Search Relevance Prediction: A case study in online retailer customers data

Chrysogonidis Georgios
School of Science and Technology
International Hellenic University (IHU)
Thessaloniki, Greece
g.chrysogonidis@ihu.edu.gr

Nikiforidis Vasileios
School of Science and Technology
International Hellenic University (IHU)
Thessaloniki, Greece
v.nikiforidis@ihu.edu.gr

***Abstract*—Search engines are designed to help users to quickly find useful information on the Web. Each search engine uses different indexing and/or hashing methods and different metrics to evaluate a query result. Online retailers borrowed this idea, applied on the Web for many years, and adapted it to their e-commerce platforms. A high quality search is mainly derived from a high relevance score that helps customers find the desired information or products in the website; although a single search engine has not the best performance all the time. Previous studies have shown that the performance of search engines depends on the performance measures used and application domains. The aim of this paper is to predict the relevance score of a product with respect to a query made in an e-commerce website. The conducted experiments consist of data description, several preprocessing techniques, feature engineering, implementation of different machine learning algorithms and evaluation. The results show that XGBoost with ten-fold cross validation in grid search achieved the best performance for the particular dataset.**

***Index Terms*—stemming, spell correction, scaling, relevance score prediction, cross-validation , grid search, XGBoost.**

I. INTRODUCTION

Due to the rapid evolution of the web and any web-based technology, manual performance evaluation of a search engine has become a costly procedure, both in time and resources. Daily user activity on e-commerce platforms has grown so massive that automatic evaluation of the relevance between a search-query and a result, is becoming more and more mandatory in order to facilitate a better user experience, by providing the right products given a customer's search. The role of the automatic evaluation is filled by machine learning models which take advantage of the massive amount of data being generated daily by users, in order to train and produce a higher relevance score. After all, in a data driven era, if one does not take advantage of the new form of currency, they are bound to fall behind competition.

II. DATA AND PROBLEM DESCRIPTION

1. Data description

The given data contain a number of products and real customer search queries from Home Depot's website and they are broken down into four csv files, training, testing, product description and the attributes of each product. Specifically, the training dataset contains 74067 samples of product ids and titles, search queries and relevance scores indicating the target variable, while the testing dataset contains 166693 records including the above features except from the relevance score. The next file contains 124428 rows with a description of each product, and the last file consists of 2044803 rows

providing extended general information about the subset of products, such as their brand and name.

2. Data preprocessing

The problem to be solved is considered a regression problem, since the target variable is a real number between 1 and 3, where 1 denotes the minimum relevance score and 3 the maximum. Specifically, as can be seen in Figure 1, almost 26% of the search queries had the highest score and only 2.847% of search terms had the minimum score. An insight we can get from this result, is that the users of this particular search engine, more often than not, found what they were looking for. If we take a closer look at the diagram, it is noticeable that the queries with score 2.33 and higher consist of nearly 68% of all search queries, which leads to the fact that even if the user did not find exactly their desired product, they at least found something relevant or close to it.

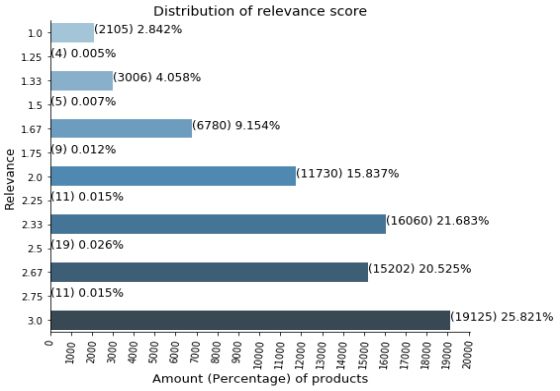

Figure 1. Distribution of relevance score.

Search terms are not unique in the training dataset, as the percentage of rows containing unique queries is 15.925% (11795 rows out of 74067 in total). More precisely, 9 different queries were searched 16 times each, occupying in total only 0.194% of the rows while we clearly observe that most queries were searched 9 times (Figure 2).

All the features in the training, testing and products' description dataset contain non null values. The dataset with the additional products' attributes contains 155 null values in the product's name and id and 2284 null values in the brand's name. Missing values in the dataset could prevent a classifier from trying to predict the unknown data or may affect in a negative way the performance of a machine learning algorithm. Null values based on "product_uid" were dropped, resulting in 2129 brand's name that filled with the "unknown" value. In order to make the most of all the columns, a series of joins on "product_uid" took place that unified all datasets to a single one that contained 250760 records. Since the datasets contained text, Natural Language Processing techniques were utilized that were carefully selected.

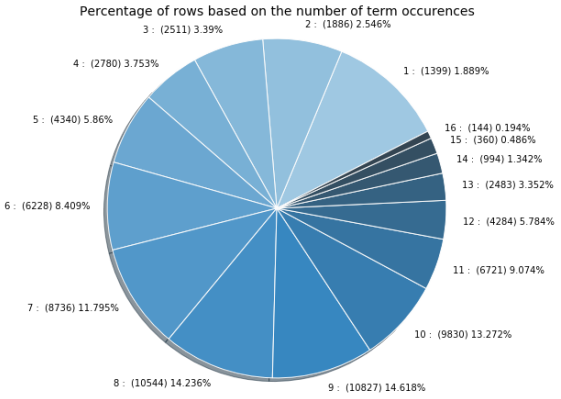

Figure 2. Percentage of rows based on the number of term occurrences

To start with, there are some proven standards when it comes to text preprocessing methods. As with everything, there are exceptions to these but in most cases, the following techniques are the starting point when one needs to preprocess text. The features that participated in this phase were: "search_term", "product_title", "brand" and "product_description".

General inconsistencies in those features were observed that were handled with a couple of regular expressions, leaving space after dots and question marks and separating words that were accidentally concatenated with other words, such as "informationRevives" and "coatingResists" in the product's description.

Same measurement units were also spotted with different formats and abbreviations that were tried to be corrected. Some examples of those are "centimeters" that converted to "cm" and "sq feet", "square feet" or "sq ft" that all converted to "sqft".

Lowercasing, although commonly overlooked, is one of the simplest and most effective forms of text preprocessing, especially in this particular problem. Most users do not consider writing a search query exactly as they expect to find its result, thus without lowercasing the relevance score would have taken a significant hit.

Moving on, another go-to tactic for text preprocessing is to remove all stop words from the text (depending on the language these may vary). Stop words are a set of commonly used words in a language e.g. "is", "a", "the", etc. which mostly have no information to offer on the context of the query. Removing these words, allows the algorithm to focus on the more important ones instead.

This technique was performed separately from stemming so that we have better control over their removal.

In the same scope as stop words removal, is punctuation removal that performed subsequently. Again, punctuation, in most cases, offers no information regarding the context of the query, thus it is only confusing the algorithm.

Stemming is another classic approach to text preprocessing. It is the process of normalizing text with the goal of keeping the vocabulary small to help improve the accuracy of language modelling tasks. For example, the words "information" and "informative" both have the same root, "inform" which is what is left of the two words after stemming is performed. This helps us minimize the vocabulary while still being in the same context of the query. In general, there are 3 most commonly used stemmers, in order of "aggressiveness": Porter, Porter2(Snowball) and Lancaster. Porter is also the oldest stemming algorithm, but was considered too soft in this case. We went with the Snowball stemmer since it had the best trade-off between speed and aggressiveness. Lancaster was the fastest of the three but led to many shorter words losing their meaning completely.

Finally, it seemed necessary to implement some kind of spell checking for the search queries, as in most cases, one does not take time to spell check what they write in a search engine. Some search engines implement a "did you mean" feature after the user has input their query, which was not something the community found useful. If the engine can correct a misspelled query, why would it not perform the corrected search directly instead. Thus, we attempted to create a method to spell correct search queries. The initial idea was to query Google for each search term and get back the "did you mean" of the response. Unfortunately, Google has obfuscated the responses to their search engine since the API is no longer available for public use. Other search engines, like Bing and Yahoo! were also tested, but they had a limit to the public requests allowed per day (~5000) so they were dropped as well. After that, a probabilistic approach was taken based on an essay by Peter Norvig[2]. After further reading, it was discovered that an earlier form of Google's search engine was loosely based on the same algorithm. For this algorithm to work, first we need to construct a vocabulary of words. For this dataset, the vocabulary was constructed based on the product's title, description and brand, so that the algorithm will learn words from brand and title and not try to correct them since they could be some kind of custom words. In a nutshell, the algorithm tries to predict the most likely spelling based on the vocabulary it uses, through implementation of Bayes theorem. Essentially, it tries to find the corrected text, let us call it *Corr*, out of all possible corrections for the input *Text*, that maximize the probability that *Corr* is the intended correction given the original *Text:*

$$argmax_{corr \in corrections} P(corr|Text)$$

After applying Bayes theorem we get:

$$argmax_{corr \in corrections} P(corr)\, P(Text|corr) / P(Text)$$

Here, *P(Text)* is the same for any possible input that is does not offer any particular value to the equation to help us maximize the result, so in essence we just need to compute:

$$argmax_{corr \in corrections} P(corr)\, P(Text|corr)$$

Breaking this down a bit further, we arrive at 4 separate components for the above expression:

1. argmax: Choice of the correction with the highest probability.
2. corr ∈ corrections: which corrections are taken into account.
3. P(corr): The probability that the *corrected* text appears in our vocabulary.
4. P(Text|corr): The probability that *Text* would be typed in when the user would actually mean *corr*.

These four components are computed by the algorithm and we finally get the result. This process turned out to be quite accurate, but time consuming. So after running it once, as is seen in the Python code file, we decided to create a Python dictionary with keys being the search queries and their respective corrections as values, and store this dictionary in a JSON file for subsequent predictions. The whole process of constructing the vocabulary and creating this dictionary took around thirty (30) minutes. After creating the dictionary once, it was then read from the JSON file and stored in a Python dictionary. Now, all the spell checking method had to do was take the query as input and return its correction directly from the existing dictionary which took less than 0.2 seconds. Some examples of the spell correction results are: "gas mowe": "gas mower", "three wayy": "three way", "foof leaf broom": "food leaf broom", "lawn sprkinler": "lawn sprinkler".

3. Data transformation

In the sphere of "data transformation" many techniques, such as smoothing, aggregation, generalization, normalization and feature construction, fall in.[1] Since

the datasets did not contain any numerical features, feature engineering played an important role in the experiments. Counting features calculated from the length of the product's title, description, brand and search's term were added in the dataset. Additionally, the number of common words between the search term and the product's title, brand's name and product's description were created successively. The intuition behind these features is that the more common words a query has with the brand, title and description individually, the more likely will appear the desired product in the customer's search. Some statistical features also generated from the constructed features; that is the ratio of common words in title and the ratio of common words in product's description with respect to the length of the query and the proportion of the length of title and the length of description with respect to the length of the query. The Jaccard similarity between search term and title, search term and description, search term and brand's name were constructed and added in the dataset rounded to the fourth decimal point. Jaccard similarity of sets A and B is the ratio of the size of the intersection of A and B to the size of their union:

$$\text{Jaccard' sim}(A,B) = |A \cap B| / |A \cup B|$$

The last features that participated in feature engineering were the number of common bigrams between the query and the rest text columns. The n-grams are all the combinations of adjacent words or letters of length n that you can find in a source text. In the 17 newly created columns, we performed standardization using Standard Scaler. This standardization method scales to unit variance; all values are converted to their z-score in a way that a value of 1 means that the value is 1 standard deviation higher than the mean of the column. Similarly, a value of -1 means that the value is 1 standard deviation lower than the mean of the column[4].

III. COMPARATIVE EXPERIMENTS AND RESULTS

Six regression algorithms were selected to participate in the experiments and presented in the table 1 among with their initial results. The selection of the regressors was based on the existing knowledge of each algorithm and the ability to solve regression problems. Using the same split of the dataset for every regressor with a fixed number in the random state parameter, the training set contains 51846 samples (70% of the dataset) and the test set contains 22221 samples (30% of the dataset). Moreover, the 10-fold cross validation method was utilized for each regressor in order to minimize the error. The k-fold cross validation is a resampling approach in which the dataset is randomly split into k mutually exclusive subsets (folds) of approximately equal size. The machine learning algorithm is trained and tested k times where in each fold one of the k subsets is used as the test set and the rest k-1 subsets together are used as a training set. The evaluation metric is then averaged over all k folds to get the overall performance of the model. Experiments have been shown that cross validation with k around 10 and 20 reduces the variance and increases the bias[3]. The evaluation of each regressor was based on root mean squared error (RMSE). RMSE is the square root of the average of squared differences between prediction ŷ and actual observation y:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

The initial experiments show that Gradient Boosting and XGBoost had the best performance while the regressors with the poorest performance were Support Vector Regressor (SVR) and Random Forest (Table 1). The performance of the models decreased when Standard Scaler applied in the dataset and we decided to exclude the scaling. Additionally, the moderate performance of linear regressors was not capable of keeping them in the second round of experiments where SVR and Random Forest also excluded.

Regressor	RMSE
Linear Regression	0.4901
Ridge	0.4901
SVR	0.4980
Random Forest	0.5021
Gradient Boosting	0.4799
XGBoost	0.4798

Table 1. Initial performance of each regressor

Grid search with 10-fold cross validation method utilized in Gradient Boosting and XGBoost. Grid search is an exhaustive search through a manually specified subset of the hyperparameter space of a learning algorithm. Grid search trains the model with each combination of possible values of user-provided hyper-parameters on the training set and evaluates the performance on the test set. This method overcomes the downside of manual search of the best

hyper-parameters.[5] In order to evaluate the performance of the regressors with grid search 10-fold cross validation, we created manually RMSE as this metric was not provided in the available values of the scoring parameter. The results show that Gradient Boosting and XGBoost provided similar RMSE with XGBoost outperform Gradient Boosting.

Regressor	RMSE
Gradient Boosting	0.4790
XGBoost	0.4788

Table 2. Performance of Gradient Boosting and XGBoost

XGBoost regressor's best parameters derived from this method are presented in table 3 while the most important features are displayed in figure 3.

Parameter	Value
colsample_bytree	0.75
learning_rate	0.1
max_depth	6
min_child_weight	4
subsample	7

Table 3. Best parameters of GridSearchCV in XGBoost regressor

Note that different parameters in grid search would provide different results, but still this technique outperforms a single 10 cross validation with the default model's parameters in this dataset.

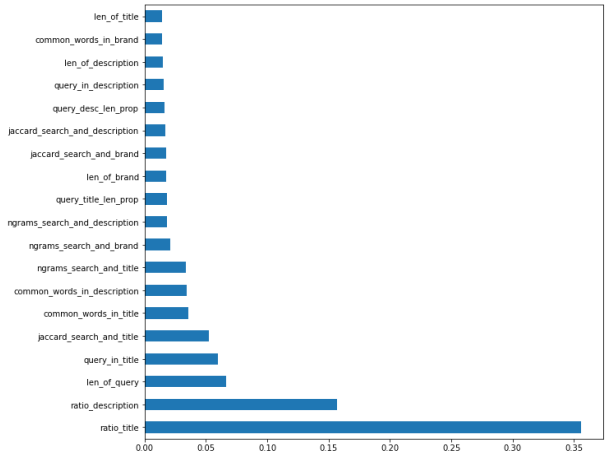

Figure 3. Feature importances of XGBoost regressor

Worth mentioning is that feature importances are derived given the parameters of grid search. Importance depicts a score indicating the usefulness of each feature in the construction of the boosted regression trees inside the model and it does not provide the importance of features in a business perspective.

IV. CONCLUSIONS

With the rapid growth of E-commerce platforms, product search engines that are able to successfully retrieve the most relevant products become more competitive and lead to satisfied customers.. Thus, most product search engines are based on the implementations of machine learning algorithms to make more accurate predictions and retrieve the most relevant product with respect to the user's query. In this paper, we proposed a methodology to predict the relevance score between a product and a search query based on given datasets. The conducted experiments comprise various NLP preprocessing techniques and machine learning algorithms to obtain the best result. The regression models derived from Linear Regression, Ridge, SVR, Random Forest, Gradient Boosting and XGBoost exhibit an initial RMSE under 0.51. The lowest error was achieved using XGBoost regressor with 0-fold cross validation in grid search method while similar performance was also obtained by Gradient Boosting regressor.

REFERENCES

[1] Uma K, M. Hanumanthappa. *Data Collection Methods and Data Preprocessing Techniques for Healthcare Data Using Data Mining*. International Journal of Scientific & Engineering Research Volume 8, Issue 6, June-2017. ISSN 2229-5518

[2] http://norvig.com/spell-correct.html

[3] Ron Koha. (1995) *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*

[4] J. Lawman, Preparing Data – Scaling and Normalization, October 2017.

[5] Jia Wu, XIu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei,Si-Hao Deng. *Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization*. DOI: https://doi.org/10.11989/JEST.1674-862X.80904120