dataset at our disposal, we decided to eliminate all the observations from the training set that contained missing values. As a result, the number of observations in the training set were now reduced to approximately 42k.

Once the dataset was modified as desired, we went ahead and performed some basic data preprocessing. Since the data that we were dealing with was mostly textual data, applying appropriate and adequate preprocessing techniques was found to be pretty crucial for this project. We performed the data preprocessing using the "Natural Language Toolkit (NLTK)" package of Python in the following three steps:

1. Since the dataset contained words with a mix of upper and lowercase letters, we first decided to format the dataset to contain only lowercase letters. This step helped us to standardize the dataset and perform further text processing easily.

2. Next, we used the "Snowball stemmer" to bring all the inflected words to their root form. For example, the words like "running","catching" and "catlike" as "run","catch" and "cat" respectively. This step is of great importance since it aids greatly in modeling the similarity between two words.

3. Fianlly, we wanted to eliminate words that were redundant and did not provide much useful information. Such words, also referred to as stop words, were removed in this step. Some of the common stop words are short function words such as "the","which","is","on" and "at" amongst many others.

4. Besides these, we also removed some punctuation marks from the dataset.

3 Feature Engineering

A very important step in this project was 'feature engineering'. None of the features were directly available to us, we had to extract them from the dataset. Also, we can't pass textual data to a model. So we needed a way via which we could convert text to numbers. Just doing this would also not suffice. We miss out on the interaction between search and product if we only pass their numerical data to any machine learning algorithm. We need features which could show how the product information and search query are related to each other. Below we describe our overall feature engineering process and how we grouped them before starting with modeling.

3.1 Text Processing Features

3.1.1 Document Term Matrix

This step involved converting our textual data into numbers. It is done is by converting a text into something called as a Document Term Matrix (DTM). Every observation becomes a document and the terms are the unique words that occur in them. If a document has a particular word, then that value is filled with '1' in the matrix else with '0'. We have replaced the '1' with the frequency of the term. This is the general representation of a DTM:

	Term 1	Term 2	Term 3	Term 4
Document 1	1	1	0	0
Document 2	0	1	0	1
Document 3	0	1	1	0
Document 4	1	0	1	0

Table 1: Document Term Matrix

The DTM matrix was calculated for the search query, product title and description and for a matrix comprising of search, title and description.

3.1.2 Term Frequency - Inverse Document Frequency (TF-IDF)

A TF-IDF value is calculated for every term of the document. It makes use of the DTM which was calculated in the previous step. The TF for a term t in a document d is calculated using the formula shown below:

$$TF(t,d) = 0.5 + 0.5.\frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}$$

where $f_{t,d}$ is the raw frequency of the term which can be looked up in the DTM and $f_{t',d}$ is the maximum frequency of any term in that document. The formula for IDF or Inverse Document Frequency is as follows:

$$IDF(t,d) = log\frac{N}{1 + |\{d \text{ in } D\} : t \in d|}$$

where N is the total number of documents or observations, the denominator means the total number of documents where the term t occurs. So the TF-IDF value for a term in a document is the product of the term frequency and inverse document frequency.

$$TFIDF(t,d) = TF(t,d) * IDF(t,d)$$

3.1.3 Singular Value Decomposition (SVD)

The result of converting a textual data to numbers was a high dimensional matrix. For product title and descriptions, the dimensions were almost a *hundred thousand*. We used Singular Value Decomposition (SVD) to find a low-dimensional representation of this high-dimensional matrix. SVD

reduces a large matrix M into its components U, Σ and V. The way it reduces the dimensionality is by setting the smallest of singular values to zero. If we set the d smallest singular values to zero, we can eliminate d rows of U and V. Thus we get a low-dimensional representation of M. We reduced the dimensions of each of the TF-IDF matrices obtained from the previous step to 50 dimensions.

At the end of this step, we had truncated TF-IDF matrices for each textual information. These form an important set of features for us.

3.2 Features extracted from the dataset

Since the dataset was entirely textual, we used this to extract a lot of features from the data. Following is the list of features that were obtained:

- **Length of query, title, description, brand:** These features gave us the *number of words in the search query, product title, product description* and *brand* respectively. This can be an important parameter in deciding the significance of the query. If the length of the query is long, it gives a lot of information about what kind of product is the user expecting. Hence the list of products that could be displayed as a search result to a longer query could be lot more significant than the results to a shorter query.

- **Query in title and description:** We needed a feature which could tell us *how many times does the entire query directly appear in the title and description* of the product. If the words of the search query appear entirely in the title or description, it could signify greater relevance than in cases where the query does not occur in the title or description.

- **Query last word in title and description:** The last word of any query gives a lot of information about what the user is asking for. This feature explains this intuition that is the *number of times the last word of the search query occurs in product title and description.* We also tried experimenting with boolean values for this feature. If the last word occurs atleast once, then make that feature '1' else '0'.

- **Query words in title, description and brand:** Apart from the last word, we also need to consider the occurrence of other words of the search query in title and description. This parameter incorporates the *total occurrence of any word of the search query in title and description.*

  We also tried to take ratio of these parameters and introduce them as features.

- **Ratio title:** $\frac{\text{Query words in title}}{\text{Length of query}}$
- **Ratio description:** $\frac{\text{Query words in description}}{\text{Length of query}}$
- **Ratio brand:** $\frac{\text{Query words in brand}}{\text{Length of query}}$

3.3 Similarity Features

For every observation, we had the product information vector and and a vector for the search query. We can measure the similarity between two vectors using similarity indices. We decided to use ***Cosine Similarity***. There is no specific reason so as to why we choose this over other similarity measuring formulae. Our main intention behind calculating this feature was to see what difference does a similarity coefficient make to a model. Our intuition was that it should make some difference to the model because it directly gives the interaction between the search and product vectors. Cosine similarity for two vectors A and B is given the formula:

$$\text{Cosine Similarity} = \frac{A.B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

where A_i and B_i are the components of the vectors A and B.

3.4 Feature Sets

Our feature extraction stage gave us many features to choose from. We grouped them in the following manner:

- **Feature Set 1:** Extracted Features + SVD of Search TF-IDF Matrix + SVD of Product Title and Description TF-IDF Matrix
- **Feature Set 2:** Extracted Features + SVD of Product Title, Description and Search Query TF-IDF Matrix
- **Feature Set 3:** Extracted Features + SVD of Search TF-IDF Matrix + SVD of Product Title and Description TF-IDF Matrix + Cosine Similarity
- **Feature Set 4:** Extracted Features + Cosine Similarity
- **Feature Set 5:** Extracted Features
- **Feature Set 6:** Cosine Similarity

4 Modeling and Learning

We were ready with our features and labels. We gave the various features sets mentioned above to some classic machine learning algorithms and tried to see which one of them performed well.

4.1 Support Vector Regression (SVR)

The SVM algorithm can be applied to classification as well as regression problems. The idea is to learn a non-linear function by a linear learning machine which maps the data to a kernel feature space. SVR maps the input x into an m-dimensional feature space using some kind of mapping.

SVR uses a loss function called as ϵ-insensitive loss function. It is also called as the margin of tolerance. Support vector algorithms are used mostly used with kernels. We decided to go ahead with a linear and radial kernel. Our intuition was that a linear kernel will perform well. Mainly because a linear kernel works well when there are a lot of features. Another advantage is that it is much faster. Just for a comparison purpose we also tried with a radial kernel. We had to use a truncated dataset (2K) for training a SVM model, as our laptops were not capable of performing such heavy processing. We tried with the full dataset (30K), but the execution never completed. Execution was done using the `svm` function of the `e1071` package of `R`.

4.2 Random Forest Regression

Random forest in general performs well for various regression and classification. It builds a number of decision trees on the bootstrapped samples of the training data. Along with this, when it is time to split a tree, it considers a random sample of m predictors as possible candidates. This helps in reducing the overall variance. By not allowing the tree to choose from all possible predictors, it restricts it to only consider the randomly selected ones. Hence every tree looks different and need not choose the strong predictors everytime for splitting the node of a tree. It was interesting for us to know how does the random forest model perform in the textual domain. Since we had a lot of varied features, the bagging and random selection of features approach could help in making better predictions. We used the `randomForest` function from the `randomForest` package of R. We set the number of trees as 200 and number of features to be selected during each split as 10.

4.3 Neural Network Regression

Neural networks or perceptrons have been giving great performances for a wide variety of applications. One reason for their good results is the back propagation algorithm. But the downside is that is makes them computationally expensive. It took close to four hours to run each model on our machines. The implementation was achieved using `neuralnet` library and function in `R`. Since there is no thumb rule to decide the number of layers and neurons, we went ahead with the default values of the function. The default was one hidden layer and the logistic function for smoothing the result.

4.4 XGBoost Regression

XGBoost is a very powerful learning algorithm which has helped a lot of people win various data science competitions on Kaggle. They go a step ahead of random forests by incorporating boosting. Apart from the boosting part, it also adds a regularization term. The learning becomes an additive process. The new tree learns from the mistakes of the previous tree. It does so by minimizing the residual error. Hence it becomes a gradient descent problem. Apart from this, the regularization term helps prevent the tree from over fitting. The library `xgboost` and its function `xgboost` were used for implementation of this algorithm.

5 Results and Analysis

We trained every model with six different features sets. The training was done on 70% of the dataset. We then tested every model on the remaining 30%, which formed our testing data. The table shows us the RMSE values for the testing data. (FS stands for feature set).

	FS1	FS2	FS3	FS4	FS5	FS6
SVR (Linear)	0.495	0.497	0.496	0.496	0.497	0.532
SVR (Radial)	0.484	0.497	0.488	0.491	0.494	0.533
Random Forest Regression	0.466	0.480	0.479	0.489	0.494	0.620
XGBoost	0.312	0.319	0.474	0.488	0.320	0.526
Neural Network Regression	0.479	0.485	0.479	0.487	0.486	0.531

Table 2: RMSE Values

This is our analysis from the results we obtained:

- We had expected the support vector regression wih a linear kernel to perform pretty well. But the RMSE values weren't that great as compared to other models. It is a difficult to reason this out. One reason we could think of is that the data needs to be linearly separable for SVM to do well. We had two textual data sets with high interaction between them. This could have made the data inseparable.

- Neural network seems to have done well than SVR on some of the feature sets. For the feature set 6, which had only cosine similarity, neural network did the worst. Just a single feature seems to be insufficient for the model to do well.

- The regression trees turned out to be the best performers for us. Random forest did well as compared to all other non-tree based methods. XGBoost seems to have out-performed other methods. It gave us the lowest RMSE values. Like many other people on Kaggle, XGBoost is the method we choose for this problem. We can conclude that a bagging and boosting methodology can do wonders as compared to other methods.

- The RMSE values also vary across feature sets. Feature sets three, four and six which had cosine similarity did decently well. Except XGBoost, the RMSE values for three and four are lower as compared to other feature sets. This means that adding a similarity feature worked. The reason for XGBoost to do bad with the cosine similarity features seems difficult to reason out. Feature set 6, which had only cosine similarity as the feature, did not do well at all. This means that just a similarity measure is not enough for the model to learn.

6 Conclusion

Currently, e-commerce marketplaces like Home Depot employ human raters to analyze and evaluate the search algorithms that their organization uses. However, owing to time constraints

and lack of resources, such a technique is not feasible in the long run. This project devises a solution to this problem by developing a model that predicts the relevance of a search result to the search query typed in by the user on its own by the use of several feature selection techniques and machine learning algorithms. With the appropriate selection of features and models to fit the data, the need for human intervention while performing the analysis of search algorithms can be completely eliminated. This could largely help the e-commerce industries in understanding how well their search algorithm is performing while also utilizing their resources in a much more efficient manner.

7 Future Work

In this project, we have tried to explore the effect of cosine similarity measure on our model. However, we could also try to use similarity measures of a similar flavor such as Jaccard indices and Dice coefficents. It would be interesting to see how our model performs for each of these measures individually as well as on a combination of the measures.

Moreover, since we observed that selecting the right features is crucial, especially in applications dealing with textual data, we could try to implement concepts like query expansion and extraction of n-grams and bi-grams from the data.

Finally, since the data that was provided to us was extracted real-time from the Home Depot website, it consisted of quite a lot of spelling mistakes and redundant words. According to us, the efficiency of the project can be further improved by applying much more sophisticated text preprocessing techniques such as spelling corrections and synonym replacement.

8 Acknowledgments

We would like to thank Prof. Chad Hazlett for all the expert guidance he provided us throughout this quarter. We gained an ample amount of practical knowledge and learned many different machine learning algorithms while working on this project. We are sure the skills we have acquired throughout this course will take us a long way in the field of statistics and data science.

References

[1] https://xgboost.readthedocs.org/en/latest/R-package/xgboostPresentation.html [Online]

[2] https://www.kaggle.com/c/home-depot-product-search-relevance [Online]

[3] https://pypi.python.org/pypi/nltk [Online]

[4] http://scikit-learn.org/stable/modules/metrics.html [Online]

[5] https://www.kaggle.com/c/crowdflower-search-relevance/forums/t/15186/1st-place-winner-solution-chenglong-chen/87205#post87205 [Online]