Amy Edwards
William Chirciu
Final Project
CSC 575 - Online 810

Kaggle Ranking public: 8th
Kaggle Ranking private: 7th - 0.48571

The objective of this project was to predict the relevancy scores of product search results from HomeDepot.com. When a user enters a query into the search engine, they want to receive results that are significant. The results should be ranked, so that the results most relevant to the search appear first. If HomeDepot had a bad ranking algorithm, no one would want to shop online for their products. In the current age of internet shopping, this would limit their customer pool and negatively affect their profits. A user also expects their search to return a result extremely quickly. So minimizing the run time of the algorithm is critical.

We were given four data files to work with. The main data file was the training data with the following features: query id, product_uid, product_title, search_term, and relevance. Relevance is on an ordinal scale between 1 and 3. 1 signifies the query is not relevant to the document, 2 is mildly relevant, 3 is highly relevant. The supplementary data files contained the product descriptions and the product attributes. Not all products had attributes and some had more than one, but all products had a description. Our target variable is the average relevancy scores and is to be predicted across all search terms using a regression model. The relevance scores were determined by human raters that used information on the product, including photos, to rank the relevancy. We were given a set of data to test our regression model and submit our results to the Kaggle project page.

We first created a skeleton in order to get a general idea of the format and make our first submission. At minimum we needed one numerical feature that relates to the search query. Then we could run the training data through a regression in order to generate a model and perform predictions. We made a template with a variety of Regression Models to test: Linear, Decision Tree, Random Forest, Gradient Boosting, Ada Boosting, Support Vector Machine, and Neural Network. For evaluating the models, we looked at minimizing Root Mean Square Error and maximizing explained variance. RMSE is a measure of the differences between the observed values (ground truth) and the values (from a sample or population) that are predicted by a model or an estimator. The next sections will go through our individual and collaborative efforts to solve this problem.

Major Parts that Amy worked on ---

The first feature created was just the cosine similarity between the query and the product title. We then added cosine similarity between the query and product description and cosine similarity between the query and attributes (if any). We also added the number of words in the query, the description, and the title. At this point, Amy started to work on model evaluation and deciding which regression models have the best performance. She made a template with a variety of Regression Models to test: Linear, Decision Tree, Decision Tree with bagging, Random Forest, Gradient Boosting, Ada Boosting, Support Vector Machine, and Neural Network. To evaluate, she looked at minimizing Root Mean Square Error, maximizing explained variance, and runtime for each model. Decision Trees with boosting methods is producing the lowest RMSE values. Neural network and linear had similar results on the training set. But linear performed poorly on the Kaggle prediction set and neural networks take much longer to run than desired. Unfortunately, the support vector model is unable to run due to the size of the data. There is some literature that says that SVMs are great models for this type of regression, so this can be future work for parallel computing.

Regression Models, with cross validation								
	Linear	Decision Tree	Decision Tree with bagging	Random Forest	SVM	Gradient Boosted	Ada Boosted	Neural Network
RMSE	0.51 (+/- 0.02)	0.70 (+/- 0.02)	0.54 (+/-0.02)	0.52 (+/- 0.02)	n/a	0.50 (+/- 0.02)	0.51 (+/- 0.01)	0.51 (+/- 0.02)
Explained Variance	0.10 (+/- 0.04)	-0.76 (+/- 0.11)	-0.01 (+/- 0.05)	0.07 (+/- 0.05)	n/a	0.11 (+/- 0.05)	0.10 (+/- 0.03)	0.10 (+/- 0.05)
Runtime (s)	0.051	1.98	9.73	51.9	> 2 h	13.13	8.59	21.23

After performing univariate feature selection and calculating feature importance with Extra-Trees, we discovered that out of the six features we currently have, cosine between the title and the query and cosine between description and the query are unequivocally the most significant. We tried removing the least significant feature, count of words in the description, but it did not have any effect on the model RMSE. Yet, we had only unearthed 6 features so far, and tried one similarity metric. Next, we tested the model with Jaccard similarity between the query and the title instead of cosine. This did not have a statistically significant advantage over using cosine. However, including both similarity metrics in our model did help with the model score.

Next, Amy created a binary state column for if the product title or the product description contains the exact phrasing of the search query. For example, for the first query, “angle bracket”, if the product title or product id contains those words in that order with spaces surrounding them (to isolate the terms), then the column would have a 1 for True in it. This feature was called “exact” for exact phrasing. Products with descriptions or titles that match the exact phrase of the query should be more relevant because it is exactly what the user is looking for. We found this to be true, because when incorporating it into our model features, our RMSE decreased.

Amy also tried a binary feature based on if brand names were contained in the query, then they were also in the title, or description. The brand names were gathered from popular products on Home Depot.com. If the query’s product title or description has a brand name in it along with the query, that makes the search more specific. If it has a brand name in it, and the brand name is carried by Home Depot, then the search should return items specifically relating to that brand. Brands often have a theme of items that they make. So even if searching for one item from a brand brings up an additional item from the brand, it is likely related to the user’s query. When running the algorithm on our limited list of brands, we found we were getting very few hits. So we did not include this feature in our model because the vector was almost all zeros and would have not had any impact on our model. It would have been nice if Home Depot had a verified list of brands that they carry, to ensure that we are accounting for all of them.

Amy added features for the number of words in the query, number of words in the title and the number of words in the description. We thought that the brevity, or lack thereof, of the search query could indicate whether or not the person would receive a relevant result. Similarly, products that have a lengthy description or title have more words that can match in a search result. This means there are more chances for a user to search the “right” words that will show the items that they are attempting to find. We think this is important because a user does not always have the right words to communicate the item they are envisioning in their head.

She also added features of ratios for product titles and product descriptions. These are the ratio of the length of the title to the length of the query and the ratio of the length of the description to the ratio of the length of the query. For example, if the query length is 2 words and the title length is 8 words, the ratio is 8/2 or 4. We thought adding this detail might assist the model in predicting the relevancy. In addition to this ratio, we added a feature for the percentage of the search query in the product title and the percentage of search query in the product description. For example, for the query of “angle bracket”, and the title of “Simpson Strong-Tie 12-Gauge Angle”, the word angle appears in both the query and the title, but bracket does not. The ratio of common terms is 50% because half of the words in the search query appear in the title. If the query was “angle bracket strong” with the same title, the ratio of common terms would be 66% because 2 out of the 3 terms in the search query appear in the title.

Major parts that WIlliam worked on --

Because we had such a large dataset, figuring out how to minimize the run times of our algorithms was one of our top priorities. Because of this, William worked on building an inverted index. This involves creating a sorted dictionary of words that contains details such as their Inverse Document Frequencies (IDF) and their term frequencies within each document. An inverted index allows for fast text lookup at the cost of a very long setup time. The inverted index for the descriptions would take around 3 hours to build. However, this paid off in the long run when we started computing basic cosine similarities. What was taking upwards of 3 hours was now taking between 10 and 20 minutes. An inverted index was created for the project titles, descriptions, and attributes. A large issue that needed to be tackled was the way that the words were tokenized. The descriptions were incredibly messy and it was hard to match a lot of the words to the query, even though the query had a high relevance score with the document. After many attempts at formatting strings , we came upon a piece of code that manage to do this pretty well created by Susan Li [1] and sourced on GitHub. This coupled with tokenization led to more comprehensible terms and simpler comparison. After building the indexes and computing the tf x IDF vector lengths of the documents, we were ready to try computing the cosine similarities between the queries and the three types of documents available to us.

William used homework 4 as the template for computing the cosine similarities with an inverted index. From there it was just a matter of fitting it to our problem. After computing all of the cosines across the titles, descriptions, and attributes, we noticed that we were getting a lot of 0 values. To double check this validity, we went and combed through our original training/test data. A lot, if not most of these 0 values were unwarranted, as their corresponding queries had high enough relevance with the document. Obviously, this was a problem as the similarity values were not correctly representing the data. This is the point when we decided to incorporate WordNet.

WordNet is a lexical database for the English language. Using this, we are able to extract a list of synonyms, hypernyms, and hyponyms of a single word. Our thinking was that a lot of the time the exact word might not be in the document. However, if we were to search across all words related to our query term, we might get more hits and more accurate similarity scores. It is possible that the title has used similar words to the query, but not the exact search words. Accounting for this would increase the similarity between the search query and the product title and that can help return results that are relevant to the user. So when given a query term, we tried to get a match across all related words in the document. This was a bit tricky to implement as we had to consider that there might be more than 1 occurrence of a word in a document, both in its exact form and across its synonyms. So we took a hand at computing the document scores by including all words in the synsets, not just the first word encountered in the document. Once implemented we were getting a lot fewer zeros in our similarity lists. Our first approach where we tried to find the first query term (or related word) in the document and score it based on that, was the approach that actually made it in the model. We implemented the augmented approach searching for the query term and ALL related words in the document, but couldn't apply it to the model before the deadline due to extensive run times. However, we hypothesize that it would have further improved the RMSE on the test set.

Having done everything we could to create a robust cosine similarity algorithm, it was time to come up with some more features. William tried to explore word embeddings. Word embeddings are a vector representation of individual terms. Using this, we could take the average of these vectors to find the distance between terms. To get a workable feature, we were thinking we could find the magnitude of vectors. To explain more clearly, let's say we have the query "angle bracket". We would find a vector representation of 'angle' and a vector representation of 'bracket'. To get the average vector we would sum the vectors and divide by 2. From this we can calculate the magnitude using the square root of the sum of squares. This value would represent the 'word_embed' feature for this query. To get the actual word vectors, we had to download them from an external source called glove. This source contained pre-trained word vectors with up to 1000 dimensions. For the purposes of this project we stuck with 100 dimensions. Unfortunately, this process was cut short as a lot of the query terms could not be found in the vocabulary. Not knowing how to handle these cases, we decided to set the idea aside so as to work on engineering additional features.

Conclusion

After trying all these features, we worked on tuning our final model. We used Gradient Boosted Decision Trees with 5 fold cross validation. The parameters were: loss = 'ls', n_estimators = 100, and learning_rate = 0.1. Our set of features was with cosine between query and title, cosine between query and description, cosine between query and attribute, jaccard between query and title, count of words in query, count of words in the title, count of words in description, ratios between query length and title length, ratio between query length and description length, percent of search query in title, percent of search query in description, and the binomial feature for if the exact query phrase is in the title or description. We tried our best to make the model generalizable to the test set.

Our final model had RMSE 0.48571 which landed us in 7th place. We are happy to have improved our score so much since the beginning where we had around 0.52 RMSE. We felt that title and description were the best indicators of the relevancy of the query with respect to the resources we've been given. With more time, we could consider utilizing the product attributes in a more efficient way than just the cosine similarities.

A huge limitation in this project was computing power. It would take hours to run some of these algorithms and it was demoralizing to find out that it was done incorrectly. Having access to a stronger cpu or a distributed network would definitely help in further optimizing our solution. We would also like to test more regression methods, especially since SVM seems to be popularly used. However, we were unable to run it because of the excessive run times. Additionally being able to run the WordNet augmented similarities in a reasonable amount of time could have potentially improved our RMSE.

We think the notion of using brands and word embeddings as features is worth more consideration as well. We were unable to find a satisfying link between brands in the search query and brands in the title or description, but perhaps there is an avenue of relating the two that we have not thought of. As for word embeddings, it might have been worth trying to train the word vectors on our own corpus, as the number of words that actually existed in the glove vocabulary was limited with respect to Home Depot's domain.

Final Comments:

This project was hard to conceptualize in comparison to the homeworks assigned in class. The bulk of time was spent working on creating features for regression and fine tuning the model. It would have been better with more computing power as well, because a single machine was not enough to run SVM Regression or use full WordNet features. In addition, building the inverted indexes took hours. Making small changes such as altering tokenization techniques would set us back a while because the inverted indexes had to be reprocessed. Even computing the WordNet augmented similarities or performing brand matching was computationally taxing and often resulted in us having to interrupt the kernel so we could try running other algorithms.

On the other hand, we learned a lot about feature engineering which is a very important skill to have as a data scientist. Trying to extract information that wasn't obvious from the available data was an interesting, challenging task. We were also able to hone our programming skills both in correctness and efficiency. This was also our first Kaggle competition and we look forward to exploring this platform a lot more in the future.

[1] S. Li, "str_stem," *GitHubGist*, Oct-2018. [Online]. Available: https://gist.github.com/susanli2016/b83d148de7394821509bd5172d2c96d3