Συστήματα Ανάκτησης Πληροφοριών

Προγραμματιστική Εργασία: Μέρος Γ

Σοφία-Ζωή Σωτηρίου, p3210192

Για την εκπαίδευση του μοντέλου word2vec ενσωματώθηκε στον κώδικα του μέρους Β της εργασίας ο εξής κώδικας που δόθηκε από τα εργαστήρια για:

- τη δημιουργία μια λίστας με ένα string ανά doc η οποία θα δοθεί στο μοντέλο για εκπαίδευση.

```
def sentence_generator(jsonl_path, fields):
    sentences = []
    with open(jsonl_path, 'r', encoding='utf-8') as f:
        for line in f:
            try:
                data = json.loads(line)
                full_text = " ".join(str(data[field]) for field in fields if field in data)
                tokens = simple_preprocess(full_text, deacc=True)
                if tokens:
                    sentences.append(tokens)
            except json.JSONDecodeError:
                continue
    return sentences
✓ 0.0s
```

- Την αρχικοποίηση κάποιων πεδίων που χρειάζονται για την εκπαίδευση του μοντέλου αλλά επειδή χρησιμοποιούνται και αργότερα στον κώδικα θεωρήθηκε σκόπιμη η αρχικοποίηση τους για να είναι σίγουρη η ομοιογένεια των παραμέτρων κατά μήκος του κώδικα.

```
filepath = "read/scidocs/corpus.jsonl"
text_fields = ["title", "text", "authors", "year", "cited_by", "references"]
vector_dimensions = 50

sentences = sentence_generator(filepath, text_fields)
✓ 12.9s
```

Το πεδίο text fields περιλαμβάνει όλα τα πεδία ενός doc. Για να εξακριβωθεί αν είναι όλα απαραίτητα ή δημιουργούν θόρυβο έγιναν πειράματα με διάφορους συνδυασμούς, με τα αποτελέσματα να είναι ανάλογα του πλήθους των text fields. Ενδεικτικά παρακάτω παρατίθενται τα αποτελέσματα 2 πειραμάτων στα οποία ακολουθήθηκε ακριβώς η ίδια διαδικασία με ακριβώς τις ίδιες παραμέτρους, με μόνη αλλαγή ότι στο πρώτο πείραμα (δεξιά) χρησιμοποιήθηκαν όλα τα πεδία, ενώ στο άλλο χρησιμοποιήθηκαν μόνο τα πεδία title και text (αριστερά), τα οποία είναι και τα δύο πεδία με λέξεις και άρα τα πιο σχετικά για αναζήτηση συνωνύμων.

titleTextExpanded.txt

num_rel	all	4928
num_rel_ret	all	1351
map	all	0.0573
gm_map	all	0.0024
P_5	all	0.0444
P_10	all	0.0514
P_15	all	0.0589
P_20	all	0.0675

bestCase.txt

num_rel	all	4928
num_rel_ret	all	1354
map	all	0.0575
gm_map	all	0.0024
P_5	all	0.0450
P_10	all	0.0519
P_15	all	0.0592
P_20	all	0.0677

Όπως φαίνεται, η ύπαρξη παραπάνω πεδίων δεν επηρεάζει ούτε θετικά ούτε αρνητικά τα αποτελέσματα (ούτε τον χρόνο εκτέλεσης σε σημαντικό επίπεδο), οπότε έγινε η αυθαίρετη επιλογή να κρατηθούν όλα.

- Τη συνάρτηση εκπαίδευσης του μοντέλου, την εκπαίδευση του και την αποθήκευση του.

```python
def train_word2vec(sentences):
    model = Word2Vec(
        sentences=sentences,
        vector_size=vector_dimensions,
        window=5,
        min_count=2,
        workers=4,
        sg=0
    )
    return model
```
✓ 0.0s

```python
model = train_word2vec(sentences)
model.save("my_word2vec.model")
print("'my_word2vec.model' saved")
```
✓ 1m 9.5s

```
'my_word2vec.model' saved
```

Οι παράμετροι εκπαίδευσης που φαίνονται από πάνω είναι οι βέλτιστες όπως διαπιστώθηκε μετά από σχετικά πειράματα. Το αποτέλεσμα αυτό είναι λογικό, καθώς το corpus περιέχει ~25.000 κείμενα, αριθμός ο οποίος για δεδομένα εκπαίδευσης του word2vec είναι σχετικά μικρός, οπότε είναι λογικό το sg=0 (Αρχιτεκτονική CBOW, λειτουργεί καλύτερα με μικρότερα corpus κειμένων) και οι γενικά μικρότερες παράμετροι (embedding=50, παράθυρο συμφραζόμενων=5) να αποφέρουν τα σχετικά καλύτερα αποτελέσματα. Παρακάτω παρατίθενται ενδεικτικά τα αποτελέσματα που απέφερε το παραπάνω configuration (δεξιά) και τα αποτελέσματα του (vector_size = 200, window = 10, min_count = 5, sg = 1) configuration (αριστερά).

highTrainingParams.txt

num_rel	all	4928
num_rel_ret	all	95
map	all	0.0036
gm_map	all	0.0000

P_5	all	0.0040
P_10	all	0.0036
P_15	all	0.0040
P_20	all	0.0047

bestCase.txt

num_rel	all	4928
num_rel_ret	all	1354
map	all	0.0575
gm_map	all	0.0024

P_5	all	0.0450
P_10	all	0.0519
P_15	all	0.0592
P_20	all	0.0677

Έπειτα, δημιουργείται ένα νέο index, το mapping του οποίου είναι ίδιο με αυτό που υπάρχει στο Β μέρος της εργασίας με ένα πρόσθετο πεδίο όπως φαίνεται παρακάτω. Το ευρετήριο με τα καλύτερα αποτελέσματα που θα χρησιμοποιηθεί για τη σύγκριση με τις προηγούμενες φάσεις ονομάζεται w2v.

```
                "allContent_vector": {
                    "type": "dense_vector",
                    "dims": vector_dimensions
                }
            }
        }
    }

    # create an index with the configuration above
    client.indices.create(index='w2v', body=mapping)
```
✓ 3.2s

```
ObjectApiResponse({'acknowledged': True, 'shards_acknowledged': True, 'index': 'w2v'})
```

Για το indexing χρησιμοποιείται μια βοηθητική συνάρτηση η οποία για κάθε κείμενα δημιουργεί το διάνυσμα που μπαίνει στο πεδίο allContent_vector δημιουργώντας ένα string ανά κείμενο, το οποίο μετά σπάει σε tokens και για κάθε token δημιουργεί το αντίστοιχο διάνυσμα.

```python
def create_vector(doc):
    full_text = " ".join(str(doc[field]) for field in text_fields if field in doc)
    tokens = simple_preprocess(full_text, deacc=True)
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    return np.mean(vectors, axis=0).tolist() if vectors else [0.001] * vector_dimensions   #if it's 0.000 cosine similarity crashes
```

✓ 0.0s

Το indexing ακολουθεί το ίδιο μοτίβο με τη φάση 2, μόνο που εδώ γεμίζουν όλα τα fields μαζί και στη συνέχεια μέσω της βοηθητικής συνάρτησης γεμίζει και το πεδίο allContent_vector

```python
documents = []
for doc in formatted_data:
    docID = doc.get('uniqueID')
    doc.pop('uniqueID')
    documents.append({
        "_index": "w2v",
        "_id": docID,
        "_source": {
            **{k: v for k, v in doc.items() if k != 'uniqueID'},
            "allContent_vector": create_vector(doc)  # vector
        }
    })

# Indexing με helpers.bulk
bulk(client, documents, refresh=True)

count = client.count(index='w2v')["count"]

print(f"Index '{'w2v'}' contains {count} documents.")
```

✓ 5m 6.2s

```
Index 'w2v' contains 25657 documents.
```

Τέλος, η τελευταία τροποποίηση που έγινε στον κώδικα σε σχέση με τον κώδικα της φάσης 2 είναι το χτίσιμο των queries, όπου πρώτα δημιουργείται το query όπως στη φάση 2, έπειτα τα πεδία του query μετατρέπονται σε ένα ενιαίο string το οποίο μετατρέπεται σε διάνυσμα (και κανονικοποιείται μέσω βοηθητικής συνάρτησης ώστε το cosineSimilarity να μην επηρεάζεται από το μήκος του διανύσματος) και τέλος πραγματοποιείται η αναζήτηση χρησιμοποιώντας το cosineSimilarity για να συγκριθεί το διάνυσμα του query με το allContent_vector κάθε κειμένου. Το τελικό σκορ προκύπτει προσθέτοντας το σκορ της αναζήτησης στο allContent πεδίο και στο allContent_vector πεδίο προσθέτοντας 1 για την αποφυγή αρνητικών σκορ. Τέλος, γράφονται στο αρχείο results όπως και στη φάση 2.

Βοηθητική συνάρτηση κανονικοποίησης

```python
def normalize(vec):
    norm = np.linalg.norm(vec)
    return (vec / norm).tolist() if norm > 0 else vec
```

✓ 0.0s

Αναζήτηση

```
with open("write/results.test", "w") as file :
    for query in formatted_queries :
        base_query = {
            "bool": {
                "should": [
                    {"match": {"allContent": query.get("text", "")}},
                    {"match": {"allContent": safe_query(query.get("authors", ""))}},
                    {"match": {"allContent": query.get("year", "")}},
                    {"match": {"allContent": safe_query(query.get("cited_by", ""))}},
                    {"match": {"allContent": safe_query(query.get("references", ""))}},
                ]
            }
        }

        combined_text = " ".join([
            str(query.get(field, ""))
            for field in ["title", "text", "authors", "year", "cited_by", "references"]
        ])

        query_vector = normalize(create_vector({"text": combined_text}))
        search_results = client.search(
            index="w2v",
            size=20,
            query={
                "script_score": {
                    "query": base_query,
                    "script": {
                        "source": "_score + (cosineSimilarity(params.query_vector, 'allContent_vector') + 1.0)",
                        "params": {"query_vector": query_vector}
                    }
                }
            }
        )

        c = 1
        for hit in search_results["hits"]["hits"] :
            doc_id = hit["_id"]
            score = hit.get("score", None)
            run_name = "STANDARD"
            rank = c
            c+=1

            line = f"{query["uniqueID"]} Q0 {doc_id} {rank} {score} {run_name}"
            file.write(line + "\n")
✓ 1m 58.8s
```

Τα αποτελέσματα αυτού του κώδικα είναι τα αποτελέσματα του bestCase.txt με το οποίο έγιναν οι συγκρίσεις και στα πειράματα των προηγούμενων βημάτων.

```
num_rel                 all     4928
num_rel_ret             all     1354
map                     all     0.0575
gm_map                  all     0.0024
```

```
P_5                     all     0.0450
P_10                    all     0.0519
P_15                    all     0.0592
P_20                    all     0.0677
```

Το αποτέλεσμα είναι σημαντικά χειρότερο από τα αποτελέσματα της φάσης 2, επομένως έγιναν πειράματα μετρίασης της σημασίας του word2vec. Ο κώδικας τροποποιήθηκε ώστε να υπάρχουν βάρη με τον εξής τρόπο:

Η γραμμή: "source": "_score + (cosineSimilarity(params.query_vector, 'allContent_vector') + 1.0)",

Έγινε: "source": "_score*0.09 + (cosineSimilarity(params.query_vector, 'allContent_vector')*0.01 + 1.0)",

Τοποθετώντας έτσι πολύ μικρή σημασία στην αναζήτηση με word2vec

Παρακάτω φαίνονται τα αποτελέσματα της αναζήτησης με τα βάρη (αριστερά) και χωρίς (δεξιά):

withWeights.txt

```
num_rel                 all     4928
num_rel_ret             all     1351
map                     all     0.0572
gm_map                  all     0.0024

P_5                     all     0.0444
P_10                    all     0.0514
P_15                    all     0.0589
P_20                    all     0.0675
```

bestCase.txt

```
num_rel                 all     4928
num_rel_ret             all     1354
map                     all     0.0575
gm_map                  all     0.0024

P_5                     all     0.0450
P_10                    all     0.0519
P_15                    all     0.0592
P_20                    all     0.0677
```

Τα βάρη όπως φαίνεται επηρεάζουν ελάχιστα το αποτέλεσμα, αλλά αφού αυτή η επιρροή παραμένει αρνητική, το bestCase παραμένει ως η βέλτιστη λύση.

Τέλος, όπως και στις προηγούμενες φάσεις της εργασίας έγινε και ένα πείραμα, στο οποίο γίνεται το εξής preprocessing στα δεδομένα:

```python
def preprocess_text(text):

    text = text.lower()
    text = re.sub(r'[/()@|:?!.\-,+\d]', ' ', text)

    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    text = " ".join(filtered_words)

    return re.sub(r'\s+', ' ', text).strip()
```

✓ 0.0s

Το οποίο επέφερε τα εξής αποτελέσματα:

preprocessing.txt

```
num_rel                 all     4928
num_rel_ret             all     1275
map                     all     0.0539
gm_map                  all     0.0019

P_5                     all     0.0420
P_10                    all     0.0493
P_15                    all     0.0557
P_20                    all     0.0637
```

bestCase.txt

```
num_rel                 all     4928
num_rel_ret             all     1354
map                     all     0.0575
gm_map                  all     0.0024

P_5                     all     0.0450
P_10                    all     0.0519
P_15                    all     0.0592
P_20                    all     0.0677
```

Όπως φαίνεται, το preprocessing των δεδομένων (όπως και στις φάσεις 1 και 2) μειώνει ελαφρώς όλα τα αποτελέσματα οπότε τα αποτελέσματα που θα χρησιμοποιηθούν για τις συγκρίσεις με τις φάσεις 1 και 2 θα είναι αυτά του bestCase.txt, δηλαδή η απλή υβριδική αναζήτηση.

Αριθμός σχετικών ανακτηθέντων κειμένων από τα 4928 που έπρεπε (num_rel_ret)

	k=20	k=30	k=50
indexino	2309	2586	2851
nouns	2534	2771	3001
nv	2569	2801	3027
w2v	1354	1485	1653

Mean Average Precision (MAP)

	k=20	k=30	k=50
indexino	0.1616	0.1634	0.1647
nouns	0.1700	0.1689	0.1677
nv	0.1721	0.1699	0.1681
w2v	0.0575	0.0455	0.0330

Geometric Mean Average Precision (gmMAP)

	k=20	k=30	k=50
indexino	0.0176	0.0258	0.0356
nouns	0.0345	0.0441	0.0504
nv	0.0377	0.0464	0.0527
w2v	0.0024	0.0025	0.0025

Precision στα 5 πρώτα ανακτηθέντα κείμενα (P5)

	k=20	k=30	k=50
indexino	0.1362	0.1340	0.1344
nouns	0.1422	0.1364	0.1360
nv	0.1436	0.1376	0.1360
w2v	0.0450	0.0304	0.0196

Precision στα 10 πρώτα ανακτηθέντα κείμενα (P10)

	k=20	k=30	k=50
indexino	0.1330	0.1279	0.1265
nouns	0.1406	0.1316	0.1276
nv	0.1423	0.1319	0.1278
w2v	0.0519	0.0332	0.0192

Precision στα 15 πρώτα ανακτηθέντα κείμενα (P15)

	k=20	k=30	k=50
indexino	0.1213	0.1138	0.1115
nouns	0.1309	0.1181	0.1129
nv	0.1323	0.1189	0.1134
w2v	0.0592	0.0365	0.0227

Precision στα 20 πρώτα ανακτηθέντα κείμενα (P20)

	k=20	k=30	k=50
indexino	0.1155	0.1010	0.0970
nouns	0.1267	0.1060	0.0985
nv	0.1285	0.1063	0.0988
w2v	0.0677	0.0403	0.0227

Συμπεράσματα

Το βασικότερο συμπέρασμα που μπορεί να εξαχθεί από τα παραπάνω δεδομένα είναι ότι με τη προσθήκη του word2vec σε όλες τις μετρικές τα αποτελέσματα είναι εμφανώς χειρότερα. Τα χειρότερα αυτά αποτελέσματα ήταν αναμενόμενα γιατί το μέγεθος του corpus (~25.000 documents) είναι πολύ μικρό για την αποτελεσματική εκπαίδευση ενός μοντέλου που χρησιμοποιεί word2vec, επομένως το μοντέλο είναι ανεπαρκώς εκπαιδευμένο και δεν μπορεί να δώσει σωστά αποτελέσματα, ρίχνοντας έτσι το Precision. Ακόμα, το corpus περιλαμβάνει κείμενα επιστημονικού περιεχομένου, τα οποία πιθανότατα περιέχουν ειδικούς όρους και ορολογίες, δηλαδή λέξεις τις οποίες το μοντέλο δεν είναι προετοιμασμένο να διαχειριστεί ή για τις οποίες η συνήθης εννοιολογική επεξεργασία δεν ισχύει, με αποτέλεσμα το μοντέλο να «πέφτει έξω» ακόμα περισσότερο. Αυτό το γεγονός, σε συνδυασμό με το πολύ μικρό μέγεθος του Training data δείχνει ότι είναι απολύτως λογικό τα αποτελέσματα της τρίτης φάσης να είναι χειρότερα από αυτά της δεύτερης και της πρώτης.

Σε ότι αφορά τα αποτελέσματα ως συνάρτηση του αριθμού των κειμένων που επιστρέφει η αναζήτηση, επικρατούν τα ίδια μοτίβα που επικρατούσαν και στις προηγούμενες φάσεις (το num_rel_ret αυξάνεται όσο αυξάνεται το k ενώ το map και τα p5, p10, p15, p20 μειώνονται), απλά σε πολύ μικρότερους αριθμούς.

Επομένως, το βασικό συμπέρασμα παραμένει το γεγονός ότι η προσθήκη του word2vec σε ένα μικρό και γεμάτο ειδικούς όρους και ορολογίες σώμα κειμένων δεν βελτιώνει την αναζήτηση, αντίθετα τη παρεμποδίζει σημαντικά.