Ανάπτυξη Λογισμικού
για Πληροφοριακά Συστήματα
ΠΑΡΑΔΟΤΕΟ 2

Άννα Αλεξάνδρου: 1115201700202
Ελένη-Πολυξένη Παπαδάκη: 1115201700116
Μαρία Μαυράκη: 1115201700081

Μεταγλώττιση και εκτέλεση Προγράμματος:

*Η μεταγλώττιση και εκτέλεση του κώδικα πρέπει να γίνεται στον ίδιο φάκελο με τον φάκελο camera_specs (και με χρήση του large/medium datasetW, επίσης στον ίδιο φάκελο).

*Ο φάκελος camera_specs ΔΕΝ περιέχεται.

Για μεταγλώττιση: επισυνάπτεται αρχείο Makefile

- make project2: για μεταγλώττιση του κώδικα
- make test_project2 : για μεταγλώττιση του κώδικα για το Software Testing

Για εκτέλεση:

- ./project2
- ./test_project2 ή ./test_project2 όνομα_συνάρτησης

Εκκαθάριση:

make clean: διαγράφει τα εκτελέσιμα, τα object files και τα αρχεία εξόδου

Υλοποίηση:

1) Γίνεται χρήση hashTable για την αποθήκευση και την εύρεση των specs.
2) Γίνεται χρήση hashTable για την αποθήκευση των λέξεων που υπάρχουν στην περιγραφή κάθε spec (local vocabulary).
3) Γίνεται χρήση hashTable για τη δημιουργία ενός vocabulary που περιέχει όλες τις λέξεις που εμφανίζονται στις περιγραφές των specs συνολικά (vocabulary).
4) Κάθε spec αποθηκεύεται σε μία δομή που σε σχέση με την 1η εργασία περιέχει επιπλέον τα εξής:
    a. 2 vectors (bow και tf-idf)
    b. αριθμός λέξεων που περιέχει το spec
    c. μέγεθος του πίνακα που κρατάει τις tf τιμές
    d. local vocabulary
5) Κάθε spec διαθέτει μία λίστα (local vocabulary) στην οποία αποθηκεύονται οι λέξεις που εμφανίζονται σε αυτό το spec (κάθε λέξη εμφανίζεται μόνο μία φορά). Επιπλέον, για κάθε μία από τις λέξεις , αποθηκεύεται στη λίστα ένας αριθμός που δείχνει πόσες φορές εμφανίζεται αυτή η λέξη στο συγκεκριμένο spec.
6) Δημιουργούμε pointers από την κάθε κλίκα θετικών συσχετίσεων πίσω στο bucket (υλοποίηση της 1-1 αντιστοιχίας μεταξύ των κορυφών του γράφου και των specs των προϊόντων).
7) Κάθε spec μέσα στο hashtable δείχνει σε μία δομή head.
   Κάθε head περιέχει έναν pointer σε κλίκα (η οποία αποτελείται από τα ids των specs που αναφέρονται στο ίδιο αντικείμενο με το spec που αντιστοιχεί στο συγκεκριμένο head), μία λίστα που περιέχει τις αρνητικές συσχετίσεις και έναν pointer σε επόμενο head.
8) Όταν λαμβάνουμε μία αρνητική συσχέτιση μεταξύ 2 specs, βρίσκουμε τα heads αυτών των 2 specs στο hashtable και στην λίστα των αρνητικών συσχετίσεων του καθενός προσθέτουμε pointer στο head του άλλου.
9) Για την υλοποίηση του προγράμματος χρησιμοποιούμε 2 λίστες. Η μία αποτελεί μία κλίκα (clique), δηλαδή μία λίστα από specids και η άλλη (AccessCliques) κρατάει τα heads για να έχουμε γρήγορη πρόσβαση σε αυτά. Κάθε στοιχείο της λίστας AccessCliques δείχνει σε μία κλίκα (δηλαδή στην αρχή μιας λίστας clique).

Παραδοχές:

1) Μέσα στα buckets του hashtable αποθηκεύουμε ολόκληρα τα specs ενώ μέσα στις κλίκες (δηλαδή τις λίστες clique) μόνο τα spec_ids και δείκτες πίσω στα specs.

2) Υποθέτουμε ότι για κάθε κατηγορία spec που διαβάζουμε οι χαρακτήρες είναι λιγότεροι από 51.000 και τα spec_ids περιέχουν λιγότερους από 50 χαρακτήρες.

3) Για κάθε λέξη ενός spec κρατάμε στο local vocabulary (εκτός από τη λέξη) και το πόσες φορές εμφανίζεται αυτή η λέξη στο spec.

4) Υποθέτουμε ότι για τη διαδικασία χρήσης του μοντέλου θα δοθεί ένα αρχείο της μορφής:

left_spec_id, right_spec_id

www.*//*, www.*//*

www.*//*, www.*//*

www.*//*, www.*//*

...

Αρχεία:

words.c:

Περιέχει την υλοποίηση των εξής βοηθητικών συναρτήσεων που καλούνται κατά την αποθήκευση των specs:

- `void lower (char* word)`
- `int isStopword (char* word, FILE* file)`

Η πρώτη συνάρτηση παίρνει ως όρισμα μία λέξη, μετατρέπει όλα τα γράμματά της σε πεζά και την επιστρέφει.

Η δεύτερη συνάρτηση, δοθέντος ενός αρχείου που περιέχει stopwords, ελέγχει αν η λέξη που της δίνεται ως πρώτο όρισμα υπάρχει μέσα σε αυτό το αρχείο.

bowvector.c:

Περιέχει την υλοποίηση των συναρτήσεων που σχετίζονται με το BagOfWords (BOW).

`int createVector (struct Spec* spec, int vsize):`

Δημιουργεί ένα διάνυσμα μεγέθους vsize και αρχικοποιεί όλα τα κελιά του σε 0.

`int setVector (struct Spec* spec):`

Δημιουργεί ένα bow διάνυσμα για το spec που έχει λάβει ως όρισμα ως εξής:

Πρώτα, καλεί την createVector() για να δημιουργήσει ένα μηδενικό διάνυσμα. Έπειτα, διαβάζει το tf-idf διάνυσμα του spec που λαμβάνει ως όρισμα και για κάθε κελί ελέγχει την τιμή του. Αν αυτή είναι >0, ενημερώνει το αντίστοιχο κελί στο bow διάνυσμα βάζοντας την τιμή 1.

logistic.c:

```
void feedModel (struct Entry* hashTable, char* argv, float *weight, float trainSize, char mode, char* y_true, float** y_fpred):
```

Η συνάρτηση καλείται με mode TRAIN ή TEST.

Σε κάθε περίπτωση (TRAIN/TEST), καλεί τις κατάλληλες συναρτήσεις και δίνει τα σωστά δεδομένα ώστε το μοντέλο να επιτελέσει την αντίστοιχη λειτουργία.

Ως πρώτο όρισμα δίνεται ένας δεκαδικός αριθμός που αντιπροσωπεύει το ποσοστό του αρχείου που θα χρησιμοποιηθεί για το training του μοντέλου. Στο mode περνιέται η σταθερά TRAIN ή TEST, ώστε να προσδιορίζει αν η συνάρτηση καλείται στη φάση του training ή στη φάση του testing, ώστε να εκτελέσει η συνάρτηση τον κατάλληλο κώδικα. Τα ορίσματα y_true και y_fpred είναι NULL για τη φάση του training. Στο testing χρησιμεύουν ώστε η συνάρτηση να επιστρέψει στο y_true τα πραγματικά labels του datasetW, ενώ στο y_fpred τις προβλέψεις που έκανε το μοντέλο.

Έχουμε αποθηκεύσει τις θετικές συσχετίσεις σε ένα αρχείο output1.csv και τις αρνητικές σε ένα αρχείο output0.csv.

Η feedModel, στο κομμάτι του training, διαβάζει εναλλάξ από τα παραπάνω αρχεία μέχρι να φτάσει σε κάποια τελική συνθήκη (είτε βάση των epochs είτε βάση τελικής τιμής των βαρών) που θεωρείται ότι έχει εκπαιδευτεί το μοντέλο.

```
char* threshold (float* y_fpred)
```

Δέχεται σαν όρισμα έναν πίνακα με τις προβλέψεις που έχει παράξει το μοντέλο. Για κάθε μία τιμή εφαρμόζει έναν έλεγχο αν αυτή είναι μεγαλύτερη ή μικρότερη από ένα καθορισμένο όριο

(εδώ 0.5) και ανάλογα με το αποτέλεσμα καταγράφει κατ' αντιστοιχία σε έναν νέο πίνακα την τιμή 1 ή 0.

`void evaluate (char* y_true, char* y_pred):`

Βάσει των πραγματικών και των προβλεπόμενων τιμών που λαμβάνει ως ορίσματα, υπολογίζει τα κατάλληλα στατιστικά στοιχεία για να μετρήσει την αποδοτικότητα του μοντέλου.

Συγκεκριμένα, μετράει τα: truePositives, falsePositives, trueNegatives, falseNegatives και υπολογίζει επιπλέον το f1-score.

`void print_scores (int tp, int fp, int tn, int fn, float f1):`

Εκτυπώνει τα στατιστικά στοιχεία που υπολογίζει η evaluate().

`void test(struct Entry* hashTable, char* argv, float *weight,float trainSize):`

Εκτελεί το test για το μοντέλο που έχει κατασκευαστεί και προπονηθεί στο πλαίσιο της παρούσας εργασίας.

`void logistic_regression(float trainSize,struct Entry* hashTable,char* argv):`

Καλείται από τη main για να υλοποιηθεί το logistic regression.

Στη συνάρτηση αυτή αρχικοποιείται η τιμή της μεταβλητής b και καλείται η συνάρτηση feedModel για train και test αντίστοιχα.

`void init_w(float *weight):`

Αρχικοποιεί τα βάρη

`float *update_w(float *weight,float *gradient):`

Η συνάρτηση αυτή ανανεώνει τα βάρη με βάση τον τύπο $w^{t+1} = w^t - \eta * \text{ανάδελτα}(J(w,b))$ και ελέγχει αν είναι η τελική ανανέωση των βαρών σύμφωνα με τον τύπο $w^{t+1} - w^t < \varepsilon$ ή με το πλήθος των epochs.

`float f(float *w, float *x, float b):`

Υπολογισμός του τύπου b+ w^t *x.

`float sigmoid(float f_x):`

Υπολογισμός του τύπου 1/(1+ $e^{-f(x)}$)

`float gradient(int j,float *weight,float *input, float b, int label):`

Υπολογισμός του prediction για συγκεκριμένο input με βάση την sigmoid(f) και τα βάρη εκείνης της στιγμής και στη συνέχεια υπολογισμός της τιμής $\partial / \partial wj$ J(w,b) = (σ(w^t * $x^{(i)}$ + b)- $y^{(i)}$) * $xj^{(i)}$

BOW.c:

`int hashFunction2(int HashtableNumOfEntries,char* hashName):`

Είναι η hashfunction του vocabulary hashtable και λαμβάνει ως όρισμα ένα hashName και τα entries του vocabulary hashtable.

Αν το hashName έχει length = 1, τότε ελέγχει αν αυτός ο χαρακτήρας είναι νούμερο ή γράμμα και το τοποθετεί στο entry που προκύπτει από:

[τον ascii κωδικό του αφαιρώντας την τιμή ‘0’ (αν είναι νούμερο) ή ‘a’ (αν είναι γράμμα)]%entries

Αν το hashName έχει length > 1, τότε ελέγχει τους 2 πρώτες χαρακτήρες του και το τοποθετεί στο entry που προκύπτει από την πράξη:

entry= [(first2Letters[0] + first2Letters[1])*first2Letters[0]] % entries

όπου first2Letters[0] ο ascii κωδικός του πρώτου χαρακτήρα και first2Letters[1] ο ascii κωδικός του δεύτερου χαρακτήρα.

`int addWordInVocabulary(char* word, int HashtableNumOfEntries, struct VocabEntry* HashTable, int bucketSize):`

Προσθέτει μία καινούρια λέξη στο vocabulary hashtable.

Ακολουθεί τη διαδικασία εισαγωγής που είχε υλοποιηθεί στο spec hashtable με τη διαφορά οτι θέσεις των buckets αποτελούνται μόνο από μία λέξη, έναν αριθμό αναγνωριστικό της λέξης (αύξοντα αριθμό) και έναν αριθμό που δείχνει σε πόσα specs εμφανίζεται η συγκεκριμένη λέξη.

`int addWordInLocalVocabulary(char* word, int HashtableNumOfEntries, struct VocabEntry* HashTable, int bucketSize):`

Ίδια λογική με την addWordInVocabulary

`int searchWordInVocabulary(struct VocabEntry* hashTable, int HashtableNumOfEntries, int bucketSize, char* word,int entryNum, int delete):`

Η συνάρτηση αυτή ψάχνει μία συγκεκριμένη λέξη μέσα στο vocabulary hashtable. Αν τη βρει επιστρέφει τον αύξοντα αριθμό της λέξης, διαφορετικά επιστρέφει -1. Χρησιμοποιείται για να μην βάζουμε 2η φορά κάποια λέξη στο λεξιλόγιο.

`int getWordPosition(struct VocabEntry* hashTable, int HashtableNumOfEntries, int bucketSize, char* word):`

Βρίσκει μία λέξη στο vocabulary hashtable και επιστρέφει τον αύξοντα αριθμό της.

Η συνάρτηση αυτή χρησιμοποιείται κατά των σχηματισμό των vectors των specs για bow και tf-idf αντίστοιχα, έτσι ώστε η θέση μιας λέξης στο vector να αντιστοιχεί με τον αύξοντα αριθμό της στο vocabulary hashtable.

HashTable.c:

`void searchSimilar(struct Entry* hashTable, int HashtableNumOfEntries, int bucketSize, char* id,char* dest_id):`

Κάθε φορά που λαμβάνουμε ένα ζευγάρι θετικών συσχετίσεων, η συνάρτηση αυτή ψάχνει στο hashtable τα heads των 2 specids που δόθηκαν.

Αν αυτά ταυτίζονται, τότε η συνάρτηση δεν κάνει τίποτα.

Αν δεν ταυτίζονται, τότε προσθέτει στο τέλος της κλίκας στην οποία δείχνει το destination head, την κλίκα στην οποία δείχνει το origin head και ταυτίζει το origin head με το destination head.

`void searchDifferent(struct Entry* hashTable, int HashtableNumOfEntries, int bucketSize, char* id,char* dest_id):`

Κάθε φορά που λαμβάνουμε ένα ζευγάρι αρνητικών συσχετίσεων, η συνάρτηση αυτή ψάχνει στο hashtable τα heads των 2 specids που δόθηκαν. Κοιτάει αν υπάρχει το head του ενός spec στη λίστα αρνητικής συσχέτισης του άλλου και αν δεν υπάρχει, τότε προσθέτει στη λίστα και των 2 specs την κεφαλή-head του άλλου.

`struct Spec* findSpecInHashTable(struct Entry* hashTable, int HashtableNumOfEntries, int bucketSize, char* id):`

Λαμβάνει ως όρισμα το specid, ψάχνει στο hashtable το συγκεκριμένο spec που έχει ζητηθεί και το επιστρέφει.

ReadX.c

Αλλαγές σε:

`void set_spec(struct Spec *spec, FILE *spec_file, char* dir_name, char* file_name,struct VocabEntry* vocabulary, FILE* stop_file,int HashtableNumOfEntries,int bucketSize)`

Καθώς η συνάρτηση set_spec διαβάζει τις πληροφορίες για τις κατηγορίες και τις λεπτομέρειες, ξεχωρίζει μια μια τις λέξεις και καλεί για αυτές την συνάρτηση procedure. Αφού διαβάσει όλες τις λέξεις από το αρχείο, υπολογίζει τις τιμές tf για κάθε μια από αυτές.

Αφαιρεί τους κωδικούς unicode

Αποθηκεύει αριθμούς

Αποθηκεύει ως μέρος της λέξης σύμβολα τα οποία προσδιορίζουν τη σημασία της λέξης, π.χ. 3"

Έχουν προστεθεί τα:

`int endofword(char current, bool flag)`

Η συνάρτηση αυτή λαμβάνει ένα χαρακτήρα και ένα flag και αποφασίζει αν αποτελεί διαχωριστικό ανάμεσα σε λέξεις. Επιστρέφει 1 αν αποφασίσει ότι έχουμε φτάσει στο τέλος μιας

λέξης και 0 αν ο χαρακτήρας αποτελεί μέρος της λέξης. Αν διαβάσει flag == true, τότε ο χαρακτήρας αυτόματα δεν αποτελεί το τέλος μιας λέξης.

void set_flags(char prev_ch, char ch, bool help_flag, bool* changeword)*

Η παραπάνω συνάρτηση έχει ως στόχο να αποδώσει τις σωστές τιμές στα flags που ελέγχουν την λειτουργία διαχωρισμού των λέξεων που διαβάζονται. Πχ. Για εύρεση τελείας '.' ανάμεσα σε αριθμό, η συνάρτηση θέτει ένα flag = true, έτσι ώστε να μη θεωρηθεί ως αλλαγή λέξης.

*void add_string_end(struct WordList** list, char* word)*

Τοποθετεί μία λέξη στο τέλος μιας λίστας λέξεων

void procedure(struct Spec spec, char* word, struct VocabEntry* vocabulary,FILE* stop_file)*

Η συνάρτηση αυτή καλείται για κάθε λέξη του spec (όπως αυτές φιλτράρονται στην set_spec). Για κάθε λέξη:

-μετατρέπει τα κεφαλαία γράμματα σε μικρά με την χρήση της lower

-ελέγχει αν αποτελεί stopword

Αν δεν αποτελεί, τότε την εισάγει στο τοπικό λεξιλόγιο του spec (με τους κατάλληλους ελέγχους για διπλοτυπα). Αν δεν υπάρχει στο τοπικό λεξιλόγιο εισάγεται στο καθολικό λεξιλόγιο του προγράμματος.

tfidf.c

void init_tf(struct Spec spec, struct VocabEntry *general_voc)*

Καλείται κάθε φορά που διαβάζουμε ένα spec. Διαπερνάει το τοπικό λεξιλόγιο του spec και υπολογίζει, για όσες λέξεις διάβασε μέχρι τώρα, τον αριθμό tf που τους αντιστοιχεί.

*float *init_idf(int num_of_specs, struct VocabEntry* vocabulary,int numOfEntries)*

Καλείται στην main.c, αφού έχουν διαβαστεί όλα τα spec του datasetX.

Δεσμεύει και αρχικοποιεί ένα πίνακα μεγέθους totalvocabularywords, όπου σε κάθε θέση υπολογίζει την τιμή idf για την αντίστοιχη λέξη στο λεξιλόγιο. (η σύνδεση λέξης και θέσης idf γίνεται μέσω του vocabulary.num).

float calculate_tfidf_mean(struct Entry* hashTable, int numOfEntries, int bucketSize, float* idf)*

Καλείται στην main.c, αφού έχουν υπολογιστεί οι idf τιμές.

Δεσμεύει και αρχικοποιεί ένα πίνακα μεγέθους finalvectorsize, στον οποίο θα αποθηκευτούν οι μέσοι όροι tfidf για κάθε λέξη.

Για κάθε spec του hashtable, υπολογίζουμε τις tfidf τιμές της κάθε στήλης και ταυτόχρονα προσθέτουμε τις τιμές αυτές στον πίνακα που έχουμε δεσμεύσει.

Στο τέλος διαιρείται κάθε τιμή του πίνακα tfidf_mean με τον αριθμό των specs, ;έτσι ώστε να υπολογιστούν οι μέσοι όροι.

Επιστρέφει τον πίνακα με τον μέσο όρο των τιμών tfidf (tfidf_mean).

void final_tfidf(struct Spec spec, float *tfidf_mean)*

Δέχεται ένα spec για το οποίο υπολογίζει το τελικό vector tfidf απομακρύνοντας τις τιμές/στήλες που αντιστοιχούν σε μικρό μέσο όρο.

Δεσμεύει ένα νέο πίνακα tfidf για το spec ο οποίος έχει μέγεθος ίσο με το τελικό μέγεθος των vectors (finalvectorsize), όπου εκεί μεταφέρονται όλες οι τιμές που θα κρατήσουμε.

result.c

*int print_results_all(struct AccessCliques *start)*

Εκτυπώνει κάθε ζευγάρι από specs τα οποία ανήκουν στην ίδια κλίκα σε ένα αρχείο output1.csv και όλα τα ζευγάρια από specs που σίγουρα δεν ανήκουν στην ίδια κλίκα σε ένα αρχείο output0.csv.

collision.c

*struct PredStruct *set_PredStruct(struct Entry* hashTable, char* specid1, char* specid2, float y)*

Δέχεται δύο specids και ένα prediction για αυτά, βρίσκει τη θέση τους στις κλίκες και αποθηκεύει σε ένα struct τις δοθείσες πληροφορίες και τον AccessCliques κόμβο που αναφέρεται στην κλίκα στην οποία ανήκουν.

*bool collisions(struct PredStruct pred_struct, struct PredList *predictions)*

Επιστρέφει την τιμή true αν για την συγκεκριμένη πρόβλεψη βρεθεί κάποιο collision, και false αν δεν βρεθεί.

Ελέγχει για collisions με βάση τις ήδη δημιουργημένες κλίκες και τα negative correlations τους με τη χρήση της *col_in_dataset* και με τη χρήση της *col_in_predictions* ελέγχει για collisions με βάση προηγούμενα predictions.

bool col_in_dataset(struct PredStruct pred_struct)

Ελέγχει για collisions με βάση τις ήδη δημιουργημένες κλίκες και τα negative correlations τους.

Επιστρέφει true αν βρεθεί collision ή false αν δεν βρεθεί.

Για πρόβλεψη που τείνει προς 1, έχουμε collision όταν τα δύο specids βρίσκονται σε κλίκες όπου μεταξύ τους υπάρχει αρνητική σύνδεση (negative correlation).

Για πρόβλεψη που τείνει προς 0, έχουμε collision αν τα δύο specids βρίσκονται στην ίδια κλίκα.

*bool col_in_predictions(struct PredStruct pred_struct, struct PredList *predictions)*

Ελέγχει για collisions με βάση τα προηγούμενα predictions.

Επιστρέφει true αν βρεθεί collision ή false αν δεν βρεθεί.

Για τα δύο specid, βρίσκει αν υπάρχει κάποιο κοινό specid με το οποίο έχουν και τα δύο πρόβλεψη. Αν οι τρεις προβλέψεις δεν συμβαδίζουν, αποφασίζεται με τη χρήση της συνάρτησης findgreater() που δέχεται ως ορίσματα τις αποστάσεις των προβλέψεων από το μηδέν ή ένα, ποια πρόβλεψη είναι η “χειρότερη”. Αν η πρόβλεψη αυτή είναι αυτή που μόλις υπολογίστηκε και ελέγχεται για collision, τότε επιστρέφεται true και δεν εισέρχεται στη λίστα προβλέψεων. Αλλιώς, αν είναι κάποια από τις δύο άλλες τιμές, τότε αυτή αφαιρείται από τη λίστα με τη χρήση της *removeFromPredictions()* και επιστρέφεται η τιμή false έτσι ώστε η νέα πρόβλεψη να ενταχθεί στη λίστα.

*void addInPredictions(struct PredList **predictions, struct PredStruct *pred_struct)*

Προσθέτει στην λίστα με τις προβλέψεις τη νέα πρόβλεψη.

*void removeFromPredictions(struct PredList **predictions, struct PredList *pred_node)*

Αφαιρεί από τη λίστα με τις προβλέψεις, μία πρόβλεψη που επιλέχθηκε ως “κακή” κατά τον έλεγχο για collisions.

char findgreater(float ac_diff, float ab_diff, float bc_diff)

Επιστρέφει ένα χαρακτήρα που προσδιορίζει ποια τιμή από τις τρεις είναι η μεγαλύτερη.

*int print_predictions(struct PredList *predictions)*

Εκτυπώνει τις προβλέψεις που περιέχονται στη λίστα προβλέψεων σε ένα αρχείο predictions.csv

Χρήση Μοντέλου Μηχανικής Μάθησης (test model):

Για να χρησιμοποιήσουμε το μοντέλο μηχανικής μάθησης, μεταγλωττίζουμε ως εξής:

make project2_testmodel

και τρέχουμε ως εξής:

./project2_testmodel tf-idf ή

./project2_testmodel bow

Αφού ξεκινήσει η εκτέλεση της main ζητείται από το χρήστη να δώσει ένα δικό του dataset (αν δεν δοθεί καινούριο dataset, χρησιμοποιείται ένα default), ώστε να χρησιμοποιήσει το περιεχόμενο του για να τεστάρει το μοντέλο.

Παράγεται αρχείο predictions.csv με τις προβλέψεις για κάθε ζευγάρι που δόθηκε στο μοντέλο.

Αρχεία Output:

Δημιουργούνται τα εξής αρχεία:

best_weights.txt: περιέχει τις τιμές των βαρών (w & b) όπως υπολογίστηκαν από τη διαδικασία της λογιστικής παλινδρόμησης.

output0.csv: περιέχει όλα τα ζευγάρια specid που σύμφωνα με τις λίστες αρνητικών συσχετίσεων, σίγουρα δεν αναφέρονται στο ίδιο αντικείμενο. Είναι της μορφής:

left_spec_di, right_spec_id, 0

output1.csv: περιέχει όλα τα ζευγάρια specid που σύμφωνα με τις κλίκες, αναφέρονται στο ίδιο αντικείμενο (αρχείο εξόδου εργασίας 1). Είναι της μορφής:

left_spec_di, right_spec_id, 1

predictions.csv: περιέχει τα ζευγάρια που χρησιμοποιήθηκαν κατά τον έλεγχο του μοντέλου, καθώς και την πρόβλεψη για αυτά. Σε αυτά δεν συμπεριλαμβάνονται όσα αφαιρέθηκαν λόγω collision.

Software testing:

Χρησιμοποιήθηκε η βιβλιοθήκη “acutest.h”.

Ελέγχονται οι συναρτήσεις στο αρχείο test2.c:

- setVector
- hashFunction2
- searchSimilar
- searchDifferent
- addWordInVocabulary
- searchWordInVocabulary
- getWordPosition
- findSpecInHashTable
- sigmoid
- f
- update_w
- init_tf
- init_idf
- calculate_tfidf_mean
- final_tfidf
- endofword