Fetch Project: Similarity Search via Text Inputs

Developed by Chen Zhang, 7/30/2023

1. Introduction

1.1 Problem Statement

We were provided with a dataset of offers and associated metadata, such as the retailers and brands that sponsor the offer. We were also provided with a dataset of brands that we support on our platform, and the categories to which those products belong.

The goal is to develop a tool to perform the following queries:

- **Category search**: If a user searches for a category (e.g., diapers), the tool should return a list of offers that are relevant to that category.
- **Brand search**: If a user searches for a brand (e.g., Huggies), the tool should return a list of offers that are relevant to that brand.
- **Retailer search**: If a user searches for a retailer (e.g., Target), the tool should return a list of offers that are relevant to that retailer.
- **Similarity score**: The tool also returns the score that was used to measure the similarity of the text input with each offer.

1.2 Outline

In this project, I developed a command-line tool that enables users to intelligently search for offers via text input from users. The tool is built on natural language processing (NLP) models for performing similarity search on a dataset of product categories. The methodology involves the following five aspects:

- **Similarity definition**: An NLP model is used to define the similarity metric that will be used to measure the similarity between two product categories.
- **Data cleaning**: The dataset of product categories is cleaned to remove any errors or inconsistencies.
- **Table operation**: The cleaned dataset is loaded into a database table.
- **Query search**: The user enters a query, and the NLP model is used to generate a list of product categories that are similar to the query.
- **Metrics analysis**: The results of the similarity search are analyzed to determine which metrics work best for certain types of queries.

The notebook provides detailed responses to each problem in Section 1.1, with a focus on the production pipeline surrounding the model. In addition, I discuss the comparison of multiple NLP models and the selection of the right similarity metrics. I also outline plans for future work on this project, which include:

- Training a sentence transformer model and using it to perform similarity search.
- Investigating the use of other NLP models for similarity search.
- Developing a user interface for the tool.

1.3 Run Locally

Please see Example.ipynb for details.

1.4 Git Repository

https://github.com/zchen163/Fetch/tree/main

2. Production Pipeline

2.1 Libraries Prerequisite

This notebook requires the installation of the following packages:

- numpy: A library for scientific computing.
- pandas: A library for data analysis.
- sklearn: A library for machine learning.
- thefuzz: A library for fuzzy string matching.
- sentence_transformer: A library for sentence embedding.

To install these packages, you can use the following commands:

- pip install numpy
- pip install pandas
- pip install sklearn
- pip install thefuzz
- pip install sentence_transformer

Once you have installed the packages, you can load them into your notebook using the following code:

In [4]:
```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import sklearn
```

```python
from thefuzz import fuzz
from sentence_transformers import SentenceTransformer
import re
from sklearn.metrics.pairwise import cosine_similarity
pd.set_option('display.max_colwidth', 30)
```

2.2 Data Cleaning

My first step took in processing language data is **data cleaning**. It is important to remove noise from the data so that the machine can more easily detect patterns. Text data can contain a lot of noise, such as special characters and inconsistent capitalization. These elements can be difficult for computers to understand, so they need to be removed.

One example is shown in the snapshot of the *offer_retailer.csv* file below. As we can see, the data contains some items that start with a column character. Additionally, the second and third rows only differ in the capitalization of the letter "L". These inconsistencies need to be addressed before the data can be processed.

offer_retailer

OFFER	RETAILER	BRAND
:ratio™ KETO* Friendly Cereal OR Granola		RATIO
12 Pack OR 2 Liter AND Whole Pizza at Casey's	CASEYS GENERAL STORE	CASEYS GENERAL STORE
12 pack OR 2 liter AND Whole Pizza at Casey's	CASEYS GENERAL STORE	CASEYS GENERAL STORE
12 Pack OR 2 Liter AND Whole Pizza Pie at Casey's	CASEYS GENERAL STORE	CASEYS GENERAL STORE
2 Pack OR 2 Liter AND Whole Pizza at Casey's	CASEYS GENERAL STORE	CASEYS GENERAL STORE

The data was cleaned using the following steps:

- Remove special characters: Special characters such as punctuation marks and symbols were removed using regular expressions.
- Normalize capitalization: All words were lowercased to ensure that the machine saw them as the same entity.
- Remove stop words: Stop words are common words that add little meaning to the data, such as "the", "a", and "is". I remove these words to reduce the size of the data and improve the performance of the learning algorithms.

The code below performs these steps. To track the changes that were made to the text, a new column with the clean text was added. The output is shown below the code.

In [5]:
```python
def clean(file):
    # change to lower case
    df = pd.read_csv(file, dtype = str)
    for col in df.columns.values:
        df[col] = df[col].str.lower()

    # remove special characters in the offer column
    if file == 'offer_retailer.csv':
        for index, row in df.iterrows():
            txt = row['OFFER']
            df.loc[index, "OFFER"] = re.sub(r"[^a-z0-9 ]", "", txt)
    # remove duplicate rows
        df.drop_duplicates(subset=['OFFER'], inplace = True)
    return df
```

The three datasets were cleaned and the results are shown below. A few duplicate lines were removed from the offer_retailer file. Missing values in the Retailer column were filled with empty strings. The column names were also renamed for the purpose of future joining.

In [6]:
```python
# Offer_brand table
offer = clean('offer_retailer.csv')
offer.replace(np.nan,'',regex=True, inplace=True)
print(offer.head())
print(offer.shape)
```

```
                           OFFER             RETAILER         BRAND
0  spend 50 on a fullpriced n...            sams club     sams club
1  beyond meat plantbased pro...                        beyond meat
2  good humor viennetta froze...                         good humor
3  butterball select varietie...  dillons food store    butterball
4  gatorade fast twitch 12oun...               amazon      gatorade
(369, 3)
```

In [7]:
```python
brand = clean('brand_category.csv')
brand.rename(columns={"BRAND_BELONGS_TO_CATEGORY": "CATEGORY"}, inplace=True)
brand['RECEIPTS'] = pd.to_numeric(brand['RECEIPTS'])

print(brand.head())
print(brand.shape)
```

```
             BRAND          CATEGORY  RECEIPTS
0  caseys gen store  tobacco products   2950931
1  caseys gen store            mature   2859240
2           equate      hair removal    893268
3        palmolive       bath & body    542562
4             dawn       bath & body    301844
(9906, 3)
```

In [8]:
```python
category = clean('categories.csv')
category.rename(columns={"PRODUCT_CATEGORY": "CATEGORY",
                      "IS_CHILD_CATEGORY_TO": "PARENT_CATEGORY"}, inplace=True)

category.drop(columns=['CATEGORY_ID'], inplace = True)

print(category.head())
print(category.shape)
```

```
                        CATEGORY    PARENT_CATEGORY
0               red pasta sauce        pasta sauce
1  alfredo & white pasta sauce        pasta sauce
2              cooking & baking             pantry
3              packaged seafood             pantry
4              feminine hygeine  health & wellness
(118, 2)
```

2.3 Table Operation

The next step is to combine all tables into one master table for easier handling.

In [9]:
```python
# join the brand and category table, using pandas merge function which is equal to SQL inner join
brand_cat = brand.merge(category, right_on = ['CATEGORY'], left_on = ['CATEGORY'])

print(f'After joining the category (shape {category.shape}) and brand table (shape {brand.shape}), the output has shape of {brand_cat.shape}')
print(brand_cat.head())
```

```
After joining the category (shape (118, 2)) and brand table (shape (9906, 3)), the output has shape of (9906, 4)
                BRAND          CATEGORY  RECEIPTS    PARENT_CATEGORY
0   caseys gen store  tobacco products   2950931             mature
1  rj reynolds vapor  tobacco products        21             mature
2   caseys gen store            mature   2859240             mature
3             equate      hair removal    893268  health & wellness
4           barbasol      hair removal    283926  health & wellness
```

In [10]:
```python
full = offer.merge(brand_cat, how = 'left', right_on = ['BRAND'], left_on = ['BRAND'])
full.replace(np.nan,'',regex=True, inplace=True)

# rearrange columns
cols = ['OFFER', 'RETAILER', 'BRAND', 'CATEGORY', 'PARENT_CATEGORY', 'RECEIPTS']
full = full[cols]
print(full)
print(full.shape)
```

```
                              OFFER   RETAILER              BRAND  \
0    spend 50 on a fullpriced n...  sams club          sams club
1    beyond meat plantbased pro...                   beyond meat
2    beyond meat plantbased pro...                   beyond meat
3    beyond meat plantbased pro...                   beyond meat
4    good humor viennetta froze...                    good humor
..                             ...        ...                ...
792              thomas bagel thins                        thomas
793              thomas bagel thins                        thomas
794              thomas bagel thins                        thomas
795          spend 270 at pavilions  pavilions          pavilions
796    back to the roots soils se...    walmart  back to the roots

                   CATEGORY PARENT_CATEGORY RECEIPTS
0
1             packaged meat          pantry     30.0
2           plant-based meat  meat & seafood   1584.0
3    frozen plant-based meat          frozen    313.0
4              dips & salsa          snacks    596.0
..                      ...             ...      ...
792                  bakery   deli & bakery     24.0
793        frozen breakfast          frozen    258.0
794                   bread          pantry  26490.0
795         cooking & baking          pantry     15.0
796    packaged meals & sides          pantry    146.0

[797 rows x 6 columns]
(797, 6)
```

It is worth noting that the master table contains duplicate rows for some categories that belong to more than one parent category. For example, frozen pizza is categorized as both "frozen" and "pantry," which is logical because frozen pizza can be found in both the frozen food aisle and the pantry aisle of a grocery store. Additionally, a left join was used to retain the information in the offer_retailer table. This means that even if a category is not present in the offer_retailer table, it will still be present in the master table. With the master table now prepared, the next step is to investigate input similarity search.

2.4 Basline Method: Similarity Search Based on Fuzzy String

In this section, I developed a baseline tool, called Fuzzy string, to perform similarity search. This method serves as a starting point which will be improved in the next section. Fuzzy string provides a ranked list of similarity scores based on several criteria, including ratio, partial ratio, token sort ratio, token set ratio, and partial token sort ratio. These criteria are used to measure the similarity between two strings.

- **Ratio** is the simplest measure of text similarity. It is calculated by dividing the number of common words between the two strings by the total number of words in the longer string.
- **Partial ratio** is a more sophisticated measure of text similarity. It takes into account the order of the words in the two strings. It is calculated by dividing the number of common words in the same order by the total number of words in the longer string.
- **Token sort ratio** is another sophisticated measure of similarity. It takes into account the order of the words in the two strings, but it also considers the capitalization of the words. It is calculated by dividing the number of common words in the same order and capitalization by the total number of words in the longer string.
- **Token set ratio** is a simple measure of similarity that only considers the unique words in the two strings. It is calculated by dividing the number of common unique words by the total number of unique words in the longer string.
- **Partial token sort ratio** is a sophisticated measure of similarity that considers the order of the unique words in the two strings, but it also considers the capitalization of the words. It is calculated by dividing the number of common unique words in the same order and capitalization by the total number of unique words in the longer string.

To illustrate the use of these criteria, I consider the example of the input string "meat".

In [8]:
```python
name = "meat"
similarity = category.copy()
partialRatio, ratio, tokenSortRatio, tokenSetRatio, partialTokenSortRatio = [], [], [], [], []

for index, row in similarity.iterrows():
    val = row['CATEGORY']
    a = fuzz.partial_ratio(name, val)
    b = fuzz.ratio(name, val)
    c = fuzz.token_sort_ratio(name, val)
    d = fuzz.token_set_ratio(name, val)
    e = fuzz.partial_token_sort_ratio(name, val)
    partialRatio.append(a)
    ratio.append(b)
    tokenSortRatio.append(c)
    tokenSetRatio.append(d)
    partialTokenSortRatio.append(e)
similarity['Partial Ratio'] = partialRatio
similarity['Ratio'] = ratio
similarity['Token Sort Ratio'] = tokenSortRatio
similarity['Token Set Ratio'] = tokenSetRatio
similarity['Partial Token Set Ratio'] = partialTokenSortRatio

for criteria in ['Ratio', 'Partial Ratio', 'Token Sort Ratio', 'Token Set Ratio', 'Partial Token Set Ratio']:
    print(f'Top 10 matched category using {criteria}')
    print(similarity.nlargest(10, criteria)[['CATEGORY', criteria]])
    print()
```

```
Top 10 matched category using Ratio
             CATEGORY  Ratio
112            mature     60
52                tea     57
90      packaged meat     47
6               cream     44
51              bread     44
71              water     44
23         condiments     43
72   plant-based meat     40
74        fresh pasta     40
92             makeup     40

Top 10 matched category using Partial Ratio
                       CATEGORY  Partial Ratio
47            jerky & dried meat            100
72              plant-based meat            100
90                 packaged meat            100
105      frozen plant-based meat            100
16   meal replacement beverages             75
19                 frozen meals             75
23                   condiments             75
24         packaged meals & sides             75
35                 sexual health             75
36                malt beverages             75

Top 10 matched category using Token Sort Ratio
               CATEGORY  Token Sort Ratio
112              mature                60
52                  tea                57
90        packaged meat                47
6                 cream                44
51                bread                44
71                water                44
23           condiments                43
18          snack mixes                40
47   jerky & dried meat                40
72     plant-based meat                40

Top 10 matched category using Token Set Ratio
                    CATEGORY  Token Set Ratio
47        jerky & dried meat              100
72          plant-based meat              100
90             packaged meat              100
105  frozen plant-based meat              100
112                   mature               60
52                       tea               57
6                      cream               44
51                     bread               44
71                     water               44
23                condiments               43

Top 10 matched category using Partial Token Set Ratio
                       CATEGORY  Partial Token Set Ratio
47            jerky & dried meat                      100
72              plant-based meat                      100
90                 packaged meat                      100
105      frozen plant-based meat                      100
16   meal replacement beverages                       75
19                 frozen meals                       75
23                   condiments                       75
24         packaged meals & sides                     75
35                 sexual health                      75
36                malt beverages                      75
```

As you can see, even if the first few top-matched items contain the word "meat," other matched words may not make much sense. This is because the nature of fuzzy search is **edit distance**. Edit distance is a measure of how many changes need to be made to one string to make it the same as another string. For example, the edit distance between "meat" and "meal" is 1, because only one letter needs to be changed (the "e" to an "a").

Fuzzy search algorithms use edit distance to find strings that are similar to the input string. The lower the edit distance, the more similar the two strings are. However, edit distance does not take into account the *meaning* of the words. For example, the words "meat" and "meal" have different meanings, even though they have the same edit distance.

This is why it is important to carefully consider the results of a fuzzy search. Just because a string has a low edit distance to the input string does not mean that it is a good match. It is important to consider the meaning of the words in the string as well.

In the example of the input string "meat," the first few top-matched items may contain the word "meat" because it is a common word. However, the other words in the matched items may not make much sense, because they may be semantically unrelated to the input string. This is because fuzzy search algorithms do not take into account the meaning of the words when they are making their matches.

In the next section, I will use **semantic similarity** to improve the results of a fuzzy search. Semantic similarity is a measure of how similar the meanings of two strings are. By using semantic similarity, it is possible to find strings that are similar to the input string in terms of both their meaning and their edit distance.

2.5. Modern Method: Similarity Search Using Transfomer Model

In this section, I developed a number of transformer models for the task of similarity search. **Transformer models** are a type of neural network that have been shown to be very effective for natural language processing tasks. They are able to learn long-range dependencies between words, which makes them well-suited for tasks such as **semantic similarity** and natural language inference. Text/sentence similarity is one of the clearest examples of the power of transformer models. These models can be used to compute the similarity between two pieces of text, even if they are very different in length or structure. This makes them ideal for tasks such as category search, where the goal is to find documents that are similar to a given query.

I used the "meat" query as an example to test the performance of the different models. I evaluated several well-established models including **all-MiniLM-L6-v2**, **all-mpnet-base-v2**, **all-distilroberta-v1**, and **bert-base-nli-mean-tokens**. The bert-base-nli-mean-tokens model is a sentence-transformers model that has been trained on a dataset of text and category pairs. This model is able to map sentences to a 768-dimensional vector space, where the distance between two vectors is a measure of the semantic similarity between the sentences.

The full model architecture is shown below. The model consists of the following layers:

-A **tokenization** layer that converts the text into a sequence of tokens.

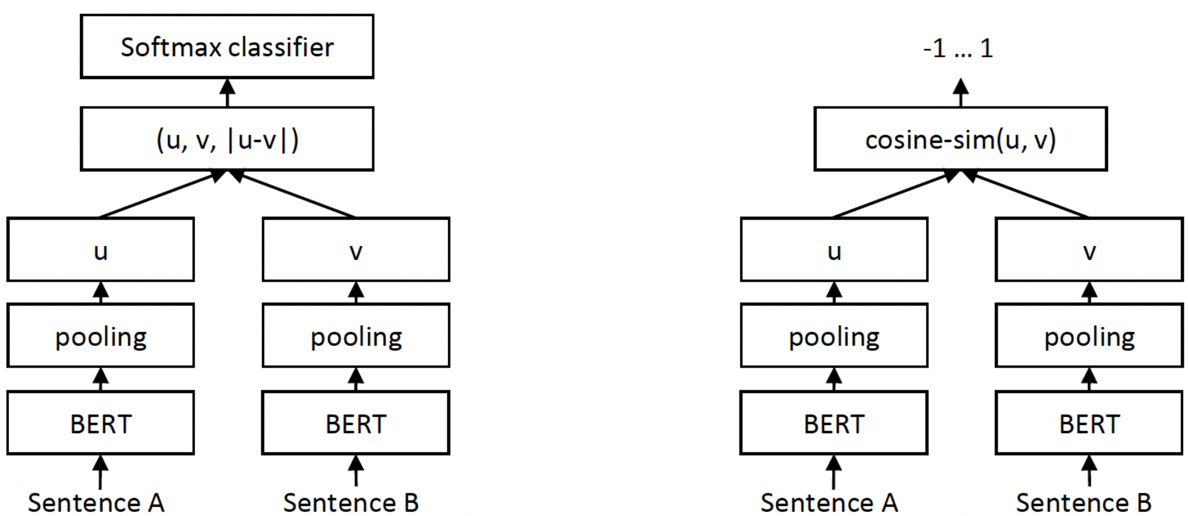
-A **word embedding** layer that maps each token to a vector representation.

-A **Transformer** layer that learns to predict the next token in a sequence.

-A **pooling** layer that aggregates the output of the Transformer layer into a single vector representation.

-A **similarity** layer that computes the similarity between the query and the document representations.

The full model architecture is shown below.



Here I provide step-by-step instructions on using transformer models for similarity search.

First, I create a **bert-base-nli-mean-tokens** model using the sentence_transformers library. This model can be used to map sentences and paragraphs to a 768-dimensional dense vector space, where the distance between two vectors is a measure of the semantic similarity between the sentences.

In [2]:
```
from sentence_transformers import SentenceTransformer

model_name = 'bert-base-nli-mean-tokens'
# model_name = 'all-MiniLM-L6-v2'
# model_name = 'all-mpnet-base-v2'

model = SentenceTransformer(model_name)
```

Second, I create a category dictionary from the ['CATEGORY'] column. In this example,

In [11]:
```
catDict = category['CATEGORY'].to_list()
catDict_vecs = model.encode(catDict)
print(catDict_vecs.shape)
```

```
(118, 768)
```

Below are the embedding vectors for the category dictionary.

In [12]:
```
catDict_vecs
```

Out[12]:
```
array([[-0.41283208,  0.19538327, -0.7930067 , ..., -0.17972049,
         0.6738818 ,  0.5252664 ],
       [-0.34384283,  0.20766301, -0.21792002, ..., -0.37926593,
         0.02805232,  0.63666254],
       [ 0.54358596,  0.8228612 ,  1.6034012 , ..., -0.3250906 ,
         0.2267657 ,  0.13139722],
       ...,
       [-0.51006913,  1.0867724 , -0.1981626 , ..., -0.18807282,
         0.77340156,  0.3120817 ],
       [-0.18643278,  0.87110186,  0.6632734 , ..., -0.37362155,
        -0.26959875,  0.02258943],
       [ 0.26655945,  0.2495401 ,  1.5600967 , ...,  0.7145732 ,
         0.29977995,  0.2982676 ]], dtype=float32)
```

Next, I apply cosine similarity to calculate the pairwise similarity between the input string and all items in the category column.

In [14]:
```
query = model.encode(['meat'])
print(query.shape)
cos = cosine_similarity(query, catDict_vecs)
similarity = category.copy()
similarity['Cosine'] = cos.reshape(-1, 1)

print(similarity.nlargest(20, 'Cosine')[['CATEGORY', 'Cosine']])
```

```
(1, 768)
                      CATEGORY    Cosine
90               packaged meat  0.899200
47           jerky & dried meat  0.781604
72              plant-based meat  0.755113
105      frozen plant-based meat  0.655589
115                  frozen beef  0.632474
62                food storage  0.621770
73                        eggs  0.609283
84                 dog supplies  0.607903
30                 soup & broth  0.600675
114              frozen chicken  0.570397
24        packaged meals & sides  0.570185
23                   condiments  0.569826
86               prepared meals  0.569633
63                       cheese  0.568969
85                pickled goods  0.568233
64            frozen vegetables  0.564005
17                     pretzels  0.547607
16   meal replacement beverages  0.529368
113                frozen turkey  0.528729
51                        bread  0.522049
```

I found that the top 20 matched categories are more semantically meaningful than the fuzzy search results. Not only are categories that contain the word "meat" identified, but the remaining high-scoring categories are also food-related, such as eggs and frozen chicken. I will proceed to address the actual problem using this pre-trained model.

3. Responses to each problem in Section 1.1

Task 1. Category Search

The previous experiments demonstrated that we can query a category name and return similar categories. In this section, I developed a command-line tool for each of the tasks in Section 1. I extended the search to the full table and returned all offers under those similar categories in the order of relevance, namely similarity score.

The next cell defines the query and the desired number of answers.

In [15]:
```
def categorySearch(queryLst: list, N = 20):
    """
    query: a list of string inputs of search in category. length of the list is n.
    N: top N offers selected by the matching. default 20.
    return: n csv files stored in the folder, each stores the top N selected matching for the corresponding query.
    """
    catDict = full['CATEGORY'].to_list()
    catDict_vecs = model.encode(catDict)
    similarity = full.copy()
    query = model.encode(queryLst)
    cos = cosine_similarity(query, catDict_vecs).round(4)
    for i, q in enumerate(queryLst):
        similarity[q] = cos[i].reshape(-1, 1)
    df = similarity.copy()

    # Display the result, top N matched offer
    for q in queryLst:
        df.drop_duplicates(subset=['OFFER'], inplace = True)
        a = df.sort_values(by=[q, 'RECEIPTS'], ascending=[False, False])
        a.rename(columns={q: "SCORE"}, inplace=True)
        # top N offers
        a = a.iloc[:N][['OFFER', 'RETAILER', 'BRAND', 'CATEGORY', 'RECEIPTS', 'SCORE']]
        fname = f'Category Search Top {N} {q}.csv'
        a.to_csv(fname)
```

To display the result (stored in variable 'res'), I selected the top N offers based on their similarity score, from highest to lowest. Note that duplicate offers were removed, as some offers may belong to multiple categories. In the output, the last column, named as the query input string, denotes the similarity score. The offers are ranked by highest score to lowest. If there is a tie in the similarity score, the offers are ranked by the number of receipts.

The output is stored in a CSV file.

In [16]:

```
queries = ['meat', 'coffee']
res = categorySearch(queries)
```

The results of top 20 offers by searching 'meat' in category are shown below.

Category Search Top 20 meat

	OFFER	RETAILER	BRAND	CATEGORY	RECEIPTS	SCORE
103	tyson products select varieties spend 20 at sams club	sams club	ball park frank	packaged meat	3186.0	0.8992
584	tyson products select varieties spend 15 at walmart	walmart	aidells	packaged meat	854.0	0.8992
1	beyond meat plantbased products spend 25		beyond meat	packaged meat	30.0	0.8992
47	beyond steak plantbased seared tips 10 ounce at target	target	beyond meat	packaged meat	30.0	0.8992
66	beyond steak plantbased seared tips 10 ounce buy 2 at heb	h-e-b	beyond meat	packaged meat	30.0	0.8992
486	beyond steak plantbased seared tips 10 ounce at heb	h-e-b	beyond meat	packaged meat	30.0	0.8992
523	beyond steak plantbased seared tips 10 ounce buy 2 at target	target	beyond meat	packaged meat	30.0	0.8992
552	beyond meat plantbased products spend 15		beyond meat	packaged meat	30.0	0.8992
592	beyond meat plantbased products spend 20		beyond meat	packaged meat	30.0	0.8992
777	jack links select varieties		jack links	jerky & dried meat	1614.0	0.7816
60	gortons at select retailers	shop rite	gortons	jerky & dried meat	14.0	0.7816
81	gortons air fried butterfly shrimp at walmart	walmart	gortons	jerky & dried meat	14.0	0.7816
36	egglife egg white wraps at aldi	aldi	egglife	eggs	27.0	0.6093
95	cooked perfect meatballs at walmart	walmart	cooked perfect	frozen chicken	253.0	0.5704
194	cooked perfect meatballs homestyle or turkey at walmart	walmart	cooked perfect	frozen chicken	253.0	0.5704
153	back to the roots grow seed starting pots or germination trays at walmart or target	target	back to the roots	packaged meals & sides	146.0	0.5702
186	back to the roots soils select varieties and sizes at walmart	walmart	back to the roots	packaged meals & sides	146.0	0.5702
277	back to the roots dry plant food 5 pounds at the home depot	the home depot	back to the roots	packaged meals & sides	146.0	0.5702
289	back to the roots garden soil 1 cubic foot at lowes home improvement	lowes home improvement	back to the roots	packaged meals & sides	146.0	0.5702
464	back to the roots raised bed gardening kit with soil seeds and plant food at target	target	back to the roots	packaged meals & sides	146.0	0.5702

The results of top 20 offers by searching 'coffee' in category are shown below.

Task 2. Brand Search

Similar to the category search, if the brand entered is correct or close enough to brands in the offer_retailer file, the corresponding offers will be directly outputted, just like the category search. However, this is not always the case. Note that some brands may not match with the offer_retailer table. In such cases, I will search the brand in the brand table, and use the top N (default = 3) matched categories to return all offers based on that.

In [17]:

```
def brandSearch(query: str, cutoff = 0.9, N = 10):
    """
    Searching the brand in the joined full table.
    query: a single string input of search in brand
    return: a dataframe with the last column is the similarity score between the query and brand in each row
    """
    query = query.lower()
    brandDict = full['BRAND'].to_list()
    brandDict_vecs = model.encode(brandDict)
    similarity = full.copy()
    q = model.encode([query])
    cos = cosine_similarity(q, brandDict_vecs).round(4)
    similarity['SCORE'] = cos.reshape(-1, 1)
    df = similarity.copy()
    df.sort_values(by=['SCORE', 'RECEIPTS'], ascending=[False, False])
    brandOut = df[df['SCORE'] >= cutoff]

    # case one: if the brand search can identify something
    if brandOut.shape[0] > 0:
        brandOut.drop_duplicates(subset=['OFFER'], inplace = True)
        # top N offers
        b = brandOut.iloc[:N][['OFFER', 'RETAILER', 'BRAND', 'CATEGORY', 'RECEIPTS', 'SCORE']]
        fname = f'Brand Search Top {N} {query}.csv'
        # print the result here, and save to a csv file
#         print(fname)
#         print(b)
        b.to_csv(fname)
    # case two: if the brand search cannot identify any matching
    else:
        print('No good matching found in the offer table. Searching the brand_category list instead. ')
        brandOut = brandToCatSearch(query)
        if brandOut.shape[0] == 0:
            print('No matching found, please double check your input')
            return
        return brandOut
```

```
In [18]: def brandToCatSearch(query, N = 30, k = 3):
    """
    Searching the brand in the brand_category table instead.
    query: a string input of search in brand
    return: a dataframe with the last column is the similarity score between the query and brand in each row
    """

    brandDict = brand['BRAND'].to_list()
    brandDict_vecs = model.encode(brandDict)
    similarity = brand.copy()
    q = model.encode([query])
    cos = cosine_similarity(q, brandDict_vecs).round(4)
    similarity[query] = cos.reshape(-1, 1)
    df = similarity.copy()
    sortdf = df.sort_values(by=[query, 'RECEIPTS'], ascending=[False, False])

    match = sortdf.iloc[:k]
    # this match returns the top k matched category, by searching in the brand-category table
    print(f'Top {k} matched category')
    print(match)

    queryLst, scores = match['CATEGORY'].to_list(), match[query].to_list()

    catDict = full['CATEGORY'].to_list()
    catDict_vecs = model.encode(catDict)
    similarity = full.copy()
    query = model.encode(queryLst)
    cos = cosine_similarity(query, catDict_vecs).round(4)
    for i, q in enumerate(queryLst):
        similarity[q] = cos[i].reshape(-1, 1)
    df = similarity.copy()

    df[queryLst] = df[queryLst]*scores
    df[queryLst].round(4)
    df['SCORE'] = df[queryLst].max(axis=1)
    df.drop(columns=queryLst, inplace = True)
    df.drop_duplicates(subset=['OFFER'], inplace = True)

    sortdf = df.sort_values(by=['SCORE', 'RECEIPTS'], ascending=[False, False])

    out = sortdf.iloc[:N]
    fname = f'Brand Search through Category - Top {N}_{q} Match.csv'
    out.to_csv(fname)
    print()
    print(fname)
    print(out)
    return out
```

Example 1: The searching input is in the offer_retailer table. The search results are shown below:

```
In [19]: brandQuery = 'cvs'
res = brandSearch(brandQuery)
```

Example 2: The search input is in the offer_retailer table. Note that KFC is in the offer_retailer table, but not in the brand_category table. The search results are shown below:

```
In [20]: brandQuery = 'kfc'
res = brandSearch(brandQuery)
```

Example 3: The search input is not in the offer_retailer table. In this case, the algorithm will:

Find the top k (default 3) matched categories by searching in the brand_category table. Then, search these categories in the full table and return the top N (default 30) scored offers. If the scores are the same, the offers are ranked by receipts. The score is calculated by the similarity between the brand and category (step 1) multiplied by the score in category (step 2).

```
In [21]: brandQuery = 'Kroger'
res = brandSearch(brandQuery)
```

```
No good matching found in the offer table. Searching the brand_category list instead.
Top 3 matched category
      BRAND            CATEGORY  RECEIPTS  kroger
6    kroger              bakery    251276     1.0
430  kroger  household supplies      5255     1.0
473  kroger               water      4823     1.0

Brand Search through Category - Top 30_water Match.csv
                             OFFER         RETAILER  \
264  artesano buns buy 2 at wal...          walmart
605  glad trash bags 4 or 8 gallon
657  glad forceflex max strengt...
501            ballpark buns buy 2
473  core hydration select vari...          walmart
684  core hydration select vari...          walmart
396  sign up for mcalisters del...  mcalisters deli
675  snuggle liquid fabric soft...          walmart
132  sara lee bread select vari...          walmart
333  sara lee bread select vari...          walmart
765  sara lee delightful bread ...
137  sara lee or alfaros artesa...
275  sara lee or alfaros artesa...
590  sara lee or alfaros artesa...
718  sara lee or alfaros artesa...
654  brownie brittle snacks sel...
451  bimbo sweet baked goods buy 2
217  little bites spend 10 at w...          walmart
258       thomas bagel thins buy 2
520  thomas select varieties sp...
792             thomas bagel thins
14   arnold brownberry oroweat ...          walmart
326       brita pitcher and filter
430  arnold brownberry oroweat ...
529  brita standard or elite fi...
672         arnold grains almighty
735     brita pitcher or dispenser
225  purex laundry detergent se...          walmart
331           bays english muffins
723          the rustik oven bread

                         BRAND            CATEGORY     PARENT_CATEGORY  \
264          sara lee artesano              bakery      deli & bakery
605                       glad  household supplies  household supplies
657                       glad  household supplies  household supplies
501          ball park pop ups              bakery      deli & bakery
473             core hydration               water           beverages
684             core hydration               water           beverages
396            mcalisters deli              bakery      deli & bakery
675                    snuggle  household supplies  household supplies
132                   sara lee              bakery      deli & bakery
333                   sara lee              bakery      deli & bakery
765                   sara lee              bakery      deli & bakery
137                    alfaros              bakery      deli & bakery
275                    alfaros              bakery      deli & bakery
590                    alfaros              bakery      deli & bakery
718                    alfaros              bakery      deli & bakery
654            brownie brittle              bakery      deli & bakery
451                      bimbo              bakery      deli & bakery
217                entenmanns              bakery      deli & bakery
258                     thomas              bakery      deli & bakery
520                     thomas              bakery      deli & bakery
792                     thomas              bakery      deli & bakery
14   arnold brownberry oroweat              bakery      deli & bakery
326                      brita  household supplies  household supplies
430  arnold brownberry oroweat              bakery      deli & bakery
529                      brita  household supplies  household supplies
672  arnold brownberry oroweat              bakery      deli & bakery
735                      brita  household supplies  household supplies
225                      purex    laundry supplies  household supplies
331                       bays               bread              pantry
723                rustik oven               bread              pantry

    RECEIPTS   SCORE
264   7912.0  1.0000
605   6822.0  1.0000
657   6822.0  1.0000
501   5920.0  1.0000
473   4595.0  1.0000
684   4595.0  1.0000
396   2808.0  1.0000
675   2214.0  1.0000
132    916.0  1.0000
333    916.0  1.0000
765    916.0  1.0000
137    783.0  1.0000
275    783.0  1.0000
590    783.0  1.0000
718    783.0  1.0000
654    563.0  1.0000
451     57.0  1.0000
217     47.0  1.0000
258     24.0  1.0000
520     24.0  1.0000
792     24.0  1.0000
14      20.0  1.0000
326     20.0  1.0000
430     20.0  1.0000
529     20.0  1.0000
672     20.0  1.0000
735     20.0  1.0000
```

```
225     18.0  0.8176
331   1066.0  0.7833
723    292.0  0.7833
```

Brand Search through Category - Top 30_water Match

	OFFER	RETAILER	BRAND	CATEGORY	PARENT_CATEGORY	RECEIPTS	SCORE
264	artesano buns buy 2 at walmart	walmart	sara lee artesano	bakery	deli & bakery	7912.0	1.0
605	glad trash bags 4 or 8 gallon		glad	household supplies	household supplies	6822.0	1.0
657	glad forceflex max strength trash bags		glad	household supplies	household supplies	6822.0	1.0
501	ballpark buns buy 2		ball park pop ups	bakery	deli & bakery	5920.0	1.0
473	core hydration select varieties at walmart	walmart	core hydration	water	beverages	4595.0	1.0
684	core hydration select varieties buy 2 at walmart	walmart	core hydration	water	beverages	4595.0	1.0
396	sign up for mcalisters deli rewards tap for details	mcalisters deli	mcalisters deli	bakery	deli & bakery	2808.0	1.0
675	snuggle liquid fabric softener at walmart	walmart	snuggle	household supplies	household supplies	2214.0	1.0
132	sara lee bread select varieties buy 2 at walmart	walmart	sara lee	bakery	deli & bakery	916.0	1.0
333	sara lee bread select varieties buy 2	walmart	sara lee	bakery	deli & bakery	916.0	1.0
765	sara lee delightful bread buy 2		sara lee	bakery	deli & bakery	916.0	1.0
137	sara lee or alfaros artesano bread buy 5		alfaros	bakery	deli & bakery	783.0	1.0
275	sara lee or alfaros artesano bread spend 8		alfaros	bakery	deli & bakery	783.0	1.0
590	sara lee or alfaros artesano bread buy 2		alfaros	bakery	deli & bakery	783.0	1.0
718	sara lee or alfaros artesano bread spend 20		alfaros	bakery	deli & bakery	783.0	1.0
654	brownie brittle snacks select varieties buy 2		brownie brittle	bakery	deli & bakery	563.0	1.0
451	bimbo sweet baked goods buy 2		bimbo	bakery	deli & bakery	57.0	1.0
217	little bites spend 10 at walmart	walmart	entenmanns	bakery	deli & bakery	47.0	1.0
258	thomas bagel thins buy 2		thomas	bakery	deli & bakery	24.0	1.0
520	thomas select varieties spend 10		thomas	bakery	deli & bakery	24.0	1.0
792	thomas bagel thins		thomas	bakery	deli & bakery	24.0	1.0
14	arnold brownberry oroweat small slice bread at walmart	walmart	arnold brownberry oroweat	bakery	deli & bakery	20.0	1.0
326	brita pitcher and filter		brita	household supplies	household supplies	20.0	1.0
430	arnold brownberry oroweat keto bread buy 2		arnold brownberry oroweat	bakery	deli & bakery	20.0	1.0
529	brita standard or elite filters		brita	household supplies	household supplies	20.0	1.0
672	arnold grains almighty		arnold brownberry oroweat	bakery	deli & bakery	20.0	1.0
735	brita pitcher or dispenser		brita	household supplies	household supplies	20.0	1.0
225	purex laundry detergent select varieties at walmart	walmart	purex	laundry supplies	household supplies	18.0	0.8176000118255620
331	bays english muffins		bays	bread	pantry	1066.0	0.78329998254776
723	the rustik oven bread		rustik oven	bread	pantry	292.0	0.78329998254776

Task 3. Retailer Search

Retailer search is more challenging than the other two, as the **offer_retailer** table is the only table that contains retailer information. Therefore, I only search in the full table and return related retailers without linking them to categories (like brand search).

In [22]:
```
def retailerSearch(query: str, N = 10):
    """
    query: a single string inputs of search in retailer.
    N: top N offers selected by the matching. default 10.
    return: a csv files stores the top N selected retailer matching.
    """
    retDict = full['RETAILER'].to_list()
    retDict_vecs = model.encode(retDict)
    similarity = full.copy()
    q = model.encode([query])
    cos = cosine_similarity(q, retDict_vecs).round(4)
    similarity['SCORE'] = cos.reshape(-1, 1)
    df = similarity.copy()

    # Display the result, top N matched offer
    df.drop_duplicates(subset=['OFFER'], inplace = True)
    a = df.sort_values(by=['SCORE', 'RECEIPTS'], ascending=[False, False])
    # top N offers
    a = a.iloc[:N][['OFFER', 'RETAILER', 'BRAND', 'CATEGORY', 'RECEIPTS', 'SCORE']]
    fname = f'Retailer Search Top {N} {query}.csv'
    a.to_csv(fname)
```

In [23]:
```
queries = 'CVS'
res = retailerSearch(queries)
```

The results are shown below:

Retailer Search Top 10 CVS

	OFFER	RETAILER	BRAND	CATEGORY	RECEIPTS	SCORE
664	spend 10 at cvs	cvs	cvs	medicines & treatments	39210.0	1.0
737	spend 30 at cvs	cvs	cvs	medicines & treatments	39210.0	1.0
474	when you join costco as a gold star member new members only	costco	costco			0.7774
495	when you join costco as an executive member new members only	costco	costco			0.7774
682	butterball select varieties spend 10 at marianos	marianos	butterball	nut butters & jam	2107.0	0.75
63	shop 2 times at acme	acme	acme	medicines & treatments	449.0	0.7479
300	spend 90 at acme	acme	acme	medicines & treatments	449.0	0.7479
314	spend 250 at acme	acme	acme	medicines & treatments	449.0	0.7479
466	any acme receipt	acme	acme	medicines & treatments	449.0	0.7479
600	spend 130 at acme	acme	acme	medicines & treatments	449.0	0.7479

Task 4. Similarity Score

The similarity score between the text input and each offer is stored in the score column of each output CSV file. This score is a real number between 0 and 1, where 0 indicates no similarity and 1 indicates perfect similarity. The score is calculated via cosine similarity in the transformer model. It is measured by the cosine of the angle between two embedding vectors: one from the input string, and the other from each candidate category/brand/retailer.

7. Summary

In summary, I have presented a command-line tool that enables users to intelligently search for offers via text input from users. The tool is built on natural language processing (NLP) models for performing similarity search on a dataset of product categories.

The tool has two different methods for similarity search: fuzzy string and transformer model based sentence embedding. The fuzzy string method is a simple and easy-to-implement method, but it lacks the sematic similarity. The transformer model and sentence embedding method are more accurate. I evaluated the performance of the two methods on provided dataset. The results showed that the transformer model and sentence embedding method was more accurate than the fuzzy string method.

Some potential optimizations that could further improve the searching include:

- **Making the tool dynamic and real-time**: The current tool depends on the static, offline data provided. I plan to make the tool more scalable so that it can be used to process large datasets.
- **Training a sentence transformer model**: Sentence transformer models are a type of NLP model that have been shown to be effective for similarity search. By training a sentence transformer model on a dataset of product categories, I can create a model that can effectively measure the similarity between two product categories.
- **Investigating the use of other NLP models**: There are a variety of other NLP models that could be used for similarity search. I plan to investigate the use of these models to see if they can improve the performance of our tool.
- **Developing a user interface**: The current tool is a command-line tool. I plan to develop a user interface for the tool so that it can be used by a wider range of users.

8. References:

Sentence-Transformer library: https://www.sbert.net/docs/usage/semantic_textual_similarity.html
bert-base-nli-mean-tokens model: https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens
all-MiniLM-L6-v2 model https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
cosine similarity: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html#sklearn.metrics.pairwise.cosine_similarity

In [ ]:

In [ ]: