**CS 6375- Machine Learning**

**Project Report**

**Topic: Product search relevance score prediction using Home Depot Dataset**

**Team Members :**

|  |  |
| --- | --- |
| ***Ali Aflaki***  ***Juhi Bandari***  ***Revanth Nyalakonda***  ***Panchami Rudrakshi***  ***Ranjani Suresh*** | ***axa154731***  ***jxb161830***  ***rxn162130***  ***pgr150030***  ***rxs144930*** |
|  |  |
|  |  |

**INDEX**

|  |  |
| --- | --- |
| **Sl No.** | **Topic** |
| 1 | Introduction |
| 2 | Problem Statement |
| 3 | Related Work |
| 4 | Dataset description |
| 5 | Data preprocessing methods |
| 6 | Proposed solution |
| 7 | Pseudocode |
| 8 | Results |
| 9 | Conclusion |
| 10 | References |

**1. INTRODUCTION**

The Home Depot is a home improvement supplies superstore that sells tools, construction products and services. It has a huge customer base and has several products and its descriptions. The customer search relevance score of search results, given the searching queries, resulting product titles and product descriptions is of our utmost importance and this is what enables the customers to find the right products.

**2. PROBLEM STATEMENT1**

In this project, we are developing a model that can accurately predict the relevance of search results.

Search relevancy is an implicit measure Home Depot uses to gauge how quickly they can get customers to the right products. Currently, human raters evaluate the impact of potential changes to their search algorithms, which is a slow and subjective process. By removing or minimizing human input in search relevance evaluation, Home Depot hopes to increase the number of iterations their team can perform on the current search algorithms.

**3. RELATED WORK**

This dataset has already been used and the concept of model ensembling was already proposed for the prediction of search relevance by a team called the Turing rest for the Kaggle competition where they won the first position. We are trying to expand the same by using different regression methods and model ensembling methods which can be used for the search relevance score prediction quantitatively.

**4. DATA SET DESCRIPTION**

The Home Depot dataset consists of two basic datasets one for training and the other for testing.

* **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.

To get an overview of the Train and Test Datasets, and also to get familiar with words used both in product description, title, attributes and also in search phrases / keywords. Several simple steps have been taken which will be elaborated in this section.

After getting familiar with Dataset details, we could design our data preprocessing which of course is highly correlated with the structure and content of Datasets, which also will be detailed in this section.

# **Training & Test Datasets:**

As the very first step in reviewing the structure of the dataset few first lines of the both Train.CSV and Test.CSV will be printed here;

First three lines form Train.CSV:

"id","product\_uid","product\_title","search\_term","relevance"

2,100001,"Simpson Strong-Tie 12-Gauge Angle","angle bracket",3

3,100001,"Simpson Strong-Tie 12-Gauge Angle","l bracket",2.5

Formatted in Table;

x`

First three lines form Test.CSV:

"id","product\_uid","product\_title","search\_term"

1,100001,"Simpson Strong-Tie 12-Gauge Angle","90 degree bracket"

4,100001,"Simpson Strong-Tie 12-Gauge Angle","metal l brackets"

Formatted in Table;



Training Dataset consists of 74,067 instances of training, among which there are 54,667 unique PRODUCT\_UID’s. As shown in above table each instance consists of five columns in order of appearance from left to right these columns are;

ID: search instance id number

PRODUCT\_UID: unique ID number for each product which has shown as a result of a search

PRODUCT\_TITLE: title of the product

SEARCH\_TERM: search keyword(s) of that search instance

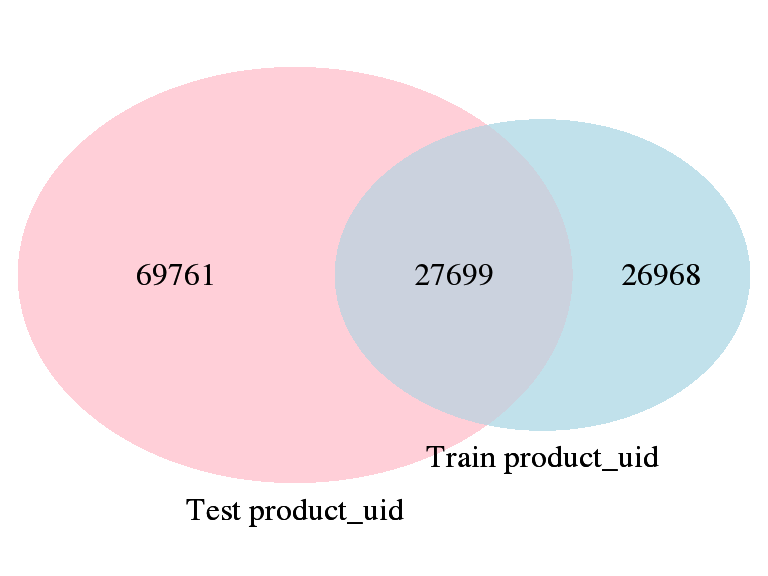
RELEVANCE: average score given by human raters ot the relevancy of this product to the search a real number between 0 and 3. 3 being the most relevant.

As we expected Test Dataset lists all the headers in Training Dataset except for the Relevance which is the value our classifier will generate.

Test Dataset lists 166,693 instances, among which there are 97,460 unique PRODUCT\_UID’s.

We have done further analysis on both datasets to see how many PRODUCT\_UID’s are common between the two and how many appears only in either of the two. Basically in this step we have cross referenced the two datasets and counted the number of PRODUCT\_UID’s appearing in both and also in only either of the two. Diagram below represents the results of this investigation It shows that there products in Training dataset that are not referred to in the Test dataset and also there are products in Test dataset that have not been presented to the system in the training dataset.

1. **Attribute Datasets:**

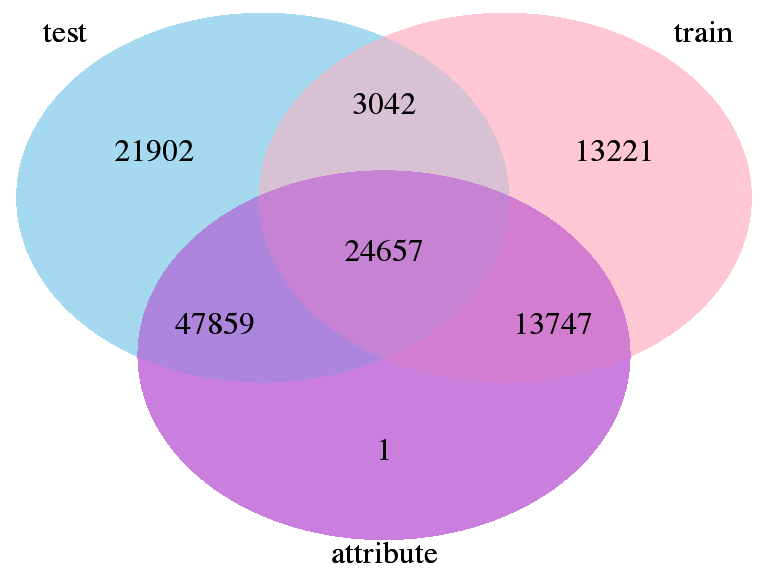
  
Illustration 1: Product\_UID Distribution across Training and Test Datasets

Along with the Training and Test datasets there is a CSV file which contains Attributes of products. This file contains 2,044,803 rows which covers 86,264 unique PRODUCT\_UID’s.

Surprisingly the number of unique PRODUCT\_UID’s covered in this file is smaller than unique PRODUCT\_UID’s which appear in Test dataset. Also we noticed that Attibutes file does not contain attributes of all of the products.

Further analysis of Attributes file and comparison of that with Training and Test Datasets revealed the overlap of PRODUCT\_UID’s in these three files. Which is represented in the diagram above: HomeDepot First Data Exploration[[1]](#footnote-1)

This observation suggests that though utilization of product attributes in training of our ML system might be beneficial. Attributes cannot help in all cases because the information is limited to few Products and as can be seen for considerable percentage of PRODUCT\_UID’s in Test Dataset no attribute information is available. This fact which might affect our strategy in implementation of the ML system, certainly is worth noting.

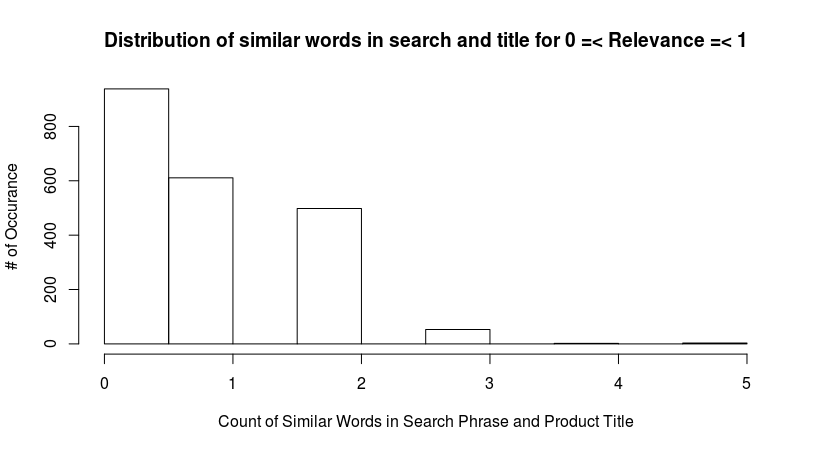
  
Illustration 2: Product\_UID distribution across Datasets

# **c ) Search Keyword and Product Titles Analysis:**

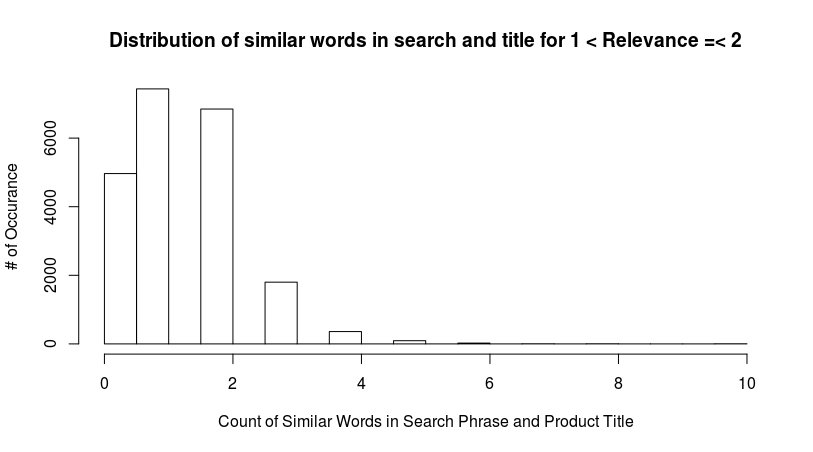
Considering the fact that a product’s is expected to reveal useful information to help customers make their purchase decision. We think that emphasis should be placed on product titles in this project. But to evaluate how relevant the product title information and search keywords are. We decided to analyze the Training Dataset in this section, and measure the accuracy of this idea in this particular case.

To do so we have designed a study in which for each of the training instances, we will compare the relevancy rank given to the instance with the similarity of the product title and the search keywords in that instance. If our idea holds true then we expect to see higher relevance in training instance were the search keyword has higher similarity with the product title.[[2]](#footnote-2)

Relevance in Training Dataset has got 13 distinct values ranging from Zero to Three, for this analysis relevance has been categorized into three buckets; [0,1], (1,2], (2,3] and for each bucket of relevance values distribution of similarity between search keywords and product title has been analyzed.



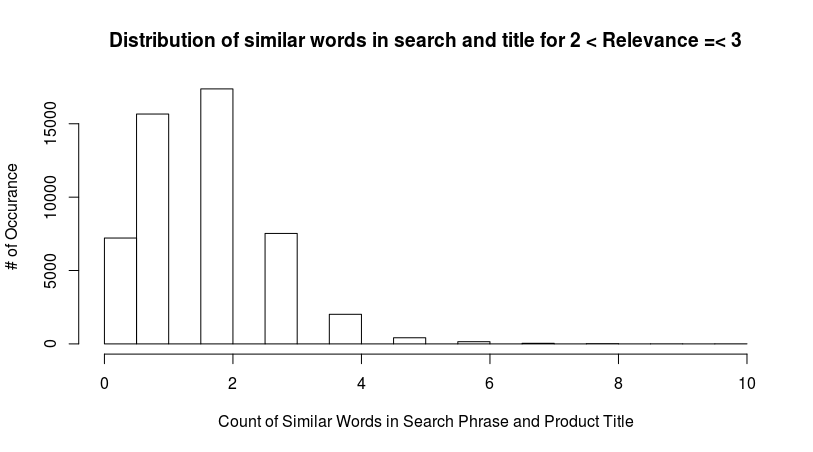
Above figure shows the distribution of count of similar words in Search keywords and product title for instances with Relevance between zero and one inclusive. Instances with no similar words in search keyword and product title are most frequent in this group.



Below figure shows the distribution of count of similar words in Search keywords and product

title for instances with Relevance between one and two. Comparing to previous figure notice that the distribution consists of considerably more matching words in search phrase and product title.

Above figure shows the distribution of count of similar words in Search keywords and product title for instances with Relevance greater than two. Comparing to both previous figures it is apparent that the distribution consists of considerably more matching words in search phrase and product title. This can be seen by observing the peak movement from Zero to greater values in figures.



**5. DATA PREPROCESSING METHODS**

## 5.1 Spell Checking and Correction

We started learning about spell checking in Text datasets by reviewing the article at [CleaningText] were we got introduced to [Aspell] which appeared to be Kevin’s effort in improving upon [Ispell] but going through its details we realized that around 2013 another spell checker system has raised which gained more popularity which is called [Hunspell] and it appeared to be the one used in LibreOffice and Mozilla Firefox applications as well as some proprietary systems including Mac OS x, InDesign, Opera and etc. So we decided to learn and use Hunspell for this project.

Then we figured out about hunspell in R at this link [HunspellR]. We implemented the spell checker and also spell correction algorithms using the Hunspell package, which works well.

After being able to correct spellings of the text information next step would be to extract the stem of the words, though Hunspell provides two functions for this (stem and analyze) none seem to work as expected, they only can remove simple pre/suffixes from the words, such as removing -ing and -ed. But drove which is past tense of drive is not recognized by Hunspell methods, even worse “running” which get -ning also is not recognized. I got the impression that Hunspell though very good in spell checking did not appear promising for stemming. So, we carried on more research on stemming, which will be covered in next section.

## 5.2 Stemming

Stemming or reducing inflected or derived words to their base or root. [StemmingWiki] Conflation: Many [search engines](https://en.wikipedia.org/wiki/Search_engine) treat words with the same stem as [synonyms](https://en.wikipedia.org/wiki/Synonym) as a kind of [query expansion](https://en.wikipedia.org/wiki/Query_expansion), a process called conflation.[StemmingWiki] Learned about Martin Porter’s Snowball framework. And going to work on that here [Snowball]. We installed the package SnowballC in R and used its wordStem method.

We encountered a problem that it cannot really get through long affixes, for example once we tried “friendliness” and experimented both SnowballC and tm package (stemDocument method) which also wraps the SnowballC and observed below responses:

> wordStem("friendliness")

[1] "friendli"

> wordStem(wordStem("friendliness"))

[1] "friendli"

> wordStem("friendli")

[1] "friendli"

> wordStem("friendliness")

[1] "friendli"

> wordStem(wordStem("friendliness"))

[1] "friendli"

> stemDocument("friendliness")

[1] "friendli"

> stemDocument(stemDocument("friendliness"))

[1] "friend"

as you can see the shortcoming in case of tm.stemDocument can be overcome by running the word again through the algorithm but in case of SnowballC even that would not help. Going to do more research on this matter. Hence we decided to carry on utilizing the tm package’s stemDocument method in our implementation.

## 5.3 Features

As explained above we decided to first clean the datasets which are mainly in Text, the core of data are Product Titles and Descriptions and User Search Queries which are all in text. So, to extract features we first preprocessed the data through below steps;

* Convert all to lowercase
* remove punctuation characters
* remove whitespace from the text
* split / expand text in to arrays of words
* remove stop words such as; a, an, the, of, …
* Spell Correction
* convert all words to their stem’s
* remove duplicates from the stem’s

From this stage onwards we will only deal with the extracted stem’s or converted datasets. But apparently still we are to deal with a very large size of text data, hence next step would be to define, create and calculate features which are to be used with our ML system for this task.

We have defined four features for this DATASET. As defined below;

1. X1 = count of elements of intersect(ProductTitle[i],SearchQuery[i])
2. X2 = X1 / (count of elements of union(ProductTitle[i],SearchQuery[i])
3. X3 = X1 / count of elements of ProductTitle[i]
4. X4 = X1 / count of elements of SearchQuery[i]

where X2 is the Jaccard Distance which a very popular text feature, but for this specific problem we decided to introduce three more features as listed above, where X1 is simply the count of common stems in product title and search query, X2 is X1 divided by the count of union of all stems in both strings, X3 is X1 divided count of words in product title, and finally X4 is X1 divided by count of words in Search Query. Idea is to penalize long phrases, since apparently the longer the phrase it will have more stems/words and chances of being a match to any search query might increase, similarly a search query with a very long phrase also might match many titles, X3 and X4 are there to penalize these extreme cases, whereas in normal situation they are not supposed to affect the system greatly.

To extract the introduced features we have compared the STEM’s extracted from each instance of test and train dataset and have created these features for each instance.

## 5.4 Pseudo Code (Pre-Processing)

Note below pseudo code and code is written in R and using this code we have preprocessed the datasets and saved the results in two CSV files, which are also submitted along the report.

File#1: testFeatures-all.csv- Features of the Test Dataset

File#2: trainingFeatures-all.csv- Features of the Training Dataset

# Loading required libraries

require(tm)

require(hunspell)

require(data.table)

#Load the data into environment

hdTrain<-data.table::fread("train.csv",header=TRUE,encoding="Latin-1")

hdTest<-data.table::fread("test.csv",header=TRUE)

# loading training Set into local variables

# Training DataSet

P <- hdTrain$product\_title

Q <- hdTrain$search\_term

R <- hdTrain$relevance

# Test DataSet

testP <- hdTest$product\_title

testQ <- hdTest$search\_term

# Spell Correction gets an array of words and returns it fixed

correctSpelling <- function(words) {

for (i in 1:length(words)) {

if (!hunspell\_check(words[i])) {

#cat(paste("wrong word: ", words[i], " at index: ", i))

words[i] = hunspell\_suggest(words[i])[[1]][1]

#cat(paste(" replaced by: ", words[i], "\n"))

}

}

return (tolower(words));

}

# preprocess Text

preprocessText <- function (myText)

{

myText = tolower(myText);

myText = gsub("[[:punct:]]", " ",myText);

myText = removeWords(unlist(strsplit(stripWhitespace(removePunctuation(myText))," ")), stopwords("en"))

myText = myText[myText!=""];

if (length(myText) > 0) {

mytext = correctSpelling(myText);

myText = unique(stemDocument(myText));

}

return(myText);

}

# preprocessing the training data

cat("\n\n\n\nProcessing Product Titles...;");

preProcessedP = lapply(P, preprocessText);

cat("Completed Processing Product Titles...;");

cat("\n\n\n\nProcessing Search Queries...;");

preProcessedQ = lapply(Q, preprocessText);

cat("Completed Processing Search Queries...;");

#encoding problem in Product Title so they have been converted to UTF-8

testP = iconv(testP, from="ISO-8859-1", to="UTF-8");

# preprocessing the test data

cat("\n\n\n\nProcessing Product Titles...;");

preProcessedTestP = lapply(testP, preprocessText);

cat("Completed Processing Product Titles...;");

cat("\n\n\n\nProcessing Search Queries...;");

preProcessedTestQ = lapply(testQ, preprocessText);

cat("Completed Processing Search Queries...;");

# Experiment 1

# x1 = count of elements of intersect(P[i],Q[i])

# x2 = x1 / (count of elements of union(P[i], Q[i]))

# x3 = x1 / count of elements of P[i]

# x4 = x1 / count of elements of Q[i]

X1 = NULL

X2 = NULL

U1 = NULL

X3 = NULL

X4 = NULL

# extractFeatures is a function that extracts features from preprocessed product titles and search queries, results are stored in X1,X2,X3 & X4.

extractFeatures <- function(processedProductTitle, processedSearchQuery)

{

endIndex = length(processedProductTitle);

for (i in 1:endIndex)

{

X1[i] = length(intersect(processedProductTitle[i][[1]], processedSearchQuery[i][[1]]))

U1[i] = length(union(processedProductTitle[i][[1]], processedSearchQuery[i][[1]]))

X2[i] = X1[i] / U1[i]

X3[i] = X1[i] / length(processedProductTitle[i][[1]])

X4[i] = X1[i] / length(processedSearchQuery[i][[1]])

}

output <- list(X1,X2,X3,X4);

names(output) <- c("X1","X2","X3","X4");

output <-as.data.frame(output);

return(output)

}

# fixing and saving Trainig Dataset's features

features <- extractFeatures(preProcessedP,preProcessedQ)

features[,5] = R

names(features)[5] = "Relevance"

summary(features)

features$X4[is.na(features$X4)] = 0

dim(features)

write.csv(features, file = "trainFeatures-all.csv", col.names = TRUE, row.names = FALSE)

# fixing and saving Test Dataset's features

# in Test Dataset there is no Relevance information (Target Value)

features <- extractFeatures(preProcessedTestP,preProcessedTestQ)

summary(features)

features$X4[is.na(features$X4)] = 0

dim(features)

write.csv(features, file = "testFeatures-all.csv", col.names = TRUE, row.names = FALSE)

**6. Proposed Solution**

The classifier we use are Neural networks, Random Forest and Bagging to predict the relevance.

**6.1 Neural Networks Implementation:**

For this task we have also designed an Artificial Neural Network. As explained in the preprocessing section we managed to get the datasets converted into four features of X1, X2, X3, and X4, as listed below;

1. X1 = count of elements of intersect(ProductTitle[i],SearchQuery[i])
2. X2 = X1 / (count of elements of union(ProductTitle[i],SearchQuery[i])
3. X3 = X1 / count of elements of ProductTitle[i]
4. X4 = X1 / count of elements of SearchQuery[i]

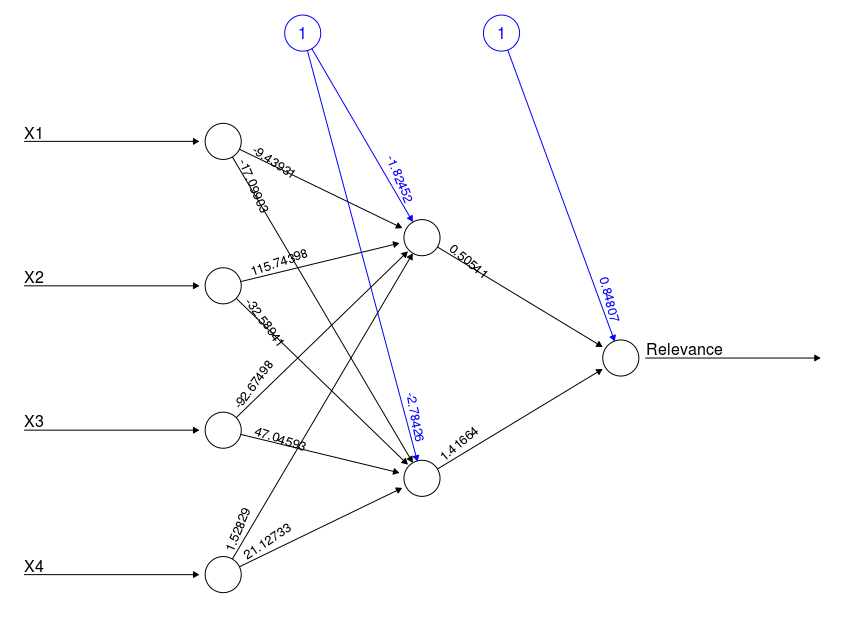


Illustration 1: Neural Network Structure & weights

This NN has been implemented in R using ‘neuralnet’ package. Runs in Regression mode (linear output). And its structure is as follows;

Three layers; input, hidden and output. Input layer consists of four input nodes and a bias node, hidden layer consists of 2 hidden nodes[Model Selection ANNs] and one bias node and output layer consists of one node. Below diagram shows this network weights correspond to last trial of the 5-fold cross validation done on training data.

In this network following the idea shared in this article [Model Selection ANNs] since there are 4 input nodes and one output node, we tried three and two nodes in one hidden layer, the architecture with two hidden node (plus a bias node) appeared to be the suitable configuration for our neural network, hence the design was shaped as illustrated in above figure.

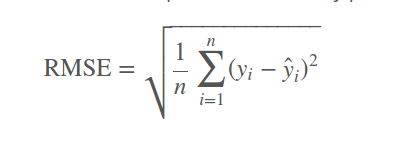
## Cross Validation

In this particular task where we did not have the target values on Test Dataset, for the sake of training and testing our neural network we decided to do cross validation on Training Dataset. We had more than 67,000 instances in our dataset, and decided to do a five fold cross validation on this Dataset. Hence as illustrated and implemented in the code we have randomized the sequence of instances and for five iterations have sequentially put aside one fifth of the instances for test and rest for training. Results of which will be reported in a later section.

This approach enabled us evaluate and optimize the neural network’s designed for this task.

### Performance metric

As explained in kaggle, source of this data and task, the performance measure for this regression task is RMSE, root mean squared error, which is defined as;

  
Illustration 2: RMSE Formula

**6.2 Random Forest classifier Implementation:**

Random forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks, that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees. Trees that are grown very deep tend to learn highly irregular patterns: they [overfit](https://en.wikipedia.org/wiki/Overfitting) their training sets, i.e. have [low bias, but very high variance](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff). Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.

**Basic algorithm**

Random forests train a set of decision trees separately, so the training can be done in parallel. It uses feature bagging. The algorithm injects randomness into the training process so that each decision tree is a bit different. Combining the predictions from each tree reduces the variance of the predictions, improving the performance on test data. At each candidate split in the learning process, a [random subset of the features](https://en.wikipedia.org/wiki/Random_subspace_method) is selected. The spark.ml is used for implementing random forests for regression using both continuous and categorical features.

**Training**

The randomness injected into the training process includes:

* Subsampling the original dataset on each iteration to get a different training set (a.k.a. bootstrapping).
* Considering different random subsets of features to split on at each tree node.

Apart from these randomizations, decision tree training is done in the same way as for individual decision trees.

**Prediction**

To make a prediction on a new instance, a random forest aggregates the predictions from its set of decision trees. The label is predicted to be the average of the tree predictions. Since the target variable is a real valued number, we fit a regression model to the target variable using each of the independent variables. Then for each independent variable, the data is split at several split points. Sum of Squared Error (SSE) is calculated at each split point between the predicted value and the actual values. The variable resulting in minimum SSE is selected for the node. Then this process is recursively continued till the entire data is covered.

Our Random forest classifier has been built using spark.ml package :

Spark ML standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. It is built on top of [DataFrames](https://spark.apache.org/docs/1.6.1/sql-programming-guide.html" \l "dataframes) using the following methods:

[**DataFrame**](https://spark.apache.org/docs/1.6.1/ml-guide.html#dataframe): Spark ML uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types.

E.g., It has different columns storing text, feature vectors, true labels, and predictions.

* [**Transformer**](https://spark.apache.org/docs/1.6.1/ml-guide.html#transformers): A Transformer is an algorithm which can transform one DataFrame into another DataFrame.

E.g., an ML model is a Transformer which transforms DataFrame with features into a DataFrame with predictions.

* [**Estimator**](https://spark.apache.org/docs/1.6.1/ml-guide.html#estimators): An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer.

E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

* [**Pipeline**](https://spark.apache.org/docs/1.6.1/ml-guide.html#pipeline): A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.
* [**Parameter**](https://spark.apache.org/docs/1.6.1/ml-guide.html#parameters): All Transformers and Estimators now share a common API for specifying parameters.

**6.3 Bagging Implementation:**

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.

There are libraries and packages available in R for performing bagging regression. The randomForest package is used to devlop the model. We then read the training and test datasets and assign the test relevance 0.0. The random forest model is developed with the trained data and the prediction is done taking into account all the features of the training dataset. This is where bagging differs from random forest as the latter performs random selection of the features whereas the former takes into consideration all the features. Now for the cross validation, we split the training dataset 80-20 and the 20% is the subset taken as test set. The relevance mean and the plot are predicted and displayed.

**7. PSEUDO CODE**

**7.1 Neural Network**

1. Load the required libraries and packages
2. Load, examine and fix the Dataset removing NA values
3. Draw the summary
4. Remove the instances with all features = 0 and Relevance != 1 (irrelavant)
5. Normalize the dataset- the only feature which need be centered is the X1 other features are normalized by definition
6. Relevance is assumed to be between [1-3]
7. Perform Cross Validation k-fold with k = 5
8. Run the network in iterations
9. Find RMSE and plot the model
10. Print the results and architecture of the network

**7.2 Random Forest :**

1. Load and import all required packages and libraries
2. Build the Spark Context
3. Load the featureset
4. Create schema
5. Loading train and test data
6. Create a Vector Assembler
7. Convert the RDD into dataframes
8. Create Feature indexers
9. Sample the dataset into 80-20 with 20% as the test dataset
10. Building a Random Forest Regression
11. Generate a Pipeline
12. Perform Cross Validation using a parameter grid and a regression evaluator
13. Use no of folds as 5 for regression evaluator
14. Predict the RMSE value using the evaluator
15. Print and store the results in a file

**7.3 Bagging:**

1. Load the required libraries
2. Load the training and test dataset
3. Extract the columns and draw the summary
4. Assign the test relevance value as 0.0
5. Generate a random forest with the train data and the no of features as 4
6. Predict the corresponding relevance value for test data
7. Sample the dataset into 80-20 with 20% as the test dataset
8. Perform cross validation on the latter
9. Plot the predicted relevance and find the mean squared error
10. Write the results into a file

# **8. RESULTS**

**Neural Network:**

# Average Root Mean Squared Error on 5-Fold CV: 0.00814682654187679

# RMSE of each iteration:

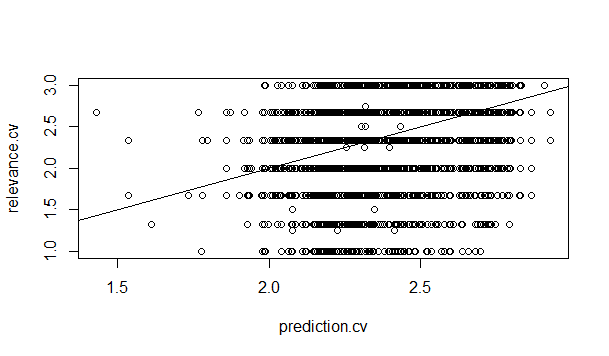
# [1] 0.0098348079515 0.0013240595257 0.0006317040866 0.0090597745138 0.0198837866317

**Random Forest:**

Root Mean Squared Error (RMSE) on test data = 0.489768

**Bagging:**

Plot for Bagging- Relevance prediction cross validation



# **8. CONCLUSION**

This Dataset investigation has provided us with few highlights and facts about the problem we have at hand. First and foremost, it is critical to remember that we do not have attributes for the Products which are being referred to in the Test Dataset. With the involvement of the attributes and the description the way of handling would have been different. As this is out of our scope and as this hasn’t been provided as well, the approach was pretty straight forward.

On the other hand the latter analysis of similarity between the search keywords and product titles though is in-line with our expectation as the closer the search phrase to the product title the more relevant the instance should be. Still even in the case of highest value of Relevancy bucket (2 < R =< 3) still considerable number of instances are among zero or one matching words. Since HomeDepot training Dataset is the result of human raters reviewing the information and ranking the relevancies. We believe there are more similar words but they might be not exactly in the same format, such as synonyms or words with same stem but in different form, also we observed several cases of mis-spelled search phrases. Since human rater would automatically consider the synonyms identical and even the mis-spelling also in most cases are ignored by human because they will assume the person searching have made a mistake and meant the correct spelling.

This whole idea has made us believe and more text processing especially search keywords will help improve overall systems performance. Hence we are considering work stemming as well as spell correction for search keywords in our preprocessing as well.

Rather than classification, we have performed regression with the given dataset and predicted continuous values with the categorical features defined above. Data set is regression based data set. The prediction of relevance is quantitative and we have given the prediction values based on the defined features.

## 10. REFERENCES

1) CompetitionDetails: KaggleTeam, Home Depot Product Search Relevance, [online] <https://www.kaggle.com/c/home-depot-product-search-relevance>

2) DataExploration: Brian Carter, HomeDepot First Data Exploration, 2016, [online] <https://www.kaggle.com/briantc/home-depot-product-search-relevance/homedepot-first-dataexploreation-k>

3) Data set description: <https://www.kaggle.com/c/home-depot-product-search-relevance/data>

4) CleaningText: Schaun Wheeler, Automatic cleaning of messy text data,

5) Aspell: Kevin Atkinson, GNU Aspell,

6) Ispell: UCLA, International Ispell,

7) Hunspell: Hunspell Team, Hunspell,

8) HunspellR: Jeroen Ooms, Hunspell: Spell Checker and Text Parser for R,

9) StemmingWiki: Wikipedia, Stemming,

10) Snowball: Wikipedia, Snowball (programming language)

11) Bagging : Springer Texts in Statistics by G. Casella, S. Fienberg, I. Olkin

12)Random forest implementation: <https://spark.apache.org/docs/1.6.1/ml-classification-regression.html#random-forest-regression>

13) Model Selection ANNs: doug, How to choose the number of hidden layers and nodes in a feedforward neural network?

1. We have done this analysis in R using data.table package. Code for R has been submitted along this report. Refer to Code Segment [PreProcess-DataAnalysis]. [↑](#footnote-ref-1)
2. This analysis is implemented in R and its code is submitted along with this report. Refer to Code Segment [PreProcess-KeywordSimilarityAnalysis] [↑](#footnote-ref-2)