Ανάκτηση Πληροφορίας

Υλοποιητική άσκηση

Χειμερινό Εξάμηνο 2022

Κωνσταντίνος Ντούνης

Τμήμα: HMTY

Ερώτηση 1

Ως πρώτο βήμα πρέπει να φορτώσουμε στην elasticsearch τα δεδομένα από το αρχείο των βιβλίων με την χρήση του Bulk API – συνάρτηση helpers.bulk().

Δημιουργούμε το εξής generator function ώστε να μην αποθηκεύσουμε όλες τις πληροφορίες στη μνήμη πριν από την εκτέλεση του bulking:

(αρχείο load_books.py)

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

Για την εφαρμογή της αναζήτησης των σχετικών βιβλίων συνεχίζουμε στο αρχείο q1.py , όπου χρησιμοποιούμε το παρακάτω query για να λάβουμε τα αποτελέσματα της αναζήτησης του κειμένου search_term, το οποίο δίνεται από τον χρήστη κατά την διάρκεια της εκτέλεσης του προγράμματος. Έχει

```
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
```

ρυθμιστεί ώστε να επιστρέφει το 10% των βιβλίων, μετά από εύρεση του αριθμού τους.

Επιστρέφει ένα dictionary τύπο που περιλαμβάνει την μέγιστη βαθμολογία, και για τα σχετικά βιβλία όλες τις πληροφορίες τους καθώς και το score που έχουν για την συγκεκριμένη αναζήτηση σύμφωνα με την μετρική της elasticsearch.

Για να φτιάξουμε την δική μας μετρική, παίρνουμε την λίστα των αποτελεσμάτων της elasticsearch και την κάνουμε sort χρησιμοποιώντας την συνάρτηση aggregate. Σε αυτήν, κανονικοποιούμε το elasticsearch score και λαμβάνουμε για το κάθε βιβλίο την αξιολόγηση του χρήστη. Τέλος, η δημιουργούμενη μετρική συνυπολογίζει κατά 50% την βαθμολογία της elastic και κατά 50% του χρήστη, αν υπάρχει.

```python
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)
```

```python
# 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"])
```

Για παράδειγμα, η διαφορά των 2 μετρικών για τον χρήστη 6575 που έχει βαθμολογήσει με 9 το βιβλίο “Horse Heaven” λαμβάνει για την αναζήτηση “Horse” τα παρακάτω αποτελέσματα:

```
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
```

Ενώ διαφορετικά τα βιβλία στην 3η και 4η θέση θα εμφανίζονταν πρώτα:

```
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
```

Παρατήρηση:
Στη γενική περίπτωση τα αποτελέσματα θα πρέπει πρώτα να επιστρέφονται από την elasticsearch, οπότε δεν θα δούμε τελείως άσχετα βιβλία που, π.χ. έχουν πάρει 10 από τον χρήστη

Ερώτηση 2

Για να ομαδοποιήσουμε τους χρήστες με βάση την ηλικία και την χώρα που κατοικούν, χρησιμοποιώντας τον αλγόριθμο K-means κάνουμε τα εξής βήματα

1. Προεπεξεργασία των δεδομένων:
Εύρεση της χώρας μετά από splitting της στήλης ‘location’ για τους χρήστες. Ο αλγόριθμος K-means δεν μπορεί να ομαδοποιήσει κατηγορικά δεδομένα όπως η χώρα, αφού μετρά ευκλείδειες αποστάσεις πολυδιάστατων δεδομένων. Άρα πρέπει να μετατρέψουμε τη νέα κατηγορική μεταβλητή "country" σε αριθμητικές τιμές με χρήση one-hot encoding. Στη python παρέχεται ως sklearn.preprocessing.OneHotEncoder().

Επίσης κάνουμε ένα ξεκαθάρισμα, καθώς υπάρχουν ηλικίες >110 και χρήστες που δεν περιλαμβάνουν χώρα ή ηλικία.

```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. Καθορίζουμε τον αριθμό των συστάδων σε 100, ώστε να έχουμε πιο ασφαλή συμπεράσματα για την κάθε συστάδα, αλλά και για να μειώσουμε τον χρόνο εκτέλεσης καθώς μειώνοντας το μέγεθος κάθε συστάδας μειώνονται και οι βαθμολογίες που έχουν καταχωρήσει και πρέπει να συμπληρωθούν για όλους τους χρήστες (ακόμα και για 100 συστάδες το μέγεθος των συμπληρωμένων βαθμολογιών καταλήγει να είναι 800MB). Σε άλλη περίπτωση βρίσκουμε ιδανικότερο σημείο από το elbow διάγραμμα που δημιουργείται ώς:

```python
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. Εκπαιδεύουμε το μοντέλο K-means της sklearn.cluster στον νέο πίνακα δεδομένων χρησιμοποιώντας την καθορισμένη τιμή του k = 100. Αποθηκεύουμε το cluster του κάθε χρήστη σε νέα στήλη.

```python
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. Χρησιμοποιούμε το προσαρμοσμένο μοντέλο για να συμπληρώσουμε τις βαθμολογίες για κάθε χρήστη, όσων βιβλίων δεν έχει βαθμολογήσει εκείνος αλλά έχει βαθμολογήσει η συστάδα του. Χρησιμοποιούμε τις μέσες τιμές, από το grouping του DataFrame ‘clusterbooks’ και επαναλαμβάνουμε για κάθε cluster την συμπλήρωση των κενών που προκύπτουν από όλους τους συνδυασμούς uid-isbn μέσα στο cluster.

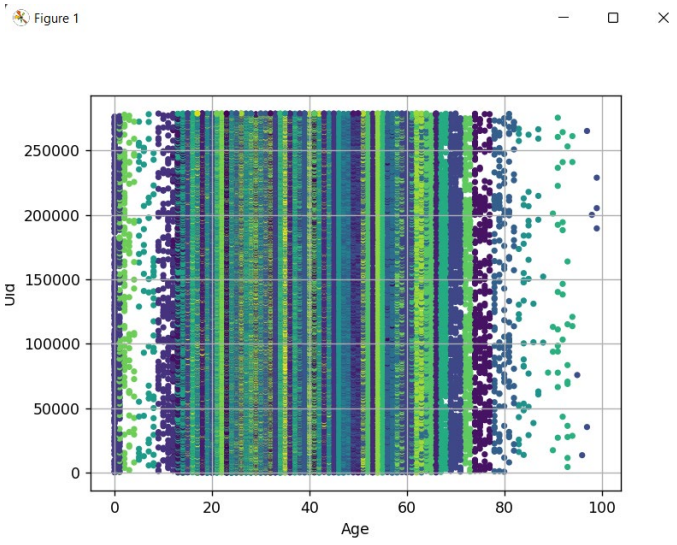
```
def fill_all():
    users = pd.read_csv('clustered_users.csv').dropna()
    rat = pd.read_csv('BX-Book-Ratings.csv')
    # keep useful ratings their user clusters
    ratings = pd.merge(rat, users[['uid', f'cluster_{K}']], how="inner", on=["uid"])
    clusterbooks = ratings.groupby([f'cluster_{K}', '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. Για οπτικοποίηση δημιουργούμε ένα διάγραμμα διασποράς όπου σε κάθε συστάδα αποδίδεται διαφορετικό χρώμα και ο άξονας x αντιπροσωπεύει την ηλικία και ο άξονας y αντιπροσωπεύει τον αριθμό του χρήστη (δεν έχει νόημα να παρουσιάσουμε μια κατηγορική μεταβλητή όπως η χώρα)



Αποτέλεσμα

Οι συμπληρωμένες βαθμολογίες έχουν τώρα αποθηκευτεί στο αρχείο “complete_ratings.csv” και μπορεί να δοκιμασθούν στο ερώτημα 1, με φόρτωση του νέου αρχείου.

Ερώτηση 3

Στο 3ο ερώτημα δημιουργήθηκε ένα νευρωνικό δίκτυο το οποίο εκπαιδεύουμε για συγκεκριμένη συστάδα χρηστών με είσοδο τις περιλήψεις των βιβλίων που η συστάδα έχει βαθμολογήσει και έξοδο την βαθμολογία τους.

Η είσοδος πρέπει πρώτα να υποστεί προεπεξεργασία, αφού είναι απλώς ένα σύνολο λέξεων. Αυτό το πετυχαίνουμε με τα Word Embeddings τα οποία δημιουργούνται για κάθε χρήσιμη λέξη της περίληψης και τις αντιστοιχίζει σε διάνυσμα που επιλέγεται να έχει 100 διαστάσεις. Το τελικό διάνυσμα της περίληψης είναι η μέση τιμή αυτών. Σημειώνεται πως για να κρατήσουμε τις χρήσιμες λέξεις αφαιρούμε τα σημεία στίξης και τα stopwords της αγγλικής, που παρέχονται από το 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
```

Τα διανύσματα για το λεξιλόγιο των περιλήψεων δημιουργούνται με το Word2Vec της genism. Τα wordvectors αποθηκεύονται για χρήση σε επόμενες εκτελέσεις.

```
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...")
```

Η μετατροπή όλων των περιλήψεων σε διανύσματα:

```python
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)
```

```python
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')
```

Στη συνέχεια δημιουργούμε το νευρωνικό δίκτυο που θα βαθμολογήσει τα διανύσματα περιλήψεων που θα του εισάγουμε κάθε φορά.

Η δομή του θα είναι η εξής: Input layer, 1 Hidden layer, Output layer

```python
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
```

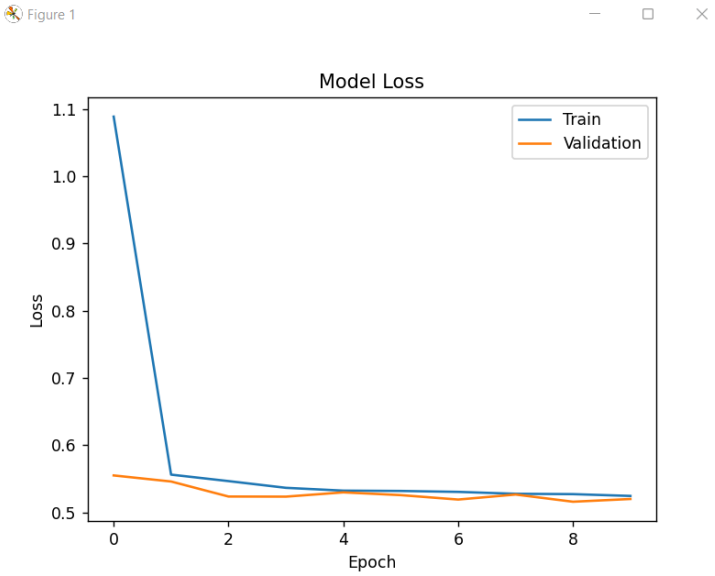
Χωρίζουμε τα δεδομένα εκπαίδευσης για τον cluster 1 πχ που έχει υλοποιηθεί σε 80%-20% για εκπαίδευση και validation αντίστοιχα και αποθηκεύουμε το μοντέλο μας. Δημιουργούμε και το αντίστοιχο διάγραμμα που καταγράφει τη μείωση του σφάλματος σε κάθε epoch.

```python
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
```

Η συνάρτηση που θα χρησιμοποιήσουμε και δέχεται το isbn του βιβλίου και επιστρέφει την προβλεπόμενη εκτίμηση είναι η εξής:

```
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]
```



Για παράδειγμα εκτιμούμε το 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
```

Για την χρήση της πρόβλεψης βαθμολογιών στην elasticsearch πρέπει να προβλέψουμε τις βαθμολογίες για τα επιστρεφόμενα βιβλία από την κάθε αναζήτηση .

Το βασικό πρόγραμμα υλοποιείται ως εξής:

```
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)
```

Και η αναζήτηση γίνεται με τις εξής συναρτήσεις και την print_custom του 1ου ερωτηματος:

```
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)
```