

"INFORMATION RETRIEVAL" IMPLEMENTATION PROJECT

Georgios Petrakis Student
of tHMTY

AM: 1066516

E-mail: up1066516@upnet.gr

DESCRIPTION OF THE IMPLEMENTATION ENVIRONMENT

Programming language:

Anaconda with Python 3.9 - 3.10

Software libraries:

"elasticsearch", "elasticsearch_dsl", "string", "re", "nltk", "tensorflow",
"keras", "scikit-learn", "gensim", "numpy", "matplotlib", "pandas"

Observations on the execution of the programmes:

1. In all queries, it is assumed that the csv files of the pronunciation are in the same directory as the code file
2. In the first 2 queries the Python interpreter version is 3.9, but due to a problem when installing tensorflow in the base anaconda environment, we had to create a new environment with interpreter 3.10 for the remaining queries.
3. The programs of the 3^{ou} and 4^{ou} queries, when first executed, create and store the necessary models in their directory. For speed, in subsequent runs the models are loaded from the folder.

ASK 1

Before running the program, we activate the elasticsearch client from the file with path ".../elasticsearch-8.3.3/bin/elasticsearch.bin".

The code shown in the screenshot below performs the following actions:

1. Loads the data from the "BX-Books.csv" file into a Pandas DataFrame
2. "Connects" to the elasticsearch local client on port 9200
3. It uses the "helpers" of the "elasticsearch" library to load the data into a client index using the "generator()" function. The name of the index defaults to "books"
4. With the "userinput_query()" function, the program accepts an alphanumeric from the user and returns the results of the query in an "elasticsearch Series Object"
5. Prints the results in descending order, along with their scores according to the elasticsearch metric

```
if __name__ == '__main__':
    df = pd.read_csv("BX-Books.csv")
    try:
        esclient = Elasticsearch("http://localhost:9200")
    except:
        print("Not connected to elasticsearch client")
    try:
        helpers.bulk(esclient, generator(df))
    except:
        print("OK! bulking successful")
    #after connecting to client and feeding the data, we listen for user input
    sm=Search(using=esclient)
    query_res=userinput_query(esclient, sm)
    for i, hit in enumerate(query_res):
        print(f'Book n{i} title: "{hit.book_title}" with ES score {hit.meta.score}')
```

The functions are shown below:

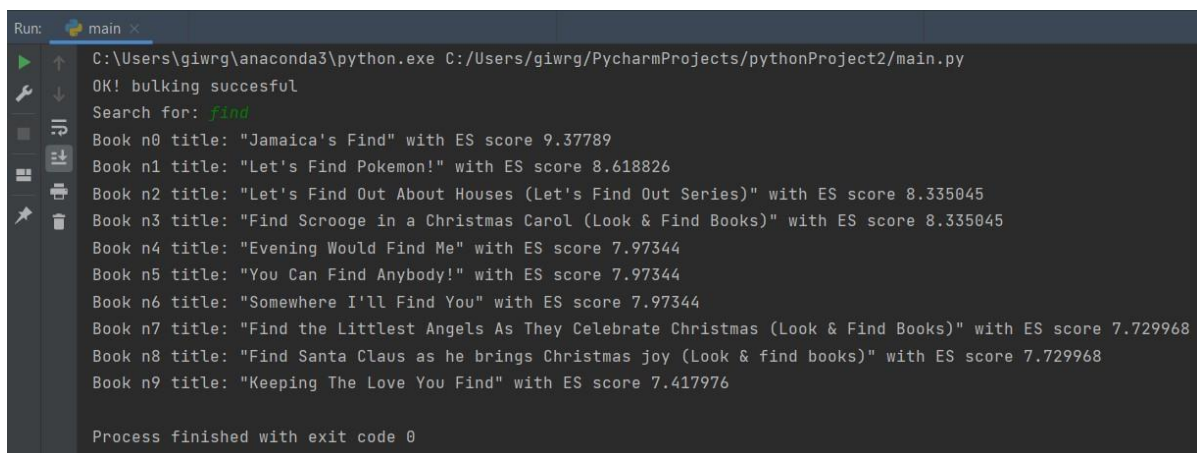
```
def generator(frame):
    iterable=df.iterrows()
    for i, doc in iterable:
        yield {
            "_index": 'books',
            "_source": doc.to_dict()
        }
    raise StopIteration
```

```
def userinput_query(client, searchreq):
    search_string=input("Search for: ")
    res=searchreq.query("query_string" ,query=search_string)
    return res
```

Comments

The "generator()" function returns the next object of the dataframe in appropriate form each time it is called by elasticsearch's "bulk()". To make the code more readable, the "userinput_query()" function used the "Search.query()" function of the "elasticsearch_dsl" library.

Indicative Results



```
Run: main
C:\Users\giwrg\anaconda3\python.exe C:/Users/giwrg/PycharmProjects/pythonProject2/main.py
OK! bulking succesful
Search for: find
Book n0 title: "Jamaica's Find" with ES score 9.37789
Book n1 title: "Let's Find Pokemon!" with ES score 8.618826
Book n2 title: "Let's Find Out About Houses (Let's Find Out Series)" with ES score 8.335045
Book n3 title: "Find Scrooge in a Christmas Carol (Look & Find Books)" with ES score 8.335045
Book n4 title: "Evening Would Find Me" with ES score 7.97344
Book n5 title: "You Can Find Anybody!" with ES score 7.97344
Book n6 title: "Somewhere I'll Find You" with ES score 7.97344
Book n7 title: "Find the Littlest Angels As They Celebrate Christmas (Look & Find Books)" with ES score 7.729968
Book n8 title: "Find Santa Claus as he brings Christmas joy (Look & find books)" with ES score 7.729968
Book n9 title: "Keeping The Love You Find" with ES score 7.417976
Process finished with exit code 0
```

ASK 2

For the second code file we needed to create two new functions, which will reorder the results returned by "userinput_query()" according to a new metric. The new metric chosen

for the results is a "weighted sum" of the original elasticsearch score (weight 0.5), the user's review (weight 0.4) and the average score from all users (weight 0.1). The new metric "prefers" results that have reviews (even low ones) to those that do not have any.

The body of the code is similar to the 1^{ou} query, so only the two new functions are listed.

```

44 def display_custom_metric_results(results, df_ratings):
45     userID=input("For results based on custom metric, enter userID: ")
46     userID=int(userID)
47     sorted_results = sorted(results, key=lambda x: custom_metric(x, userID, df_ratings), reverse=True)
48     for i, hit in enumerate(sorted_results):
49         print(f'Book n{i+1} title: "{hit.book_title}"')
50

```

```

27 def custom_metric(book,userid, ratingsframe):
28     book_code = book.isbn
29     book_rating = ratingsframe.query("isbn == @book_code") # dataframe with all the ratings for this book
30     if book_rating.empty:
31         print("no ratings for this book")
32         return 0.5*book.meta.score
33     else:
34         ratingslist = book_rating['rating'].to_list()
35         mean_component = sum(ratingslist) / len(ratingslist)
36         # get user rating
37         user_rating_frame = ratingsframe.query("isbn == @book_code and uid == @userid") #dataframe with empty
38         if user_rating_frame.empty:
39             user_component=0
40         else:
41             # exoume apomonwsei tin grammi me to rating apton xristi kai prepei na eksagoume tin vathmologia apto dataframe
42             user_component=user_rating_frame.iloc[0]['rating']
43     return 0.5*book.meta.score + 0.4*user_component + 0.1*mean_component

```

The function "display_custom_metric_results()" asks for the user ID, accepts the "Series object" (which is "sortable") and prints the "sorted" its version according to the second function "custom_metric()". This function uses Pandas DataFrame queries to "retrieve" the reviews for the result books and returns the weighted sum reported for each book.

Indicative Results

For the same search as the first query, but for the user with userID:8

```

Search for: find
For results based on custom metric, enter userID: 8
Book n1 title: "Let's Find Out About Houses (Let's Find Out Series)" with custom metric score: 5.1675225
Book n2 title: "You Can Find Anybody!" with custom metric score: 4.853386666666667
Book n3 title: "Jamaica's Find" with custom metric score: 4.688945
Book n4 title: "Find Santa Claus as he brings Christmas joy (Look & find books)" with custom metric score: 4.364984
Book n5 title: "Let's Find Pokemon!" with custom metric score: 4.309413
Book n6 title: "Keeping The Love You Find" with custom metric score: 4.275654666666667
Book n7 title: "Find Scrooge in a Christmas Carol (Look & Find Books)" with custom metric score: 4.1675225
Book n8 title: "Evening Would Find Me" with custom metric score: 3.98672
Book n9 title: "Somewhere I'll Find You" with custom metric score: 3.98672
Book n10 title: "Find the Littlest Angels As They Celebrate Christmas (Look & Find Books)" with custom metric score: 3.864984

```

If the user has not uploaded a review for any of the original elasticsearch results, then the new metric slightly affects the ranking based on the average of the total reviews of each result.

ASK 3

For the 3^o question, a neural network is asked to train a neural network that will "predicts" how a user would rate a random book based on their previous reviews and the summary of the book.

First, we need to separate the training data from the overall database. This is done by the function "get_training_data()", which takes the user ID as input, and returns the books that have rated by the user and their abstracts, as well as their categories and their isbn identifiers.

```
66 def get_training_data(userID):
67     df_books = pd.read_csv("BX-Books.csv")
68     df_rating = pd.read_csv("BX-Book-Ratings.csv")
69     user_ratings = df_rating.query("uid == @userID")
70     ratings = user_ratings['rating'].tolist()
71
72     summaries = [df_books.query("isbn == @bookcode")['summary'].to_list() for bookcode in user_ratings.isbn.tolist()]
73     categories = [df_books.query("isbn == @bookcode")['category'].to_list() for bookcode in user_ratings.isbn.tolist()]
74     bookcodes = user_ratings.isbn.tolist()
75     # kratame mono ta vivlia ta opoia exoun summaries giati proseksa oti iparxoun kai merika kena
76
77     ratings_new = []
78     for i, t in enumerate(summaries):
79         if t:
80             ratings_new.append(ratings[i])
81     ratings_new = np.array(ratings_new) # numpy array gia na perastei sto modelo
82     summaries = [i[0] for i in summaries if i] # as lists
83
84     return [summaries, ratings_new, categories, bookcodes]
```

Next, we need to convert the book summaries into a suitable format to be used by the model, using the "Word Embeddings" technique. For this purpose, we will train another model, using the "word2vec()" function of the "gensim" library. This model, which will be trained on all summaries of the "BX-Books.csv" file, will take as input an alphanumeric and return a vector of 100 dimensions.

Ideally, this model will allocate a vector space to each word.

Words that have similar "meaning" or are used in similar contexts will be a short distance apart in vector space. The function that performs the above operation is the function "create_word2vec_model()":

```
129 def create_word2vec_model():
130     df_books = pd.read_csv("BX-Books.csv")
131     summaries = df_books['summary'].to_list()
132     clean_summaries = texts_preprocessing(summaries)
133     model = Word2Vec(clean_summaries, min_count=1, vector_size=100)
134     model.save("word2vecmodel.model")
135     print("Word2Vec model created and saved in project directory")
136
137     return model
```

For the most effective training , prior to introduction to the model, the texts are pre-processed. During pre-processing we use regular expressions (library "re") to remove from texts:

1. The multiple spaces
2. The hyphens between words
3. A list of English stopwords
4. The punctuation marks

Then, we split the text and remove the numeric characters and short words (<1). This is the function of the function "texts_preprocessing()".

We train the word embeddings model and save it in the directory. Because training takes time, it is not practical to repeat it every time the program is run. So, we save the model in the directory as "wordtovecmodel.model" . For even faster execution of the program, we also save all the word vectors generated in a separate file "word2vec.wordvectors".

Finally, with the function "get_document_vectors()" we convert the summaries of the training data into vectors, taking the average of the vectors of their words, and train the neural network with the function "get_sequential_model()".

```
140 def get_document_vectors(list_of_texts, wv):
141     clean_list = texts_preprocessing(list_of_texts)
142     text_vectors = []
143     for text in clean_list:
144         word_vectors = []
145         zero_vector = np.zeros(100)
146         for word in text:
147             try:
148                 word_vectors.append(wv[word])
149             except KeyError:
150                 print("word not found")
151                 word_vectors.append(zero_vector)
152         if word_vectors:
153             word_vectors = np.asarray(word_vectors)
154             text_vector = word_vectors.mean(axis=0) # text vector is the average of its word vectors
155         else:
156             text_vector = zero_vector
157         text_vectors.append(np.asarray(text_vector))
158
159     return np.asarray(text_vectors)
```



```

162 def get_sequential_model(training_data_x, training_data_y, embeddings):
163     training_vectors = get_document_vectors(training_data_x, embeddings)
164     # create model
165
166     model = Sequential()
167
168     model.add(Dense(128, activation='relu', input_shape=training_vectors[0].shape))
169     model.add(Dense(1, activation='linear'))
170     model.compile(optimizer='adam', loss='mse', metrics=['mae', 'mse'])
171
172     print("Training Neural Network . . .")
173     history = model.fit(training_vectors, training_data_y, epochs=500, verbose=0)
174     hist = pd.DataFrame(history.history)
175     hist['epoch'] = history.epoch # to use hist.tail method
176     print("Model trained")
177
178     return model, hist

```

As a "Linear Regression" model the last level consists of a single "neuron" with a linear activation function, and the error function of the model is based on the mean square error. The model is trained for 500 epochs, and the "hist" DataFrame stores the error metrics during training in case we want to print them.

Finally, with the functions "update_ratings_with_sequential()" and "display_custom_metric_results()" we update the ratings with the model predictions and display the results

For example:

```

OK! bulking succesful
Search for: find
UserID: 8
2022-08-29 02:14:55.869024: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized with
NVIDIA GPU acceleration. To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-08-29 02:14:55.876369: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default
min size of 10 threads and max size of 30 threads
Training Neural Network . . .
Model trained
Book n1 title: "Jamaica's Find" with elasticscore = 9.37789"
Book n2 title: "Let's Find Pokemon!" with elasticscore = 8.618826"
Book n3 title: "Let's Find Out About Houses (Let's Find Out Series)" with elasticscore = 8.335045"
Book n4 title: "Find Scrooge in a Christmas Carol (Look & Find Books)" with elasticscore = 8.335045"
Book n5 title: "Evening Would Find Me" with elasticscore = 7.97344"
Book n6 title: "You Can Find Anybody!" with elasticscore = 7.97344"
Book n7 title: "Somewhere I'll Find You" with elasticscore = 7.97344"
Book n8 title: "Find the Littlest Angels As They Celebrate Christmas (Look & Find Books)" with elasticscore = 7.729968"
Book n9 title: "Find Santa Claus as he brings Christmas joy (Look & find books)" with elasticscore = 7.729968"
Book n10 title: "Keeping The Love You Find" with elasticscore = 7.417976"

```

```

After rating predictions:
Book n1 title: "Jamaica's Find" with score = 8.688945"
Book n2 title: "Find Scrooge in a Christmas Carol (Look & Find Books)" with score = 6.912729332885742"
Book n3 title: "Find Santa Claus as he brings Christmas joy (Look & find books)" with score = 6.850273764404297"
Book n4 title: "You Can Find Anybody!" with score = 6.819834878336589"
Book n5 title: "Find the Littlest Angels As They Celebrate Christmas (Look & Find Books)" with score = 6.486250174316407"
Book n6 title: "Keeping The Love You Find" with score = 6.458265559916178"
Book n7 title: "Evening Would Find Me" with score = 6.374220190734864"
Book n8 title: "Let's Find Pokemon!" with score = 5.923035665405274"
Book n9 title: "Let's Find Out About Houses (Let's Find Out Series)" with score = 5.806733677825927"
Book n10 title: "Somewhere I'll Find You" with score = 5.6217957598876955"

```


ASK 4

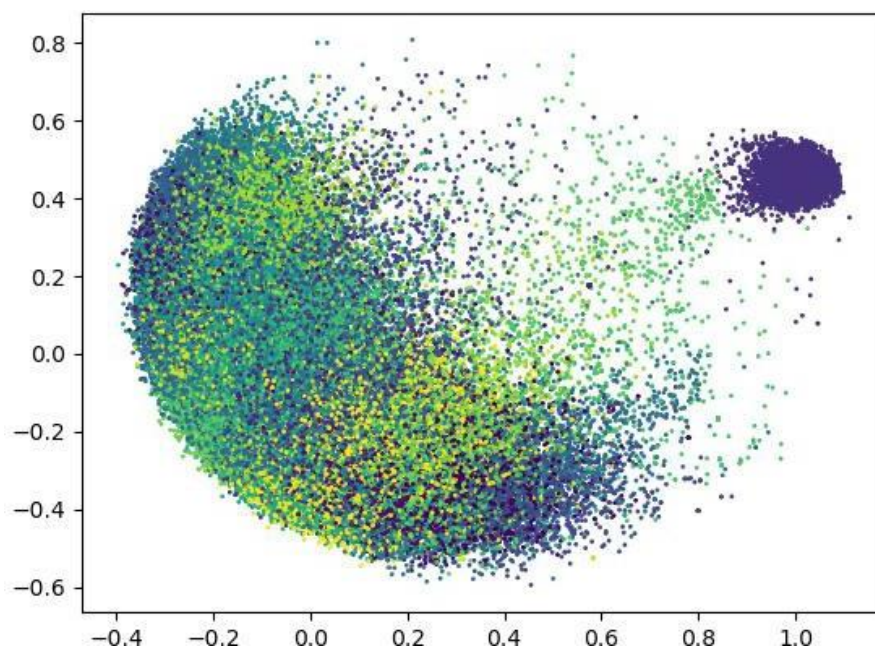
In the 4^o query, the k-means algorithm is applied to the set of books. For word embeddings the model trained in the previous question is used. In order for the kmeans algorithm to sort the samples based on cosine similarity, we normalize the vectors texts. This is because for two normalized vectors x, y the following relationship between Euclidean distance and cosine similarity holds:

$$EuclD(x, y) = 2 - 2\text{CosineSimilarity}(x, y)$$

Thus, although the default metric of Scikit-learn's kmeans is Euclidean distance, there is now a direct relationship with cosine similarity. So, by calculating the distance between a sample and the cluster center we can easily obtain the cosine similarity from the previous relation.

The scikit-learn library was used to train the classification model and the books were randomly selected to be divided into 50 clusters. The vectors before being entered into the algorithm were projected in 2D space by "Partial Component Decomposition". Then, using the "matplotlib" library we display the classification results as an illustration.

Of course, the number of samples and groups does not allow us to obtain any meaningful information from the graph.



For speed, the first time the program is run, it applies the algorithm and saves the clusters in a csv file named "BooksClustered.csv" which is accessed in subsequent runs. After clustering, the program prints the following data for each cluster:

1. Category of books with the most appearances
2. Average reviews for the books in the cluster
3. Average age of people who submitted reviews
4. Region from which most reviews come from

The program then asks the user to enter a cluster number for which more detailed graphs about the above cluster properties are displayed.

Indicative Results

```
Top genre in cluster 26 is ['fiction']
The average ratings were 3.0737169830270163
The average age of the people who rated is 36.5632183908046
Most of them were in portland, oregon, usa

Top genre in cluster 27 is ['business & economics']
The average ratings were 3.04863313698837
The average age of the people who rated is 37.472103004291846
Most of them were in houston, texas, usa

Top genre in cluster 28 is ['fiction']
The average ratings were 2.479256080114449
The average age of the people who rated is 38.975694444444444
Most of them were in chicago, illinois, usa

Top genre in cluster 29 is ['computers']
The average ratings were 3.501603135019594
The average age of the people who rated is 32.25396825396825
Most of them were in toronto, ontario, canada
```

And for cluster 13

[illegible]

Value	Occurrences (approx.)
a	13500
0	0
1	0
2	100
3	200
4	200
5	800
6	600
7	1400
8	2000
9	1300
10	1400

Countries

