Ανάπτυξη Λογισμικού για Πληροφοριακά Συστήματα

Project2

Περιγραφή προγράμματος

Το project αυτό αποτελεί επέκταση του πρώτου μέρους (project1). Προσθέτει λοιπόν στο πρόγραμμα τις παρακάτω λειτουργείες:

Κατά την διαδικασία δημιουργίας κλικών με θετικές συσχετίσεις, δημιουργεί αρνητικές συσχετίσεις προσθέτοντας σε κάθε κλίκα μια λίστα με δείκτες στις κλίκες με τις οποίες διατηρεί αρνητική συσχέτιση. Η διαδικασία αυτή εκτελείται όταν κάποια γραμμή του .csv αρχείου έχει στο πεδίο label την τιμή 0.

Κατά το parcing των πληροφοριών των json αρχείων, οι διαδοχικές λέξεις «καθαρίζονται» με τον εξής τρόπο: αφαιρούνται τα σημεία στίξης, όλα τα γράμματα μετατρέπονται σε μικρά, ενώ αν η λέξη ανήκει στα αγγλικά stopwords απορρίπτεται. Αν περάσει τον έλεγχο, προστίθεται σε μια λίστα η οποία περιέχει όλες τις διαδοχικές λέξεις κάθε αρχείου (αν βρίσκεται ήδη στην λίστα ενημερώνεται ο αντίστοιχος μετρητής εμφανίσεων της λέξης), καθώς και σε ένα νέο hashTable το οποίο περιέχει όλες τις μοναδικές λέξεις κάθε αρχείου. Αν η λέξη υπάρχει ήδη στο hashTable, ενημερώνεται ο αντίστοιχος μετρητής της. Όταν ολοκληρωθεί αυτή η διαδικασία, υπολογίζουμε για κάθε λέξη των json αρχείων το TF score της. Γνωρίζοντας την πληροφορία αυτή, υπολογίζουμε για κάθε λέξη του hashTable το IDF score (χάρη στον counter κάθε λέξης, αρκεί για κάθε λέξη να πραγματοποιηθεί μία διαίρεση, μειώνοντας την συνολική απαιτούμενη πολυπλοκότητα της διαδικασίας). Μπορούμε να υπολογίσουμε λοιπόν το TF-IDF score κάθε λέξης του hashTable ως τον μέσω όρο των διαδοχικών TF-IDF scores των λέξεων των json αρχείων.
Έπειτα, αφαιρούμε από το hashTable τις λέξεις με χαμηλό TF-IDF score, λέξεις που εμφανίζονται δηλαδή πολλές φορές και ως αποτέλεσμα δεν δίνουν κάποια σημαντική πληροφορία. Για το medium αρχείο κρατάμε περίπου τις 1000 πιο σημαντικές λέξεις ενώ για το large τις 100 σημαντικότερες.

Στην συνέχεια, δημιουργούμε έναν machine learning classifier με την τεχνική του logistic regression. Διατρέχουμε ακόμα μια φορά το .csv αρχείο και για κάθε ζεύγος αρχείων υπολογίζουμε τον πίνακα tf-idf που αναπαριστά τα δεδομένα του (προέρχεται από ένωση των 2 ατομικών πινάκων), τον φορτώνουμε σε ένα πίνακα με σκοπό να ανακατευθεί με τους υπόλοιπους. Έπειτα διατρέχουμε τον πίνακα με τις ανακατεμένες γραμμές. Στο σημείο αυτό δημιουργούνται δύο διακριτές επιλογές:

α) Αν η γραμμή ανήκει στο πρώτο 80% του πίνακα:
κάνουμε train το μοντέλο με 2 η 10 περάσματα αν το αποτέλεσμα είναι 0, 1 αντίστοιχα.

β) Αν η γραμμή ανήκει στο τελευταίο 20% του αρχείου:
κάνουμε test με τον συγκεκριμένο πίνακα και εκτυπώνουμε την πρόγνωση του μοντέλου μαζί με το πραγματικό αποτέλεσμα του ζεύγους στο αρχείο predictions.txt.

Τελικά εκτυπώνονται τα δεδομένα του classifier στο αρχείο classifier.txt, αποδεσμεύεται η δεσμευμένη μνήμη και το πρόγραμμα τερματίζει.

Περιγραφή Υλοποίησης

Βελτιώσεις πρώτου μέρους:

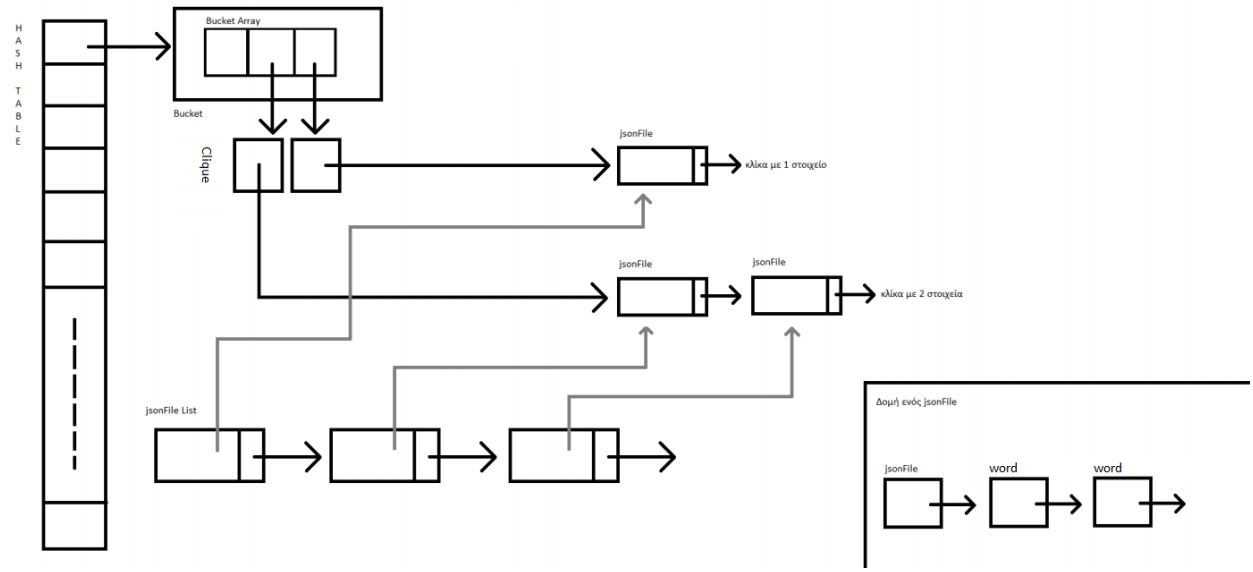
1. Απαλοιφή των warnings της παραμέτρου -Wall.
2. Αντικατάσταση περιττής δομής red-black tree με δομή κλίκας.
3. Σωστή απελευθέρωση μνήμης όλων των προγραμμάτων (project και test)

Νέες δομές δεδομένων που χρησιμοποιήθηκαν:

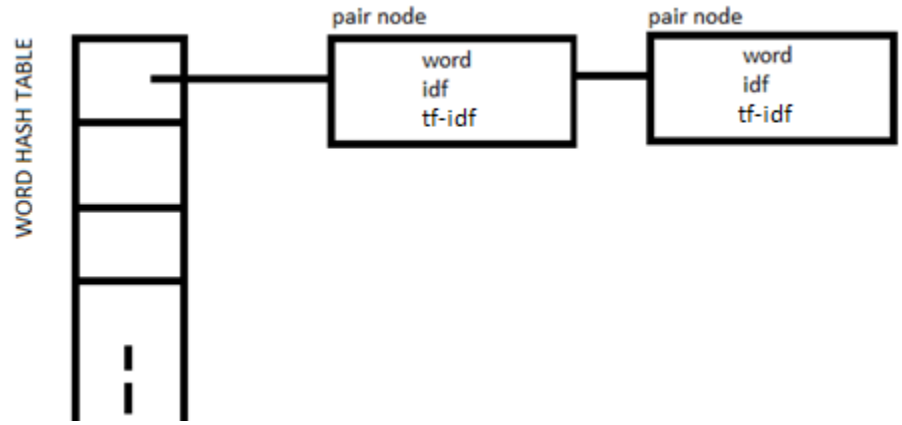
Οι βασικές δομές είναι οι ίδιες με το πρώτο μέρος, με εξαίρεση τις αλλαγές του περιγράφονται παραπάνω και τις επόμενες νέες προσθήκες:

1. hashTable για την αποθήκευση των μοναδικών λέξεων και τον υπολογισμό των TF, IDF, TF-IDF.
2. Δομή logistic regression για τον classifier η οποία περιλαμβάνει: πίνακα με τα βάρη κάθε όρου, δείκτη στον πίνακα X που εξετάζεται κάθε φορά, το αποτέλεσμα Y του προαναφερθέντος πίνακα, learning rate για το μοντέλο, error.
3. Λίστα με λέξεις και έναν μετρητή για κάθε λέξη, η οποία χρησιμοποιείται για την αποθήκευση των λέξεων κάθε json αρχείου.

Σχηματικά:



Εικόνα 1 Οι δομές και οι σχέσεις μεταξύ τους



Εικόνα 2 Το hashTable για την αποθήκευση των λέξεων

Περιγραφή Βασικών Συναρτήσεων

void parcerCSV(char *path, char *csv, Bucket **hashtable, int hashSize)
Διαβάζει το dataset και ενημερώνει τις σχέσεις μεταξύ των json αρχείων, ενώνοντας κλίκες όπου χρειάζεται και δημιουργώντας αρνητικές συσχετίσεις, όπως περιγράφεται στην εκφώνηση.

int check_neg_cliques(Bucket** ht, char* word1, char* word2,int limit)
Ελέγχει για 2 json αρχεία που δέχεται (word1, word 2), αν το ένα βρίσκεται σε κάποια από τις αρνητικές συσχετίσεις του άλλου. Χρησιμοποιείται για τον έλεγχο της ακεραιότητας των αρνητικών συσχετίσεων

void NegativeQliqueTest(char *path, char *csv, Bucket **hashtable, int hashSize)
Πραγματοποιεί τον έλεγχο της παραπάνω συνάρτησης για όλες τις αρνητικές συσχετίσεις που βρίσκονται στο Dataset. Σε περίπτωση προβλήματος, εκτυπώνει το ανάλογο μήνυμα.

word_ht* create_word_ht(int size)
Δημιουργεί και επιστρέφει ένα hashTable στο οποίο αποθηκεύονται όλες οι μοναδικές λέξεις των αρχείων. Κάθε λέξη έχει: id, μετρητή εμφάνισης, μετρητή κειμένων στα οποία εμφανίζεται, idf score, tf-idf score.

void deleteWordsWithCount(word_ht *ht, double key)
Διατρέχει το hashTable των λέξεων και διαγράφει όσες έχουν tf-idf score μικρότερο από το key.

void reCalculateHashTableWordId(word_ht *ht)
Διατρέχει το hashTable και αναθέτει νέα id για όλες τις λέξεις. Η κλήση αυτής της συνάρτησης είναι αναγκαία έπειτα από την διαγραφή λέξεων.

jsonFile* parcer(FILE *fp, char *site, char *temp, word_ht *ht)
Διαβάζει ένα json αρχείο, καθαρίζει τις λέξεις και όσες κρατάει τις τοποθετεί στο λεξικό του συγκεκριμένου jsonFile αλλά και στο hashTable με τις μοναδικές λέξεις

void calculateRecordListTF(recordList *list)
Διατρέχει όλα τα json αρχεία και υπολογίζει για κάθε λέξη κάθε αρχείου το tf score που της αναλογεί

void CalculateIDF(recordList *reclist, word_ht *wordHash, int size)
Διατρέχει το hashTable των μοναδικών λέξεων και για κάθε λέξη υπολογίζει το idf score, καθώς και το tf-idf ως τον μέσο όρο όλων των tf-idf score που της αναλογούν.

Bucket** initHashTables(int maxB, int bSize)
Δημιουργεί και επιστρέφει το hashTable στο οποίο βρίσκονται οι κλίκες που περιέχουν τα αρχεία.

Bucket* addToHashTable(Bucket* ht, char* key, int position, jsonFile* file)
Δημιουργεί μια κλίκα και τοποθετεί σε αυτή το jsonFile. Έπειτα η κλίκα τοποθετείται σε κατάλληλη θέση στο hashTable με βάση το όνομα (πχ www.ebay.com//45)

jsonFile* findJsonFromCsv(Bucket** ht,int hash, char *key)
Ψάχνει το ζητούμενο json αρχείο στο hashTable των κλικών και επιστρέφει δείκτη σε αυτό.

void CreateJsonListWordCountArray(jsonFile *list, int arraySize)
Δημιουργεί ένα πίνακα μεγέθους arraySize τον οποίο αρχικοποιεί με 0,0. Είναι ο tf-idf πίνακας που αναπαριστά τα δεδομένα του αρχείου.

void FillJsonWordCountArray(jsonFile *list, int arraySize,word_ht *wordHash)
Για κάθε λέξη του αρχείου που βρίσκεται στο hashTable, υπολογίζει το tf-idf score της και το τοποθετεί στο κατάλληλο κελί του πίνακα.

double* arrayConcat(double *startArray, double *endArray, int size)
Δέχεται 2 πίνακες tf-idf, και τους ενώνει κρατώντας για κάθε κελί την ευκλείδεια απόσταση των τιμών τους.

double** shuffleArray(double** array, int *array2, char **array3, char ** array4, int size)
Δέχεται τον πίνακα array, ο οποίος είναι οι tf-idf αναπαραστάσεις όλων των ζευγαριών του dataset, τον array2 ο οποίος περιέχει τα αποτελέσματα της κάθε γραμμής και τους array3, 4 οι οποίοι περιέχουν τα ονόματα. Αναμεταθέτει τυχαία τα περιεχόμενα των πινάκων ώστε να είναι έτοιμοι για το training, testing.

logistic_reg * create_logistic_reg(int lineSize);
Δημιουργεί το μοντέλο μηχανικής μάθησης που χρησιμοποιούμε αρχικοποιεί τις βασικές πληροφορίες και το επιστρέφει.

void fit(logistic_reg * cur,double *x,int y, int lineSize)
Τοποθετεί στο μοντέλο έναν πίνακα και το αποτέλεσμα του.

logistic_reg* logisticRegretionAlgorithm(logistic_reg *cls, int limit)
Γίνεται train του μοντέλου με τον τελευταίο πίνακα που έγινε fit. Τροποποιούνται δηλαδή τα βάρη με την αντίστοιχη μαθηματική διαδικασία που περιγράφεται στην εκφώνηση. Αν y είναι 0, γίνεται train 2 φορές με το ίδιο αρχείο, διαφορετικά 10 φορές.

double cost_function(logistic_reg *cls)
Εκτυπώνει το κόστος του training.

positiveMetrics *initPositiveMetrics(char *name)
Αρχικοποιεί την δομή η οποία υπολογίζει και εκτυπώνει στατιστικά (precision, f1, recall) για τα αποτελέσματα του testing με y = 1.

negativeMetrics *initNegativeMetrics(char *name)
Αρχικοποιεί την δομή η οποία υπολογίζει και εκτυπώνει στατιστικά (precision, f1, recall) για τα αποτελέσματα του testing με y = 0.

positiveMetrics *updatePositiveMetrics(positiveMetrics *metrics, int x, int y)
Δέχεται το αποτέλεσμα του classifier (x) καθώς και το label (y) και ενημερώνει αντίστοιχα τους μετρητές ώστε στο τέλος να υπολογίσει τις ζητούμενες μετρικές. Καλείται όταν y = 1. Αντίστοιχα λειτουργεί και η updateNegativeMetrics.

positiveMetrics *evaluatePositiveMetrics(positiveMetrics *metrics)
Υπολογίζει τα 3 score από τους μετρητές που έχει τροποποιήσει. Αντίστοιχα λειτουργεί και η evaluateNegativeMetrics.

void printPositiveMetrics(positiveMetrics *metrics)
Εκτυπώνει τα score.

double logisticRegrationTest(logistic_reg *cls, double *data, char *left, char *right, int y, FILE *fp,positiveMetrics *P_metrics, negativeMetrics *N_metrics)
Υπολογίζει για το data το prediction του classifier και εκτυπώνει το αποτέλεσμα (μορφη: json1, json2, prediction, actual result) στο αρχείο fp. Ενημερώνει επίσης τις μετρικές positive, negative.

recordList* createRecordList()
Δημιουργεί και επιστρέφει μια λίστα η οποία θα περιέχει nodes με δείκτες σε json αρχεία.

int insertRecordList(recordList * list, recordNode *node)
Τοποθετεί ένα node το (οποίο δείχνει σε ένα json αρχείο) στην λίστα με τα records.

Unit Testing

Έχει γίνει unit testing των συναρτήσεων της εργασίας χρησιμοποιώντας την βιβλιοθήκη acutest.h, η οποία και έχει συμπεριληφθεί στα αρχεία της εργασίας. Ο έλεγχος έχει επίσης ενσωματωθεί με το github μέσω της λειτουργείας github actions, και πραγματοποιείται κάθε φορά που γίνεται push στο repository. Τα αποτελέσματα του είναι τα ακόλουθα:

```
Test list_create...                                [ OK ]
Test JsonNode_create...                            [ OK ]
Test JsonFile_create...                            [ OK ]
Test RecordNode_create...                          [ OK ]
Test RecordList_create...                          [ OK ]
Test hashTable_create...                           [ OK ]
Test test_search...                                [ OK ]
Test test_word_list_and_node...                    [ OK ]
Test test_word_hash_table...                       [ OK ]
Test test_logistic_regression...                   [ OK ]
Test test_array_functions...                       [ OK ]
SUCCESS: All unit tests have passed.
```

Compiling - Εκτέλεση

Το compile του κώδικα γίνεται με την εντολή «make» καθώς έχει υλοποιηθεί makefile. Γίνεται separate compilation των αρχείων για να παραχθεί το εκτελέσιμο πρόγραμμα.

Η εκτέλεση γίνεται με τον ακόλουθο τρόπο:

make run: Ο βασικός τρόπος εκτέλεσης του προγράμματος. Εκτελεί την εντολή:
./project2 -x Datasets -c sigmod_large_labelled_dataset.csv

Υπάρχουν επίσης αρκετές επιλογές οι οποίες τρέχουν το πρόγραμμα με χρήση valgrind η gdb όπως: make valgrind, make valgrind_extreme

Τα αποτελέσματα της εκτέλεσης του προγράμματος με χρήση valgrind είναι τα ακόλουθα:

```
==15601==
==15601== HEAP SUMMARY:
==15601==     in use at exit: 0 bytes in 0 blocks
==15601==   total heap usage: 17,426,746 allocs, 17,426,746 frees, 945,561,514 bytes all
ocated
==15601==
==15601== All heap blocks were freed -- no leaks are possible
==15601==
==15601== For lists of detected and suppressed errors, rerun with: -s
==15601== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```


Κουμερτάς Σταύρος 1115201600231
Κοροβέσης Παναγιώτης 1115201700056
Κοροβέσης Χαράλαμπος 1115201700057