Kaggle – Home Depot Product Search Relevance

Problem statement

The objective of the home depot challenge is to find out the product which best matches the user input search text.
According to the description of the labeling data process, the search text will compare with both product title and product description. There are three values to represent the similarity: 1 (not match), 2 (somewhat match), 3 (perfect match).

Software version

spark, ipython, mongodb, pymongo, nltk, gensim

Iteration 1

Procedure

1. Import data into MongoDB (train, test, product_description)
2. Read data using spark, join product_description and training data according to product_uid
3. Preprocessing product_title, product_description and search_term fields using tokenizing, removing stop words and removing punctuation. I tried stemming, but word such as ‘milwaukee' will be changed to ‘milwak' which is totally different from its original meaning. Therefore, I skipped stemming part.

Feature engineering

1. Full search text match

It means to check whether the search text is fully matched with product_description or product_title. The output contains many 0 which means the full search text is not found in the product_description and product_title.

2. Unigram search text match

It means check whether each word in tokenized search_term exists in tokenized_title or tokenized_description. The output contains many non-zero values which means the split search text could find matches in the product_description and product_title.

3. Combination of full search text and unigram search text match

4. Synonym of each search term

Search synonym of each word in the search term, then calculate the similarity between synonyms and the product_description and product_title.

5. Bigram search text match

It means split the search_term, product_title, product_description in bigram way. The similarity is calculated using Jaccard coefficient.

6. LSI search text match
LSI is a topic model which can be applied to product_description & product_title. By identifying the topics of the product, we can directly match the search term with these topic.

Iteration 2

Improved feature engineering
I made some changes to the feature engineering in the previous week.
First, I removed the full text match case since the validation result shows that the probability for the whole search_term match with either product_title or product_description is really low. Such feature engineering would not lend much benefit to the model training.
Second, I changed the LSI topic model to LDA topic model. LDA is more intuitive since it converts a document to a set of meaningful topics which can be used for search_term matching.

2. End to end pipeline with new/improved features
I build up the entire data processing pipeline as below.

- Read data from mongoDB as DataFrame, split data into training, validation and test set
- Tokenize the search_term, product_description and product_title fields
- Define similarity calculator for each model
- Apply model specific data processing strategy on search_term, product_description and product_title fields
- Build the new feature column using similarity calculator
- Train RandomForestRegressor model
- Evaluate the model using validation set
- Make prediction using the test set
- Store the prediction result in mongoDB

3. Results and its interpretation (what you understand by the results).
Since I mainly used RandomForestRegressor model, the evaluation metric is the RMSE.

<u>Unigram</u>
The validation result shows the RMSE is around 0.50914 which is not very optimal.
I haven’t figure out the reason. Maybe it is due to the model configuration such as tree count, tree depth, etc.

<u>Bigram</u>
The validation result shows the RMSE is around 0.53346 which is higher than unigram, I think for below reason.
Currently, the bigram is built by combining words. For instance, if the search_term is [“A”,”B”,”C”], the bigram would be [“A B”,”B A”,”B C”,”C B”,”A C”,”C A”]. Such kind of combination is purely made by rearrange of positions without considering other aspects such as

semantics. Thus, the search hit of these bigrams would be lower than unigram.

Synonym
The validation result shows the RMSE is around 0.53349, which is higher than unigram, I think it is caused by the number of synonyms.
First, the synonym returned from nltk is limited. For specific words, there might be no synonyms. Second, I only considered one synonym of each search term. Ideally, I should involve more synonyms. But the cost would be more memory consumption which often lead to GC exception.

LDA
The validation result shows the RMSE is around 0.53347, which is higher than unigram. The reasons might be:
The generated topics for each field (product_description, product_title) may not be accurate. In this case, I chose to generate 5 topics for these fields. However, there might be record whose product_description or product_title field is too short to generate 5 topics. Another possibility is that, the topic may not include the key word that the search_term can match with. For instance, the original product_description might contain “gasoline” and the search_term contains “gasoline”, but the generated topic contains “energy”. Such kind of mismatch will bring negative impact to the RMSE result.

Model analysis
Based on the evaluation result, I would prefer to do further optimization for unigram and bigram model.

The reason for not choosing synonym model is that the optimal size of synonym is hard to decide. For instance, how many synonym should be considered for each term given different word count in the search term.

Another reason for not choosing synonym and topic model is that both of these models are memory and time consuming. Especially topic model which needs to convert the product_description, product_title into word count vector. Also, an index matrix needs to be created in order to retrieve the similarity between search_term and corresponding product. This process would cost a huge amount of time and often leads to memory issues.

Iteration 3

Improved feature engineering
The bigram prediction result was not optimal which is 0.655. The unigram prediction result was a little bit better which is 0.588. However, both of them were not good.
I made some changes to my feature engineering strategy.

1. For the search_term, I split it into unigram and bigram terms. Then I collected the numeric features like:
2. Number of matches in word unigram between query and title, query and description

3. Ratio of the unigram matches
4. Number of matches in word bigram between query and title, query and description
5. Ratio of the bigram matches
6. Topic similarity between query and title, query and description

2. Model training
I trained both regression model and classification model.

3. Optimization
I used ParamGrid to search for optimal tree number and tree depth for both RandomForestRegressor and RandomForestClassifier model.

4. Result analysis
After applying the new feature engineering, there’s only a tiny improvement on the score from 0.588 to 0.587. There could be several reasons.

First, I didn’t correct the typo in the search term. According to other students’ presentation, the typo had some negative impact on the final score. I searched online but didn’t find a good solution to do this. One possible way is to check typo using google search, but it doesn’t work under spark data frame UDF.
Another possible reason for the low score is that I didn’t use information such as product color, material, etc as an extra field to match the query_term. Since these information can be a strong indicator of high relevance.

Besides, I think the LDA topic model I used didn't bring the expected improvement to the result although it applies the TF-IDF and word2Vec on product title and description. The topic model might be way too far for this problem since I just need to check the similarity between search_term and product title & description.