Information Retrieval

Implementation Exercise

Fall Semester 2022

Konstantinos Ntounis

Department: ECE

Question 1

As a first step we need to load the data from the book file into elasticsearch using the Bulk API - helpers.bulk() function.

We create the following generator function so that we don't store all the information in memory before performing bulking:

(load_books.py file)

```
def doc_generator(dataframe):
    df_iter = dataframe.iterrows()
    for index, document in df_iter:
        yield {
            "_index": 'books',
            "_source": document.to_dict(),
        }
```

To implement the search for the relevant books, we continue in the q1.py file , where we use the following query to get the search results of the search_term text, which is given by the user during the program execution. It has

```
def elastic_query(search_term):
    total = es.count(index="books")['count']
    # Return info for 10% of total books
    resp = es.search(index="books", query={"query_string": {"query": search_term}},size=total*0.1)
    return resp
```

set to return 10% of the books, after finding their number.

Returns a dictionary type that includes the maximum score, and for related books all their information and the score they have for that search according to the elasticsearch metric.

To build our own metrics, we take the list of elasticsearch scores and sort it using the aggregate function. Inside it, we normalize the elasticsearch score and get the user's rating for each book. Finally, the generated metrics takes into account 50% of the elastic score and 50% of the user's score, if any.

```
def aggregate(book,df,uid,maxscore):
    scaled_score=book['_score']*10/maxscore
    isbn=book['_source']['isbn']

    user_rating = df.query(f"isbn=='{isbn}' and uid=={uid}"
    if user_rating.empty: agg_rating = scaled_score / 2
    else:
        rating = user_rating.iloc[0]['rating']
        agg_rating = scaled_score / 2 + rating / 2
    book['custom_rating'] = round(agg_rating,3)
```

```
# Get elasticsearch results, user id, ratings dataframe and print custom order of results
def print_custom(query_res,df,uid):
    maxscore = query_res['hits']['max_score']
    books=query_res['hits']['hits']
    srt = sorted(books, key=lambda book: aggregate(book, df, uid, maxscore), reverse=True)

    print("Got %d Hits" % (query_res['hits']['total']['value']))
    print("Top 10% of books are displayed below:")
    for i, hit in enumerate(srt):
        print(f"{i+1}. Score: {hit['custom_rating']}",
              " %(book_title)s, %(book_author)s: %(year_of_publication)s" % hit["_source"])
```

For example, the difference of 2 metrics for user "6575" who has rated the book "Horse Heaven" with a 9 receives the following results for the search "Horse":

```
User id: 6575
Search for: Horse
Got 659 Hits
Top 10% of books are displayed below:
1. Score: 7.807  Horse Heaven (Ballantine Reader's Circle), Jane Smiley: 2001
2. Score: 6.337  Barn Blind, Jane Smiley: 1993
3. Score: 5.0  The Last Castaways, Harry Horse: 2003
4. Score: 5.0  Last Cowboys, Harry Horse: 1999
5. Score: 4.997  Driving Force, Dick Francis: 1994
6. Score: 4.405  Horse Play (Horse Crazy Series), Virginia Vail: 1989
7. Score: 4.199  Gunsmith Cats: Return of Gray, Dark Horse Comics: 1998
8. Score: 4.199  Gunsmith Cats Misfire, Dark Horse Comics: 1997
```

Otherwise the books in the 3rd and 4th position would appear first:

```
Search for: Horse
Got 659 Hits
Top 10% of books are displayed below:
1. Score: 10.782821  The Last Castaways, Harry Horse: 2003
2. Score: 10.782821  Last Cowboys, Harry Horse: 1999
3. Score: 9.499819  Horse Play (Horse Crazy Series), Virginia Vail: 1989
4. Score: 9.054375  Gunsmith Cats: Return of Gray, Dark Horse Comics: 1998
5. Score: 9.054375  Gunsmith Cats Misfire, Dark Horse Comics: 1997
```

Observation:
In the general case the results should first be returned by elasticsearch, so we won't see completely irrelevant books that, e.g., have gotten 10 from the user

Question 2

To group users by age and country of residence, using the K-means algorithm, we take the following steps

1. Pre-processing of data:
Finding the country after splitting the 'location' column for users. The K-means algorithm cannot cluster categorical data such as country, since it measures Euclidean distances of multidimensional data. So we need to convert the new categorical variable 'country' into numeric values using one-hot encoding. In python it is provided as sklearn.preprocessing.OneHotEncoder().

We are also performing a filtering, as there are ages >110 and users that do not include country or age.

```python
K=100

def preprocessing(df):
    # We get 2 columns from right splitting and we keep the 2nd: country
    # we assign it to column 'country'
    df['country'] = df['location'].str.rsplit(pat=',', n=1, expand=True)[1].str.strip('\"\' ')
    df.dropna(inplace=True)
    # Many users were over 110 years old
    df = df[df['age'] < 100].copy()

    # One-hot encoding the countries
    encoder = OneHotEncoder(handle_unknown="ignore", categories="auto")
    country_enc = encoder.fit_transform(df[['country']])
    df[['onehot']] = country_enc
    # Combine the ages and one-hot encoded countries
    data = np.hstack((df[['age']].values, country_enc.toarray()))
    return df, data
```

2. We set the number of clusters to 100, in order to have more reliable conclusions for each cluster, but also to reduce the runtime as reducing the size of each cluster reduces the number of scores that have to be filled in for all users (even for 100 clusters the size of the filled in scores ends up being 800MB). Otherwise we find an ideal point from the **elbow diagram** generated as:

```
def optimise_k_means(data, max_k):
    means = []
    inertias = []

    for k in range(1, max_k):
        kmeans = KMeans(n_clusters=k, n_init='auto')
        kmeans.fit(data)
        means.append(k)
        inertias.append(kmeans.inertia_)

    fig = plt.subplots(figsize=(10, 5))
    plt.plot(means, inertias, 'o-')
    plt.xlabel("Number of Clusters")
    plt.ylabel("Inertia")
    plt.grid(True)
    plt.show()
```

3. We train the K-means model of sklearn.cluster on the new data table using the specified value of k = 100. We store each user's cluster in a new column.

```
def apply_kmeans(k, data, df):
    # Perform k-means clustering
    kmeans = KMeans(n_clusters=k, n_init='auto')
    kmeans.fit(data)

    # Get the cluster labels for each data point
    df[f'cluster_{k}'] = kmeans.labels_
    clustered_users = df.drop(columns=['location', 'onehot'])
    clustered_users.to_csv('clustered_users.csv')
```

4. We use the custom model to fill in the ratings for each user, for those books not rated by him but rated by his cluster. We use the average values, from the DataFrame 'clusterbooks' grouping, and repeat for each cluster the gap filling resulting from all uid-isbn combinations within the cluster.

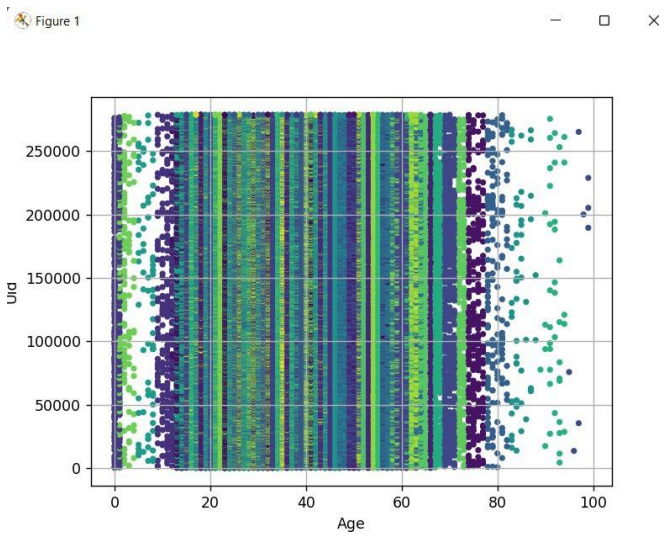
```
def fill_all():
    users = pd.read_csv('clustered_users.csv').dropna()
    rat = pd.read_csv('BX-Book-Ratings.csv')
    # keep useful ratings their user clusters
    ratings = pd.merge(rat, users[['uid', f'cluster_{K}']], how="inner", on=["uid"])
    clusterbooks = ratings.groupby([f'cluster_{K}', 'isbn'])['rating'].mean() \
        .reset_index()  # cluster, isbn, rating

    for c in range(K):
        userdata = ratings[ratings[f'cluster_{K}'] == c].drop(columns=[f'cluster_{K}'])  # uid, isbn, rating
        clusterdata = clusterbooks[clusterbooks[f'cluster_{K}'] == c][['isbn', 'rating']]  # isbn, rating

        uids = pd.Series(users[users[f'cluster_{K}'] == c]['uid'].unique(), name='uid')
        isbns = pd.Series(clusterbooks[clusterbooks[f'cluster_{K}'] == c]['isbn'].unique(), name='isbn')
        full_table = pd.merge(uids, isbns, how='cross')  # full table of uid-isbn for every user and book in the cluster
        full_table = pd.merge(full_table, userdata, how='left', on=['uid', 'isbn'])  # uid, isbn, rating

        # replace NaN with values from clusters
        to_fill = full_table[full_table['rating'].isna()][['uid', 'isbn']]
        filled = pd.merge(to_fill, clusterdata, how='left', on='isbn')
        print(f"Filled cluster {c} users..")
        rat = pd.concat([rat, filled])
    rat.to_csv("complete_ratings.csv",index=False)
```

5. For visualisation we create a scatter plot where each cluster is assigned a different colour and the x-axis represents the age and the y-axis represents the number of the user (it does not make sense to present a categorical variable like country)



Result

The completed ratings have now been saved in the file "complete_ratings.csv" and can be tested in question 1 by loading the new file.

Question 3

In the 3rd question, we create a neural network that is trained for a specific cluster of users with an input of the summaries of books that the cluster has rated and an output of their rating.

The input must first be pre-processed, since it is just a set of words. This is achieved with **Word Embeddings** which are being created for each useful word in the summary and being mapped to a vector that is chosen to have 100 dimensions. The final vector of the summary is the average of these. Note that to keep the useful words we remove the punctuation and stopwords of English, provided by nltk.corpus .

```
def clean_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.split()
    text = [word for word in text if word not in stop_words]
    return text
```

The vectors for the vocabulary of the summaries are created with genism's Word2Vec. The wordvectors are stored for use in subsequent runs.

```
def train_word2vec_model(df):
    model = Word2Vec(df['cleaned_summary'], vector_size=100, window=5, min_count=1, workers=4)
    model.train(df['cleaned_summary'], total_examples=len(df['cleaned_summary']), epochs=10)
    print("Word2Vec model trained...")
    model.save('book_summary_word2vec.model')
    model.wv.save('word2vec.wordvectors')
    print("Word vectors saved for future use...")
```

The conversion of all summaries to vectors:

```
def text_to_vector(wv,text):
    text = text.split()
    vector = np.zeros(100)
    for word in text:
        try:
            vector += wv[word]
        except KeyError:
            pass
    return vector / len(text)
```

```
def vectorize_summaries(df,using):
    df = df.drop(
        columns=['book_title', 'book_author', 'year_of_publication', 'publisher', 'category'])
    df['vector'] = df['summary'].apply(lambda x: text_to_vector(using, x))
    print("All summaries are now translated into vectors.")
    return df.drop(columns='summary')
```

We then create the neural network that will score the summary vectors that we introduce to it each time.

Its structure will be as follows: Input layer, 1 Hidden layer, Output layer

```
def make_neural_network(dim):
    model = Sequential()
    model.add(Dense(128, activation='relu', input_shape=(dim,)))
    model.add(Dense(1, activation='linear'))

    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

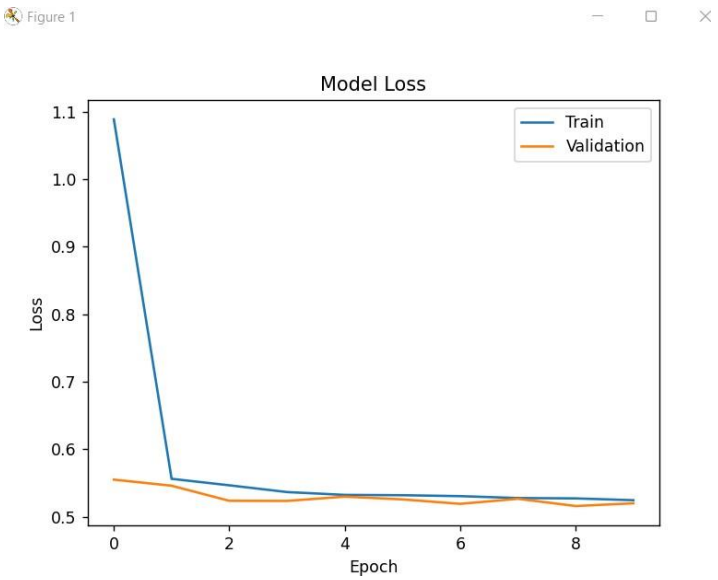
We split the training data for cluster 1 e.g. implemented in 80%-20% for training and validation respectively and save our model. We also create the corresponding graph that records the reduction in error at each epoch.

```
def train_ratings_model(model,df_vec_summaries,df_ratings,cluster):

    training_df = pd.merge(df_ratings.query("cluster_100==@cluster"),df_vec_summaries,how="inner",on="isbn")
    X = training_df['vector'].tolist()
    y = training_df['rating'].tolist()
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
    history = model.fit(np.array(X_train), np.array(y_train), epochs=10, batch_size=32,
              validation_data=(np.array(X_val), np.array(y_val)))
    print(f"Model trained with cluster {cluster} ratings...")
    plot_loss(history)
    model.save("model_cluster_1.h5")
    return model
```

The function we will use that takes the ISBN of the book and returns the predicted estimate is:

```
def predict_rating(df_vectors,isbn):
    summary_vector=df_vectors[df_vectors['isbn']==isbn]['vector'].iloc[0].tolist()
    prediction = model.predict([summary_vector])
    return prediction[0][0]
```



For example we value ISBN "0374157065"

```
    # We use cluster 1 ratings to train our model,
    # and we save model because it takes 10 mins to train
    if not exists("model_cluster_1.h5"):
        model = make_neural_network(dim=100)
        model = train_ratings_model(model,vectored_summaries_df,ratings_df,1)
    else: model = keras.models.load_model("model_cluster_1.h5", compile=False)

    isbn='0374157065'
    pre = predict_rating(vectored_summaries_df,isbn)
    print(f"Predicted rating for isbn {isbn} is: ", pre)
```

```
Summaries cleaned!
All summaries are now translated into vectors.
```

```
Predicted rating for isbn 0374157065 is:  3.4121177

Process finished with exit code 0
```

To use score prediction in elasticsearch we need to predict the scores for the returned books from each search.

The main programme shall be implemented as follows:

```
if __name__=="__main__":
    ratings_df = pd.read_csv('complete_ratings.csv')
    clusters_df = pd.read_csv('clustered_users.csv')
    ratings_df = pd.merge(ratings_df,clusters_df.drop(columns=['age','country']),how='left',on='uid')
    summaries_df = pd.read_csv('BX-Books.csv')


    if not exists(WV_NAME): train_word2vec_model(summaries_df)
    else: print("Word2Vec model and vectors already exists.\nLoading it...")
    wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')

    # Clean Summaries
    summaries_df['cleaned_summary'] = summaries_df['summary'].apply(clean_text)
    print("Summaries cleaned!")
    # Vectorize
    vectored_summaries_df = vectorize_summaries(summaries_df,using=wv)  # isbn, vector
    vectored_summaries_df.to_csv("vectored_summaries.csv")
    print("Summary vectors created and saved.")



    # For example, suppose user 54 -> cluster=1
    # We use cluster-1 ratings to train our model,
    # and we save model because it takes 10 mins to train
    if not exists("model_cluster_1.h5"):
        model = make_neural_network(dim=100)
        model = train_ratings_model(model,vectored_summaries_df,ratings_df,1)
    else: model = keras.models.load_model("model_cluster_1.h5", compile=False)

    new_search(vectored_summaries_df)
```

And the search is done with the following functions and the print_custom of the 1st question:

```
def get_isbn_list(query_res):
    maxscore = query_res['hits']['max_score']
    books=query_res['hits']['hits']
    L=[]
    for hit in books: L.append(hit['isbn'])
    return L

def new_search(vec_sum):
    uid = int(input("User id: "))
    df_ratings = pd.read_csv("BX-Book-Ratings.csv").query("uid==@uid")

    query = input("Search for: ")
    res = elastic_query(query)
    isbns = get_isbn_list(res)
    for isbn in isbns:
        pred = predict_rating(vec_sum,isbn)
        df_ratings.append(pd.Series([uid,isbn,pred]))
    print_custom(res,df_ratings,uid)
```