**Capstone Project 2 Report**

**Problem:**

Improving Home Depot customers' shopping experience by developing a model that can accurately predict the relevance of search results.

**Client:**

The client is Home Depot which is a retail company and they have an e-commerce platform. They care about this problem because they try to improve customer experience by increasing the accuracy of search results, in order to import the conversation rate.

**1. Data:**

The data is from: <https://www.kaggle.com/c/home-depot-product-search-relevance/data>. This data set contains a number of products and real customer search terms from Home Depot's website. The challenge is to predict a relevance score for the provided combinations of search terms and products. To create the ground truth labels, Home Depot has crowdsourced the search/product pairs to multiple human raters.

The relevance is a number between 1 (not relevant) to 3 (highly relevant). For example, a search for "AA battery" would be considered highly relevant to a pack of size AA batteries (relevance = 3), mildly relevant to a cordless drill battery (relevance = 2), and not relevant to a snow shovel (relevance = 1)

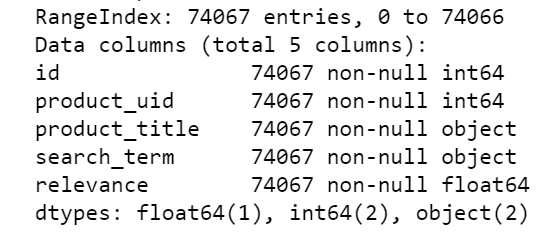
**File Description:**

* 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 of 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.

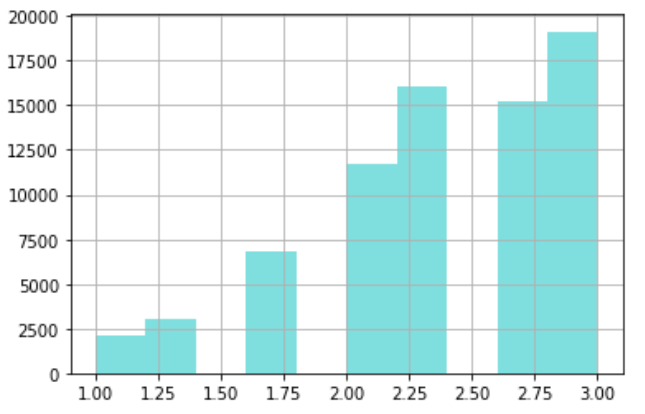
**Data Files:**

* id - a unique Id field which represents a (search\_term, product\_uid) pair
* product\_uid - 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

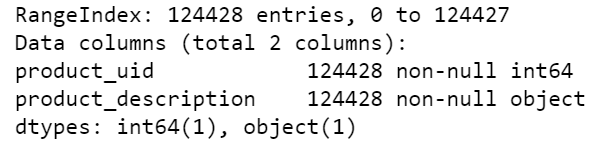
There are a total of 74067 rows in train.csv:



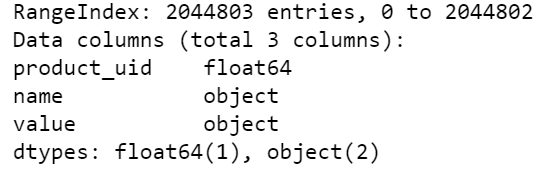
The distribution of relevance score is heavily towards greater than 2:



There are a total of 124428 rows in product\_descriptions.csv:



There are a total of 2044803 rows in attributes.csv:



**2. Test Pre-processing:**

Text pre-processing includes spell checking, creating a column of brands, removing tags, removing accented characters, removing special characters, tokenization, stemming, lemmatization. Text pre-processing will help improve the accuracy of the model since it removes not necessary words.

**2.1 Spell Checking**: Used a static corpus that is provided on Kaggle to correct the spelling of the search terms.

**2.2 Column of Brands:** Created the name of the brands column from the name column of attributes.csv.

**2.3 Column of Attributes:** Cleaned the text on the value column of attributes.csv, and created the column of products attributes.

**2.4.1 General Processing:** Splitting words, lowercase, removing special characters, removing text between parentheses/brackets, replacing word numbers to numerical expression, and standardize the unit's representations. For example, replacing “inches” and “ inch” to “in”.

**2.4.2 Removing Tags:**  Removing HTML tags.

**2.4.3** **Removing Accented Characters:** Converted and standardized into ASCII characters.

**2.4.4 Tokenization:** **Tokenization**is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

**2.4.5** **Stemming:** Word stems are usually the base form of possible words that can be created by attaching affixes like prefixes and suffixes to the stem to create new words. This is known as inflection. The reverse process of obtaining the base form of a word is known as stemming. A simple example is the words **WATCH**ES, **WATCH**ING, and **WATCH**ED. They have the word root stem **WATCH** as the base form.

**2.4.6 Lemmatization:** Lemmatizationis very similar to stemming, where we remove word affixes to get to the base form of a word. However, the base form, in this case, is known as the root word but not the root stem. The difference being that the root word is always a lexicographically correct word (but the root stem may not be so.

|  |  |
| --- | --- |
| Before Lemmatization | After Lemmatization |
| **Not only do angles make joints stronger…** | Not angle make joint stronger… |
| **BEHR Premium Textured DECKOVER is an innovativ…** | Behr premium textured deckover innovativ… |
| **Update your bathroom with the Delta Vero Single…** | Update bathroom delta vero single |

**2.4.7 Removing Stop Words:** Words that have little or no significance especially when constructing meaningful features from the text are known as stop words or stop words. These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus. Words like **a**, **an**, **the**, **and**so on are considered to be stop words.

**3. Features:**

I created a total of 34 features such as length of words, word matching counts, words ratio, TF-IDF, Word2Vec, FastText, cosine distance, Jaccard distance, and Levenshtein distance.

**Tf-idf Common Feature:**

* search\_term: customers search terms
* product\_title: title of the products
* prod\_desc\_merge: a combination of brands, descriptions, and attributes of products
* product\_uid: products ID

**Length of Words:**

* len\_of\_query: length of the search terms
* len\_of brand: length of the brands
* len\_of\_attribute: length of the attributes

**Search Terms Matching:**

* **w**ord\_in\_title: number of words that search terms match the product titles
* word\_in\_description: number of words that search terms match the product descriptions
* word\_in\_attributes: number of words that search terms that match the product attributes
* brand\_in\_search: number of words that search terms that match the brands
* first\_word\_title\_match: the first word of the search terms match product titles
* first\_word\_description\_match: the first word of the search terms match the product descriptions
* last\_word\_title\_match: the last word of the search terms match the product titles
* last\_word\_description \_match: the last word of the search terms match the product descriptions

**Word Ratio:**

* ratio\_brand: the ratio of the number of matching words in the brands and search terms
* ratio\_title: the ratio of the number of matching words in the product titles and search terms
* ratio\_description: the ratio of the number of matching words in the product descriptions and search terms
* ratio\_attributes: the ratio of the number of matching words in the product attributes and search terms

**Jaccard Distance:**

* jacc\_in\_title: the Jaccard distance between product titles and search terms
* jacc\_in\_desc: the Jaccard distance between product descriptions and search terms
* jacc\_in\_attr: the Jaccard distance between product attributes and search terms

**Levenshtein distance:**

* leven\_in\_title: the Levenshtein distance between product titles and search terms
* leven\_in\_desc: the Levenshtein distance between product descriptions and search terms
* leven\_in\_attr: the Levenshtein distance between product attributes and search terms

**Word2Vec:**

* word\_sim\_in\_title: the cosine distance between product titles and search terms based on a Word2Vec model
* word\_sim\_in\_title: the cosine distance between product descriptions and search terms based on a Word2Vec model
* word\_sim\_in\_attr: the cosine distance between product attributes and search terms based on a Word2Vec model

**Doc2Vec:**

* doc\_sim\_in\_title: the cosine distance between product titles and search terms based on a Doc2Vec model
* doc\_sim\_in\_title: the cosine distance between product descriptions and search terms based on a Doc2Vec model
* doc\_sim\_in\_attr: the cosine distance between product attributes and search terms based on a Doc2Vec model

**FastText:**

* ft\_sim\_in\_title: the cosine distance between product titles and search terms based on a FastText model
* ft\_sim\_in\_title: the cosine distance between product descriptions and search terms based on a FastText model
* ft\_sim\_in\_attr: the cosine distance between product attributes and search terms based on a FastText model

**4. Predictive Model:**

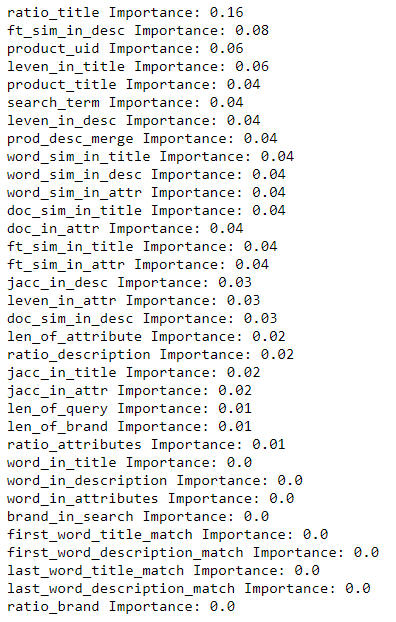
I used Lasso, elastic net, ridge regression, SVR, bagging, random forest, Adaboost, Gradient boost, XGboost, LightGBM, and voting regression algorithms to build the model and choose the smallest root mean square error model among them.

|  |  |  |
| --- | --- | --- |
|  | **Algorithms** | **RMSE** |
| **4.1** | Lasso | 0.5251 |
| **4.2** | Elastic Net | 0.5250 |
| **4.3** | Ridge Regression | 0.4659 |
| **4.4** | SVR | 0.5529 |
| **4.5** | Bagging | 0.4744 |
| **4.6** | Random Forest | 0.4761 |
| **4.7** | AdaBoost | 0.4794 |
| **4.8** | Gradient Boost | 0.4794 |
| **4.9** | XGBoost | 0.4576 |
| **4.10** | LightGBM | 0.4530 |
| **4.11** | Voting Regression | 0.4576 |

From the chart, we can see for bagging methods, random forest performed the best. For boosting methods, LightGBM performed the best.

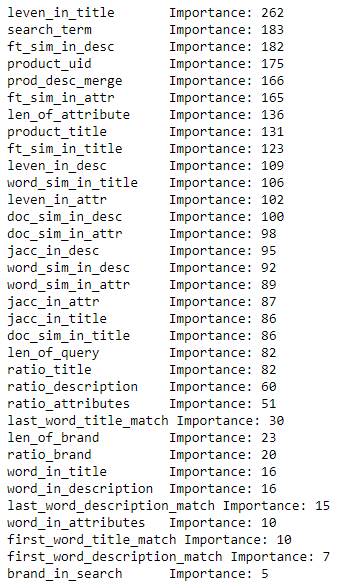
**4.12 Feature Selection:**

**Features importance for the random forest model:**

****

I deleted the features that are zero importance and perform random forest on the model again, the RMSE did not reduce. The RMSE is **0.4773**.

**Feature importance for the lightGBM model:**



**4.12 & 4.13 Hyperparameter Tuning:**

I performed a random and grid search on both the random forest and lightGBM model. After hyperparameter tuning, random forest model had RMSE at **0.4522** and lightGBM model had RMSE at **0.4530**

**Best Parameters for Random Forest:**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| bootstrap | False |
| max\_depth | 11 |
| max\_features | sqrt |
| min\_samples\_leaf | 1 |
| min\_samples\_split | 2 |
| n\_estimators | 1600 |

**Best Parameters for LightGBM:**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| learning\_rate | 0.05 |
| max\_bin | 64 |
| max\_depth | 15 |
| min\_data\_in\_leaf | 40 |
| num\_leaves | 200 |

**4.13 Final Model:**

After hyperparameter tuning, I used the stacking method to stack the random forest, and lightGBM model together and used lightGBM again. The final model has the best RMSE of **0.4521** among all.