Γλωσσική Τεχνολογία

Παρλαπάνης Αντώνιος

ΑΜ 1059709

1. Web crawler

Για τη προσκόμιση των ιστοσελίδων χρησιμοποιήθηκε η βιβλιοθήκη Beatiful Soup. Παρόλο που το beatiful soup είναι parsing library και όχι web scraping framework, κατάφερα (μέσω του navbar) να παίρνω άρθα απο διαφορετικές σελίδες δίνωντας μονο το root url.

Απο τα websites dailymail και echonomist παίρνω τα άρθρα απο όλα τα subpages ενώ απο τα υπόλοιπα (europe news,huffpost,the sun) παίρνω τα άρθρα μόνο απο μία σελίδα του website, γιατί θα είχα παρα πολλά άρθρα και αργούσε πολύ η διαδικασία .

Μόνο το αρχείο dailymail.py περιέχει σχολιασμένο κώδικα γιατί όλοι οι crawlers δουλέυουν με τον ίδιο τρόπο.

2. Προεπεξεργασία δεδομένων

Στο σύστημα αποθηκέυω μόνο το κείμενο,τους τίτλους και τα links.

Οσο αναφορά την προεπεξεργασία των δεδομένων, αρχικά κάνω import μια λίστα με 179 πολύ συχνά εμφανιζόμενα stopwords (the,and,me etc) απο το nltk.corpus.Ετσι, σε συνδυασμό με την αφαίρεση των μη αλφαριθμητικών χαρακτήρων μειώνω σημαντικά το μέγεθος των άρθρων και συνεπώς το χρόνο που χρειάζεται για την επεξεργασία τους.

```
for article in articles:
    iid = iid + 1
    article = [word for word in article if word not in basic_stopwords]  # remove basic stopwords
    article = [word for word in article if word.isalnum()]  # to keep only the alphanumeric characters
    tagged_article = nltk.pos_tag(article) #classifying words into their parts of speech and labeling them accordingly
    stopwords = gather_stopwords(tagged_article)  # creating a list with the stopwords found
    article = [custom_lemmatizer(word[0],word[1]) for word in tagged_article] #lemmatizing words with the help of pos tags.
    article = [word for word in article if word not in stopwords]#remove stopwords found from pos tags
    articles_processed[iid] = article #id -> article map
```

Στη συνέχεια χρησιμποιώντας το nltk.pos_tag δίνουμε σε κάθε λέξη το κατάλληλο tag.Έπειτα με τη χρήση της συνάρτησης WordNetLemmatizer() του nltk γίνονται οι κατάλληλες αλλαγές αναλογα με το αν η λέξη είναι ρήμα,ουσιαστικό,επίθετο η επίρρημα.Τέλος, παράλληλα με τη ληματοποίηση δημιουργήθηκε και μια λίστα stopwords απο λεξέις η οποίες ανήκαν στα closed class categories,και χρησιμοποιείται για την αφαίρεση επιπλέον λέξεων που δεν αφαιρέθηκαν στη πρώτη φάση.

Για τη δημιουργία του ανεστραμμένου ευρετηρίου χρησιμοποιώ ένθετο dictionary.

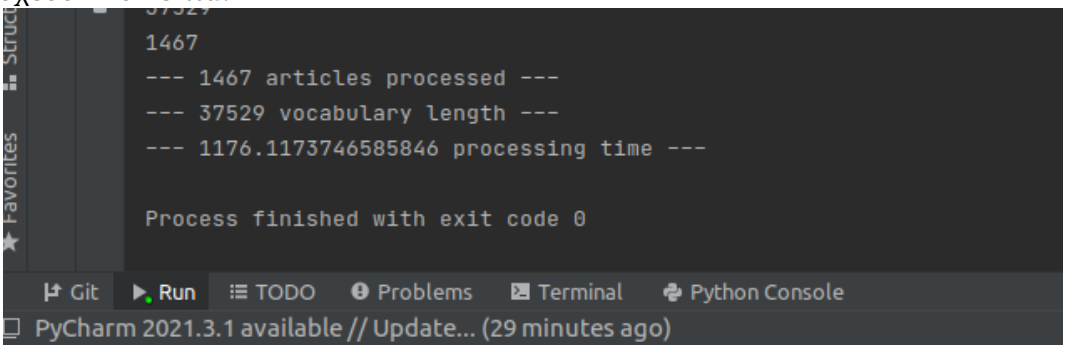
```
def create_index(articles_processed): #creating the inverted index
    inverted_index = {}
    for word in vocabulary:
        wordd = {} #using this dictionary to create the final nested dictionary
        for i in articles_processed:
            if word in articles_processed[i]:#if word exists in an article, calculate the tfidf value on that article and append it on the wordd dictionary
                tf = termfreq(articles_processed[i], word)
                idf = inverse_doc_freq(word)
                value = tf * idf
                wordd[i] = value
        inverted_index[word] = wordd #append the wordd dictionary to the inverted_index dictionary using as key the word
    return inverted_index
```

Για κάθε λέξη ψάχνω σε κάθε άρθρο.Αν η λέξη υπάρχει, υπολογίζω το tfidf value της λέξης σε εκείνο το άθρο και το βάζω στο dictionary wordd με key το id του άρθρου.Το ευρετήριο αποθηκέυετε σε μορφή json.


```
0.01268609342363223, "120": 0.010366621118565143, "121": 0.009245439134028149, "122": 0.011386288769571552, "123":
 0.00635756169285002, "172": 0.010747599457545295}, "vital": {"38": 0.0023180997407081976, "155": 0.025169984221640238": 0.002154551092355849, "92": 0.0028581111398596824, "107": 0.006445332264377446}, "raquel": {"38": 0,
115}, "science": {"38": 0.0015299846974336719, "83": 0.008970810275952763, "110": 0.014951350459921272, "116": 0,
.024577562399870582, "137": 0.016510693759422265, "148": 0.015291153879464937, "149": 0.011213512844940954, "160":
ing": {"38": 0.0025486086139931115}, "sink": {"38": 0.0025486086139931115}, "genetically": {"38": 0.0023180997407081
```


3. Αξιολόγηση ευρετηρίου

Για την αξιολόγηση του ευρετηρίου μετρήθηκε αρχικά ο χρόνος που χρειάζεται για την προεπεξεργασία 1467 άρθρων και τη δημιουργία του ανεστραμμένου ευρετηρίου.Για 1467 άρθρα χρειάστηκαν 1176 δευτερόλεπτα δηλαδή σχεδόν 20 λεπτά.



Στο μηχανισμό υποβολής ερωτημάτων υποβλήθηκαν 100 ερώτηματα.Χρειάστηκε

σχεδόν 1 δευτερόλεπτο.Αρα ο μέσος χρόνος απόκρισης είναι 0,01 δευτερόλεπτα. Οι μετρήσεις έγιναν στο αρχείο test.py.Ο μηχανισμός υποβολής ερωτημάτων είναι στο αρχείο merosAquery.py.

```
--- 0.9739956855773926 seconds ---
--- 0.009739956855773925 time per query  ---

Process finished with exit code 0
```

4. Μέρος Β

Χρησιμοποιούμε το Tfidfvectorizer του Scikit-learn για να μετατρέψουμε τη συλλογή απο raw κείμενα σε ενα matrix απο TF-IDF χαρακτηριστικά.

```
tf = TfidfVectorizer(stop_words='english', max_features=8000,use_idf=True) #Initializing the vectorizer.Setting 8000 fe
tfidf = tf.fit_transform(x_train)#fit_transform learns vocabulary and idf, returns document-term matrix.
```

Δίνουμε σαν παράμετρο ότι χρειαζόμαστε 8000 χαρακτηριστικά,να αφαιρεθούν stopwords και να χρησιμοποιηθεί idf.
Με τη χρήση του fit_transform δημιουργείτε ο χώρος χαρακτηριστικών,υπολογίζετε το idf κάθε λέξης για κάθε κείμενο και επιστρέφεται ένα document-term matrix με tfidf τιμές.

```
]: print(tfidf)
```

```
  (0, 1981)     0.08461100813909206
  (0, 4056)     0.02648576450181207
  (0, 29)       0.05000062345558237
  (0, 5361)     0.0387185237363595
  (0, 7570)     0.049829870998868646
  (0, 321)      0.056922108705844986
  (0, 7096)     0.019290752634784514
  (0, 6564)     0.06011884350932519
  (0, 3243)     0.045374092374743616
  (0, 4963)     0.02954481903322534
  (0, 6131)     0.0750476737092194
```

```
: print(tfidf.shape)
```

```
(11314, 8000)
```

Για τη κατηγοριοποίηση εγγράφου απο τη συλλογή Α δημιουργούμε διάνυσμα χαρακτηριστικών και το συγκρίνουμε με όλα τα κατηγοριοποιημένα έγγραφα χρησιμοποιώντας μια μετρική σχετικότητας (cosine similarity η euclidean distance).Το έγγραφο κατηγοριοποιείται στη κατηγορία του εγγράφου με το οποίο είχε τη μεγαλύτερη σχετικότητα.

```
def similarity(document,tfidf,metric):
    testvector = tf.transform(document)#create vector for each document that needs to be classified
    if metric=='cosine':
        score = cosine_similarity(testvector, tfidf)
    elif metric=='euclidean':
        score = euclidean_distances(testvector, tfidf)
    else:
        sys.exit()
    prediction = np.argmax(score, 1)#returns the index of the biggest value
    predicted = df_news_train['tag'].iloc[prediction]#locating the tag of the article that had the biggest score
    return predicted

#df=df_news_test
def classify(df,metric):
    classified = pd.DataFrame()#this is were the classified docs will be stored
    doc_count = len(df.index)
    correct = 0
    for row in df.itertuples():
        predicted = similarity(row[1],tfidf,metric)#returns the predicted class
        temp = pd.DataFrame( #creating a temporary dataframe to store the current classified doc
        {
            'document':row[1],
            'class':predicted
        },index=[0])
        classified = pd.concat([classified, temp])#merge temp dataframe with the final one
        actual = row[2]
        if actual == predicted[0]:#actual is the correct class from the df_news_test
            correct=correct + 1
    accuracy = correct/doc_count
    print(correct)
    percentage = "{:.0%}".format(accuracy)
    print(percentage)
    return classified

classified = classify(df_news_test,'cosine')
```

Το βασικό μέρος αυτής της διαδικασίας είναι οτι για τη δημιουργία των δια-

νυσμάτων χαρακτηριστικών στα έγγραφα απο τη συλλογή Α χρησιμοποιούμε το ίδιο TfidfVectorizer που ορίσαμε στην αρχή και κανουμε transform του εγγράφου και οχι τλφιτ_τρανσφορμ.Έτσι μετασχηματίζεται το έγγραφο που θέλουμε να κατηγοριοποιήσουμε σε document-term matrix χρησιμοποιώντας το vocabulary και document frequencies των εγγράφων της συλλογής Ε.
Για μετρικές σχετικότητας χρησιμοποιώ έτοιμες υλοποιήσεις απο το sklearn των cosine similarity,sigmoid kernel και euclidean distance.Το cosine similarity και το sigmoid kernel έχει 64% accuracy ενώ το euclidean distance έχει μόλις 5%.

```
45      return classified
46
47  #choose between cosine,euclidean,sigmoid
48  classified = classify(df_news_test,'cosine')
49
50
```

Στη συνάρτηση classify δίνουμε σαν όρισμα την μετρική σχετικότητας.Αποθηκεύω τα κατηγοριοποιημένα έγγραφα στο αρχείο classified.csv.
**Πρέπει να δώσετε στο path_train path_test το path των δεδομένων.

```
20
21  path_train= "/home/antonis/Downloads/20news-bydate/20news-bydate-train"
22  path_test= "/home/antonis/Downloads/20news-bydate/20news-bydate-test"
23
```

5. Παραδοτέα

Στον φάκελο crawlers βρίσκονται οιcrawlers.Στον φάκελο news βρίσκονται τα αρχεία json που περιέχουν τα άρθρα.Το αρχείο MerosA.py περιέχει την προεπεξεργασία των άρθρων και τη δημιουργία του ευρετηρίου.Το αρχείο MerosAquery.py περιέχει το σύστημα υποβολής ερωτημάτων.Το αρχείο test.py δίνει τα έτοιμα ερώτημα για την μέτρηση των χρόνων.Το αρχείο MerosB.py περιέχει το μέρος Β.Επίσης προστέθηκε και το αρχείο Meros2Notebook.ipynb για να έχετε τη δυνατότητα αν θέλετε να τρέξετε το μέρος 2 για διαφορετικές μετρικές χωρίς να τρέχει όλο απο την αρχή.Το αρχείο tf-idf.json περιέχει το ανεστραμμένο ευρετήριο.Το αρχείο article_map.json χρησιμεύει στο μηχανισμό υποβολής ερωτημάτων.