Εργασία ανάκτησης

Ανεστραμμένα αρχεία

Κάποια θεωρητικά

Τα ανεστραμμένα αρχεία (inverted files) είναι ένας μηχανισμός ο οποίος χρησιμοποιείται για να διευκολύνει την αναζήτηση και την εύρεση σχετικών κειμένων από μία συλλογή. Πιο συγκεκριμένα, κάθε κείμενο διασπάται σε λέξεις οι οποίες αποθηκεύονται σε μια δομή ανεστραμμένων αρχείων. Αυτή η δομή συσχετίζει την κάθε λέξη που υπάρχει στην συλλογή, με τα αντίστοιχα κείμενα που την περιέχουν και έτσι δημιουργείται ένα είδος ανεστραμμένης αναφοράς.

Η απλούστερη τέτοια δομή είναι ο πίνακας Term-Document, όπου κάθε γραμμή αντιστοιχεί σε μια λέξη του λεξικού, στο οποίο περιέχονται όλες οι διαφορετικές λέξεις των κειμένων και κάθε στήλη αντιστοιχεί σε ένα κείμενο. Κάθε τιμή μας δείχνει πόσες φορές η λέξη υπάρχει στο συγκεκριμένο κείμενο. Αυτή η δομή απαιτεί πολύ μεγάλο χώρο αφού πολλές φορές οι λέξεις περιέχονται σε μικρό υποσύνολο των κειμένων. Για αυτό προτιμάται η αντιστοίχιση μιας λίστας των κειμένων σε κάθε λέξη (απλό ευρετήριο).

Στην παραπάνω περίπτωση χάνεται η έννοια των φράσεων, καθώς κοιτάμε κάθε λέξη ξεχωριστά χωρίς να λαμβάνουμε υπόψη μας ότι η σειρά των λέξεων μπορεί να παίζει ρόλο. Αυτό το πρόβλημα λύνει το πλήρες ανεστραμμένο αρχείο το οποίο μαζί με την αντιστοίχιση λέξεων και κειμένων αποθηκεύει και την θέση των λέξεων μέσα στο κείμενο.

Στην εργασία μας έχουμε υλοποιήσει απλά ευρετήρια διότι αργότερα όταν τα χρησιμοποιούμε στο Vector space μοντέλο δεν λαμβάνουμε υπόψη μας την σειρά των λέξεων στο κάθε κείμενο, πράγμα που μπορεί να έχει ως αποτέλεσμα σε κάποιο βαθμό τα αποτελέσματα μας να μην είναι τόσο ακριβή (και σε σχέση με το colBert).

Πρακτική υλοποίηση (Κώδικας)

Η υλοποίηση του απλού ανεστραμμένου αρχείου γίνεται μέσω της παρακάτω συνάρτησης:

def build\_inverted\_index(documents):

inverted\_index = {}

for \_, (doc\_title, document) in enumerate(documents): # Unpack the tuple

for term in set(document):

if term not in inverted\_index:

inverted\_index[term] = defaultdict(int)

inverted\_index[term][doc\_title] = document.count(term)

return inverted\_index

Η συνάρτηση αυτή παίρνει ως όρισμα την συλλογή κειμένων που έχουμε. Αρχικοποιούμε ένα κενό λεξικό inverted\_index που θα χρησιμοποιηθεί για το ανεστραμμένο ευρετήριο. Έπειτα διασχίζουμε κάθε έγγραφο στην συλλογή και η enumerate παρέχει τον τίτλο και το ίδιο το κείμενο. Για το κάθε έγγραφο διασχίζουμε τον κάθε όρο του εγγράφου. Με το set(document) αφαιρούμε τυχόν διπλότυπους όρους στο τρέχων κείμενο. Πραγματοποιούμε έλεγχο για την ύπαρξη του όρου στο inverted\_index και αν ο συγκεκριμένος όρος δεν υπάρχει ήδη, δημιουργείται μια νέα εγγραφή στο inverted\_index με τιμή ένα κενό difaultdict(int). Μετά ενημερώνεται η συχνότητα εμφάνισης του τρέχοντος όρου στο τρέχον κείμενο. Αυτή η διαδικασία συνεχίζεται για όλα τα κείμενα. Τέλος, επιστρέφεται το inverted\_index.

Vector space model

Κάποια θεωρητικά

Το vector space model (VSM) χρησιμοποιείται στην ανάκτηση πληροφορίας και στην κατάταξη σχετικών κειμένων. Είναι ένας τρόπος αναπαράστασης των κειμένων και των ερωτημάτων ως διανύσματα στον χώρο. Τα διανύσματα αυτά αποτελούνται από τα βάρη του κάθε όρου του ερωτήματος. Το κάθε βάρος υπολογίζεται ως το γινόμενο της συχνότητας εμφάνισης του όρου στο κείμενο (tf) με τον αριθμό εμφάνισης τού όρου στα κείμενα της συλλογής (idf). Τα κείμενα που είναι πιο σχετικά με το ερώτημα αποτελούν γειτονικά διανύσματα και μπορούν να υπολογιστούν μέσω μετρικών ομοιότητας όπως είναι το cosine similarity. Ένα από τα πιο κρίσιμα σημεία του VSM είναι ο ορισμός του tf και idf για τον υπολογισμό του βάρους κάθε όρου τόσο στο ερώτημα όσο και στα κείμενα.

Πρακτική υλοποίηση (Κώδικας)

def vector\_space(query,docs, query\_weight\_func, doc\_weight\_func):

tf={}

for term in set(query):#Get query term frequency

tf[term] = query.count(term)

#Calculate query and doc weights

query\_weights = []

document\_weights = {}

for doc in docs:

document\_weights[doc[0]] = []

for term in set(query):

# if len(inverted\_index[term]):

if term in inverted\_index and len(inverted\_index[term]):

idf = math.log(len(docs)/len(inverted\_index[term]))

query\_weights.append(query\_weight\_func(tf,idf,term))

for doc in docs:

try:

tf\_doc = inverted\_index[term][doc[0]]

except:

tf\_doc = 0

w = doc\_weight\_func(tf\_doc,idf,term)

document\_weights[doc[0]].append(w)

#Calcualte cosine similarity for each doc

sim = {}

for doc in document\_weights:

sim[doc] = cosine\_similarity([query\_weights],[document\_weights[doc]])

return sorted(sim.items(), key=lambda x:x[1])[-500:][::-1]

Η συνάρτηση παίρνει ως ορίσματα το ερώτημα (query) , την συλλογή των κειμένων (docs), μια άλλη συνάρτηση για τον υπολογισμό του βάρους κάθε όρου του ερωτήματος για το ερώτημα (query\_weight\_func) καθώς και μια συνάρτηση αντίστοιχα για τον υπολογισμό βάρους κάθε όρου του ερωτήματος, για ένα κείμενο. Αρχικοποιούμε το tf ως έναν κενό πίνακα. Έπειτα υπάρχει ένας βρόγχος ο οποίος υπολογίζει την συχνότητα εμφάνισης του κάθε όρου στο ερώτημα και οι τιμές αυτές εισάγονται στον πίνακα tf.

Έπειτα αρχικοποιούμε τον πίνακα query\_weights και το λεξικό document\_weights κενά. Ύστερα, διατρέχουμε όλα τα κείμενα και για κάθε κείμενο δημιουργούμε κενή λίστα στην οποία θα εισαχθούν τα βάρη του κάθε κειμένου. Στη συνέχεια διατρέχουμε όλους τους μοναδικούς όρους του query και εξετάζουμε αν ο κάθε όρος υπάρχει στο ανεστραμμένο αρχείο που έχουμε φτιάξει. Στην περίπτωση που ο όρος υπάρχει στο inverted\_index υπολογίζουμε το idf του όρου και ακολούθως υπολογίζουμε το βάρος του όρου για το query καλώντας την αντίστοιχη συνάρτηση. Μετά διατρέχουμε για τον συγκεκριμένο όρο όλα τα κείμενα της συλλογής. Για κάθε κείμενο βρίσκουμε πόσες φορές εμφανίζεται ο όρος στο κείμενο αυτό. Μετά καλούμε την συνάρτηση υπολογισμού βάρους του όρου για τo κείμενο. Έτσι υπολογίζουμε το βάρος του όρου και το προσθέτουμε στο αντίστοιχο λεξικό (document\_weights[doc[0]]). Τέλος, με την συνάρτηση cosine\_similarity υπολογίζουμε την ομοιότητα ερωτήματος και κειμένου και επιστρέφουμε τα 500 κείμενα με την μεγαλύτερη ομοιότητα.

Μετρικές

Κάποια θεωρητικά

Εφόσον χρησιμοποιώντας το Vector Space model υπολογίζονται για κάθε ερώτημα τα έγγραφα απάντησης και εφόσον υπάρχουν τα πραγματικά σχετικά έγγραφα, μπορεί να γίνει αξιολόγηση των αποτελεσμάτων του μοντέλου. Για την αξιολόγηση λοιπόν των αποτελεσμάτων, χρησιμοποιούνται οι ποσοτικές μετρικές ακρίβεια και ανάκληση. Πιο συγκεκριμένα παρουσιάζεται διάγραμμα Ακρίβειας-Ανάκλησης και η μέση ακρίβεια των ερωτημάτων.

Πρακτική υλοποίηση (Κώδικας)

def recall\_precision\_curve(result\_docs, relevant):

RP = []

# result\_docs is an array with vector space most relevant docs for each query and relevant is an array with the true relevant

for i in range(len(result\_docs)):

# Initialize variables for precision, recall, and true positives

precision\_values = []

recall\_values = []

true\_positives = 0

# Iterate over retrieved documents

for j, doc in enumerate(result\_docs[i]):

if doc in relevant[i]:

true\_positives += 1

# Calculate precision and recall at this point

precision = true\_positives /(j+1)

recall = true\_positives / len(relevant[i])

precision\_values.append(precision)

recall\_values.append(recall)

RP.append((recall\_values, precision\_values))

return RP

Η συνάρτηση παίρνει ως όρισμα έναν πίνακα με τα κείμενα της απάντησης του vector space model και έναν πίνακα με τα σχετικά κείμενα. Για κάθε ένα ερώτημα διατρέχονται τα κείμενα απάντησης του vector space model. Για κάθε κείμενο απάντησης, γίνεται έλεγχος αν το συγκεκριμένο κείμενο υπάρχει στα σχετικά κείμενα του ίδιου ερωτήματος.

Αν υπάρχει αυξάνεται κατά μια μονάδα ένας μετρητής που χρησιμοποιείται για να δείχνει πόσα κείμενα απάντησης ανήκουν στα σχετικά κείμενα μέχρι στιγμής. Έπειτα με βάση την τιμή αυτού του μετρητή και του μετρητή που αφορά, στο ερώτημα αυτό τον αριθμό των κειμένων απάντησης που έχουν ελεγχθεί, υπολογίζονται η τιμή της ανάκλησης και της ακρίβειας και εισάγονται σε αντίστοιχους πίνακες.

Αφού ολοκληρωθεί αυτή η εργασία για ένα ερώτημα οι πίνακες που αφορούν το ερώτημα αυτό εισάγονται στην αντίστοιχη θέση ενός άλλου πίνακα που αφορά όλα τα ερωτήματα. Για το επόμενο ερώτημα, οι πίνακες που αφορούσαν το προηγούμενο ερώτημα ξανά αρχικοποιούνται. Στο τέλος επιστρέφεται ο πίνακας που αφορά όλα τα ερωτήματα.

def mean\_Average\_Values(rp):

ap\_values=[]

for i in range(len(rp)):

ap=0

recall\_values,precision\_values=rp[i]

for j in range(1,len(recall\_values)):

ap += (recall\_values[j] - recall\_values[j- 1]) \* precision\_values[j]

ap\_values.append(ap)

return np.mean(ap\_values)

Η συνάρτηση παίρνει ως όρισμα έναν πίνακα με τις τιμές του διαγράμματος ακρίβειας-ανάκλησης των ερωτημάτων. Για κάθε ερώτημα αποθηκεύονται σε αντίστοιχους πίνακες οι τιμές της ακρίβειας και της ανάκλησης και για κάθε ερώτημα υπολογίζεται το εμβαδό του διαγράμματος. Έπειτα η τιμή του εμβαδού αποθηκεύεται σε ένα πίνακα που αποθηκεύει τα εμβαδά των διαγραμμάτων όλων των ερωτημάτων. Τέλος επιστρέφεται ο μέσος όρος αυτών των εμβαδών.