Midterm Report

Yu Wen, Ehsan Rahiminasab, Shifu Xu,

1. Changes

We didn’t make any changes in our objective, or data;

Instead of using the methods in the proposal, we use the random forest to train our model, details please refer to section 3.

2. Data preprocessing

Since “test.csv” is only used for prediction, and contains no true labels(in our case, the “relevance score”) , we firstly create a development set “developmentTestSet.csv” by randomly select 20% data from “train.csv”; As for the remaining 80% data, “subTrainSet.csv”, we use it as our actual training data , this random sample selection was implemented in DevelopmentSet.java.

Then we compute the tf-idf with regards to product title and product description as our features.

As we can see, the most intuitive way to regards a “query” as more “relevant” to one product is that the query words have much “similarity” in the product title, which can be a good represent of one specific product. Intuitively, we can count the number of words of one query which appear in one specific product title (suppose we are interested in calculating the relevance score between this query and this product with this product title), but this have disadvantage: some words in the product title appears more than one time, indicating that if in one specific query, this word appears, we should give this query more “attention” since it has some “more important” word, however, only counting the number of words of one query which appear in one specific product title can’t achieve this.

As a result, we can use the term-frequency to indicate this relationship. Firstly, we have to extract the “interested item” of the product tittle such that we should remove the unrelated words appear in the product tittle such as “a”, “the”, ”for” and etc. which didn’t have any realistic meaning, which are also defined as stopping words. We achieved this preprocess by using the class “CountVectorizer” in “sklearn” package.

Then, we should found the “stem” of each word in the “search term” (also refferd as “query”) and each word in product tittles, since in English, “beauty” and ”beautiful” can have the same meaning when they were used for query. We achieved this preprocess by using the class “SnowballStemmer” from “nltk” package.

Since when we considering using product tittles to represent one product, we only have to consider different “interested items” appear in all the different product tittles. So we can treat all the product tittles as the corpus, which can be used to extract the vocabulary (the “interested item” as mentioned before). As long as we have this vocabulary of product tittle corpus, we can represent a product tittle t as an vector such that t=(t_1, t_2,..t_n), n=the number of vocabularies, the component t_i stands for the number of times vocabulary i appears in t; Also, we can use the vocabulary to build another vector to represent query q such that, q=(q_1, q_2,…q_n) such that n=the number of vocabularies and the component q_i stands for whether vocabulary i appears in q. Note the, q_i either equals to 1 which indicates vocabulary i appears in this q, or, equals to 0 if not. Clearly, the each vocabulary frequency of q (or **term frequency, also refered as “tf”** of q) appears t can be computed as t “dot multiply” q such that term frequency of q with respect to .

Although we can get the term frequency of q with regards of t as mentioned before, there are also drawbacks. Some words, though not “stop words” which didn’t contribute any meaningful meaning, are actually very common words in all the product tittle, which also will not make any meaningful meaning if one query contain it. In order to fix this problem we can compute the “idf” for each vocabulary. “idf” stands for **Inverse document frequency.** The most common way to compute “idf” is to use the methods of “inverse document frequency smooth” such that “idf” of one specific term or vocabulary is

$$idf = log(\frac{N}{1+n_t})$$

, where N is the number of whole documents in the corpus which also equals to number of different product tittle here; n_t is the number of different documents which contain this term(or t the number of different product tittles which contain this term). Clearly, if n_t is very common in all the product tittle $log(\frac{N}{1+n_t})$ will be a small value. if we multiply this with the term frequency, we can get a fair trade off: only when we encounter some term which are “locally frequent” but “globally rare”, can we count this term as relevant to this product tittle.Obviously, we add “1” in the logarithm part to avoid when n_t we will encounter an infinite quantity. Other ways to compute idf can be found on Wikipedia. Suppose we get the idf vector for each term, such that idf=(idf_1, idf_2, idf_3,…idf_n), so we can easily compute the tf-idf of q with respect to t such that

$$\text{"tf} - \text{idf"} = \sum_{i=1}^{n} t_i * q_i * idf_i$$

With this explanation, we can calculate the $"\mathrm{tf}-\mathrm{idf}"$ by batch by constructing a term frequency matrix and using the matrix multiplication. Details are in the implementation of “TfIdf-for-product-title-and-discription.py”.

Since the “product description” also contain rich information about one specific product, we assume the tf-idf for “product description” can also be very helpful for giving the relevance score. Note that, if we want to compute the tf-idf for a query with respect to some specific “product description”, we just need to change the corpus as “product description”.

In addition, since it’s possible that a search term with more numbers of words may contain more relevant information, we also use length of query as one feature.

3. Methodologies

For the base line, if we randomly guess the relevance score for each pair of query and product, we get a RMSE of 0.873. Details can be found in “RandomGuess.py”.

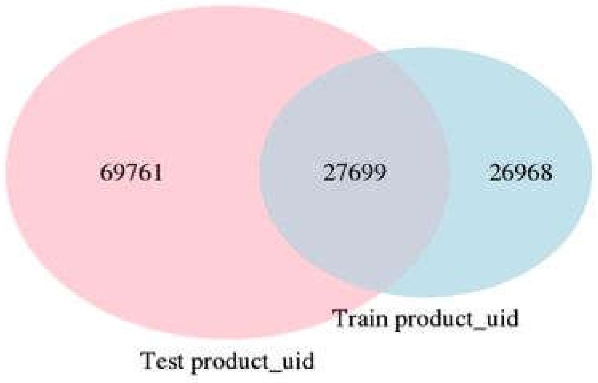
Firstly we tried regression by using SVM. The reason we are using SVM instead of the methodologies proposed in the proposal is that the proposed methods are too convoluted for our current “simple” data : for now we actually have changed the problem into an regression problem with only three input features: length of query, tf-idf for product tittle and tf-idf for product description. In addition, SVM has a high accuracy and also can be used for regression problems. As a result, we want to found out whether we can use an “simple” model SVM to achieve our goal. Another reason was that as we don’t have richful features, we could use SVM and kernels to transform the feature space to a higher dimension space, so maybe it helps to achieve better results.

The implementation of SVM is done in *SVM.py*. We have used *svm* from *sklearn* package in python.

Then we use the random forest to build another model. One reason we used this model is that, we found out that ensemble method is actually one good technique to improve the accuracy, since an ensemble have less possibility to make mistakes than its base regression model.

Another reason is that since our ground truth labels are driven from human raters, there is a strong possibility of existence of outliers and errors in our ground truth labels. Among the popular method of ensemble method such as bagging, boosting, and random forests, random forest are more robust to outliers and errors, so we choose this model to achieve our goal, and we build the random forest by using bagging in line with random attribute selection.

In addition, Random Forests are much easier to tune, and secondly they are harder to overfit. As we see in the following picture, many of the test query-product pairs have different paired product, hence we should watch carefully for the case of overfitting in the training set, therefore the use of a Random Forest for our problem seems to be a reasonable choice.



Random Forests can be used as a regression tool if we consider the average of the responses of the trees as our output, instead of just reporting the modest response.

The implementation of The Random Forest Regression model is done in *RandomForest.py*. We have used *RandomForestRegressor*, and *BaggingRegressor* from the package *sklearn.ensemble*.

4. Evaluation

Submissions in this competition are evaluated on the root mean squared error (RMSE). The use of RMSE is very common and it makes an excellent general purpose error metric for numerical predictions. We define RMSE as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$$

For the base line, if we randomly guess the relevance score for each pair of query and product, we get a RMSE of 0.873. Details can be found in “RandomGuess.py”.

The SVM model yielded the RSME of 0.57.

The Random Forest Method had yielded the RMSE of 0.48365.

This shows that Random Forest Method is more accurate than the SVM model for our data, and both the methods that we used have smaller error than random guessing, hence we are successful in yielding a model that does better than just randomly guessing the labels.

5. next step

1) In the preprocessing part:

- We should boost our methods of preprocessing, for example, the terms in CamelCase like DeckOver should be considered as two separate words, before stemming.
- For computing tf-idf score, we build a corpus consists of all product titles or product descriptions of the paired query-products in the training set, which is not true, because this will cause our corpus to have many repeated titles or descriptions, which yield to incorrect idf values. We should make a corpus of unique titles or descriptions.
- Use product attributes file to extract other features for our model.
- As numbers mostly represent sizes, we should consider whether to keep numbers in our corpus or not.

2) In the modeling part:

- We want to make the modeling to have two phases. In the first phase, we classify our data in two classes of relevant or irrelevant. Then in the second phase, we are going to use random forest regressor as a tool to do the regression job in the first and the second class. The idea behind this, is that we are going to model the brain of human raters, as a human, if we are going to score the relevancy of something, we first think is it relevant or not, then if it is relevant we try to give a good score, and if not a low score. We think that that two phases modeling can help to imitate this process.
- Playing with the attributes of the random forest regressor to cross validate that which attributes yield better RMSE.