**FMIS 3295 Data Mining**

**Spring 2016**

**Submission: April 28, 2016**

**Project #3. Status Report**

**Home Depot Product Search Relevance**

**Group #3**

**David Hautamaki**

**Yoonwoo Oh**

**Erik Saice**

**Zach Scott**

## **Table of contents**

## **Table of Contents…………………………………………....………………………….……1**

## **Executive Summary………………………………………………………………………….2**

**GitHub Screenshot………………………………………….……………………………….3**

## **Data Understanding...……………………………………..………………...……………...5**

## **Data Preparation….……...………………………………..………………………………....7**

## **Descriptive Analytics...…..……………………………...……………………………….....11**

## **Modeling and Predictive Analytics…………………..…………………………......…....19**

## **Evaluation……………………………………...…………….….………………...….………30**

**Prescriptive Analytics……………………………………………………………….………32**

**Comparisons with documented results…………….…………………………………....34**

**Deployment………………………………………………….…………………………….…..36**

**Executive Summary**

When our team was determining what kind of project we wanted to work on, we identified that we wanted it to be relevant to the some of the content we’d learned in class. At the time, we’d been discussing text mining in class. We came across the Home Depot Product Search Relevance competition through Kaggle.com and were immediately interested in it. Home Depot was looking for help in improving their customers’ shopping experience by developing a model that could accurately predict the relevance of search results. These customers, be it professionals or “weekend warriors” rely on Home Depot as an authority in home improvement products and their ability to find what they need within their first web search is critical. We understood the problem and identified the solution as related to text mining, which was exciting to us as that is what we’d been learning in class.

Home Depot uses search relevancy to gauge how quickly they can get customers to the right products. The current method of evaluating the impact of potential changes to their search algorithms is done by human raters. This is what we identified as our business analytics problem. By creating a model that can accurately predict the relevance score between a customer’s search compared to what they’re actually searching for we can help Home Depot minimize or completely eliminate the human input in search relevance evaluation.

We were provided with a training data set and a testing data set. These contained product IDs, product descriptions, search terms, and finally relevance scores. The relevance score is a number between 1 and 3 (may contain decimal) with 1 being not relevant and 3 being highly relevant. The important detail we recognized was that this task is about intent, and not word matching. We would be dealing with supervised learning with this data.

Our original plan was to address the problem by providing the best model we could develop. Being this was a supervised prediction, the model would predict the lowest root mean squared error (RMSE). We’d planned to utilize all of our tools i.e. R, Weka, Tableau, and Excel. We initially planned to start with Weka, but formatting our data to be accepted was a struggle. While creating our .arff file, we noticed that the text file was refusing to put quotation marks around the strings from the cells in excel. Luckily, we found out the Excel has a concatenate function within it so we were able to format our .arff file the correct way.

The actual practice of applying a model to the Home Depot data set was harder than originally thought. We hoped that our model would be similar to class models that we had worked with before. However Weka continually rejected our data being fed into models regardless of the format so we decided to try something else. From there we moved to RStudio to create our model. Through the use of online aids, we were able to essentially do everything we wanted to in Weka with RStudio. Suggested scripts and libraries were a helpful starting places and were quickly adapted to create the models we hoped for. In R, we were able to create a model that fared marginally compared to some of the higher positioned models on the leaderboards on Kaggle.

**GitHub Screenshot**

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## **Data Understanding**

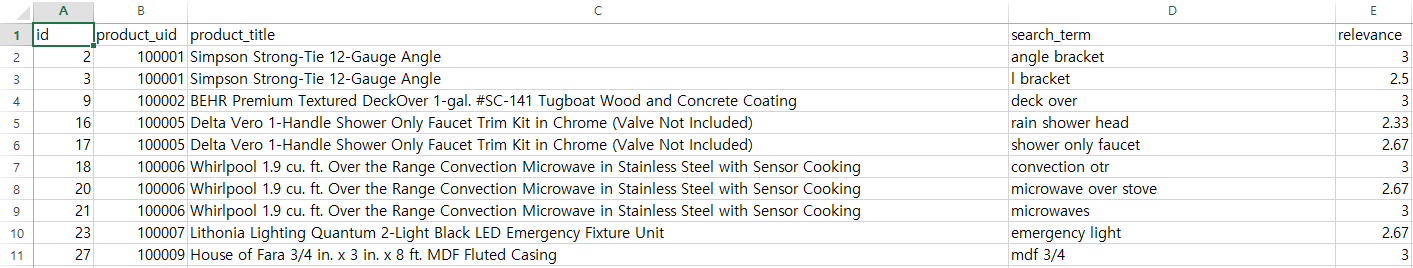
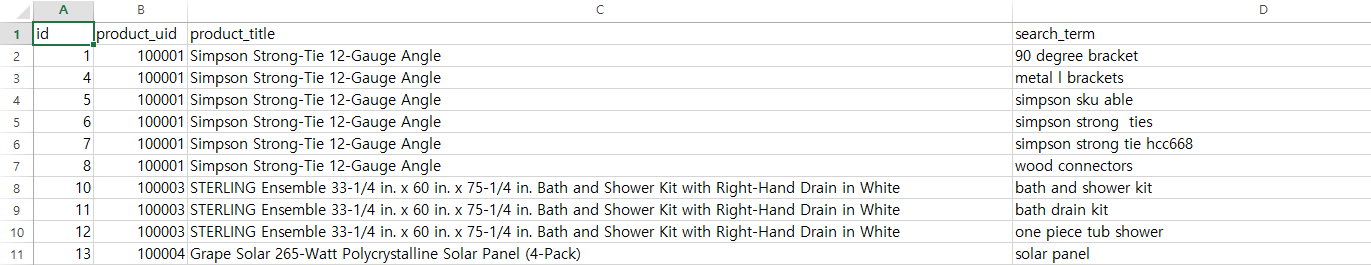
Data URL: <https://www.kaggle.com/c/home-depot-product-search-relevance/data>

The Home Depot Product Search Relevance challenge came with a training file which consisted of 3 relevant columns. Their headers for those three columns was the product name, the search term, and the relevance of that search term on a scale of 1 to 3 to find that product. Each pairing from the search term and item was coded by at least 3 human coders whose scores are averaged. These coders did not have access to product descriptions but did have access to images to the products.

The data itself is primarily strings with the relevancy score being the only numeric values. All other numbers in the datasets are treated as string. Since the product names and search terms are strings, we will be able to delimit them using spaces to break the strings into individual tokens. By doing this, we can assign value to each individual token instead of the entire search string. Finally, the relevance score will allow us to train our model to predict the strength of search words. Luckily, the data that is supplied to us is an average of at least three coders. By averaging the coders, we limit the amount of human error within our data set.

Our plan is by using the training data to create a model, we can increase the search relevances of keywords so the customers can get to the correct products that they are searching for. By doing this correctly, the keywords will have a stronger pull to certain items while also not including vague items that poorly fit the keywords.

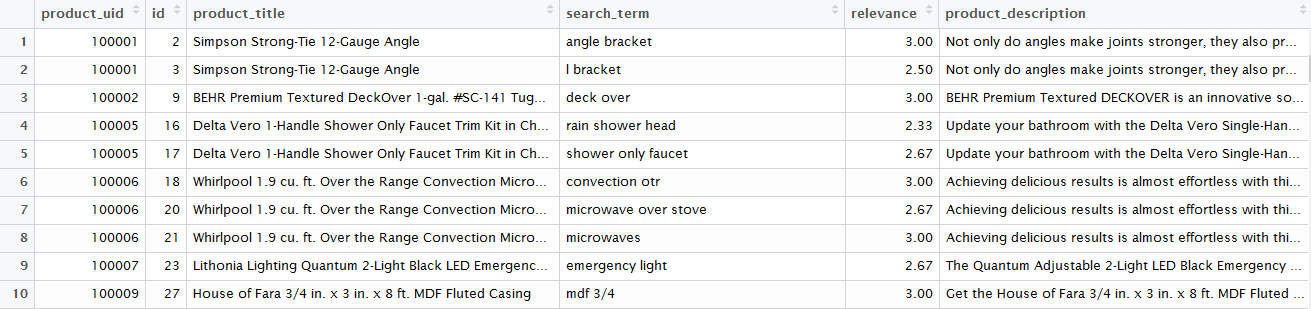
|  |  |
| --- | --- |
| Id | A unique Id field which represents a (search\_term, product\_uid) pair |
| Product\_uid | An id for the products |
| Product\_Title | The product title |
| Product\_Description | The text description of the product (may contain HTML content) |
| Search\_term | The search query |
| Relevance | The average of the relevance ratings for a given id |
| Name | An attribute name |
| Value | The attribute's value |

*Figure 1): Top 10 rows from Train.csv dataset*

*Figure 2): Top 10 rows from Test.csv dataset*

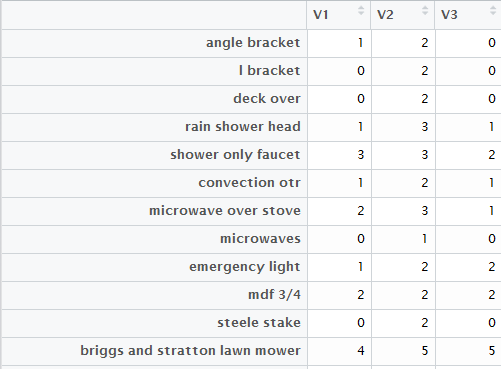
**Data Preparation**

The data in this project was obtained through Kaggle and provided by Home Depot. The data provided by Home Depot is of a very good quality. In our test, train, and product description datasets, we are presented with no missing values and all textual/string type data except for the numeric relevancy score. The mainly dirty data that we are forced to deal with are user searches that are misspelled and/or only contain part of product name or description. Product descriptions and product names do not contain spelling errors.

The first step in our data preparation is to collect the relevant data from our sources and load them into RStudio using data frames to contain the data. We want our training and test files to correlate to the product descriptions so, through a function in the script, we merge the product descriptions file with our test and training files through the product\_uid. Think of it like a join in a database. 

*Figure 1): Train data merged with product descriptions*

The second step involves further prepping and cleaning the data. This seems to really be where teams within the Kaggle competition were finding success in bring down the rmse. To start with this we use a function to break each the strings into tokens for comparison. Basically, we’re taking phrases and separating each word using space as the delimiter, and then storing them in vectors for later comparison. Words aren’t the only things that we’re looking to identify. Within the product descriptions and search terms, often numbers are used to describe the product; i.e. 90 degree angle bracket. We want the model to use numbers in its analysis rather than ignore it because users may input SKUs or 90 in reference to an angle. Put simply, product descriptions, product names, and user searches are broken into individual words and stored in vectors for each row. User searches are compared with product names and descriptions and number of words in the search term are counted. Each time there is a matching term, a count gets added to a new column. The figure below shows the user search terms, column V1 is the number of matches with product name, column V2 is the number of search terms, column V3 is the number of matches with product description. Figure 2 shows the data before it is merged with the train dataset. Figure 3 shows the data after it is merged with the train dataset.

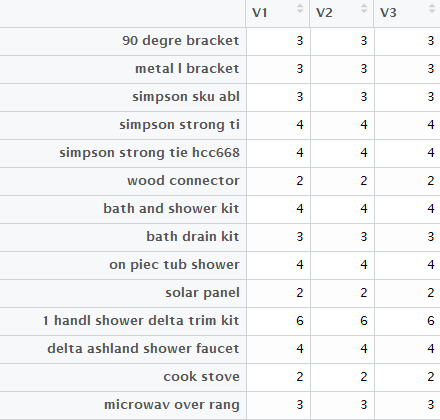


*Figure 2): Table containing counts of matching terms in train dataset before merge*

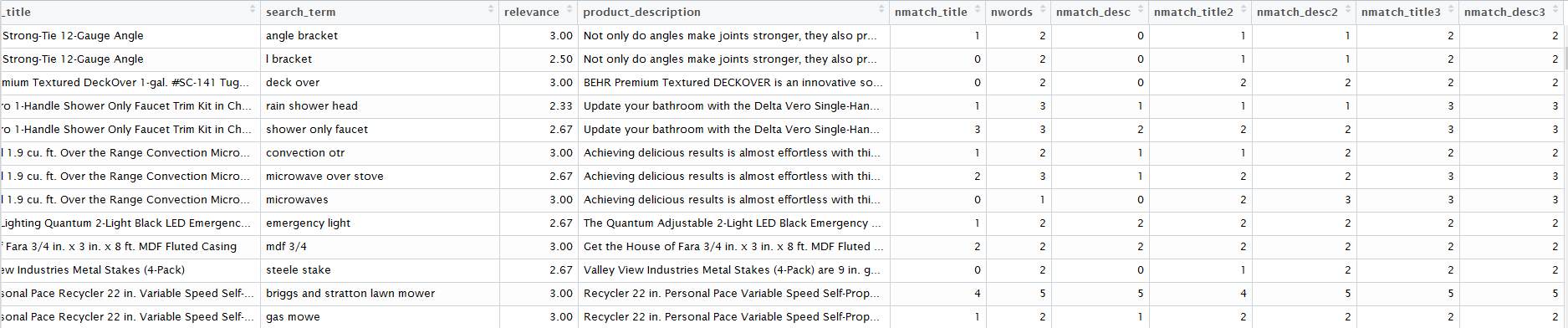


*Figure 3): Table containing counts of matching terms after merge*

The next step in our model runs our stemming portion, word, and number comparison of stemmed data. We want our tokens stemmed so that we have a baseline among words. Stemming is the process of reducing derived words to their most basic form, the “stem”. Think “running” to “run” or “instructional” to “instruct”, we don’t want these words to be worth two different values. The function within the model uses functions within a packaged we installed called SnowballC which uses the Porter stemming algorithm. Stemming will allow the model to get a more accurate comparison of words because if their more basic forms are the same then that would increase the relevancy score. The figures below gives examples. Figure 4 shows the data before merge and Figure 5 shows after the merge with the train set. Once again V1 will be counts for matches with product name, V2 is the number of search terms, V3 is the counts for matches with product description.



*Figure 4):Stemmed terms and counts for matches before merge*

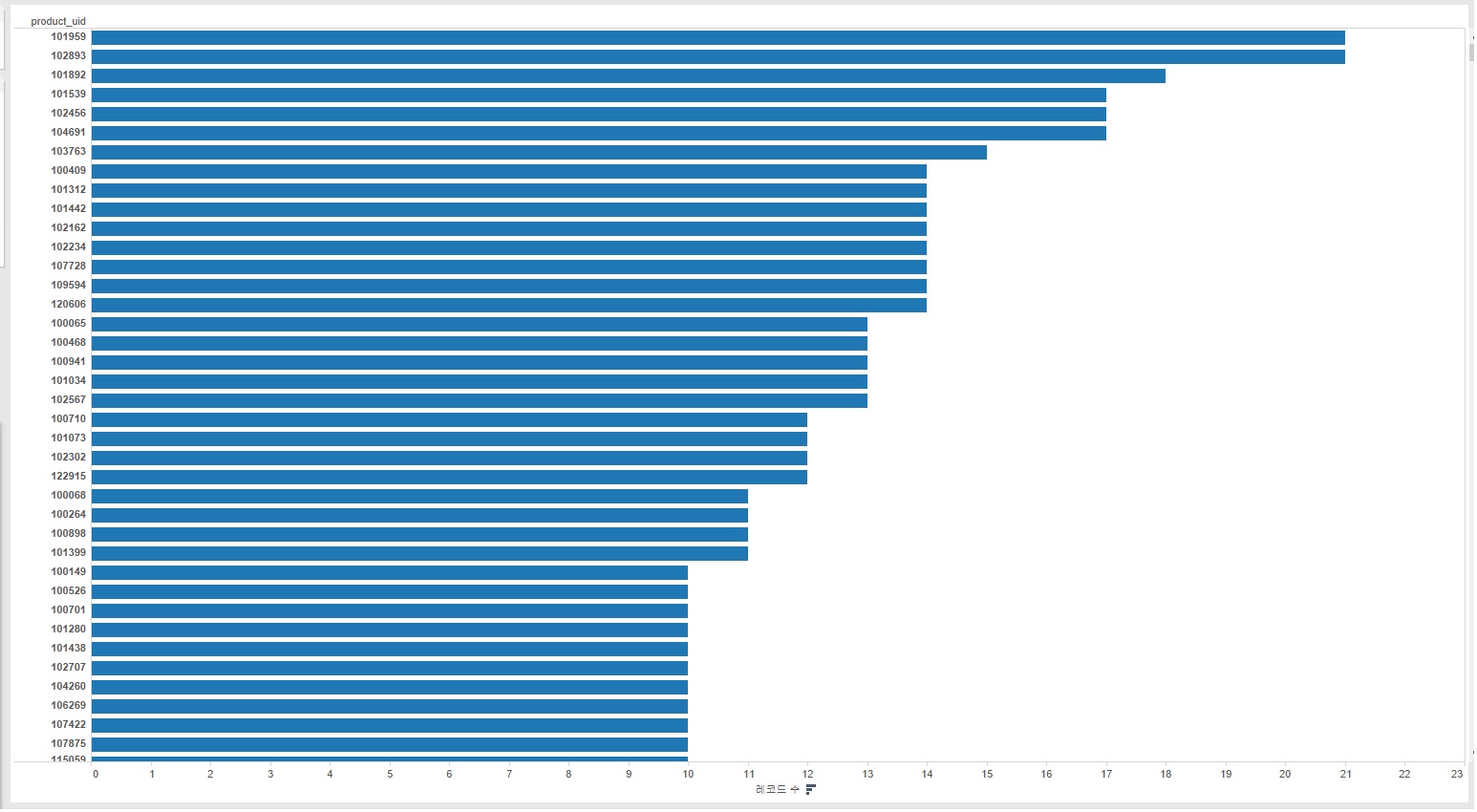


*Figure 5):Stemmed terms and counts for matches after merge*

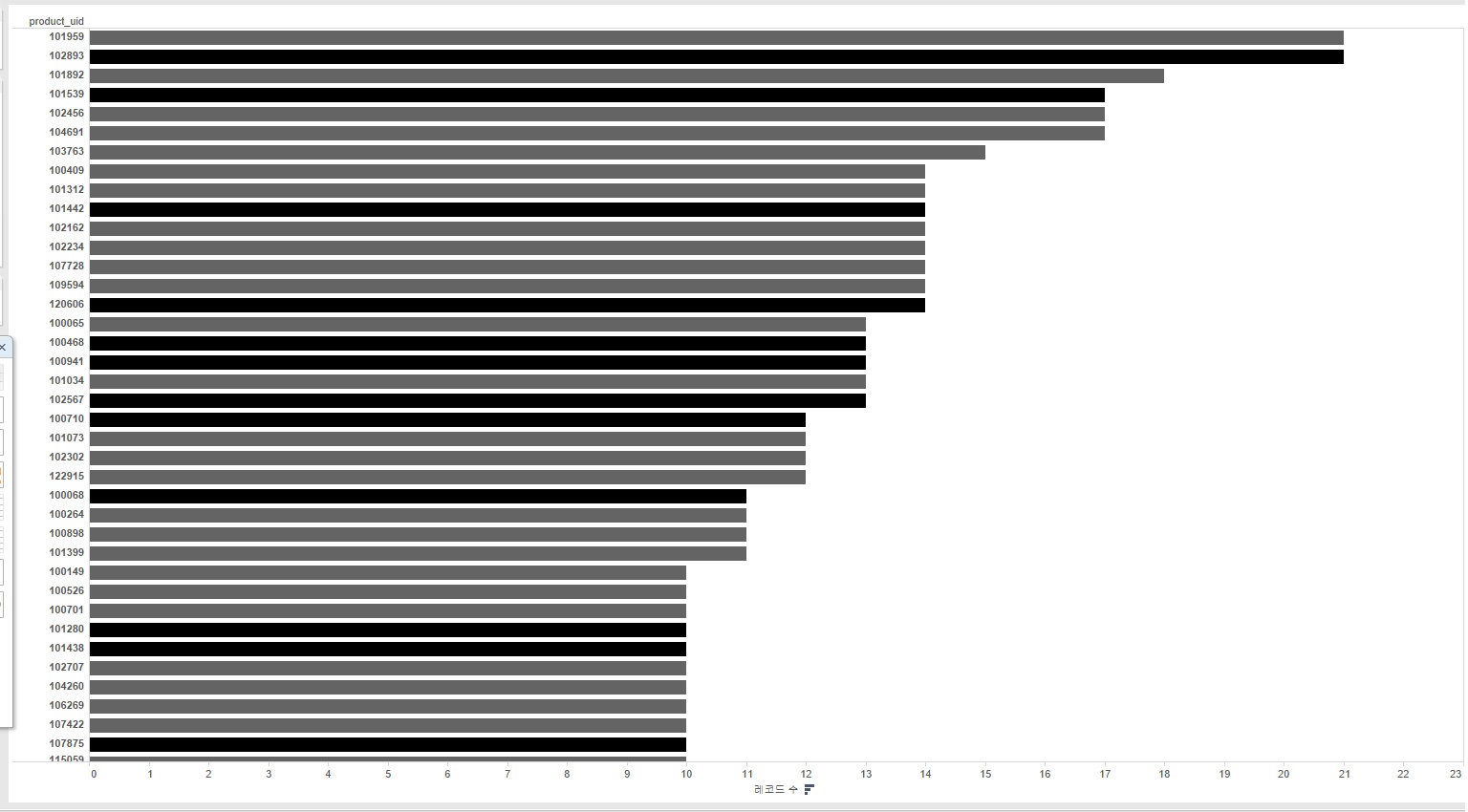
These are the final steps before the match term counts are fed into the model. The steps taken in combining and cleaning our data has definitely made the information make more sense and will allow for a more accurate prediction due to better matching of user search terms with product names and descriptions. To recap, user search terms are broken into individual terms, these terms are then compared to product name and description, then counts for matching terms are added to new columns. This process is then repeated for stemming.

**Descriptive Analytics**

Home Depot has provided training and test data sets for participants to work on. In order to understand what has happened and is currently going on, we decided to use Tableau and RStudio to analyze our data. Tableau’s function of visualization and powerful charting tools will be most efficient in showing what the numbers meant overall and easy to understand while RStudio was used to generate word clouds.



*Figure 1): Train.csv - Product search popularity top 40 instances*

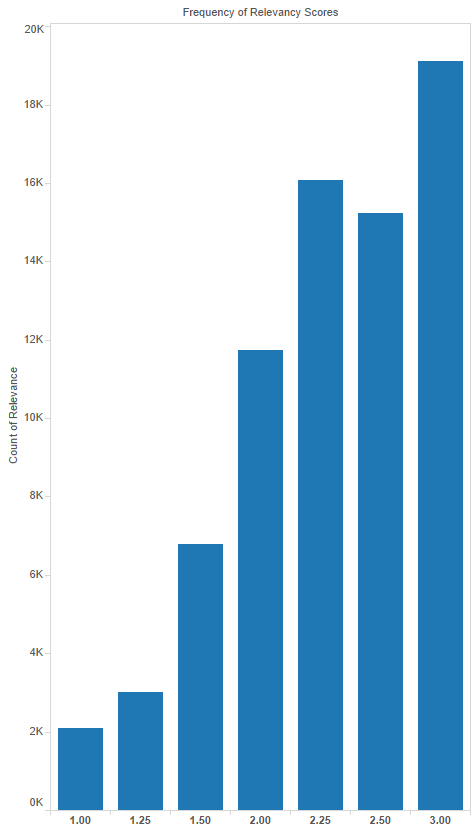


*Figure 2): Train.csv - Product search popularity top 40 instances with relevance score*

Here are figures that is based on train.csv. These diagram shows how each product is searched frequently. We first simply look at the data and wanted to know which product the customers were searching the most. We had our product ID on y-axis and count of how much each product ID appeared on x-axis. The result was came out was like figure 1. However, we also wanted to know if customers were actually getting what they were looking for. By applying relevance score as color to distinguish the graphs, we were able to figure out some interesting factors.

According to figure 2, it shows that certain products on top have very high search popularity. At the same time, the darkness of the color shows relevance score. For instance, the top two lines indicates that those two products are what people are searching for the most. The first product “*Pressure-Treated Timber #2 Southern Yellow Pine (Common: 4 in. x 4 in. x 8 ft.; Actual: 3.56 in. x 3.56 in. x 96 in.)*” with product ID of 101959 had total count of 21 with relevance score of 2.06 out of 3.00. The second product “*Lithonia Lighting All Season 4 ft. 2-Light Grey T8 Strip Fluorescent Shop Light*“ with product ID of 102893 also had total count of 21, but with relevance score of 2.52 out of 3.00. Looking at these numbers, we can tell that both products are popular to customers but the first product lacks relevance score. It’s fairly clear that we have problem with the first product’s search terminology, since the relevancy score is low meanwhile the product is quite popular.

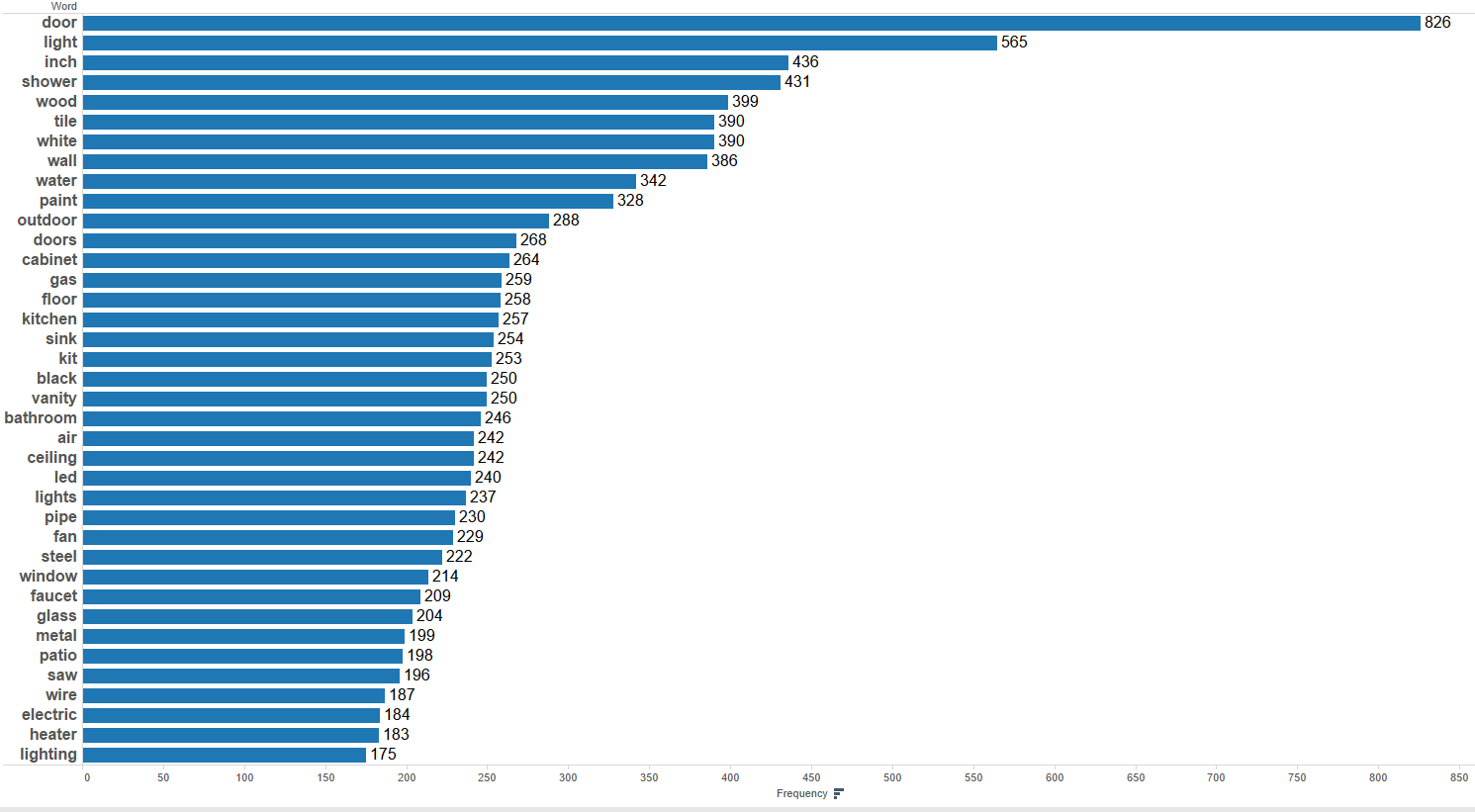
As we could seen from previous example, figure 2 tells a lot of information about the popularity, and average score the product is receiving. Looking through the diagram, we can identify those products with low relevancy scores require attention. Especially, if their popularity is high, urgent adjustment is required for higher customer satisfaction. This is closely related to our goal of helping Home Depots predicting the relevance score. It will help Home Depots to focus on those item with low relevance score but with high popularity. This can bring up the efficiency of the work.



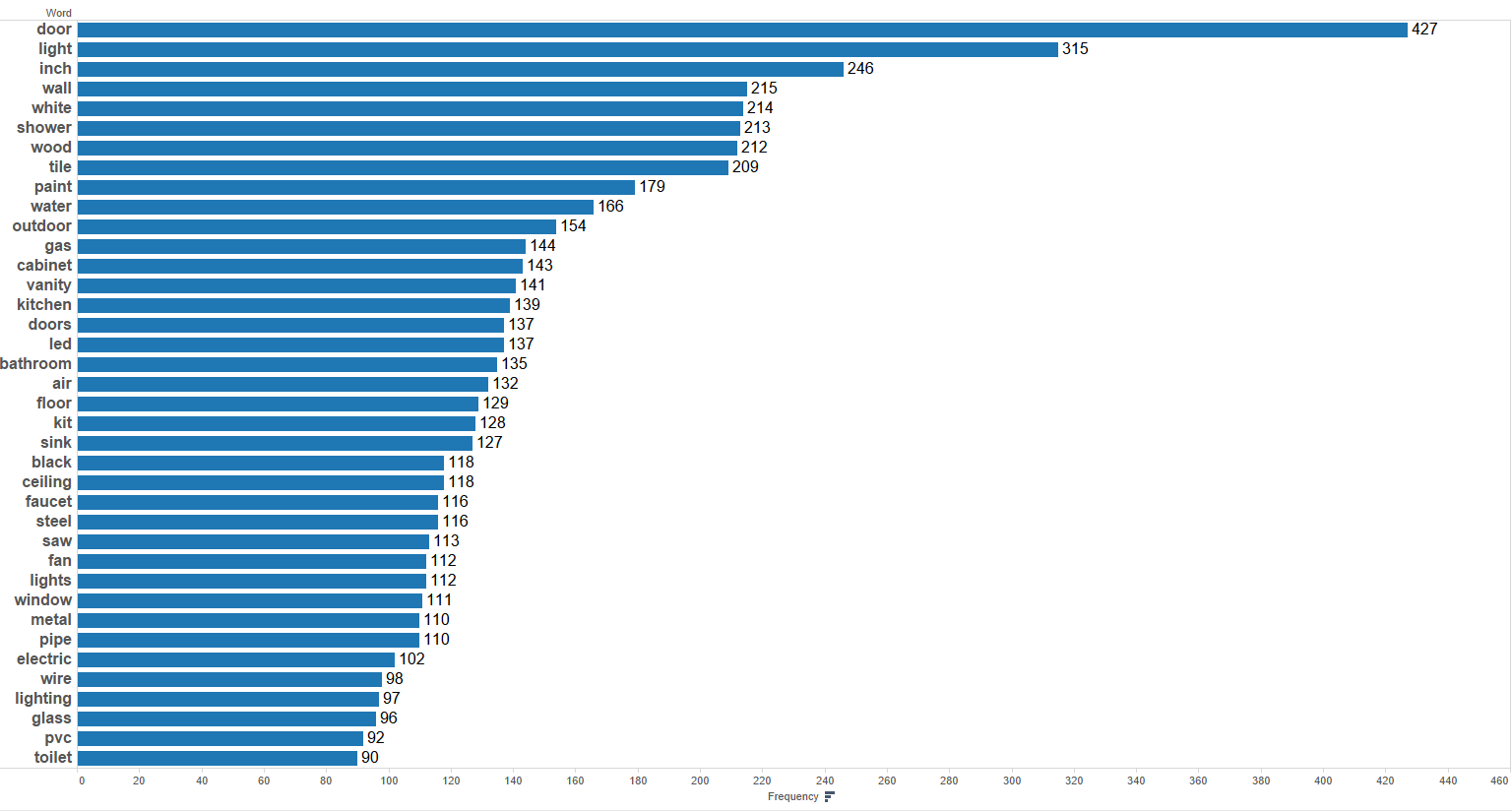
*Figure 3): Train.csv - Histogram based on relevance score*

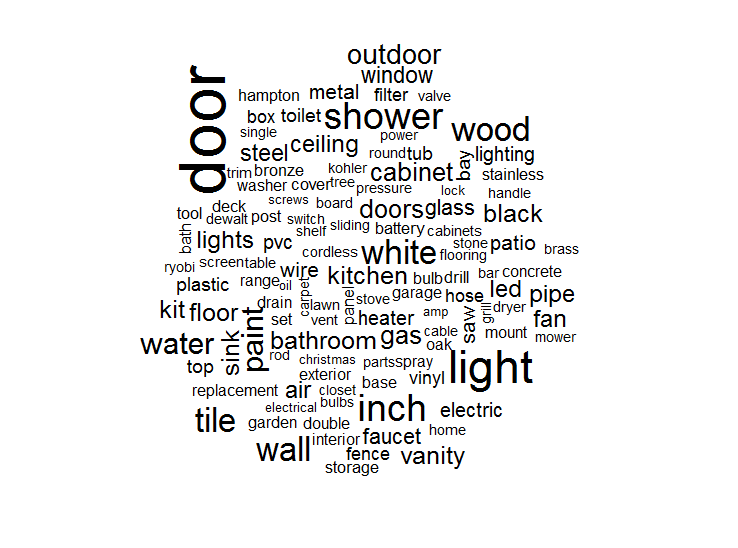
Figure 3 is a histogram of the relevance score from training set. After looking at how products had different relevance scores, we wanted to grasp an idea about how most of the products were scored in general. For this histogram, we have relevance score on x-axis and count of those values on y-axis. Based on this histogram, most of the products were scored above 2.00. However, there were over 2,000 products with relevance score of 1.00. Although the training dataset from Home Depot contains more than 70,000 products, this suggests that there is a problem with their current search engine and can be improved.

Our next focus was on search terminology. We were interested in seeing what search terms and criteria customers were using to obtain their results. Data was pulled from the test and training data sets from the search\_term column after strings were broken into individual words.

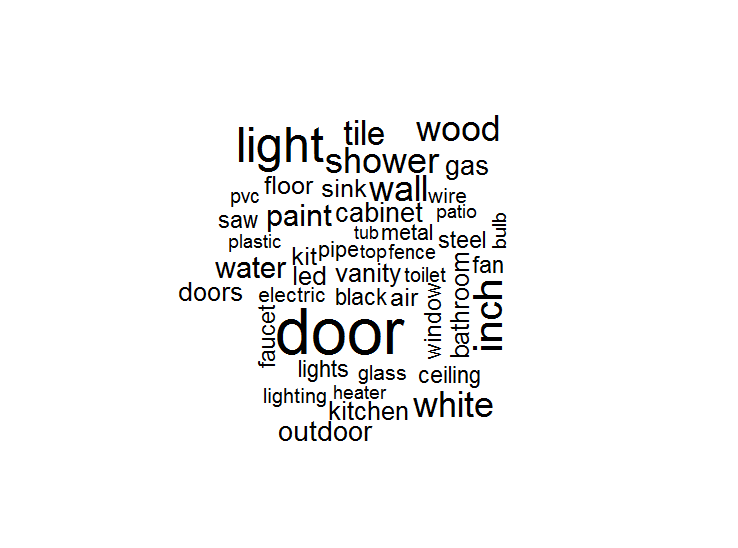


*Figure 4): Test.csv - Top 40 search terms*

*F**igure 5): Train.csv - Top 40 search term*s



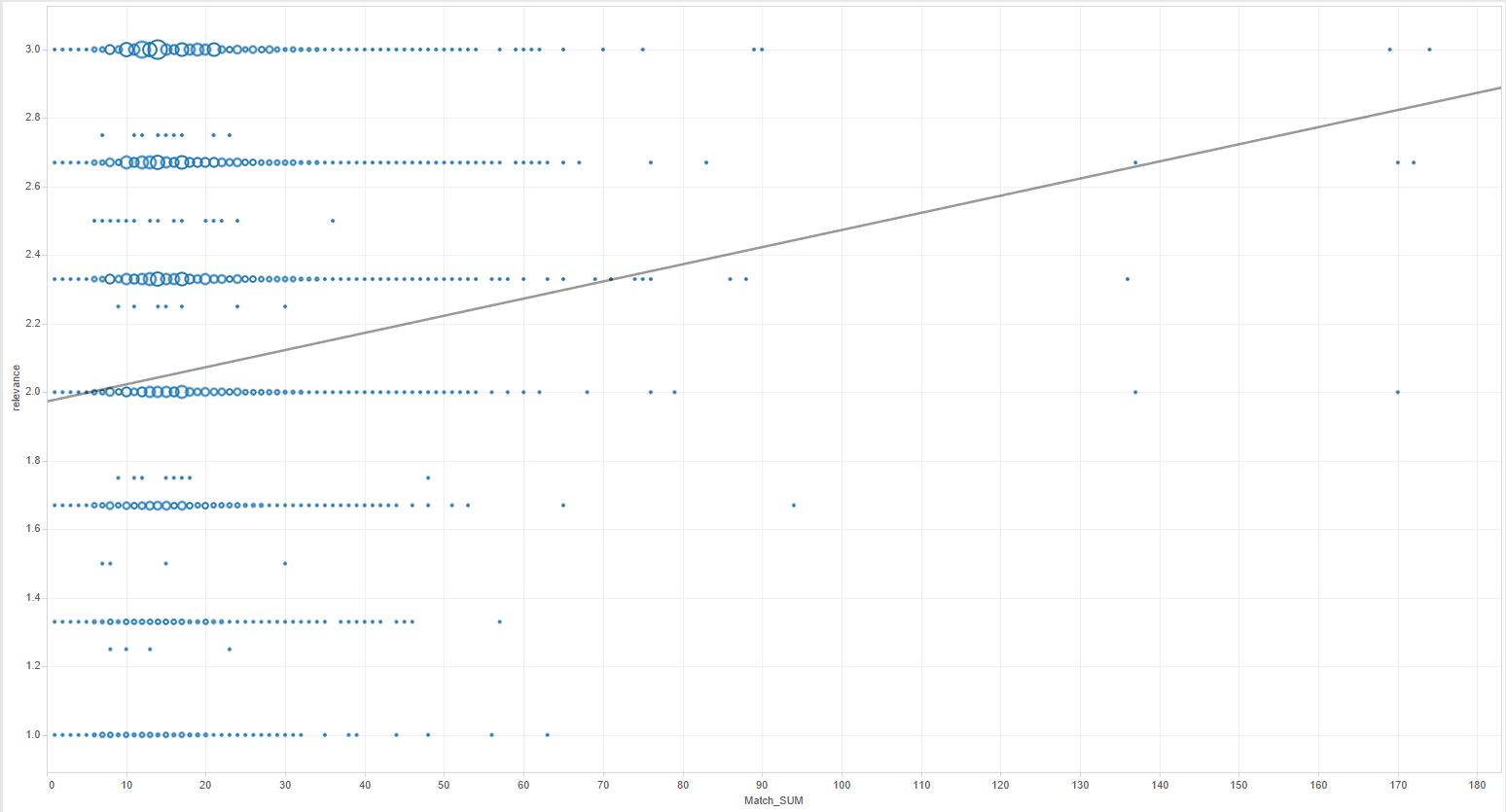
*Figure 6): Test.csv - Word cloud*



*Figure 7): Train.csv - Word cloud*

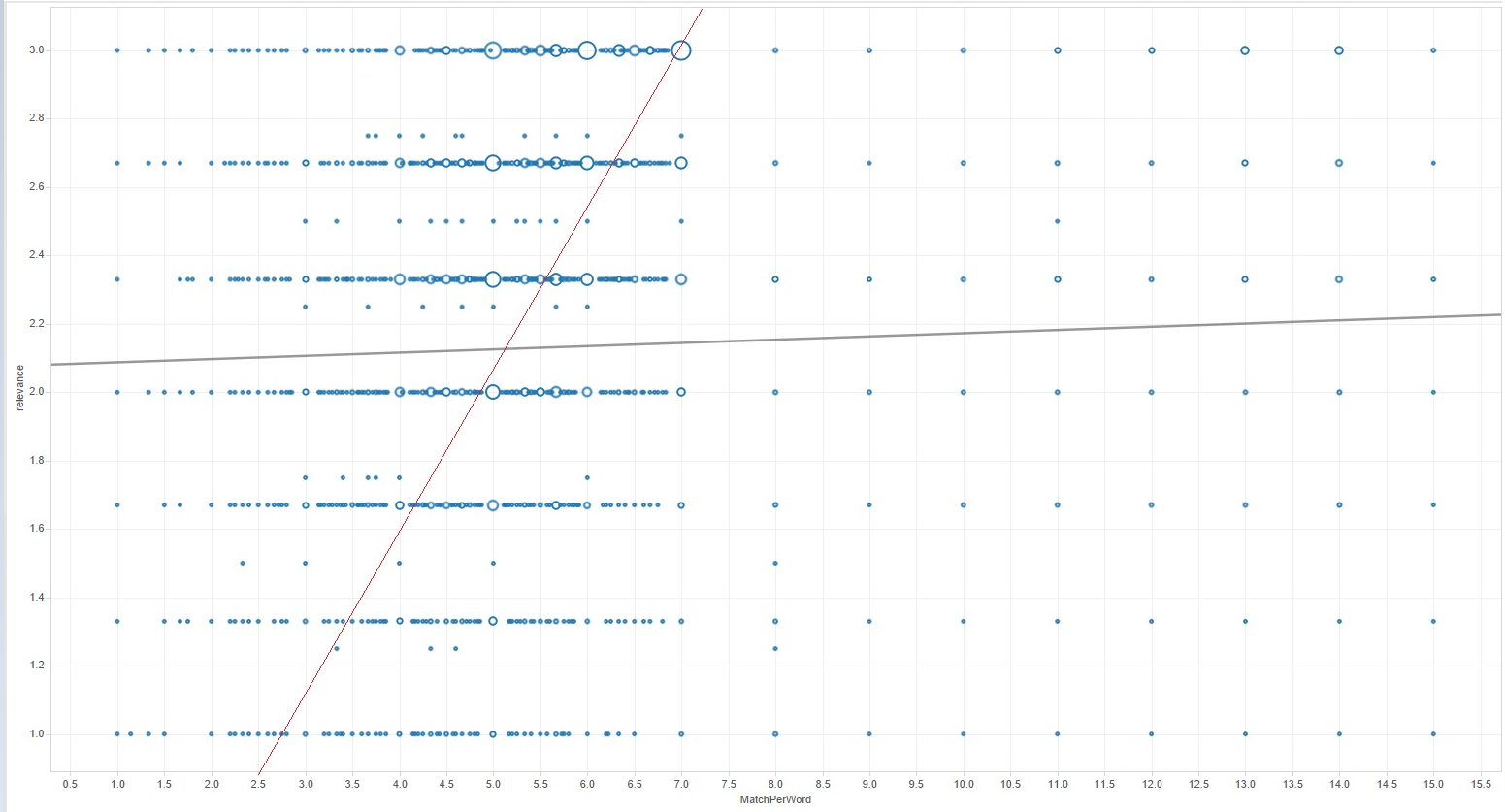
In figure 4 and 5 some of the commonalities between the datasets are made apparent. You can see that “door”, “light”, and “inch” all remain in the top 3 terms for both data sets; not surprising when dealing with a home improvement store. Figure 5 and 6 are word clouds to reinforce the barplots but with a more visually appealing representation of what words are more frequent. They too reiterate the top 3 search terms, with “door” being prominent in both clouds. Generating a frequent word list like this may be useful when trying to perform spell checking operations with a lexicon. Most frequent words may be added to the lexicon package to ensure they are considered when spell checking.

Additionally, after combining the Train.csv file with Description.csv file, we decided to see how many words were getting matched to product title, product description based on what customer types. In order to do so, we used 3 methods, starting with comparing the whole word, then comparing with only words, without any non-letter text, and finally comparing after stemming the word. After getting number of words that matches for title and description for each methods, we summed them and checked if there was any relationship with relevance score.



*Match Sum & Relevance relationship*

The graph above have sum of those words counted as x-axis and relevance score on y-axis. We also added the size filter based on the count for those sum. The trend line went very bad due to those extreme outliers, but if we carefully take a look at the graph, we can see how there is trend within the graph. A trend that as the Match Sum increases, Relevance increases as well. To make this even more precise, we changed our x-axis from sum of the counted number of matching words into average of those words.

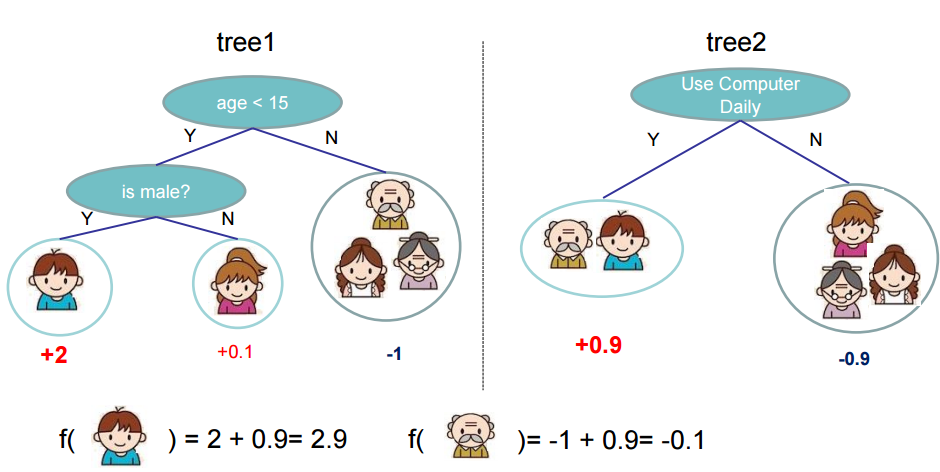
*Match Average & Relevance Relationship Deployment*

This time, the program failed to show the idealistic trend line because it counted the outliers as well. However, if we look at the graph and exclude those outliers on the rights, we can see the trend that we marked as red line. This was a better proof that a better match of words means a better relevance score. We realized how this finding will be very helpful in reaching our goal of helping Home Depots predicting the relevance score.

**Modeling and Predictive Analytics**

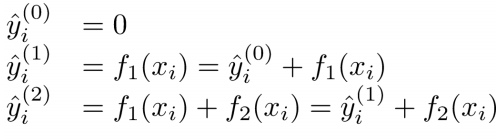
Home Depot is trying to analyze the relevancy of their website product search results based on user search input and what the user intended. This model will predict the relevancy of produced results given user search input and more refinement of the model will allow for a greater deal of accuracy. In business terms, if the most relevant items to a user’s search input and intent are displayed in results, they will then be more likely to purchase and continue using Home Depot’s website. By fine tuning the model Home Depot could apply it to their search result algorithm to provide their users with the best possible results.

Since the type of learning we are interested in doing is supervised (trying to predict our output variable), we set out to look for models that we could use to test our data provided by Home Depot. After searching for a bit, we noticed many users in the competition recommending the use of the xgb model which is short for extreme gradient boosting. Xgb is a type of classification and regression tree (CART) that uses regression tree ensembles to build the model. Basically, a bunch of weak learners are used to create a strong learner. See the image below for a tree ensemble.



(Source: xgbboost.readthedocs.org)

The prediction is then the sum of the scores predicted by each tree. The boosting part comes from additive training. For example, Y-hat is the predicted score. At training round 0, Y-hat is equal to zero. At training round 1, Y-hat is now equal to function 1. At training round 2, Y-hat is now equal to function 1 plus function 2. The process will continue on until the number of training rounds specified.

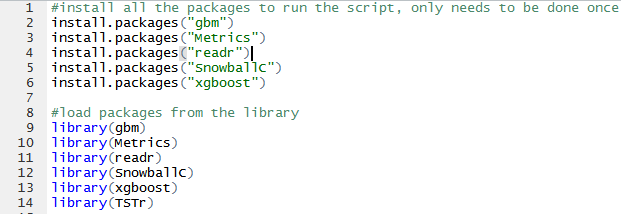


(Source: xgboost.readthedocs.org)

**Model Setup**

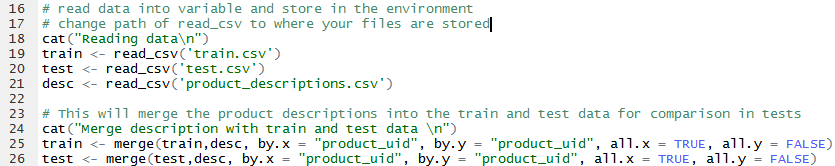
Our xgb model is currently being ran with scripts in RStudio. There are various objective functions that the model comes preloaded with, but we are using the linear regression objective function with root-mean-square error as the evaluation metric. Something like logistic regression wouldn’t work well since we are not dealing with a binary output. Below will provide an explanation on how the current iteration of the model is trained. We will also quickly reiterate how data is being transformed before being fed into the model. Here is how the model and script is setup:

The first step is to install and load the necessary packages to run the model and functions. The gbm and xgboost packages are used for running the model. Metrics is for computing the rmse. The readr package is a bit more efficient for loading and reading tabular data. Lastly, SnowballC handles stemming that is incorporated into data preparation functions.



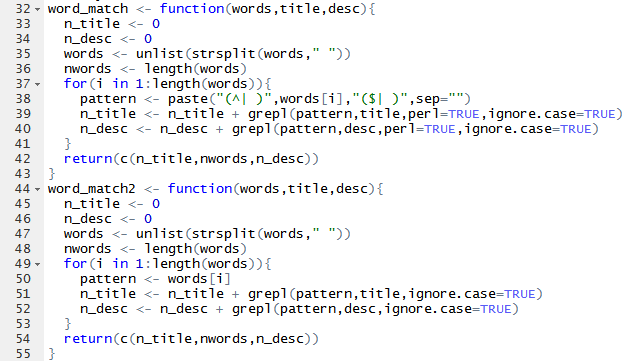
*Step 1): Packages are installed and loaded*

The next step involves merging our product descriptions data set with the test and train sets. User search needs to be compared with both product name and description and this will allow us to do it with one data frame.

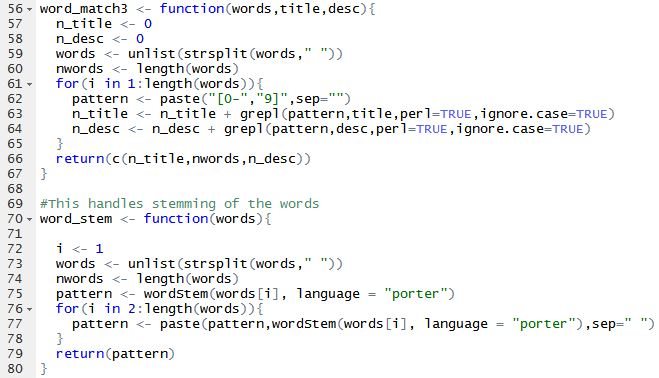


*Step 2): Product descriptions are merged with train and test sets*

Next we define the functions that will be used on the data. The functions will handle comparing the words with the product titles and descriptions after breaking the user search into individual terms and storing them in the “words” vector. Within the defined functions, the grepl function handles the comparisons and if it returns true, will add a count to the appropriate column. The terms in the words vector are used as the comparison pattern for the grepl functions. Functions are defined for pure textual terms, numeric values, and stemmed terms.

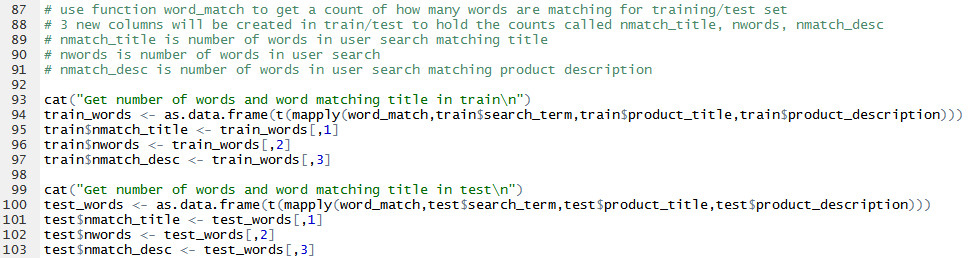


*Step 3a): Function definitions*

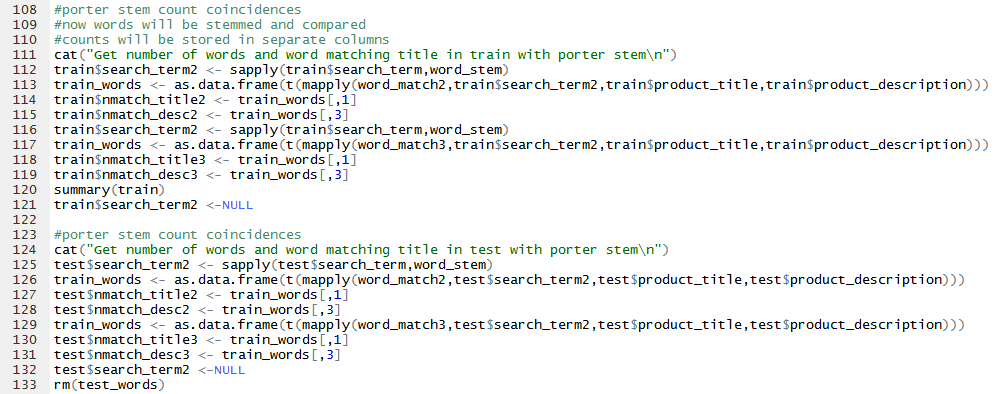


*Step 3b): Functions continued*

The next step is to actually use the functions to transform the data into the numeric counts of all the matches for each user search term with the product title and description. These are the counts that will be fed into the model to be used to predict relevancy scores. Search terms, product titles, and product descriptions from both test and train data sets are passed into the functions and 3 new columns are added to the test and train sets to contain the matching counts. This process will repeat again, but with stemming the terms and obtaining the matching counts for the stemmed terms.

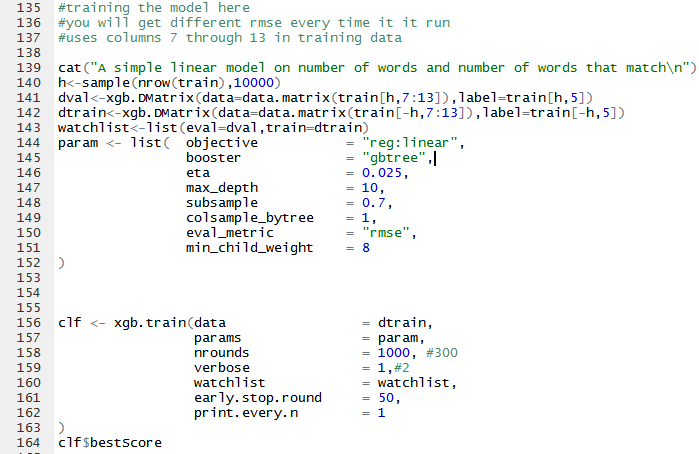


*Step 4a): Data is transformed to obtain counts of matching terms*



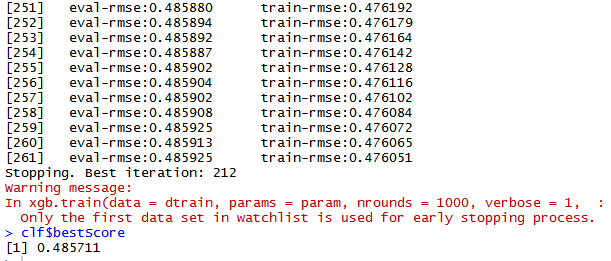
*Step 4b) Data is transformed to obtain counts of matching stemmed terms*

Lastly, we have the training model setup with parameters. The training model uses a sample of 10,000 instances from the train data set and rmse as the evaluation metric. All the rows containing the counts of matching terms are utilized (7 through 13) and the relevancy score is set as the class variable (column 5). The training model will utilize linear regression as its objective function to better handle the non-binary nature of the relevancy score. The eta parameter will control the learning rate of the tree and must be between 0 and 1. The lower eta value the more conservative the model will be and better at preventing overfitting; it will just take longer to compute and should be used with a higher number of training rounds. The model also takes into consideration how deep a tree may grow and the number of instances a child in a tree must contain. These will also help in creating a more conservative model to prevent overfitting. The model also contains parameters to exit training if the model continues to worsen after n iterations. Other parameters will be shown in the screenshot below.



*Step 5): Training model setup and parameters*

Lastly, the model will be trained with the training data. The result of every iteration will be printed to the screen until the number of rounds completes or the model terminates due to the early stop parameters. The best iteration will then be picked and the rmse score will be displayed.



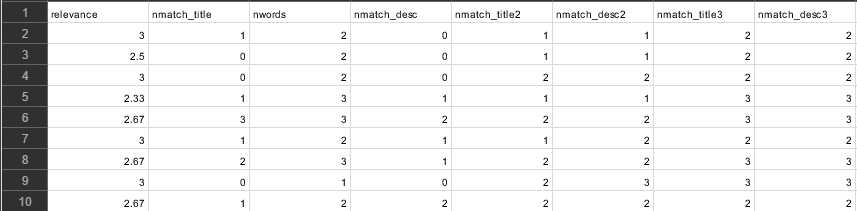
*Step 6): Running the training model*

Overall, we now have a base model that we can work with and change some of the parameters on to attempt and lower the rmse even further. For the next iteration we are looking at implementing a spell checking function into the initial round of word comparison before stemming to hopefully decrease the rmse and obtain a better model. Currently we have found another R package utilizing ternary search tree algorithms which may fulfill this purpose.

Expanded Modeling and Predictive Analytics

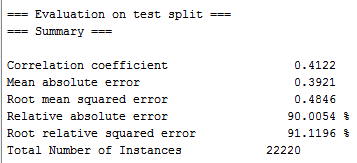
The first option that we attempted to use for expanding our xgb model was to explore the spelling correction option discussed previously. The idea was to automatically scrub the user search input for any misspellings, or unintended words, and provide possible alternatives to the user search input. These alternatives would then go through the same matching process involving product titles and descriptions. To accomplish this task we explored the use of ternary search trees using various spelling autocompletion algorithms. The general idea behind the algorithms is that user spelling error is usually limited to + or - one letter in a word, so autocompleting words or finding corrections is fairly efficient. However, while attempting to implement this method, we began running into issues with the R script and code. The code would not complete the checking of all terms and the bug appeared to be coming from within the package we were using. We decided to abandon further use of implementing the ternary search tree algorithms due to time constraints.

We decided to run another model in Weka using our data for some comparison. Initially we also had issues loading the data into Weka. The issue was resolved by removing single quotes, double quotes, and percent signs from the data set. The rest of the data was preprocessed in RStudio using the same steps as our xgb model. After importing into Weka, all unnecessary columns were removed leaving us with something like this:



Relevance was used as our class variable (what we’re trying to predict) and the other columns represent the same matches as they previously did with the xgb model.

The model we decided to use was Weka’s REPTree. There is not much documentation explaining REPTree, but Weka describes it as a fast decision tree learner that builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning. This would work well for a comparison since our xgb model is a type of tree. Same/similar parameters were set in Weka for the REPTree model. Max tree depth was set to 10, minimum number of instances in a leaf was set to 8, and the number of folds was set to 5. The REPTree was then ran using a 70% split on the loaded training test data and yielded the following results:



The REPTree, with same and similar parameters, produces a more consistent rmse than the xgb model. However, the xgb model has produced a rmse of 0.482, but can still fluctuate up to 0.488. Ultimately both of these models still have pros and cons. The pro for xgb is that we are more flexible to add additional parameters and functionality to model with R code and script. The pro with REPTree is the consistency of the score; there is no fluctuations in the evaluation metric every time the model is ran with same parameters. Because REPTree is in Weka, we are limited to what parameters we can use to try and refine the model. The biggest con for both models though, is that to get the most accurate results we will need to take significant extra steps which would also contribute to next steps the modeling would take.

One of these steps to improve the model would be to utilize n-gram algorithms. This would allow better matches of terms and provide some extra context to user searches. Instead of individual terms we could analyze phrases to see how well those were matched with search results. Secondly, the other step would be to incorporate spell check and/or search and replace. As we learned previously, this is not easy when you are a novice with R. You need to build corpuses representing all terms in the documents and then use NLP methods to try and determine user intent based on their input to select the correct word they meant to use or spell. Setting a model up after processing the data with some of these methods would allow for a much lower rmse score than our current models have produced. These were found to be some of the steps taken from previous winners of text mining competitions and have been outlined a bit in later sections.

**Evaluation**

For both the REPTree and XGBoost models we evaluated them based on the root mean square error. This will allow us to look at the differences between the population and sample predicted relevancy by the models and the actual relevancy scores defined by the training dataset. Home depot is also utilizing rmse as the competition evaluation metric. Exact figures for evaluation metrics produced by the models will be given below.

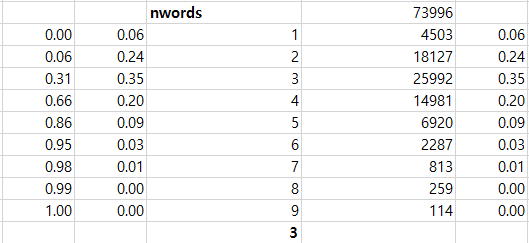
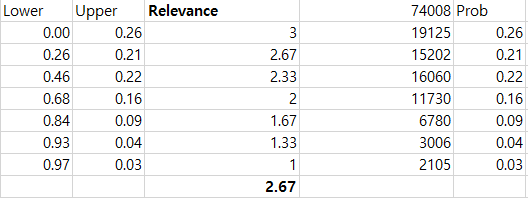
We setup both models with similar parameters where possible to enable a better comparison of the two. Overfitting our decision trees was taken into consideration in a few different ways. The first is by using subsampling. Instead of using 100% of the data, and letting the model try to fit more perfectly, we use 70% random subsamples of the data for training purposes Another parameter used to prevent overfitting is the maximum tree depth. We utilized a value of 10 in both our models. This will prevent the models from creating massive trees and result in a more conservative model. Max tree depth then leads into the minimum child weights. This is just the minimum required instances to be in each node. Both of our models used 8 as the parameter value. Once again, this will result in a more conservative model and help prevent overfitting. Lastly, one parameter available to us on the XGB model is the learning rate of the model. The value needs to be 0 < learning rate < 1 where closer to zero will make the model more robust to overfitting, but take longer to compute. In our case, we utilize the value of 0.025 for the XGB model.

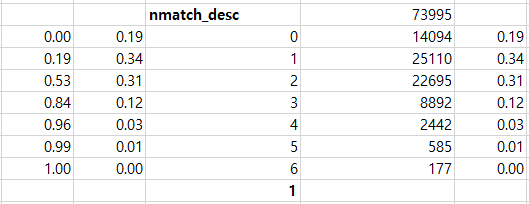
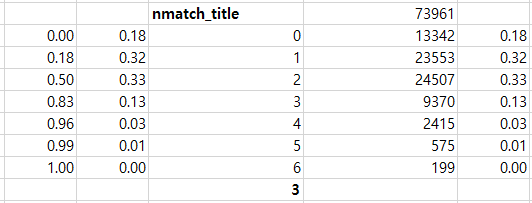
Since our models are dealing with linear regression, we do not have metrics like AUC. All metrics for the REPTree given by Weka are listed in the screenshot in the previous section. However, we have included a table to compare some of the metrics easily available to both models. This was another downside to using RStudio over Weka. Weka will provide you with relevant metrics, but RStudio requires you to write R script for any extra metrics you want the model to provide you. Here is the table:

|  |  |  |
| --- | --- | --- |
|  | XGBoost | REPTree |
| Mean Absolute Error | 0.3865691 | 0.3921 |
| Correlation Coefficient | 0.4459679 | 0.4122 |
| Root Mean Square Error | 0.482667 | 0.4846 |

In this table comparison, the XGB model is showing to be better. Less error and higher correlation scores. The values for the XGB model were taken with one of the best iterations it would produce for us to compare with the more stable REPTree rmse of 0.4846, so, as a reminder, it is important to remember the volatility of the current XGB when compared to a more stable output of the REPTree

**Prescriptive Analytics**





After putting the Train.csv file with Description.csv file, we looked at how search words were matching to actual product title and description. Since we realized that the number of word matching were related to the relevance score, we ran a test after making 1000 samples. After getting those samples, we got an average for each fields and compared the average relevance score with MatchPerWord. Surprisingly, we figured that if we divide averaged MatchPerWord in 2 and compare it with Relevance score, it showed something between 90% and 100%. This proved that MatchPerWord divided by 2 can represent Relevance score, which means we can work the same thing on Test.csv file. This proof can help predict the relevance score for Test.csv file.

## 

**Comparisons with documented results**

The Home Depot Product Search Relevance Project is again, a competition hosted through Kaggle. The submission date for the first draft is April 18th. We can see the top scores for the lowest Root Mean Squared Error (rmse), but we don’t necessarily get to see how other teams got those scores. We can only compare our training model rmse to the leaderboard rmse. We need to submit our prediction results of the training model to Kaggle and they compare that with the actual results to give us the prediction rmse. What we can do is research what methods other teams are discussing in the forums and then compare the topics to any jumps in scores. One of the largest jumps in rmse was when teams discovered that using extreme gradient boosting to help build the model. Since we wanted to use RStudio, we found that we can use the xgboost package available through CRAN. Other advances were made that resulted from differences in models and software used.

Currently, the biggest advance has been with teams finding different methods to spellcheck the search terms. Some teams are using their own spell checking algorithms while others are using Google Search and storing the corrections. At this point, they’re working to determine whether or not using Google Search violates any parameters within the rules. Currently, the admins of the competition state that using Google Search does violate competition rules because the external data can’t be verified.

Instead, we looked up CrowdFlower case on Kaggle. This case was similar in nature to the Home Depot competition, which allows us to compare methods of completion to ours. The winner of that competition won with a XGBoost model while also preprocessing it intensely. To begin, the winner started with a spelling correction method as well as a synonym replacement tool. By using a synonym replacement tool, the model takes a wide range of words that mean the same thing and apply them to a single word. By doing this, words like child, kid and youngling would all be interpreted as kid. Finally, they also stemmed their data which is something we did as well.

**Deployment**

Our deployment has the potential for a specific direction and also a very broad approach. To begin, we can take our findings through the test and train files directly to Home Depot and work with them to improve their search relevancy on their own website. However that isn’t our only option since the model can be adapted and improved to fit most other websites search functions. Because of this, we could also market our model to any website with a functional search engine to help aid in its optimization.

We do not expect any issues with the deployment of our original model to Home Depot. The only foreseeable issues would be accumulating the libraries and packages for the Rstudio script to function properly, however we circumvented this issue by adding script that automatically downloads the proper packages prior to the preprocessing of the data. By doing this, anyone would be able to copy the script and run it without error.

We have not come across any situations that could be an ethical issue. The only possible issue we could think of would be if someone had manipulated that data from its original source to skew the results in a favorable way. However this has not happened in our dataset as we were actively downloading from the source.

The first potential risk revolves around the data itself and its credibility. Luckily the data has come from a trusted vendor so we can operate under the assumption that it is set up well and concise. However if that isn’t the case, significant transformation of the data would be required to model it properly. Next, like I have previously mentioned, there was a risk of a failure of the model due to a lack of required packages. We have already mitigated that risk by adding the methods to retrieve the packages in the script itself. Also, we have contemplated that our model will not be the best. Our options regarding this are diverse and we have looked at many different options from switching to Python which seems to be a more common language as well as other more detailed methods like working with a spell checker or other methodologies. Also, we have contemplated that the output we are receiving is deceiving and not actually predicting what we were hoping for, but we have mitigated that risk by going line for line within the script to assume that the output is what we were expecting. Finally, the last risk revolves around if Home Depot uses our model to fix their search engine, however we have worked and would acknowledge that our model is not perfectly accurate and explain that creating a model that is perfect is near impossible.