# Home Depot Search and Retrieval

| Prepared by | Christian Craig<br>Vineet D'cunha |
|---|---|
| Date | 3/15/2021 |

Dataset: https://www.kaggle.com/c/home-depot-product-search-relevance/data

## Part I. Introduction

We are proposing a search / retrieval system that will attempt to show the relevance of items in a search. With our dataset we are given search and product pairs then given the task of predicating relevance for these pairs. The products have text descriptions which we can group to obtain features, then create an inverted index from. We will investigate further methods of extracting features, aside from the inverted index. We would be implementing a GUI for the search as well.

This data set contains several products and real customer search terms from Home Depot's website. Our dataset come with 2 separate components. These include a training set and product descriptions.

**File descriptions**

**train.csv** - the training set, contains products and searches

**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.

## Part II. Data Set Overview / Exploratory Analysis

Below, we can see the training set. There are 5 features, and 74066 rows (not including the header).

```
In [14]: df_train = pd.read_csv('train.csv',encoding = "ISO-8859-1")
         df_train.head()
```

Out[14]:

| | 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 |

```
In [16]: df_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74067 entries, 0 to 74066
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             74067 non-null  int64
 1   product_uid    74067 non-null  int64
 2   product_title  74067 non-null  object
 3   search_term    74067 non-null  object
 4   relevance      74067 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 2.8+ MB
```

There are 11795 unique words in the training set.

***Finding number of unique search terms in training set***

```
In [19]: train_search_terms = df_train['search_term'].unique
```

```
In [20]: print('Number of search words in training set')
         len(pd.unique(df_train['search_term']))

         Number of search words in training set

Out[20]: 11795
```

Word cloud to show the frequencies of different words in the training set.

```
In [26]: data_train = dict(zip(df_train_keywords['index'].tolist(), df_train_keywords[0].tolist()))
         wc_train = WordCloud(width=800, height=400, max_words=200).generate_from_frequencies(data_train)
         plt.figure(figsize=(10, 10))
         plt.imshow(wc_train, interpolation='bilinear')
         plt.axis('off')
         plt.show()
```


Below are the product descriptions. We will try to extract features out of these. There are 2 features, and 124427 rows.

```
In [23]: df_desc = pd.read_csv('product_descriptions.csv',encoding = "ISO-8859-1")
         df_desc.head()
```

Out[23]:

| | 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... |

```
In [24]: df_desc.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 124428 entries, 0 to 124427
         Data columns (total 2 columns):
          #   Column               Non-Null Count   Dtype
         ---  ------               --------------   -----
          0   product_uid          124428 non-null  int64
          1   product_description  124428 non-null  object
         dtypes: int64(1), object(1)
         memory usage: 1.9+ MB
```

Product description without stop words and populated in lower case.

In [47]:
```
df_desc['product_description_without_stopwords'] = df_desc['product_description'].apply(la
df_desc['product_description_without_stopwords'] = df_desc['product_description_without_st
df_desc['product_description_without_stopwords'] = df_desc['product_description_without_st
df_desc['product_uid'] = df_desc['product_uid'].astype(str)
df_desc.head()
```

Out[47]:

| | product_uid | product_description | product_description_without_stopwords |
|---|---|---|---|
| **0** | 100001 | Not only do angles make joints stronger, they ... | not angles make joints stronger also provide c... |
| **1** | 100002 | BEHR Premium Textured DECKOVER is an innovativ... | behr premium textured deckover innovative soli... |
| **2** | 100003 | Classic architecture meets contemporary design... | classic architecture meets contemporary design... |
| **3** | 100004 | The Grape Solar 265-Watt Polycrystalline PV So... | the grape solar 265watt polycrystalline pv sol... |
| **4** | 100005 | Update your bathroom with the Delta Vero Singl... | update bathroom delta vero singlehandle shower... |

## Part III. Preparing / Implementing Data for Inverted Index and Search Retrieval

**Function dict_desc()**

To split words for each product_uid. Creates a dictionary with each product_uid as the key and the product descriptions split into a list.

```
In [127]: from collections import Counter
          def dict_desc(id_col, desc_col):
              desc_dict = {}
              for k,v in zip(id_col,desc_col):
                  desc_dict[k] = v.split()
              return desc_dict

In [128]: desc_dict = dict_desc(test_df['product_uid'],test_df['product_description_without_stopwords'])
```

Output:

```
In [129]: desc_dict

Out[129]: {'100001': ['not',
            'angles',
            'make',
            'joints',
            'stronger',
            'also',
            'provide',
            'consistent',
            'straight',
            'corners',
            'simpson',
            'strongtie',
            'offers',
            'wide',
            'variety',
            'angles',
            'various',
            'sizes',
            'thicknesses',
```

**Function setup_dict() and total_freq()**

Calculates the frequency of each word as part of product descriptions. Returns the results in sorted order based on the frequency of the words.

```
In [121]: def setup_dict(dict1): #
              new_dict = {}
              for item in dict1.values():
                  for w in item:
                      if w not in new_dict:
                          new_dict[w] = 0
                      if w in new_dict:
                          new_dict[w] = new_dict[w]+1
                          #sort dict
              org_dict = OrderedDict(sorted(new_dict.items(),key = itemgetter(1),reverse = True))
              return org_dict

In [130]: word_dict = setup_dict(desc_dict)
```

In [133]:
```python
def total_freq(dict1):
    total_freq = dict(Counter(dict1))
    return total_freq
```

In [134]:
```python
total_freq = total_freq(word_dict)
```

Output:

In [135]:
```python
total_freq
```

Out[135]:
```
{'in': 198044,
 'the': 122385,
 'x': 65488,
 'easy': 58380,
 'use': 55764,
 'this': 52446,
 'design': 43740,
 'finish': 40277,
 'water': 38761,
 'door': 37224,
 'steel': 37206,
 'ft': 36754,
 'installation': 36646,
 'provides': 32596,
 'home': 32346,
 'features': 32180,
 'light': 31706,
 'wood': 30291,
```

Above, we can see the word with corresponding word count, which will be used for the total frequency in the inverted index.

**Function doc_freq()**

Calculates the document frequency for each word as part of product descriptions. Returns the results in sorted order based on the frequency of the words.

```
In [277]: def doc_freq(dict1):
              d = {}
              doc_lst = []
              for k,v in dict1.items():
                  #print(v) --> sim to doc_lst exmpl assignment1
                  for w in v:
                      if w in d.keys():
                          d[w]= d[w] + 1
                      else:
                          d[w]=1
              return d
```

```
In [278]: doc_freq = doc_freq(desc_dict_dedup)
          doc_freq
```

```
Out[278]: {'x': 2839,
           'screws': 431,
           'resistanceinstall': 15,
           'moisture': 372,
           'alonehelp': 4,
           'screw': 231,
           'common': 302,
           'sizes': 394,
           'match': 363,
           '112': 156,
           'numberversatile': 4,
           'corrosion': 354,
           'various': 219,
           'straight': 171,
           'skewed': 6,
           'model': 308,
           'provide': 1503,
           'nails': 206,
           'jobs': 135,
           'zmax': 5,
           '12gauge': 29
```

**Function idf()**
Retrieves the number of documents and calculates the idf values associated with each term.

In [279]:
```
def idf(dict1):
    d = {}
    d_idf = {}
    n = len(dict1)
    for k,v in dict1.items():
        #print(v) --> sim to doc_lst exmpl assignment1
        for w in v:
            if w in d.keys():
                d[w]=d[w]+1
                d[w] = round(np.log2(n/d[w]),3)
            else:
                d[w] = 1
                d[w] = round(np.log2(n/d[w]),3)
    for k2,v2 in d.items():
        #result_idf = round(np.log2(n/v2),3)
        d_idf[k2]= round(np.log2(n/v2),3)

    return d_idf
#idf(desc_dict)
```

In [280]:
```
idf = idf(desc_dict)
```

In [281]:
```
idf
```

```
 'images': 9.988,
 'insinkerator': 9.988,
 'sinktop': 9.98,
 'single': 9.988,
 'outlet': 9.988,
 'disposers': 9.98,
 'stylish': 9.988,
 'alternative': 9.988,
 'airactivated': 9.98,
 'mounts': 9.988,
 'sink': 9.988,
 'countertop': 9.988,
 'island': 9.988,
 'installationskit': 9.556,
 'satin': 9.988,
 'nickel': 9.988,
 'buttons': 9.988,
 'complement': 9.988,
 'decorcompatible': 10.047,
 'ac': 9.988,
 ...}
```

Above we can see the IDF value with the corresponding term. The IDF value helps tone down more frequent words across the documents. If a word occurs in various documents, then the IDF value will be lower.

**Function inv_idx()**

The below function gathers all of the search term with the corresponding Total Frequency, Document Frequency, and IDF value. Given the size of our data, we found it not optimal to calculate the list of the count of each document the search term appeared in.

```
In [284]: def inv_idx(dict1):
              keys = sorted(set(list(dict1.keys())))
              doc_lst = []
              new_dict = {}
              print('IdxTerm','--','TotalFreq','--','DocFreq','--','IDF')
              print()
              for w in keys:
                  if w in dict1.keys():
                      dict1_amt = dict1[w]
                      doc_lst.append(dict1_amt)

                  print(w,'----------',total_freq[w],'----------',doc_freq[w],'-----
          inv_idx(word_dict)
```

```
IdxTerm -- TotalFreq -- DocFreq -- IDF

0 ---------- 170 ---------- 149 --------> 9.988
00 ---------- 3 ---------- 3 --------> 9.98
000 ---------- 2 ---------- 2 --------> 10.047
000125uses ---------- 1 ---------- 1 --------> 9.556
001 ---------- 8 ---------- 5 --------> 9.988
0012 ---------- 2 ---------- 2 --------> 10.047
0017 ---------- 1 ---------- 1 --------> 9.556
002 ---------- 6 ---------- 5 --------> 9.988
0020 ---------- 4 ---------- 3 --------> 9.989
0024 ---------- 1 ---------- 1 --------> 9.556
0025 ---------- 2 ---------- 1 --------> 10.047
0028 ---------- 1 ---------- 1 --------> 9.556
003 ---------- 1 ---------- 1 --------> 9.556
0030 ---------- 2 ---------- 2 --------> 10.047
004 ---------- 1 ---------- 1 --------> 9.556
005 ---------- 1 ---------- 1 --------> 9.556
0050 ---------- 4 ---------- 4 --------> 9.989
0050gauge ---------- 1 ---------- 1 --------> 9.556
0060 ---------- 1 ---------- 1 --------> 9.556
```

**Function doc_length()**
Calculates the document length for each product_id. The function counts the distinct number of words in a product_id. Multiplies the values based on number of occurrences and idf value calculated from the previous function.

In [54]:
```
def doc_length(desc_dict):
    lst_dup =[]
    len_dict = {}
    for k,v in desc_dict.items():
        doc_len = None
        total_len = 0.0
        for distinct_value in v:
            if distinct_value not in lst_dup:
                lst_dup.append(distinct_value)
                term_count = v.count(distinct_value)
                idf_count = idf[distinct_value] * term_count
                doc_len = idf_count * idf_count
                total_len = total_len + doc_len
        len_dict[k]=np.sqrt(total_len)
    return len_dict
```

In [55]:
```
len_dict = doc_length(desc_dict)
```

In [120]:
```
len_dict
```

Out[120]:
```
{'100001': 48.853021953611005,
 '100002': 56.20488304409147,
 '100003': 42.20827999338518,
 '100004': 56.420321941655004,
 '100005': 37.74800200010592,
 '100006': 123.44514583004086,
 '100007': 40.886894844191836,
 '100008': 39.10387255758694,
 '100009': 27.54730093856747,
 '100010': 57.27060830827623,
 '100011': 62.355737586849166,
 '100012': 82.21505491088597,
 '100013': 33.29183641675539,
 '100014': 36.50777999276319,
 '100015': 34.46768860831838,
 '100016': 55.588154709434264,
 '100017': 30.09150685824823,
 '100018': 19.828828734950534,
 '100019': 31.870497736935334,
 '100020': 16.80937003578659,
 '100021': 45.787966683398395,
 '100022': 60.52371806159959,
```

By obtaining the length normalization, we can now accurately measure the impact the document length has on the document ranking.

**Function search_retrieval()**
This function searches the product_description dataframe based on the input values entered by the user. The process splits input values and the determines the idf value found from the previous function to multiplies it. The value is stored in the dictionary having the product_id as the key.
The process continues for all the words incrementing the value of dot product in the dictionary.
The value of dot product is normalized by multiplying against the document length of the input value and length of the documents.

Finally we sort and display the retrieved product_id and product descriptions in decreasing order of the similarity measure.

```
In [57]: def search_retrieval():
    search_term = entry1.get()
    search_term = search_term.lower()
    search_sim_dict = {}
    search_word_length=0.0
    search_length=0.0
    for i in search_term.split():
        for k,v in desc_dict.items():
            doc_term_count = 0
            idf_val = 0.0
            if i in v:
                doc_term_count = v.count(i) # retrieve count of occurences in every key
                idf_val = idf[i] # retrieve the idf value for the word
            if k in search_sim_dict.keys():
                search_sim_dict[k]= search_sim_dict[k] + (doc_term_count*idf_val*idf_val)  # calculate the dot product
            else:
                search_sim_dict[k]= doc_term_count*idf_val*idf_val
            if i in idf:
                search_word_length = search_word_length + (idf[i]*idf[i]) # calculate the length for the search term
    search_length = np.sqrt(search_word_length)
    sim_normalize_dict ={}
    x = []
    for k1,v1 in search_sim_dict.items():
        if v1 > 0.0:
            sim_normalize_dict[k1]=v1/(search_length.astype('float')*len_dict[k1].astype('float')) # normalize the value of D
    import operator
    sim_normalize_dict = dict( sorted(sim_normalize_dict.items(),key=operator.itemgetter(1),reverse=True)) # sort the records

    for k2, v2 in sim_normalize_dict.items():
        x.append([k2,df_desc.loc[df_desc['product_uid']==k2].product_description.to_string(index=False)]) # Get the product de
```

## Part IV. Benchmark Dataset for Precision and Recall Evaluation

In the following segment we will be preparing and implementing a benchmark dataset, so we can see the precision and recall of the selected queries that receive.

### Function evaluate_search_results()

This function processes a random sample of inputs from the training. The process splits the search terms and determines the idf value found from the previous function to multiplies it. The value is stored in the dictionary having the product_id as the key.
The process continues for all the words incrementing the value of dot product in the dictionary.
The value of dot product is normalized by multiplying against the document length of the input value and length of the documents.

Finally we sort he display the retrieved product_id and product descriptions in decreasing order of the similarity measure.
A dictionary is created with the search term as the key and list of retrieved product_id's as the values.

```
In [47]: def evaluate_search_results():
    search_sim_dict = {}
    search_word_length=0.0
    search_length=0.0
    sim_normalize_dict ={}
    keyword_dict = {}
    for j in search_keywords:
        j = j.lower()
        for i in j.split():
            for k,v in desc_dict.items():
                doc_term_count = 0
                idf_val = 0.0
                if i in v:
                    doc_term_count = v.count(i) # retrieve count of occurences in every key
                    idf_val = idf[i] # retrieve the idf value for the word
                if k in search_sim_dict.keys():
                    search_sim_dict[k]= search_sim_dict[k] + (doc_term_count*idf_val*idf_val)  # calculate the dot product
                else:
                    search_sim_dict[k]= doc_term_count*idf_val*idf_val
                if i in idf:
                    search_word_length = search_word_length + (idf[i]*idf[i]) # calculate the length for the search term
            search_length = np.sqrt(search_word_length)

        x = []
        for k1,v1 in search_sim_dict.items():
            if v1 > 0.0:
                sim_normalize_dict[k1]=v1/(search_length.astype('float')*len_dict[k1].astype('float')) # normalize the value c
        import operator
        sim_normalize_dict = dict( sorted(sim_normalize_dict.items(),key=operator.itemgetter(1),reverse=True)) # sort the rec
```

Below we will take out training data, recreate to a data frame that just shows the search_term, product_uid, and relevance.

```
In [64]: df_train.head()
```

Out[64]:

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

```
In [65]: rel_df = df_train[['search_term','product_uid','relevance']]
         rel_df
```

Out[65]:

| | search_term | product_uid | relevance |
|---|---|---|---|
| **0** | angle bracket | 100001 | 3.00 |
| **1** | l bracket | 100001 | 2.50 |
| **2** | deck over | 100002 | 3.00 |
| **3** | rain shower head | 100005 | 2.33 |
| **4** | shower only faucet | 100005 | 2.67 |
| **...** | ... | ... | ... |
| **74062** | tv riser glass | 206638 | 1.00 |
| **74063** | r20 halogen light | 206639 | 3.00 |
| **74064** | schlage lock siena half dummy knob with | 206641 | 2.33 |
| **74065** | zen garden decor | 206648 | 3.00 |
| **74066** | fine sheer curtain 63 inches | 206650 | 2.33 |

Next we create the sample dataframe that coverts the dictionary output of the evaluate_search_results() function into the dataframe. The sample data is then merger product_uid and search_term from the training set.

In [287]:
```
sample = pd.DataFrame(dict([ (k,pd.Series(v)) for k,v in keyword_dict.items() ])).melt().dropna()
```

In [288]:
```
#sample.columns = ['search_term','product_uid']
sample.head()
```

Out[288]:

| | variable | value |
|---|---|---|
| 0 | backsplash tile kitchen | 101321 |
| 1 | backsplash tile kitchen | 101354 |
| 2 | backsplash tile kitchen | 102018 |
| 3 | backsplash tile kitchen | 102560 |
| 4 | backsplash tile kitchen | 103512 |

In [73]:
```
sample['value'] = sample['value'].astype(int)
```

Merged dataset retruned from retrieval process and the sample.

The merged dataset can be used to calculate precision and recall

In [75]:
```
smpl_df = pd.merge(df_train[['product_uid', 'search_term', 'relevance']], sample, how='left', left_on=['search_term','p
smpl_df
```

Out[75]:

| | product_uid | search_term | relevance | variable | value |
|---|---|---|---|---|---|
| 0 | 100001 | angle bracket | 3.00 | NaN | NaN |
| 1 | 100001 | l bracket | 2.50 | NaN | NaN |
| 2 | 100002 | deck over | 3.00 | NaN | NaN |
| 3 | 100005 | rain shower head | 2.33 | NaN | NaN |
| 4 | 100005 | shower only faucet | 2.67 | NaN | NaN |
| ... | ... | ... | ... | ... | ... |

As we can see there are quite a few null values, so we remove those, and are left with 116 matching queries to test for recall and precision. In comparison the TIME benchmark set, this is still higher than that, as they only had 84. It is also important to note that since our evaluate_search_terms() was on a 10000 word subset of our dataset that this isn't the max amount of queries we could obtain, but for storage and speed reasons, this works better.

```
In [104]: smpl_df = smpl_df.dropna()
```

```
In [103]: del smpl_df['variable']
          del smpl_df['value']
```

```
In [106]: smpl_df
```

Out[106]:

| | product_uid | search_term | relevance |
|---|---|---|---|
| 54 | 100033 | steel shelving | 2.67 |
| 460 | 100264 | samsung front load dryer | 2.67 |
| 532 | 100303 | antenna pole | 2.67 |
| 738 | 100410 | air conditioner 12000 15x23 | 2.33 |
| 773 | 100429 | steel shelving | 3.00 |
| ... | ... | ... | ... |
| 12941 | 109539 | drill auger | 1.67 |
| 12946 | 109544 | 10 pk duplex brown | 1.33 |
| 12985 | 109586 | sliding cabinet drawers | 3.00 |
| 13058 | 109645 | rolling cart | 2.33 |
| 13254 | 109823 | corbels ad post tops | 3.00 |

116 rows × 3 columns

Below we performed operations to add in our recall and precision. As discussed in class, precision is measured by the # of relevant documents retrieved over the # of total documents retrieved. Recall is calculated by the # of relevant documents retrieved over the # of total relevant documents.

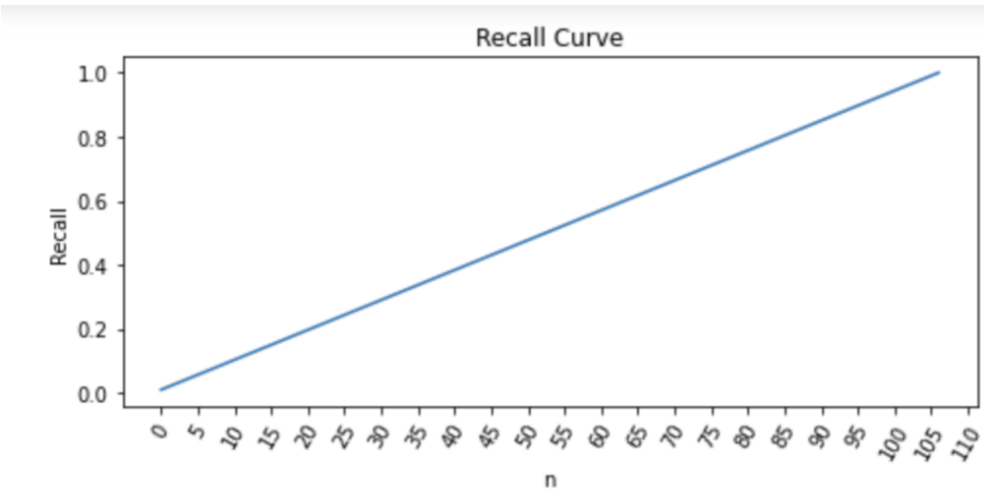
Out[192]:

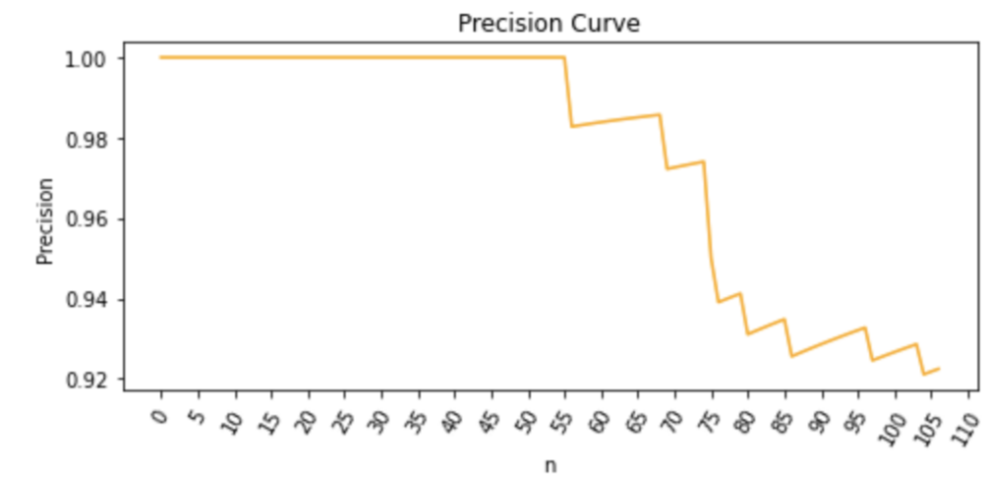
| | product_uid | search_term | relevance | relevant? | relevance_count | Recall | n | Precision |
|---|---|---|---|---|---|---|---|---|
| 54 | 100033 | steel shelving | 2.67 | yes | 1 | 0.009346 | 1 | 1.000000 |
| 460 | 100264 | samsung front load dryer | 2.67 | yes | 2 | 0.018692 | 2 | 1.000000 |
| 532 | 100303 | antenna pole | 2.67 | yes | 3 | 0.028037 | 3 | 1.000000 |
| 738 | 100410 | air conditioner 12000 15x23 | 2.33 | yes | 4 | 0.037383 | 4 | 1.000000 |
| 773 | 100429 | steel shelving | 3.00 | yes | 5 | 0.046729 | 5 | 1.000000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

In the “relevant?” column, we created a binary threshold. Since the relevance score corresponding with the product_uid, is on a scale of 1 to 3, we decided to make the threshold 1.5. Anything above or equal to 1.5 will be relevant, and anything below 1.5 will be not relevant.

**Part V. Visualizing Recall and Precision**

Below we can see the plots of the recall and precision curve. The recall curve shows, that as we look at more relevant documents, the recall increases. We eventually reach 100% recall at document 107. With Precision it is the opposite. As we look at more documents, the precision decreases. That being said, precision still maintains a fairly high rate of around 93%. We can also see that we have an average recall of about 50% and an average precision of 98%.




```
Average Recall: 0.5046728971962616
Average Precision 0.9758483273983114
```

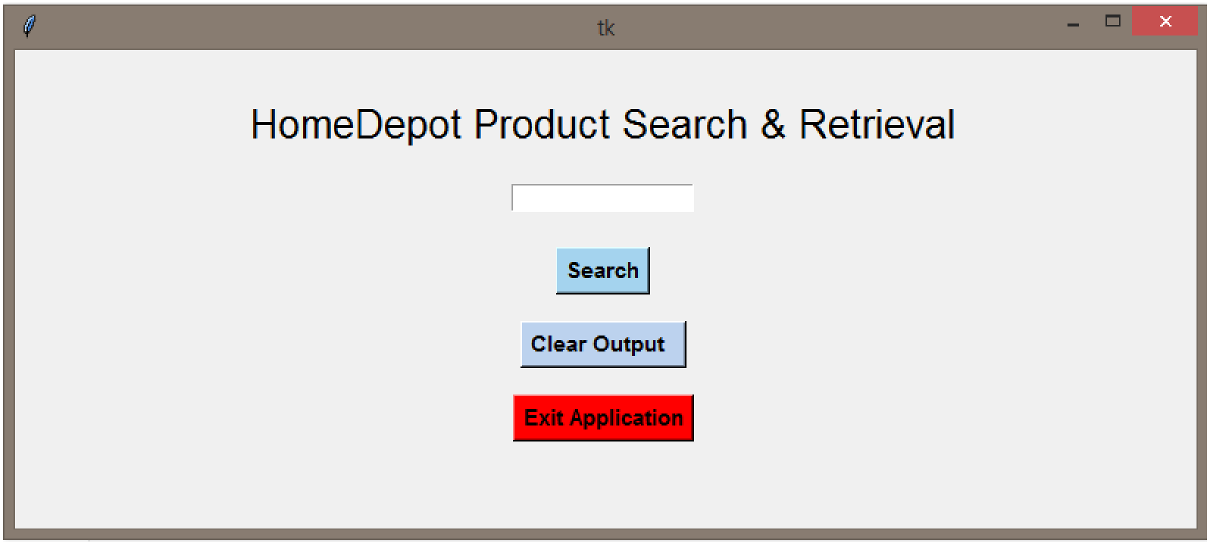
## Part VI. User Interface

### UI using tkinter package

UI is created using tkinter python library. We 1 input field and 3 button functionalities available to the user.

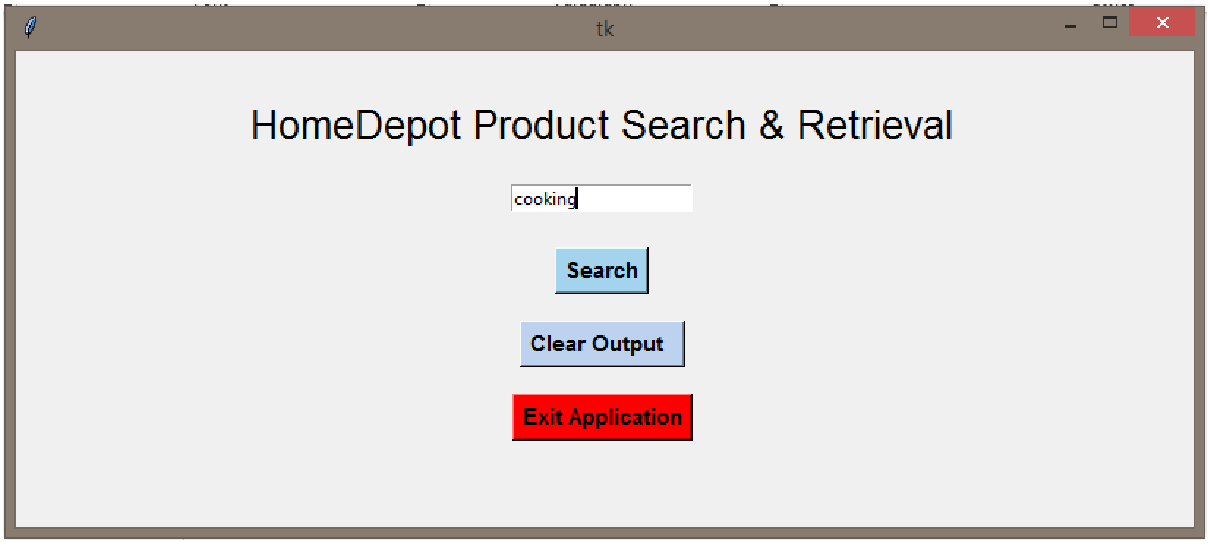
The user can enter the keywords to be searched in the input field.
On clicking the 'Search' button, product_uid and description will be retrieved in the descending order to cosine similarity measure.
The 'Clear Output' will clear the input text field and outputs.
The 'Exit Application' button will destroy the application.

### Basic UI



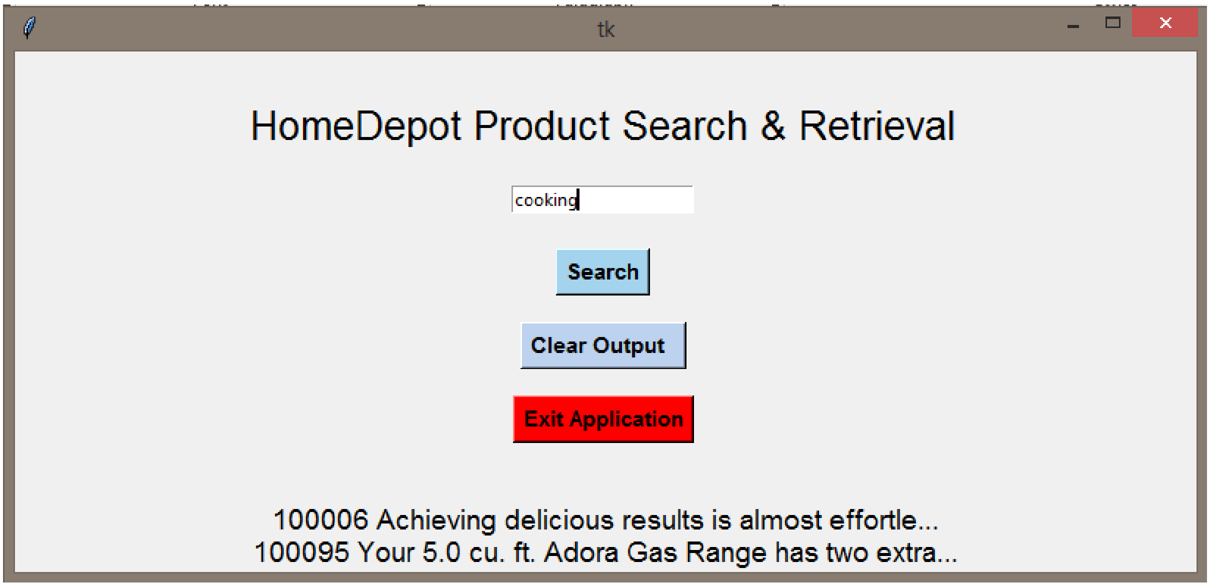

### User entering text

The user entered text is case insensitive. Our function will update the input text field to lower case.

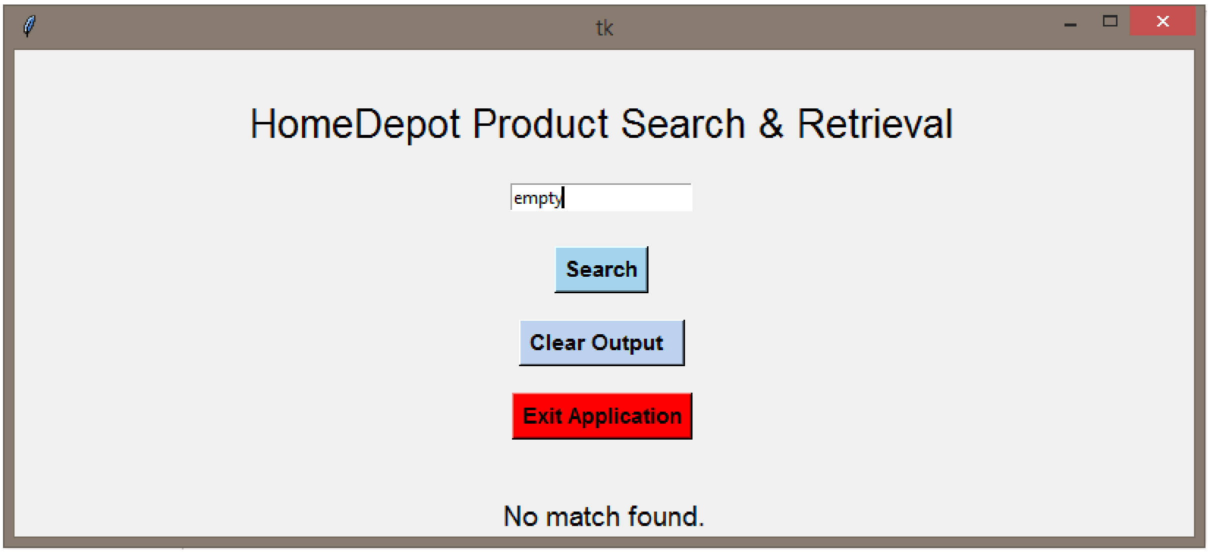



**On clicking 'Search' button**

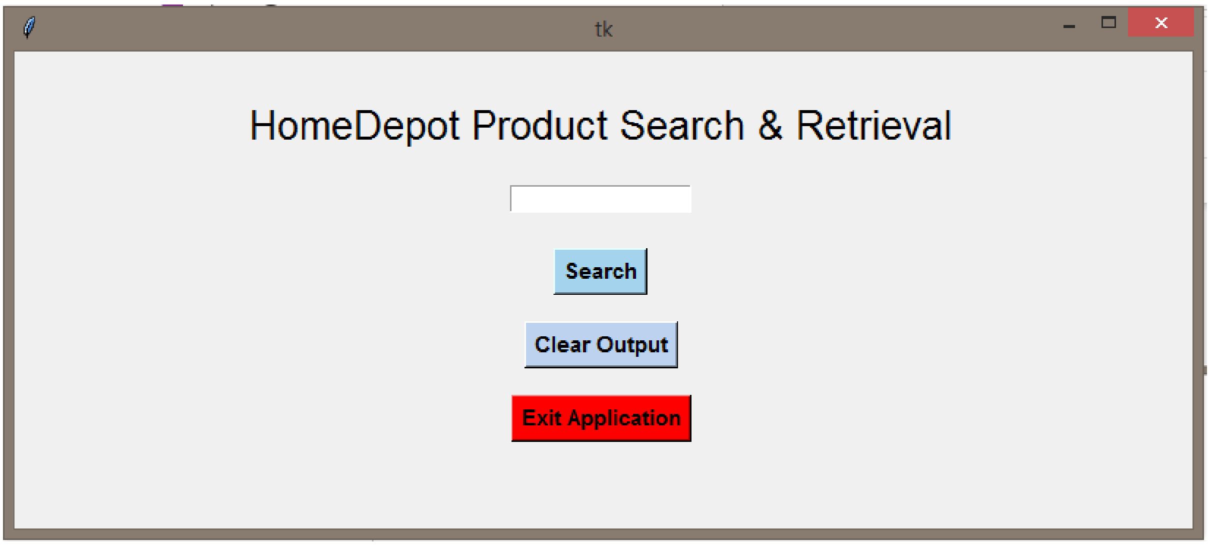
The function search_retrieval() is called which returns the list of product_uid and product_description in the descending order of similarity measure.




If the function returns no results.




**On clicking 'Clear Output' button**
The clear button will refresh the input field as well as the retrieved results in the output.




**'Exit Application' button will close the UI.**

**Part VII. Conclusion and Improvement Suggestions**

The results are retrieved by creating an inverted index on the product descriptions. Additional products can be easily appended to the list and inverted index can be recomputed with minimal impacts and computing needs. This can be done through indexing the test_df in cell 23. Through the search and retrieval function, we were able to look further into our recall and precision. We found that as the number of documents increases, recall increased and precision decreased. The average precision was high at around 98%. The GUI helped easily visualize our search function, as you can see the relevant documents provided through a search term.

Although we were able to achieve our goals in the project, there could still be further processes implemented. First, a retrieval process can be improved using user profiles and recommendation models. We were also looking into clustering to show products under the same category/make/type. We could have also looked into comparing multiple similarity measures can be used to improve the retrieved results. A final add-on would be to provide additional features like providing a link to the products can be easily implemented.