1. Data Preparation

I use 5 columns: name, lvl1, lvl2, lvl3 and type.

First, I read csv files to dataframe using library pandas. I turn all words to lower cases and fill na cells.

Second, I find some words in column “name” which appear appropriate times. They cannot appear too much or too little. Appearing too much means most of items have these words, and they are meaningless in distinguishing different items. And Appearing too little means only a few items have them, then the matrix will be too sparse. I choose the words which appear between 120 and 200 times in “name” and store them in a “frequent” list.

Third, I scan all “name” cells and take out words which are not in that vector. Now all the words in “name” column are the “frequent” words.

Fourth, I use tfidf method to calculate a value for each words in each item’s name. Actually the reason why I need to search for “frequent” words is tfidf can only handle a relatively small words set. I have tried to put all the words to tfidf method, but the program get killed. I think memory cannot calculate that many words. Then for each item, I create a dictionary, which contains word-tfidf_value pairs. After sorting each dictionary, I retain three words which have largest tfidf value. Now each item’s “name” have different words, and they are all in the previous “frequent” list.

Fifth, I append column name, lvl1, lvl2, lvl3 and type together and use Tokenizer in keras to convert from words to numbers. In this way, we can get a matrix ready for analysis.

I think feature engineering is the most important part in this project, because which features one use have great impact on the final score. I have tried many feature combinations. I record my precess below:

Firstly I combined name, lvl1, lvl2, lvl3, description and type all together, without any clean on data, and used Tokenizer to convert words to numbers. Surprisingly, the score was not bad.

Secondly I cleaned data in “description”, replaced the whole description by the most frequent words(around 1000 to 2500) as I said before. I thought at least it should be better than the first score, but surprisingly again, I got a really bad score.

I rethink my method of cleaning, and I realized that I cannot use the most frequent words. Because they appear in too many items, which means they are useless for separating different items.

Then I wanted to use the tfidf method. I got the memory overflow problem as I said before. I thought I can combine the frequent method and the tfidf method: use tfidf only on a small set of words, instead of all words.

The most challenging part is determining the range of frequency. I tried lots of lower-bond and upper-bond, however, I got no better score.

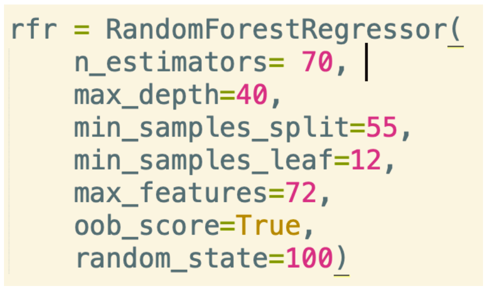
Finally, I used the “name” column instead of “description”, because I thought “description” has too many noise information, and it may influence my data preparation. I got a score a little bit better than the first one at last.

In addition, I use the tokenizer in keras instead of one hot encoding. Since there are too many dimensions of features, I thought one hot would make the matrix too long.

2. Model

I tried two models: RandomForestRegressor and neural network. The process on both of these two models are similar. I used ‘lvl1’, ‘lvl2’, ‘lvl3’ and ‘type’ and one-hot encoding in both of these models at first, then I used tfidf and tokenizer. They had similar score actually.

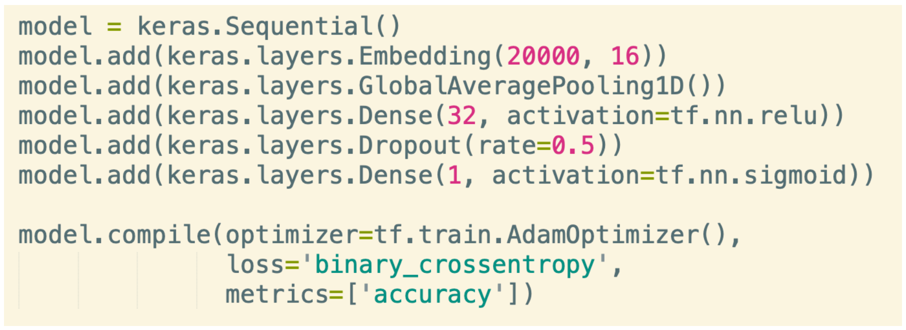
For RandomForestRegressor, I have spent lots of time on adjusting parameters, and it improves a lot, nearly 0.6.



```
rfr = RandomForestRegressor(
    n_estimators= 70, 
    max_depth=40,
    min_samples_split=55,
    min_samples_leaf=12,
    max_features=72,
    oob_score=True,
    random_state=100)
```

picture 1 RandomForestRegressor model

For neural network, the first score it gave is better than RandomForestRegressor. I also spent a lot of time adjusting kernel functions and parameters, but it improves a little. I thought the possible reason is that I’m not familiar with these kernel functions.



```
model = keras.Sequential()
model.add(keras.layers.Embedding(20000, 16))
model.add(keras.layers.GlobalAveragePooling1D())
model.add(keras.layers.Dense(32, activation=tf.nn.relu))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(1, activation=tf.nn.sigmoid))

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

picture 2 neutral network

3. Results

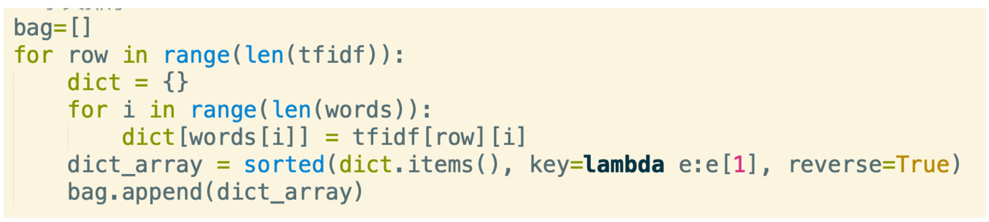
My best result for neutral network is 0.45903, and best score for RandomForestRegressor is 0.45953. I think 0.45 is a not bad score. I have tried many data preparation ways and adjust the parameters in models many times, so I believe 0.45 is nearly the best score under this model and data preparation. If I get another great ideas for processing data and new models, maybe the score can be better.

4. Lesson learned

1. Feature engineering is the most important part for data analysis. There are many possible feature combinations and you need to filter them first: which features are important, which are meaningless, which need to be cleaned, which need to be combined or separated. Actually I think if you can do feature engineering very good, you can have a good score whatever model you use.

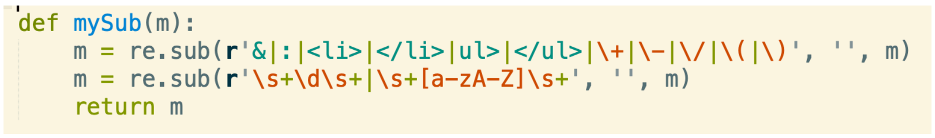
In my project, I didn't use the information in “description” and “price”, so my project still has room for improvement. In addition, for the features I used, I just simply append them together and use tokenizer to handle the string. I think a better way is adding bigger weights to lvl1, lvl2 and lvl3, because these features are relatively more important than the words I extracted from “name” by tfidf.

2. When I was doing feature engineering, I learned lots of operations in pandas and numpy, especially the convert between list, dictionary, series and dataframe. There are many great functions in pandas which can handle dataframe elegantly. For example, dataframe.apply() can parallel process cells in dataframe, it's much more faster than scanning the dataframe. I also use lambda expression when I sorting the dictionary.



```python
bag=[]
for row in range(len(tfidf)):
    dict = {}
    for i in range(len(words)):
        dict[words[i]] = tfidf[row][i]
    dict_array = sorted(dict.items(), key=lambda e:e[1], reverse=True)
    bag.append(dict_array)
```

picture 3 using lambda when sorting



```python
def mySub(m):
    m = re.sub(r'&|:|<li>|</li>|ul>|</ul>|\+|\-|\/|\(|\)', '', m)
    m = re.sub(r'\s+\d\s+|\s+[a-zA-Z]\s+', '', m)
    return m
```

picture 4 re.sub for filtering certain words

3. I learned a lot about RandomForestRegressor in sklean and neutral network in keras. They both are really lightweight model but can do pretty good estimation.

5. reference

http://yam.gift/2017/02/15/2017-02-15-list-dict-series-dataframe-ndarray-transform/, Convert between list, dictionary, series and dataframe.

https://blog.csdn.net/qq_22238533/article/details/76127966, Filter for dataframe.

https://blog.csdn.net/xiaodongxiexie/article/details/53108959, Operations on dataframe.

http://codingpy.com/article/a-quick-intro-to-pandas/, Operations on pandas.

https://keras.io, Keras.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, RandomForestRegressor.

https://zhuanlan.zhihu.com/p/27330205, Using tfidf to extract keywords.å