Συστήματα Ανάκτησης – Φάση 3

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

(**Όνομα:** Αναστάσιος Παπαπαναγιώτου, **ΑΜ:** 3200143, **Email:** p3200143@aueb.gr)

(**Όνομα:** Φοίβος Παπαθανασίου, **ΑΜ:** 3200138, **Email:** p3200138@aueb.gr)

Εισαγωγή

Αρχικά, έγινε προσπάθεια υλοποίησης της Φάσης 3 στη Java, παρόλαυτα η δοθείσα κλάση WordEmbeddingsSimilarity δεν ήταν υλοποιημένη κατάλληλα ώστε να μπορέσει να λειτουργήσει σε συνδυασμό με την IndexSearcher και κάποιον τύπο Query, πιθανώς να χρειαζόταν κάποια επέκταση της κλάσης Query που απαιτεί low-level γνώσεις του πώς λειτουργεί το scoring, ή επεξεργασία του source code κατευθείαν.

Ως λύση, δε χρησιμοποιήσαμε IndexSearcher και απλώς κάνωντας traverse το index, υπολογίσαμε για κάθε (query, document)-pair το score (αφού πρώτα είχε γίνει αναπαράσταση των queries και docs ως vectors απ'τα word embeddings).

Παρόλαυτα ακόμα και έτσι το MAP ήταν πάρα πολύ χαμηλό, δοκιμάζωντας διαφορετικές μεθόδους smoothing, μεθόδους αναπαράστασης των queries και docs ως vectors, το μέγιστο που πετύχαμε είναι 9%.

Τελικά, μετά και από αρκετούς ελέγχους και reverse engineering του source code, μάλλον ο τρόπος με τον οποίο γίνεται το tokenization ευθύνεται.

tldr: Αποφασίσαμε να υλοποιήσουμε τη Φάση 3 στην Python.

Οδηγός εκτέλεσης:

Με **venv** – Επιλέγουμε Python 3.8, εκτελούμε την εντολή *pip install -r requirements.txt* για να κατεβάσουμε τα απαραίτητα dependencies.

Με **conda** – Επιλέγουμε Python 3.8, αντικαθιστούμε το υπάρχον αρχείο requirements.txt με το αρχείο requirements.txt από το φάκελο conda requirements, εκτελούμε την εντολή *conda install --yes --file requirements.txt*

Μπορείτε να κατεβάσετε τα βάρη του μοντέλου που εκπαιδεύσαμε από το συγκεκριμένο link. Αποθηκεύουμε το μοντέλο (pretrained απ'το Wikipedia ή το δικό μας) στο φάκελο resources\models και ακολουθούμε τις οδηγίες στα σχόλια του αρχείου **main.py.**

Βήμα 1

Η εκπαίδευση του μοντέλου γίνεται με τη βιβλιοθήκη genism.

Η κλάση WordEmbeddingsModel του αρχείου **embeddings_model.py** έχει ώς instance variables το αρχείο από το οποίο θα κάνει train το μοντέλο (doc_data), το μέγεθος διαστάσεων των word embeddings (size), το μοντέλο (model_weights) και μία boolean μεταβλητή που προσδιορίζει πια αρχιτεκτονική θα χρησιμοποιηθεί για την παραγωγή των embeddings (sg = 0 -> CBOW, sg = 1 -> SkipGram).

Η μέθοδος **train()** λαμβάνει το μέγεθος παραθύρου και το πλήθος εποχών και κάνει train το μοντέλο, επίσης το αποθηκεύει. Tokens που εμφανίζονται λιγότερο από 5 φορές δεν λαμβάνονται υπόψιν κατά την εκπαίδευση του μοντέλου (παρουσιάστηκε μεγάλη βελτίωση καθώς λέξεις με πολύ μικρό αριθμό εμφανίσεων χαλάνε το context).

Εφόσον κάποιο μοντέλο έχει γίνει store, τότε με τη μέθοδο **load_model()** μπορεί να ανακτηθεί.

Μπορούν επίσης να γίνουν store τα βάρη του μοντέλου με τη μέθοδο **store_weights()** αλλά και να γίνουν load με τη μέθοδο **load_weights()**.

Η μέθοδος load_weights() θα μας φανεί χρήσιμη για το 2ο μέρος της Φάσης 3.

Βήμα 2

Η μέθοδος **get_df()** του αρχείου **preprocessing.py** έχει δύο χρήσεις:

1. Διαβάζει το αρχείο documents.txt/queries.txt και επιστρέφει ένα pandas DF που έχει όλα τα (docID, tokens)-pairs/(queryId, tokens)-pairs. Ύστερα κάνει store το DF με χρήση της βιβλιοθήκης pickle.

2. Εάν είναι ήδη αποθηκευμένο το σχετικό DF μπορεί να το κάνει read.

Στην περίπτωση χρήσης 1 (read=False, output_path μη κενό) η get_df() καλεί την **get_processed_data()** η οποία κάνει parse τις γραμμές, φτιάχνει τα (id, doc/query body)-pairs και τέλος με χρήση της **tokenize_seq()** παράγει και επιστρέφει το DF που περιέχει τα (id, doc/query tokens)-pairs.

Η **tokenize_seq()** κάνει tokenize σώμα κειμένου, αφαιρώντας stopwords και χρησιμοποιώντας lemmatization.

Βήμα 3

Χρησιμοποιούμε την κλάση Index του αρχείου index.py, έχει ως instance variables το DF των documents που περιέχει τα (docId, tokens)-pairs (doc_data) καθώς και τα βάρη ενός μοντέλου (model_weights).

Η **get_top_docs_for_queries()** επιστρέφει για ένα df που περιέχει (queryId, tokens)-pairs τα k top documents του index με βάση το score. Αρχικά – εφόσον δεν υπάρχουν – δημιουργεί τα vector representations των documents καθώς και τον queries με χρήση της μεθόδου **fit(),** στη συνέχεια καλεί για κάθε query την **get_top_results_for_query()** η οποία επιστρέφει DF με στήλες που αντιστοιχούν στα (q_id, iter, docno, rank, sim, run_id) που απαιτεί το trec_eval. Τέλος κάνει concatenate σε ένα DF όλα τα tables ώστε να πάρουμε τον τελικό πίνακα που περιέχει τα άνω στοιχεία για όλα τα query_ids.

Η μέθοδος **fit()**: Παράγει από τα tokens κάθε document/query το αντίστοιχο vector representation, παίρνωντας το mean των word embedding αναπαραστάσεων τους.

Η μέθοδος **get_top_results_for_query()**: Υπολογίζει για κάθε document vector το cosine similarity του με το query vector. Στη συνέχεια επιστρέφει DF με στήλες (score, docID) που περιέχει τα k μεγαλύτερα score. Τέλος διαμορφώνει το DF έτσι ώστε να έχει τις απαραίτητες πληροφορίες για το trec_eval (q_id, iter, docno, rank, sim, run_id).

Αποτελέσματα (Βήματα 4, 5)

Τα καλύτερα αποτελέσματα λάβαμε εκπαιδεύοντας ένα μοντέλο με τις εξής παραμέτρους: **διαστάσεις διανυσμάτων**: 300, **μήκος παραθύρου**: 5, **αρχιτεκτονική**: SkipGram, **εποχές:** 25

Παραθέτουμε τα αποτελέσματα παρακάτω:

Q	MAP	Precision@5	Precision@10	Precision@20
Q01	0.7047	0.8	0.9	0.6
Q02	0.1203	0.2	0.2	0.2
Q03	0.3299	0.6	0.5	0.3
Q04	0.0968	0.2	0.2	0.15
Q05	0.431	0.8	0.7	0.45
Q06	0.1473	0.6	0.3	0.2
Q07	0.231	0.4	0.3	0.35
Q08	0.678	1	0.7	0.5
Q09	0.2021	0.6	0.4	0.3
Q10	0.55	0.8	0.6	0.3
all	0.3491	0.6	0.48	0.335

Παρατηρούμε ότι καθώς αυξάνουμε το recall, το precision μειώνεται, το άνω μοντέλο θα απέδιδε καλά σε μία μηχανή αναζήτησης όπου οι χρήστες ενδιαφέροντα για λίγα και κατά το δυνατόν πιο ικανοποιητικά στις πληροφοριακές ανάγκες τους αποτελέσματα.

Q	MAP	Precision@5	Precision@10	Precision@20
Q01	0.7602	0.8	0.9	0.65
Q02	0.2872	0.6	0.4	0.2
Q03	0.6106	1	0.6	0.4
Q04	0.0534	0	0.1	0.15
Q05	0.2946	0.4	0.2	0.35
Q06	0.0425	0	0.1	0.15
Q07	0.2712	0.2	0.2	0.4
Q08	0.8822	1	1	0.6
Q09	0.137	0.4	0.3	0.2
Q10	0.3183	0.6	0.4	0.2
all	0.3657	0.5	0.42	0.33

Πίνακας αποτελεσμάτων Φάσης 1 (ClassicSimilarity – VSM μοντέλο)

Συγκριτικά με την 1η φάση παρατηρούμε ότι το VSM μοντέλο έχει καλύτερο precision για μεγαλύτερες τιμές recall (καθώς το MAP είναι μεγαλύτερο ενώ τα Precision @5, 10, 20 είναι μικρότερα του προηγούμενου μοντέλου). Έχει όμως μικρότερο precision @5, 10

Πίνακας καλύτερων αποτελεσμάτων Φάσης 2 (BM25 b=0, k1=3)

Q	MAP	Precision@5	Precision@10	Precision@20
Q01	0.7779	0.8	0.8	0.7
Q02	0.3017	0.6	0.3	0.25
Q03	0.6184	0.8	0.6	0.4
Q04	0.0428	0	0.1	0.15
Q05	0.2628	0.4	0.3	0.35
Q06	0.11	0.2	0.3	0.2
Q07	0.247	0.2	0.2	0.3
Q08	0.8915	1	1	0.6
Q09	0.2025	0.6	0.4	0.25
Q10	0.2861	0.6	0.4	0.2
all	0.3741	0.52	0.44	0.34

Πίνακας καλύτερων αποτελεσμάτων Φάσης 2 (BM25 b=0, k1=3)

Πάλι παρατηρούμε ότι ενώ το precision για μεγάλο recall φαίνεται λίγο καλύτερο, το WV μοντέλο έχει αρκετά καλύτερο precision @5, 10.

Ο λόγος καλύτερης απόδοσης του WV ευθύνεται στο ότι αποτυπώνεται η σημασιολογία των λέξεων με βάση το context τους.

Βήμα 6

Κατεβάζουμε το μοντέλο και αποθηκεύουμε την .bin έκδοση του στον φάκελο models, στη συνέχεια κάνουμε instantiate ένα αντικείμενο κλάσης WordEmbeddingsModel και χρησιμοποιώντας τη μέθοδο **load_weights()** φορτώνουμε το προεκπαιδευμένο μοντέλο.

Βήμα 7

Q	MAP	Precision@5	Precision@10	Precision@20
Q01	0.4721	0.8	0.7	0.4
Q02	0.1692	0.4	0.2	0.15
Q03	0.1558	0.4	0.2	0.2
Q04	0.1457	0.2	0.2	0.25
Q05	0.2092	0.6	0.5	0.3
Q06	0.0239	0	0	0.1
Q07	0.1618	0.2	0.3	0.3
Q08	0.2051	0.6	0.3	0.15
Q09	0.0676	0.2	0.2	0.1
Q10	0.3679	0.4	0.2	0.3
all	0.1978	0.38	0.28	0.225

Πίνακας αποτελεσμάτων χρησιμοποιώντας το pretrained μοντέλο σε Wikipedia dump

Παρατηρούμε ότι τα αποτελέσματα δεν είναι ικανοποιητικά συγκριτικά με τα άλλα συστήματα. Ο λόγος είναι προφανής, δεν υπάρχουν τα word-embeddings για ορισμένες λέξεις όπως “Mobility-as-a-Service” (query 6, γιαυτό έχει τόσο χαμηλό MAP), “cross-domain” (query 9, γιαυτό έχει τόσο χαμηλό MAP).

Επίσης οι λέξεις μπορεί να βρίσκονταν σε διαφορετικό context συγκριτικά με τις λέξεις των δικών μας κειμένων για τα οποία τα 10 queries που μας δίνονται έχουν λέξεις στο ίδιο «context» με των κειμένων.

Συμπέρασμα

Το WV είναι ένα αποτελεσματικό μοντέλο το οποίο μάλιστα αντιθέτως με τα υπόλοιπα μοντέλα που δοκιμάσαμε, μπορεί να βελτιωθεί περαιτέρω με την επανεκπαίδευση του μοντέλου σε μεγαλύτερη ποσότητα δεδομένων. Με βάση τα αποτελέσματα μας μόνο, θα μπορούσε να πει κανείς ότι είναι προτιμότερο για search engines και γενικά περιπτώσεις όπου ο χρήστης αναζητά λίγα κείμενα κατά το δυνατόν πιο σχετικά με τις πληροφοριακές ανάγκες του.

Πηγές

eclass