Solution to the Competition Kaggle : Home Depot Product Search Relevance

Introduction:

In this competition, Home Depot was asking Kaggle’s contestants to help them improve their customers' online shopping experience by developing a model that can accurately predict the relevance of search results.

For that aim, Home Depot have provided a dataset containing 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 descriptions

- **train.csv** - training set, contains products, searches, and relevance scores. 74067 observations
- **test.csv** - the test set, contains products and searches. We had to predict the relevance for these pairs. 166 693 observations
- **product_descriptions.csv** - contains a text description of each product. 124 428 products
- **attributes.csv** - provides extended information about a subset of the products (typically representing detailed technical specifications). 2 044 802 attributes

Data fields

- **id** - a unique Id field which represents a (search_term, product_uid) pair
- **product_uid** - an 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

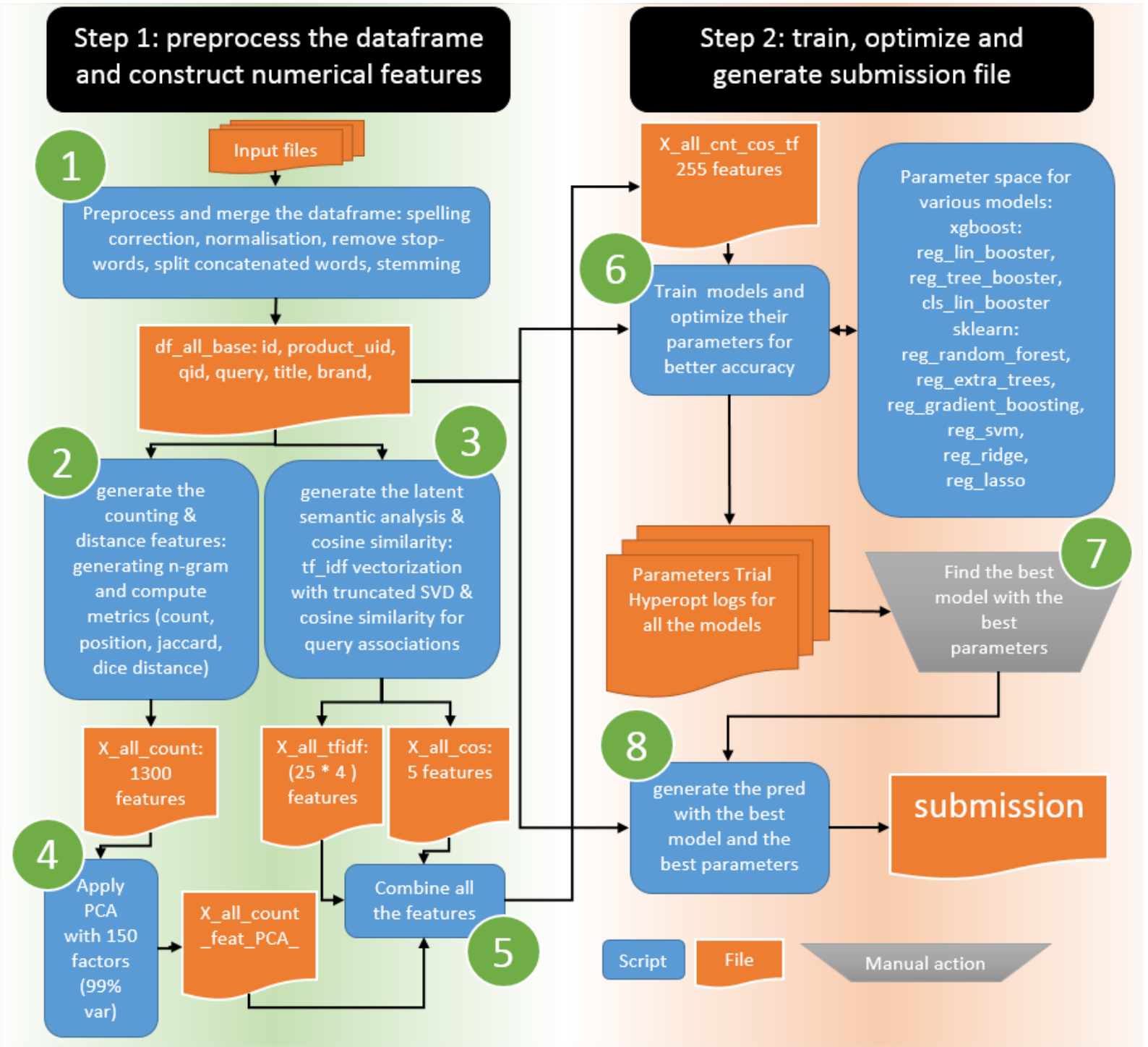
The solution:

Globally, there are two steps in the solution building.

- A first one dedicated to the preprocessing of the input files and features engineering for the models.
- A second one dedicated to the training, optimization of machine learning algorithms and generating predictions.

You can find this two steps in the Script folder, respectively with the 'feat' and 'model' subfolders.

Overview of the process:



A/ Generate the features

1/ preprocess the input dataset:

Script -> preprocess_input.py

1/ The choice of the input data to feed the model have focused on 4 fields. Of course, the "search_term" aka "query", but also, for the product information, the "product_title" aka 'title', the "product_description" aka "descr", and the brand aka "brand" which is an attribute in the corresponding file. The first step was so to design the desired dataframe by merging all this fields, mainly with the pandas library.

2/ The second operation was to normalize the text present in these fields, mainly with the reg exp "re" library:

- Lower all the characters
- Replace HTML codes
- Normalize digits
- Remove punctuation with exceptional treatment for special punctuation (example: " . " for decimal numbers, " / " or " * " for fraction or ratio, " ' " or " '' " for inch or feet measure, etc.)
- Split concatenated word-digit

3/ with this "semi-clean" data frame, the third step was to create a list of words as a reference for further treatment. Indeed, the exploration of the input data showed that some terms were unfortunately concatenated. For example, in this sentence (description for product 100001):

*"Versatile connector for various 90 connections and home repair **projectsStronger** than angled nailing or screw fastening **aloneHelp** ensure joints are consistently straight and **strongDimensions**"*,

we can see that some words are concatenated, as if the second word were the beginning of a new phrase.

To correct this, we first make a list of all words found in all fields, with counting the occurrence for each of them. This gives initially a dictionary of around 65 000 words… Using a dictionary of words costs based on the zipf's law and an algorithm to infer space, in addition with a list of English words generated with "big.txt" file, we correct the wrong words and minimize this list: 18 000 words at the end.

4/ the fourth operation was to apply different treatments on theses 4 fields.

- 1/ Eliminate the stop-words (words that are useless like "and", "the", "to", "for", etc.)
- 2/ Correct the wrong concatenated words directly in the dataframe this time

- 3/ Apply a spell correction with an algorithm trying to maximize the probability of potential candidates of correction using the reference list of words for word weighting probability
- 4/ Stem of the fields (just keep the stem of the words)
- 5/ Remove unknown stem based on the reference list.

5/ Write the final processed dataframe as a base for further computations. The same sentence above is now:

"versatil connector variou 90 connect home repair project stronger angl nail screw fasten alon help ensur joint consist straight strong dimens"

2/ Generate counting and distance features:

Script -> count_feat.py

Generating the counting & distance features in this section needed the generation of intermediary features called n-gram wich is simply the list of word found in the sequence with a n-concatenation.

Ex: *"versatil connector variou 90 connect home repair"* give the list:

- With n=1: ['versatil', 'connector', 'variou', '90', 'connect', 'home', 'repair']
- With n=2: ['versatil_connector', 'connector_variou', 'variou_90', '90_connect', 'connect_home', 'home_repair']
- With n=3: ['versatil_connector_variou', 'connector_variou_90', 'variou_90_connect', '90_connect_home', 'connect_home_repair']

This n-gram generation was heavy for the RAM and caused memory error so the workaround was to make the computation with splitting the data frame in 20 mini-batches.

With the n-grams generated, it was possible to compute these kind of features for each columns (query, title, brand, descr) and each n-grams:

Counting features:

- Basic features:
  - Count the number of n-grams
  - Count the unique number of n-grams and make the ratio "unique" on whole
  - Count digits and make the ratio of digits
- Intersect features:
  - Count the number of n-grams present in a column and also present in another one (for example how many n-gram in the query are also in the title)
  - Make their ratio with a focus on n-gram present in the query column
- Intersect Position features:
  - List the position of the n-gram of a column present in another one.
  - Apply basic statistics on it (min, max, median, mean, std)

Distance Features:

To know how likely a query is relevant to a product, the problem is to find some metrics that can express it.

The Jaccard and the Sorensen-Dice distance, which are 2 methods developed by botanists to measure the similarity of 2 different samples, are convenient statistics to describe it and are now widely used in the natural langage processing (NLP) field.

For each n-gram and each possible association, we compute these metrics.

3/ Latent semantic analysis and cosine similarity:

Script -> tfidf_feat.py

This part of the process make use of the so called tf-idf vectorization method which is widely used in the NLP field. As most of machine learning algorithms expect numerical feature vectors, the tf-idf vectorization is a method to transform raw text documents with variable length in a big sparse numerical features matrix. The process is as follow:

- **1/ tokenizing** words and giving an integer id for each possible token, for instance by using white-spaces as token separators.
- **2/ counting** the occurrences of tokens in each document. These first 2 steps are the vectorization part and are also known as the “bag of words” method.
- **3/ normalizing** and weighting with diminishing importance tokens that occur in the majority of documents. This third step use the tf-idf weighting and means term-frequency inverse document-frequency. Globally, it’s a method to appreciate the relevance of words in a document relatively to a whole corpus of document. For example, the word “the” would appear a lot of times in every document (if no stop-words discard are done) so the fact that it appears a lot of time in one document is not particularly relevant for this document and so his weight is low. In contrast, a technical word like say “hypochlorite”, would appears many times in a document whether not or very few in other documents means that this term is particularly relevant for this document and so his weight is high.

The result is a sparse matrix with dimensions m * n (with m = nb observations/documents, n = nb of possible words found in the whole corpus of documents), sparse because just a very few words appears in 1 document and the resulting matrix is mainly filled with 0. This matrix is too large to be exploited directly so we have to reduce the space using the Truncated Singular Value Decomposition (SVD). It is very similar to PCA (Principal Component Analysis), but operates on sample vectors directly, instead of on a covariance matrix. This means it can work with sparse matrices efficiently which is precisely our case. Applying a truncated SVD after a Tf-idf vectorization is known as the Latent Semantic Analysis. This dimension reduction is a necessary evil to permit further calculation, and, by finding a “smaller but better” space of representation, it construct “concepts” relatively to documents and words and permit to point out some synonymy in terms by highlighting the general idea.

1/ The first operation in this script was so to compute this latent semantic analysis for the whole corpus of text given by the query, title, brand and descr fields. The choice of the final dimension decomposition has been arbitrary chosen at 25 components for each fields, so the final dense matrix is 25 * 4 = 100 features. Note that the final variance is very poor for the entire data frame: 7.62% of explained variance… that's the game and increasing the numbers of factors didn't explained much more the variance. Furthermore, a choice had to made between explained variance and the number of features on which depend the speed of training the models in future section.

2/ The second operation was to compute the cosine similarity which simply measure the angle between two features vectors. This angle was computed respectively for the association between query and title, query and brand and query and descr. To achieve that goal, we compute the latent semantic analysis for these association respectively with n-gram = 1 and nb of SVD component = 150 and n-gram = 2 and nb of SVD component = 200. Then, we compute the cosine similarity for each association, so basically: 3 possible association * 2 set of parameters = 6 features of cosine similarity. Note that the query associated with the descr with ngram = 2 and nb component = 200 was too heavy for my RAM so I simply discarded it.

4/ Decomposition :

Script -> decompose.py (in model subfolder)

The count_feat script generates a matrix of around 1200 features, which are highly probably correlated between them. Indeed, the goals of that script is to run a decomposition (RandomizedPCA or TruncatedSVD) on this given dataset and see if fewer factors can explain almost the same amount of the initial dataset variance. After running the PCA and plot the explained variance by factors, it shows that the 25th first factors explained already around 80% of the initial features. This is non-surprising as the initial features was generated without regarding the relevance of each of them and are generated recursively with the same logic. However, keeping 150 features (instead of 25%) was typically a choice to keep the maximum of the initial variance with a threshold of 99% as this long tail could contain the information that could permit the future model to earn some point in the accuracy of the model.

5/ Combine :

Script -> combine.py

This little script make just the concatenation of the 3 files generated by the previous operation:

- 'X_all_count_feat_PCA_150' generated by the 'count_feat' and 'decompose' script
- 'X_cos_all' generated by the computation of cosine similarity in the 'tfidf_feat' script
- 'X_tfidf_all_dim_25' also generated by the 'tfidf_feat' script

To give the X_all_cnt_cos_tf file for the training of the models.

B/ Train, optimize the models and predict

The relevance of a pair (query, product) has been noted with a number between 1 (poor relevance) and 3 (high relevance) by at least 3 human raters. In the training set, we can observe …

Soon available…