Capstone 3: Home Depot Product Search Relevance Prediction

Problem identification:

Shoppers rely on Home Depot's product authority to find and buy the latest products and to get timely solutions to their home improvement needs. Customers expect the correct results to their queries. Speed and accuracy are essential.

We want to develop a model that can accurately predict the relevance of the search results. We have a dataset which collect by human. People raters give a relevance score when do those survey.

There are three key datasets for this project: train, test and product_descriptions.csv. the challenge is to predict a relevance score for the provided combinations of search terms and products/

The relevance is a number between 1(not relevant) and 3(high relevant).For instance, 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).

Each pair was evaluated by at least three human raters.

File descriptions:

- train.csv - the training set, contains products, searches, and relevance scores
- test.csv - the test set, contains products and searches. You must predict the relevance for these pairs.
- product_descriptions.csv - contains a text description of each product. You may join this table to the training or test set via the product_uid.

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

Step 1: import data

Import train, test and product description data

```
df_train = pd.read_csv('C:/Users/wuhao/Desktop/springboard/capstone_three/train.csv', encoding ="ISO-8859-1")
df_test = pd.read_csv('C:/Users/wuhao/Desktop/springboard/capstone_three/test.csv', encoding ="ISO-8859-1")
```

```
df_desc = pd.read_csv('C:/Users/wuhao/Desktop/springboard/capstone_three/product_descriptions.csv')
```

```
df_train.head(5)
```

	id	product_uid	product_title	search_term	relevance
0	2	100001	Simpson Strong-Tie 12-Gauge Angle	angle bracket	3.00
1	3	100001	Simpson Strong-Tie 12-Gauge Angle	l bracket	2.50
2	9	100002	BEHR Premium Textured DeckOver 1-gal. #SC-141 ...	deck over	3.00
3	16	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	rain shower head	2.33
4	17	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	shower only faucet	2.67

```
df_desc.head(5)
```

	product_uid	product_description
0	100001	Not only do angles make joints stronger, they ...
1	100002	BEHR Premium Textured DECKOVER is an innovativ...
2	100003	Classic architecture meets contemporary design...
3	100004	The Grape Solar 265-Watt Polycrystalline PV So...
4	100005	Update your bathroom with the Delta Vero Singl...

Merge product description column to dataset

```
In [9]: df_all = pd.merge(df_all, df_desc, how = "left", on = 'product_uid')
```

```
In [10]: df_all.head(5)
```

Out[10]:

	id	product_uid	product_title	search_term	relevance	product_description
0	2	100001	Simpson Strong-Tie 12-Gauge Angle	angle bracket	3.00	Not only do angles make joints stronger, they ...
1	3	100001	Simpson Strong-Tie 12-Gauge Angle	l bracket	2.50	Not only do angles make joints stronger, they ...
2	9	100002	BEHR Premium Textured DeckOver 1-gal. #SC-141 ...	deck over	3.00	BEHR Premium Textured DECKOVER is an innovativ...
3	16	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	rain shower head	2.33	Update your bathroom with the Delta Vero Singl...
4	17	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	shower only faucet	2.67	Update your bathroom with the Delta Vero Singl...

Step 2: data processing and wrangling

Use stemmer to obtain the root forms of search_term, product_title and product_description.

```
[11]: ### text normalization, here we use stemmer first
```

```
[12]: stemmer = SnowballStemmer('english')
      def str_stemmer(s):
          return " ".join([stemmer.stem(word) for word in s.lower().split()])
```

```
[13]: ### counter thew  numbers of a word occourred
```

```
[14]: def str_common_word(str1, str2):
          return sum(int(str2.find(word)>=0) for word in str1.split())
```

```
[15]: #apply this method to one column
```

```
[16]: df_all['search_term'] = df_all['search_term'].map(lambda x: str_stemmer(x))
```

```
[17]: df_all.head(5)
```

t[17]:

	id	product_uid	product_title	search_term	relevance	product_description
0	2	100001	Simpson Strong-Tie 12-Gauge Angle	angl bracket	3.00	Not only do angles make joints stronger, they ...
1	3	100001	Simpson Strong-Tie 12-Gauge Angle	l bracket	2.50	Not only do angles make joints stronger, they ...
2	9	100002	BEHR Premium Textured DeckOver 1-gal. #SC-141 ...	deck over	3.00	BEHR Premium Textured DECKOVER is an innovativ...
3	16	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	rain shower head	2.33	Update your bathroom with the Delta Vero Singl...
4	17	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	shower onli faucet	2.67	Update your bathroom with the Delta Vero Singl...

```
[18]: #apply this method to all columns
```

```
[19]: df_all['product_title'] = df_all['product_title'].map(lambda x:str_stemmer(x))
```

```
[20]: df_all['product_description'] = df_all['product_description'].map(lambda x:str_stemmer(x))
```

Step 3: feature engineering

Use Levenshtein package to create two new features which based on the old features(tokenize the feature, which similar to create dummy features)

```
In [24]: import Levenshtein
         Levenshtein.ratio('hello','hello world')

Out[24]: 0.625

In [25]: df_all['dist_in_title'] =df_all.apply(lambda x:Levenshtein.ratio(x['search_term'], x["product_title"]), axis
         = 1)

In [26]: df_all['dist_in_desc'] = df_all.apply(lambda x:Levenshtein.ratio(x['search_term'],x['product_description']),
         axis=1)

In [27]: # create a new column which merge the product_ title and product description for construct a corpus

In [28]: df_all['all_texts']=df_all['product_title'] + ' . ' + df_all['product_description'] + ' . '

In [29]: df_all['all_texts'][:5]

Out[29]: 0    simpson strong-ti 12-gaug angl . not onli do a...
         1    simpson strong-ti 12-gaug angl . not onli do a...
         2    behr premium textur deckov 1-gal. #sc-141 tugb...
         3    delta vero 1-handl shower onli faucet trim kit...
         4    delta vero 1-handl shower onli faucet trim kit...
         Name: all_texts, dtype: object
```

Use TF-iDF to standardize the corpus and calculate the cos-similarity between two words. Finally, fill all empty vectors with 0's, ,we then have two new numerical feature columns.

```
In [37]: from gensim.similarities import MatrixSimilarity

         def to_tfidf(text):
             res = tfidf[dictionary.doc2bow(list(tokenize(text, errors = 'ignore')))]
             return res

         def cos_sim(text1, text2):
             tfidf1 = to_tfidf(text1)
             tfidf2 = to_tfidf(text2)
             index = MatrixSimilarity([tfidf1],num_features=len(dictionary))
             sim = index[tfidf2]
             return float(sim[0])

In [38]: ## do a test for the function above

In [39]: text1 = 'hello world'
         text2 = 'hello from the other side'
         cos_sim(text1, text2)

Out[39]: 0.8566456437110901

In [40]: #apply the function above to calculate the similarities of the three columns

           File "<ipython-input-40-b7e53b5213c0>", line 1
             apply the function above to calculate the similarities of the three columns
                     ^
         SyntaxError: invalid syntax

In [41]: df_all['tfidf_cos_sim_in_title'] = df_all.apply(lambda x: cos_sim(x['search_term'], x['product_title']), axis
         =1)

In [42]: df_all['tfidf_cos_sim_in_title'][:5]

Out[42]: 0    0.274539
         1    0.000000
         2    0.000000
         3    0.133577
         4    0.397320
         Name: tfidf_cos_sim_in_title, dtype: float64

In [43]: df_all['tfidf_cos_sim_in_desc'] = df_all.apply(lambda x: cos_sim(x['search_term'], x['product_description']),
         axis=1)

In [44]: df_all['tfidf_cos_sim_in_desc'][:5]

Out[44]: 0    0.182836
         1    0.000000
         2    0.053455
         3    0.043712
         4    0.098485
         Name: tfidf_cos_sim_in_desc, dtype: float64

In [45]: # drop all the non-numerical features for modeling
```

Step 4: split the data back to train, test set

step 4: spilit the data set back to train & test set

In [49]:
```
df_train = df_all.loc[df_train.index]
df_test = df_all.loc[df_test.index]
```

In [50]:
```
test_ids = df_test['id']
```

In [51]:
```
y_train = df_train['relevance'].values
```

In [52]:
```
X_train = df_train.drop(['id','relevance'],axis=1).values
X_test = df_test.drop(['id','relevance'],axis=1).values
```

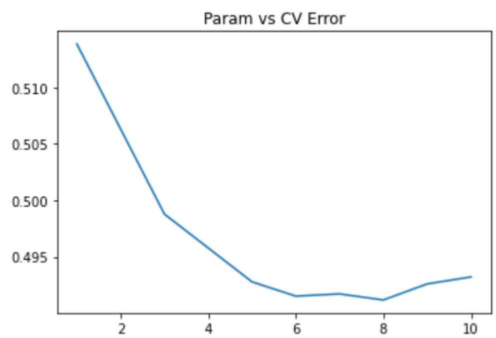
Step 5: model selection

We use Random Forest Regressor, Gradient Boosting Regressor and XGBregressor to predict. Base on different parameters, Random Forest Regressor with max_depth = 6 has the best performance.

In [55]:
```
params = [1,3,5,6,7,8,9,10]
rfr_test_scores = []


for param in params:
    rfr = RandomForestRegressor(n_estimators=30, max_depth=param)
    test_score = np.sqrt(-cross_val_score(rfr, X_train, y_train, cv=5, scoring='neg_mean_squared_error'))
    rfr_test_scores.append(np.mean(test_score))
```

In [56]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(params, rfr_test_scores)
plt.title("Param vs CV Error");
```



In [57]:
```
print(rfr_test_scores)
```

```
[0.5138314108832868, 0.4987813185023916, 0.4927863178745923, 0.4915212314009955, 0.49171811871436855, 0.491180
56549639794, 0.49260115763408513, 0.49321737945688493]
```

Step 6: show result and submission

Step 6: Choose the best model and caculate the result

```
In [64]: rfr = RandomForestRegressor(n_estimators=30, max_depth=6)

In [65]: rfr.fit(X_train, y_train)

Out[65]: RandomForestRegressor(max_depth=6, n_estimators=30)

In [66]: y_pred = rfr.predict(X_test)

In [67]: pd.DataFrame({"id": test_ids, "relevance": y_pred}).to_csv('submission.csv',index=False)

In [70]: y_pred

Out[70]: array([2.45651444, 2.0455801 , 2.20320464, ..., 2.59010177, 2.47996452,
                2.27970175])
```

Conclusion: There are many ways that can improve the result. I will just list a few here:

1. Do more steps in the processing part. For example, we can delete the stop words, delete the numbers or correct some wrong spell etc.
2. Tokenize the text with some new corpus and weighted the vectors numerical numbers.
3. Use Word2Vec to construct new features which use the similar way to TF-iDF's
4. Optimized the other parameters.
5. Model ensemble.