ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Ανάπτυξη λογισμικού για πληροφοριακά συστήματα

Τελική αναφορά στην εργασία από τους Φοιτητές:

Πλας Κωνσταντίνος 1115201700125
Στέτσικας Ελευθέριος 1115201700150

Επιμελητής: Πικραμένος Γεώργιος

Εισαγωγή

Στην αρχή του εξαμήνου κληθήκαμε να υλοποιήσουμε μία εργασία, η οποία θα έπρεπε στην αρχή να διαβάζει ένα αρχείο με ηλεκτρονικούς συνδέσμους από online καταστήματα τύπου, “left_spec, right_spec, label”, όπου το κάθε spec απεικονίζει μία ηλεκτρονική διεύθυνση και το label, δήλωνε αν αυτές οι δύο διαφορετικές ηλεκτρονικές διευθύνσεις ήταν για το ίδιο προϊόν ή όχι. Εμείς έπρεπε να υλοποιήσουμε, αρχικά ένα πρόγραμμα που να βρίσκει και να αποθηκεύει τις θετικές συσχετίσεις. Έπειτα ζητήθηκε να βρίσκονται από το πρόγραμμα και οι αρνητικές συσχετίσεις, ομοίως να αποθηκεύονται, ενώ με βάση τα δεδομένα που είχε κάθε ιστοσελίδα σε αρχεία τύπου .json να τα επεξεργάζεται το πρόγραμμα, να τα λάμβανε για να κάνει train, έτσι ώστε να μπορεί να μαντεύει τι μπορεί να ισχύει για δύο τυχαίες διευθύνσεις. Στο τελευταίο κομμάτι της εργασίας υλοποιήθηκαν νήματα (*threads*) για να γίνει η εκτέλεση του προγράμματος λιγότερο χρονοβόρα, για μεγάλα μεγέθη δεδομένων.

Σύντομη περιγραφή των δομών δεδομένων

HashTable

Μία από τις σημαντικότερες δομές που υλοποιήσαμε για αυτή την εργασία ήταν ένα HashTable. Το HashTable, αυτό, για εξοικονόμηση χώρου, αναδιαμορφωνεται όταν υπάρχει overflow στα δεδομένα για τον καλύτερο διαμοιρασμό των δεδομένων στα κελιά του. Ακόμη για να επιτευχθεί αυτό, όταν ανανεώνεται ο αριθμός των κελιών του HashTable επιλέγεται ο επόμενος μεγαλύτερος πρώτος αριθμός από το προηγούμενο πλήθος κελιών του, ώστε να είναι το νέο πλήθος κελιών του.

Στο να καταλήξουμε να επιλέξουμε τη δομή HashTable για την αποθήκευση των κλικών και των σχέσεών τους μας ώθησε ο κανόνας της μεταβατικότητας που μπορεί να ισχύει μεταξύ των κλικών. Με το HashTable μπορούμε να γνωρίζουμε χωρίς να σπαταλάμε χρόνο, για κάθε κλίκα τις

συσχετίσεις της, ενώ όταν γινόταν ένωση μεταξύ δύο διαφορετικών κλικών όλη αυτή η διαδικασία γινόταν άμεσα. Ακόμη άμεση έγινε και η αποθήκευση των αρνητικών συσχετίσεων μεταξύ των κλικών. Τέλος εξοικονομείται αρκετός χώρος αφού το HashTable, αναδιαμορφώνει μόνο του τον χώρο του, όταν χρειάζεται παραπάνω θέσεις.

BucketList

Το BucketList είναι μία απλή συνδεδεμένη λίστα που όλα τα στοιχεία της αποτελούν θετικά συσχετισμένες κλίκες.

Dictionary

Το Dictionary αποτελείται από μία απλή συνδεδεμένη λίστα η οποία χρησιμοποιείται στην αποθήκευση των στοιχείων του .json δεδομένων κάθε προϊόντος, για να μπορεί το πρόγραμμα μας να τα επεξεργαστεί κάθε στιγμή. Κάθε .json αρχείο αποθηκεύεται σε ξεχωριστή δομή Dictionary στην μνήμη όταν προσπελαύνεται. Χρησιμοποιήσαμε το dictionary καθώς θεωρήσαμε ότι η χρήση μιας πιο πολύπλοκης δομής δεν ήταν απαραίτητη λόγο της περιορισμένης πληροφορίας κάθε μεμονωμένου json.

Queue

Η δομή Queue δημιουργήθηκε για να αποθηκεύουμε τις διαφορετικές διεργασίες που πρόκειται να εκτελέσουν τα threads με τη σειρά που αυτές μπήκαν στο Queue.

RedBlackTree

Χρησιμοποιήσαμε ένα RedBlackTree για να αποθηκεύουμε τις καλύτερες τιμές του prediction. Έτσι ώστε όταν είναι να κάνουμε resolve το transitivity, χρησιμοποιούμε πάντα πρώτα τις τιμές που θεωρούμε καλύτερες, και αυτές που λαμβάνουμε πιο μετά και προκαλούν προβλήματα, είναι χειρότερες προβλέψεις από τις προηγούμενες, διότι τα έχουμε κατανέμει στο RedBlackTree με την απόσταση από το 0.5, όσο μεγαλύτερη τόσο καλύτερη η πρόβλεψη. Επιλέξαμε το RedBlackTree ως δομή για την καλύτερη ταξινόμηση των predictions. Σε κάθε κόμβο του δέντρου υπάρχει μία λίστα στην οποία αποθηκεύονται τα στοιχεία με τα ίδια κλειδιά.

SecTable

Η δομή αυτή είναι ένα γενικού τύπου HashTable. Το οποίο παρόμοια αναδιαμορφώνεται για εξοικονόμηση χώρου, απλά μπορεί κι όλας να αποθηκεύσει διαφορετικά αντικείμενα, έτσι ώστε να μπορούμε να το χρησιμοποιήσουμε πολλές φορές σε διαφορετικές καταστάσεις. Μες τη δομή λοιπόν αποθηκεύεται τι είδους αντικείμενα αποθηκεύονται, οι συναρτήσεις σύγκρισης, κατακερματισμού και διαγραφής κάθε αντικειμένου έτσι ώστε όταν είναι να αποθηκεύσουμε ένα αντικείμενο στο SecTable, δεν μας απασχολεί ο τύπος του, γιατί από το SecTable καλούνται οι κατάλληλες συναρτήσεις για να γίνουν οι πράξεις τους.

Το SecTable χρησιμοποιήθηκε για να μπορούμε να βρούμε ποιες κλίκες σχετίζονται αρνητικά, αφού ένα SecTable, περιέχεται σε κάθε κλίκα, και εκεί αποθηκεύονται οι διευθύνσεις των αρνητικών κλικών. Ακόμη χρησιμοποιήθηκε για να αποθηκεύσουμε το λεξιλόγιο, έτσι ώστε να δημιουργήσουμε το bag of Words ή tf-idf, για κάθε ηλεκτρονική διεύθυνση που έχει αποθηκευτεί,για να χρησιμοποιηθεί στην μηχανική μάθηση του μοντέλου.

SparseVector

Το SparseVector είναι μία δομή από δύο πίνακες όπου στον πρώτο πίνακα αποθηκεύεται ο index που θα είχε η τιμή αυτή στον κανονικό πίνακα, και στον άλλο αποθηκεύεται η τιμή. Τα SparseVector δημιουργήθηκαν έτσι ώστε να μειώσουμε το χρόνο εκτέλεσης, απορρίπτοντας τις άχρηστες πληροφορίες.

Σύντομη περιγραφή του τρόπου εκτέλεσης του προγράμματος main

1ο μέρος: Αποθήκευση δεδομένων

Αρχικά το πρόγραμμα μας προσπελαύνει το φάκελο που περιέχονται τα .json αρχεία και τα αποθηκεύει στο HashTable ως www.**********.com//xxxx όπου το xxxx αναπαριστά τον αριθμό του αρχείου .json (xxxx.json) και η διεύθυνση είναι ο φάκελος που βρίσκεται το αρχείο json. Τα αποθηκεύει σε ένα *Dictionary* και δημιουργεί μια μεμονωμένη κλίκα, *BucketList,* για κάθε στοιχείο. Παράλληλα δημιουργεί και το λεξιλόγιο κάθε spec. Έπειτα

προσπελαύνεται ένα δεδομένο .csv αρχείο που περιέχει τις συσχετίσεις μεταξύ των κλικών και δημιουργούνται κατάλληλα στο HashTable.

2ο μέρος: Μηχανική Μάθηση

Λαμβάνοντας τις νέες δημιουργημένες κλίκες δημιουργείται ένα μοντέλο για την μηχανική μάθηση. Κατά τη διάρκεια της εκμάθησης του μοντέλου προσπελαύνεται ένα αρχείο με συσχετίσεις, ανακατεύονται τα δεδομένα από το πρόγραμμα, έτσι ώστε να διαμοιραστούν τυχαία και ξεκινά η εκμάθηση. Τα δεδομένα χωρίζονται σε μικρότερους πίνακες, ενώ το κύριο πρόγραμμα ξεκινά διαφορετικά νήματα, και διαμοιράζει τους πίνακες κατάλληλα σε αυτά. Τα βάρη του μοντέλου αρχίζουν να αλλάζουν, και για να γίνεται αντιληπτή η πορεία του μοντέλου, τυπώνονται τα metrics μετά από κάθε ολοκληρωμένο κύκλο εκμάθησης. Όταν τελειώνει κάποιος κύκλος εκμάθησης, τότε το πρόγραμμα ξεκινάει και προβλέπει τις τιμές που θα μπορούσαν να πάρουν αυτά τα δεδομένα και με βάση κάποιου threshold επιλέγει τις καλύτερες προβλέψεις, τις ταξινομεί σε ένα *RedBlackTree,* και χρησιμοποιεί τα καινούργια δεδομένα έτσι ώστε να εισάγει κι άλλα δεδομένα στο training set, αφότου επιλυθούν τα προβλήματα που μπορεί να καταπατούν τον κανόνα της μεταβατικότητας. Η διαδικασία αυτή επαναλαμβάνεται για αρκετούς κύκλους και σταματάει όταν το threshold φτάσει στο 0.5, όπου θεωρούμε ότι το μοντέλο μας έχει εκπαιδευτεί αρκετά.

3ο μέρος: Αποθήκευση των δεδομένων

Στο τέλος του προγράμματος αποθηκεύονται τα βάρη του μοντέλου, και όλα τα χαρακτηριστικά του τρόπου εκτέλεσης του προγράμματος σε ένα αρχείο, έτσι ώστε να μπορούμε να επαναχρησιμοποιήσουμε το μοντέλο, χωρίς να χρειάζεται να εκτελέσουμε την χρονοβόρα διαδικασία της μηχανικής μάθησης. Ακόμη αποθηκεύεται και το λεξιλόγιο, που χρησιμοποιήθηκε, για να μπορούμε να το χρησιμοποιήσουμε σε διαφορετικά προγράμματα, αφού αν πάμε να το δημιουργήσουμε από την αρχή σε άλλο πρόγραμμα μπορεί να δημιουργηθεί διαφορετικά σε 2η εκτέλεση, ή μπορεί να θέλουμε να εκτελέσουμε με τα δεδομένα αυτά το πρόγραμμα μας σε διαφορετικό υπολογιστή, έτσι δεν χάνουμε σημαντικές πληροφορίες.

Σύντομη περιγραφή του τρόπου εκτέλεσης του προγράμματος inference

1ο μέρος: Προσπέλαση δεδομένων

Αρχικά το πρόγραμμα προσπελαύνει τα αρχεία που είναι για να δημιουργήσει κατάλληλα το μοντέλο, το λεξιλόγιο και δημιουργεί και τις μεμονωμένες κλίκες που δημιουργούσε και το προηγούμενο πρόγραμμα από τον φάκελο που περιέχονται τα .json αρχεία.

2ο μέρος: Προβλέψεις και metrics

Το πρόγραμμα προσπελαύνει ένα δεδομένο αρχείο που είναι να γίνουν οι προβλέψεις. Χωρίζει το αρχείο σε μέρη, δίνει σε διαφορετικά νήματα να δώσουν τη πρόβλεψη για διαφορετικά ζητούμενα του αρχείου, οι τελικές προβλέψεις καταγράφονται στη μνήμη και υπολογίζεται το metrics score του αρχείου.

Σύντομη περιγραφή της παραλληλίας του προγράμματος

Threads

Για να μπορέσουμε να μειώσουμε το χρόνο εκτέλεσης του προγράμματος δημιουργήσαμε threads, τα οποία λειτουργούν ταυτόχρονα έτσι ώστε να εκπαιδεύσουν το μοντέλο ή να βγάλουν κάποια πρόβλεψη για κάποιο δεδομένο εκπαιδευμένο μοντέλο. Τα threads δημιουργούνται από το κύριο πρόγραμμα, και με το που τους ανατεθεί κάποια δουλειά απο το JobScheduler, τότε την αναλαμβάνει κάποιο νήμα και την εκτελεί.

JobScheduler

Για να μπορούν τα threads να εκτελούν διαφορετικές διεργασίες, χωρίς να μας νοιάζουν τα ορίσματα, ούτε τι τύπος είναι η διεργασία και χωρίς ο κώδικας να γίνεται εξαιρετικά περίπλοκος δημιουργήσαμε τον JobScheduler. Ο JobScheduler, λαμβάνει δεδομένα τύπου JobScheduler, και τα προσθέτει στην ουρά εκτέλεσης. Όταν υπάρχει ένα νέο αντικείμενο στην ουρά, στέλνει σήμα στα threads για να πάει κάποιο να το εκτελέσει. Όταν είναι να σταματήσουν τα threads, ο JobScheduler, περιμένει τα threads όλα να

τελειώσουν τις διεργασίες που τους έχουν ανατεθεί, και στέλνει σήμα ότι πρέπει να τερματίσουν. Ακόμη ο JobScheduler ελέγχει να μην υπάρχει κάποια παρεμβολή στην μνήμη που μοιράζονται τα threads και τους δίνει κατάλληλα τα ορίσματα από κάθε διεργασία που είναι να εκτελέσουν.

Παρατηρήσεις που κάναμε κατά τη διάρκεια της υλοποίησης

1. Στην αρχή το HashTable δεν έπαιρνε πρώτους αριθμούς για την αναδιαμόρφωση του μεγέθους του, αφού το αλλάξαμε παρατηρήσαμε ότι υπήρχαν όλο και λιγότερες αναδιαμορφώσεις του.
2. Στην αρχή δεν είχαμε SparseVectors, αλλά κανονικούς πίνακες για την μηχανική μάθηση, η δραστική μείωση του χρόνου εκτέλεσης, και η λιγότερη μνήμη που δέσμευαν αποδείχθηκε πολύ καλύτερη επιλογή.
3. Τα προβλήματα με τον κανόνα της μεταβατικότητας, ήταν να επιλυθούν με ένα binary Heap αλλά παρατηρήθηκε ότι η μνήμη που δεσμευόταν ήταν αρκετή και αντικαταστάθηκε από Red Black Trees.

Έλεγχοι σωστής εκτέλεσης και απόδοσης του προγράμματος

Για να μπορέσουμε κατά τη διάρκεια της υλοποίησης του προγράμματος να ελέγξουμε αν το πρόγραμμα δημιουργείται σωστά, χρησιμοποίηθηκαν διαφορετικά εργαλεία. Χρησιμοποιήθηκε ο **Valgrind** για να ελέγχουμε αν η μνήμη έχει απελευθερωθεί σωστα, όπως και ο **GDB** όταν βρίσκαμε κάποιο σοβαρό πρόβλημα κατά τη διάρκεια εκτέλεσης των προγραμμάτων. Για να ελέγχουμε το χρόνο εκτέλεσης του προγράμματος χρησιμοποιούσαμε την εντολή **time** των linux, η οποία μας βοήθησε να μειώσουμε δραστικά το χρόνο εκτέλεσης του προγράμματος. Τέλος για να δούμε αν οι δομές μας και οι συναρτήσεις μας είναι σωστές, χρησιμοποιούσαμε την βιβλιοθήκη **acutest**, όπου φτιάχναμε test, έτσι ώστε, όταν κάποιο μέρος του κώδικα αποτύγχανε σε αυτά, ελέγχαμε τον κώδικα σε εκείνο το σημείο και τον διορθώναμε.

Στατιστικά στοιχεία και μετρήσεις

Οι μετρήσεις του προγράμματος κατά τη διάρκεια της εκτέλεσης ήταν αναγκαστικές για να μπορούμε να διακρίνουμε την πορεία του. Παρακάτω παραθέτω μερικά γραφήματα με τις διάφορες μετρήσεις του προγράμματος.

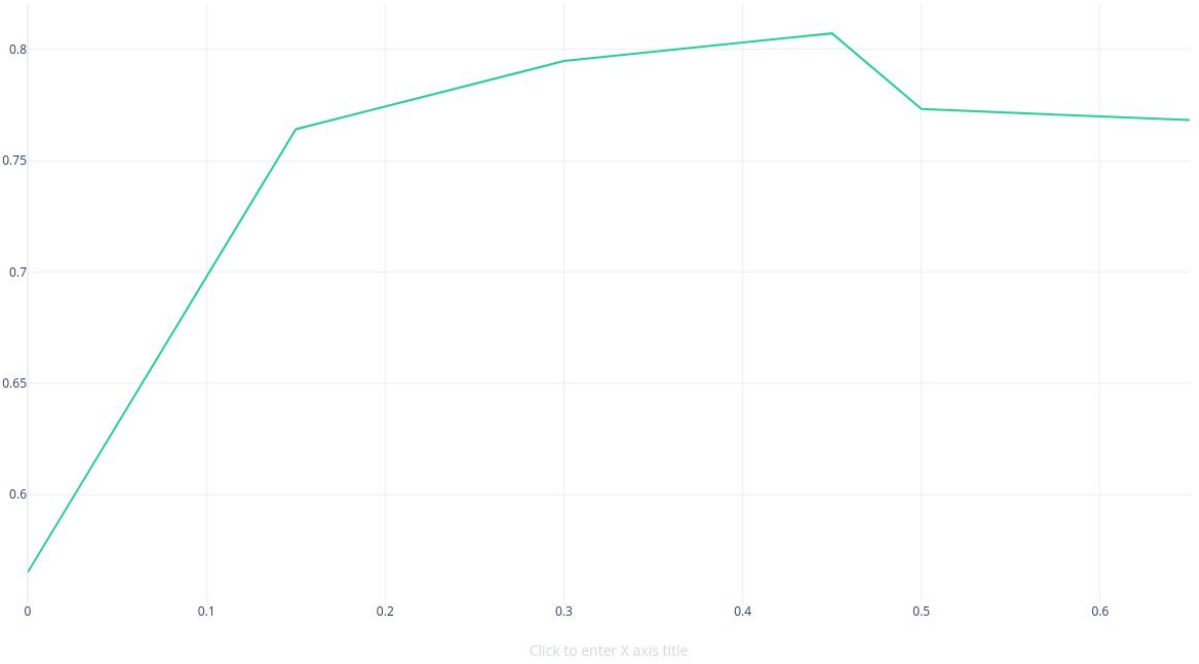
Εκτέλεση προγράμματος με την εντολή:

./main -f sigmod_large_labelled_dataset.csv -n on -v abs -b tf-idf -m 1000

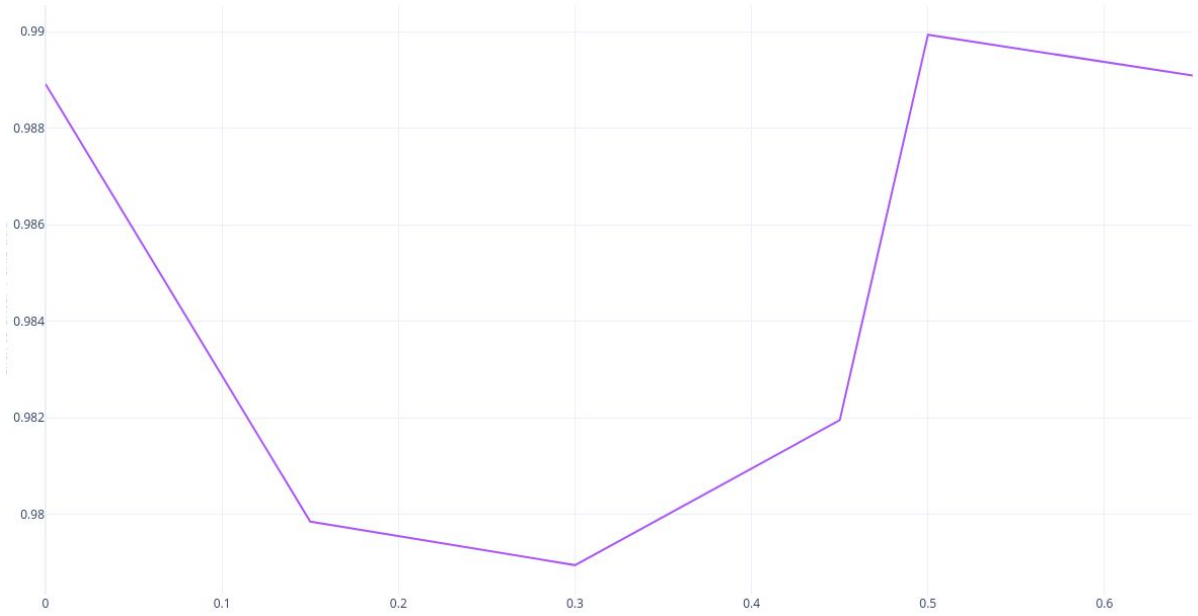
(x : Threshold Value, y : Precision for Positive relations)

(x : Threshold Value, y : Recall for Positive relations)

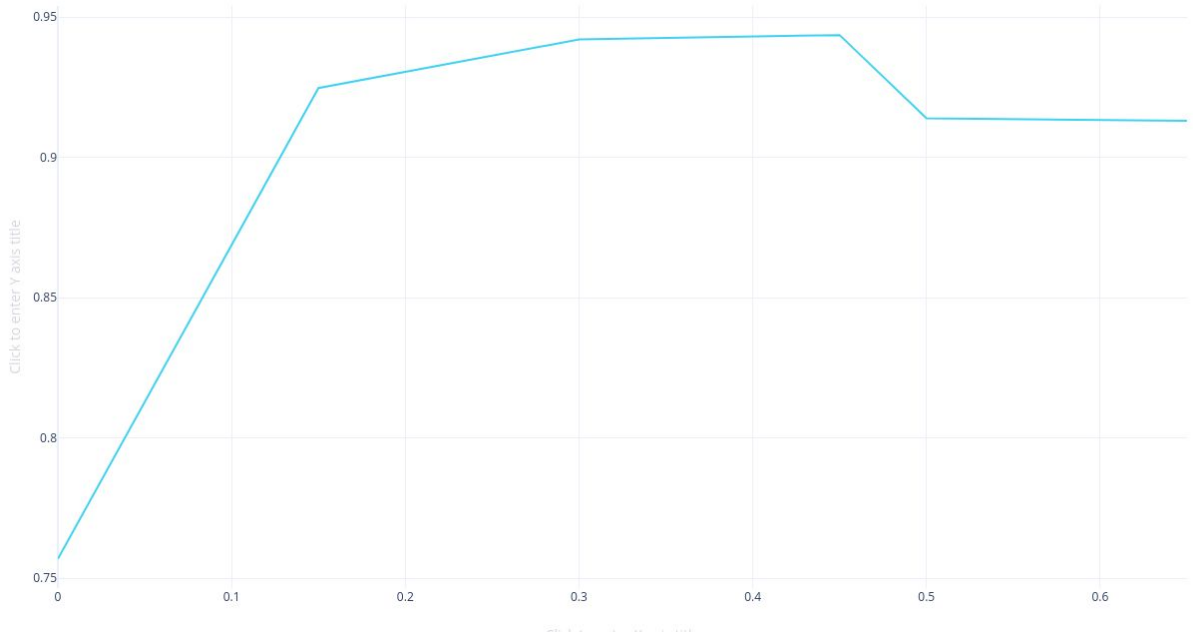
(x : Threshold Value, y : F1 score for Positive relations)



(x: Threshold Value, y : Precision for Negative relations)

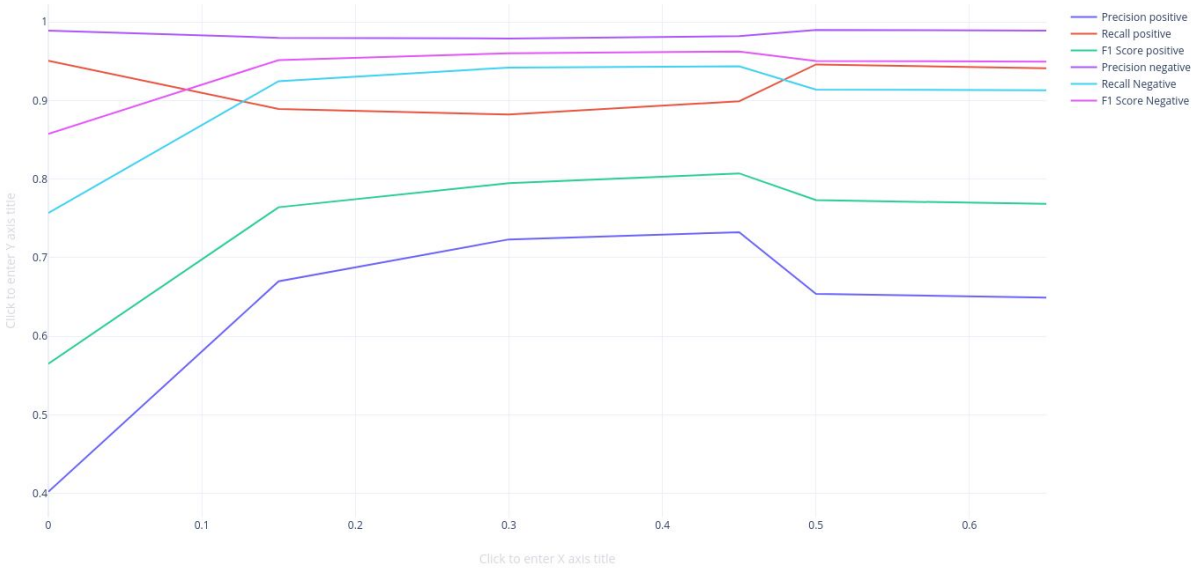


(x: Threshold Value, y : Recall for Negative relations)



(x: Threshold Value, y : F1 Score for Negative relations)

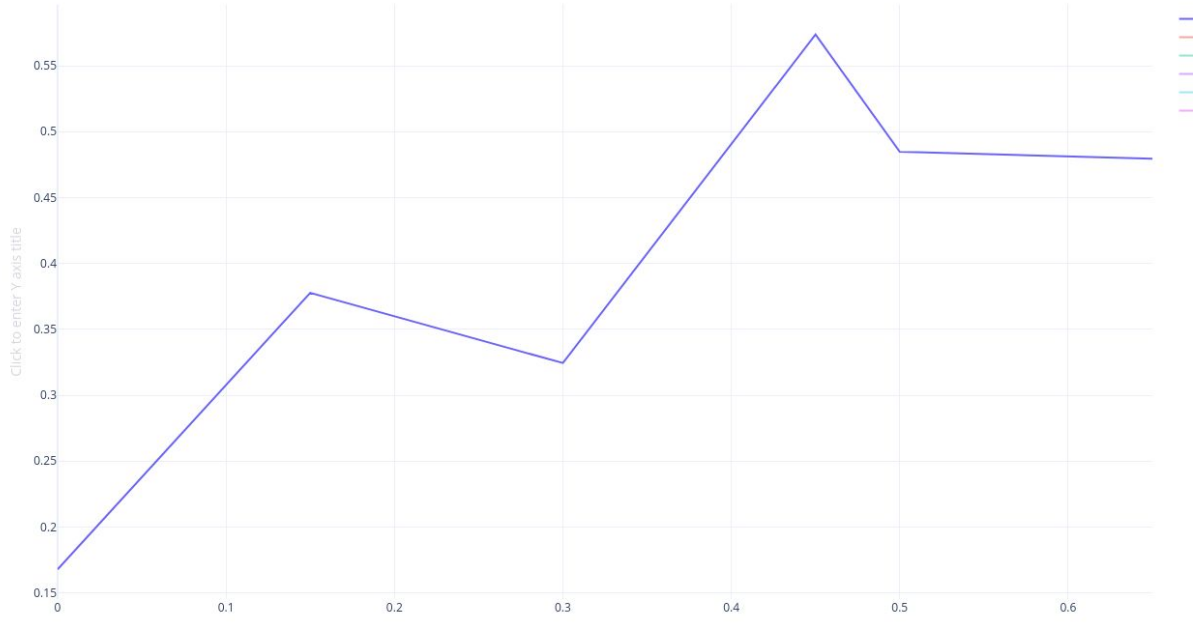
Όλα μαζί:



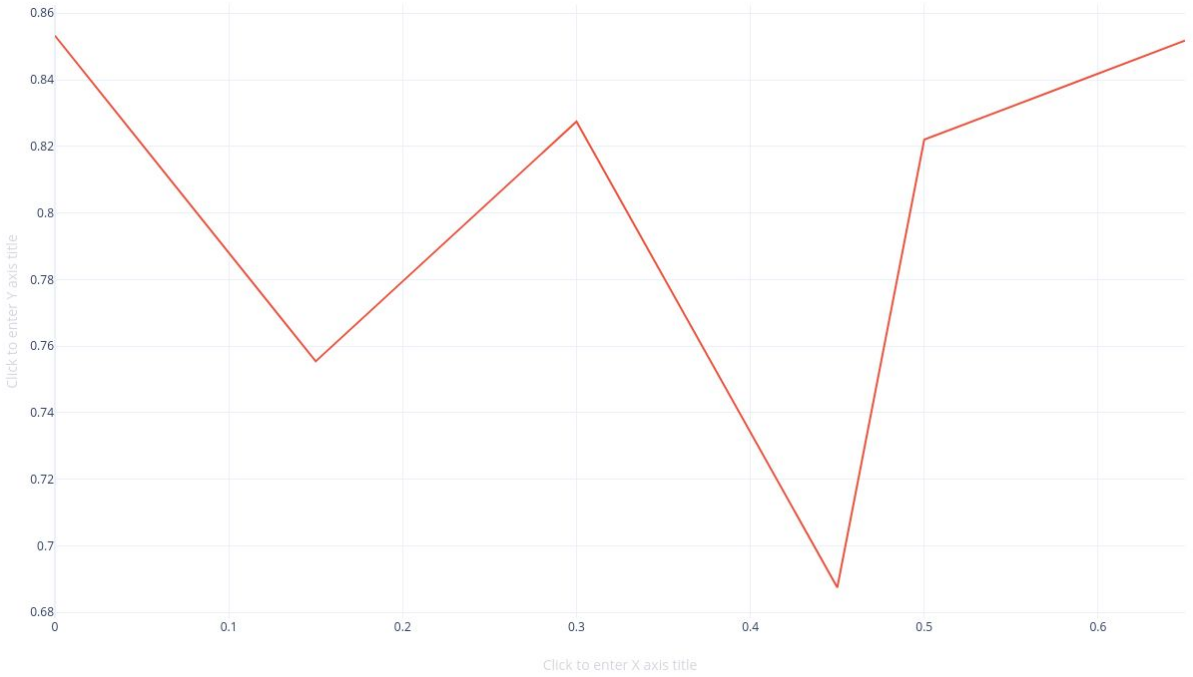
Για την εντολή εκτέλεσης:

./main -f sigmod_medium_labelled_dataset.csv -n on -v abs -b tf-idf -m 1000

(x : Threshold Value, y : Precision for Positive relations)

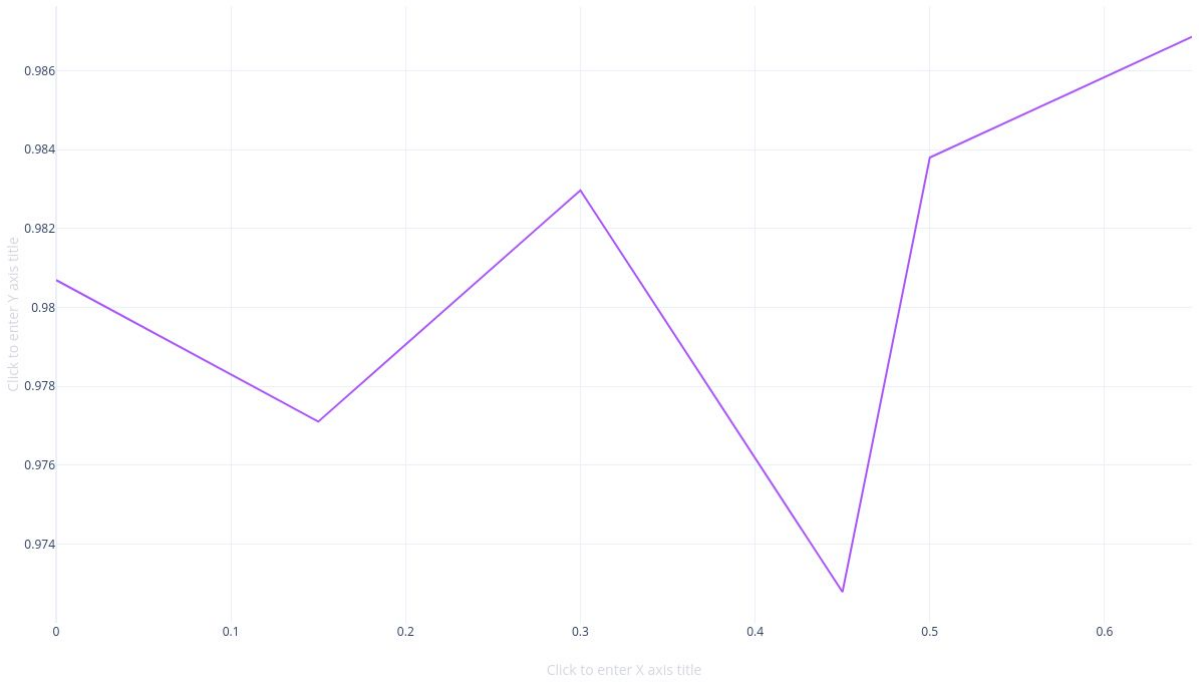


(x : Threshold Value, y : Recall for Positive relations)

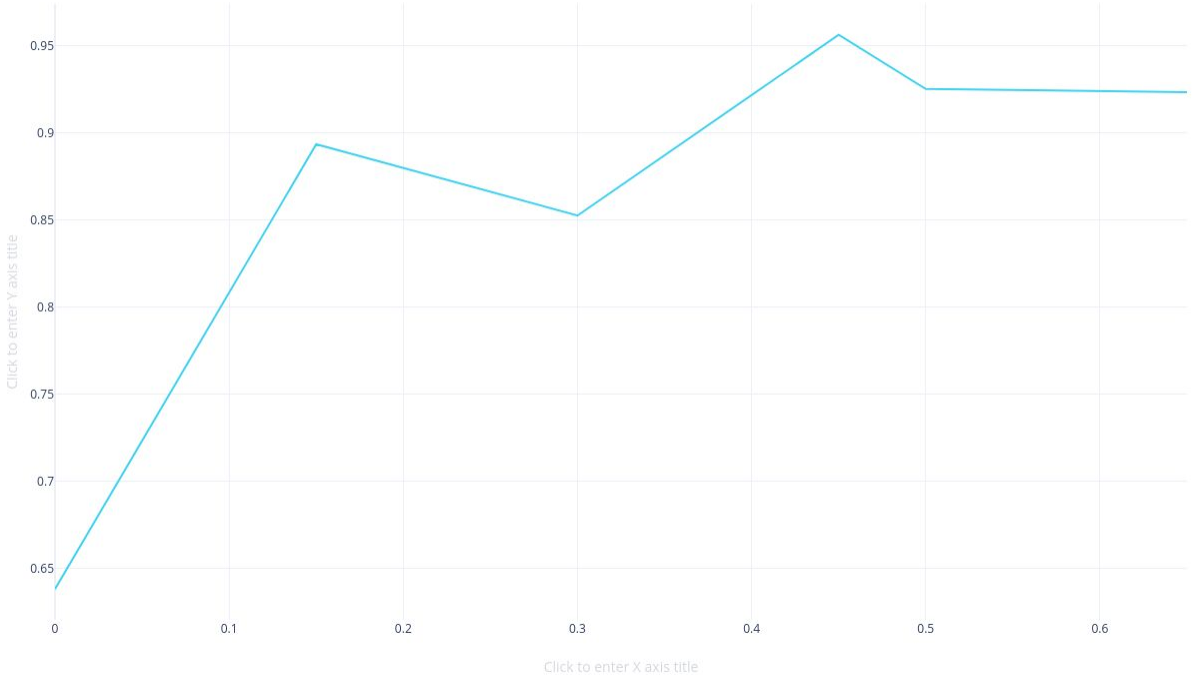


(x : Threshold Value, y : F1 score for Positive relations)

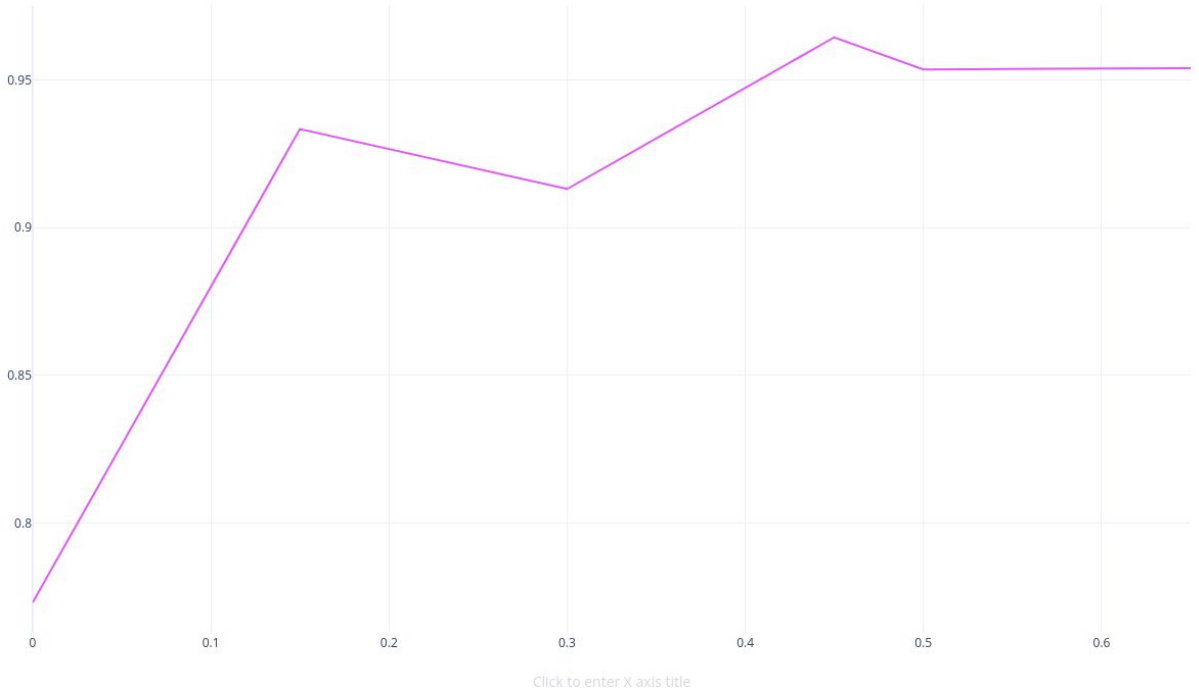
(x: Threshold Value, y : Precision for Negative relations)



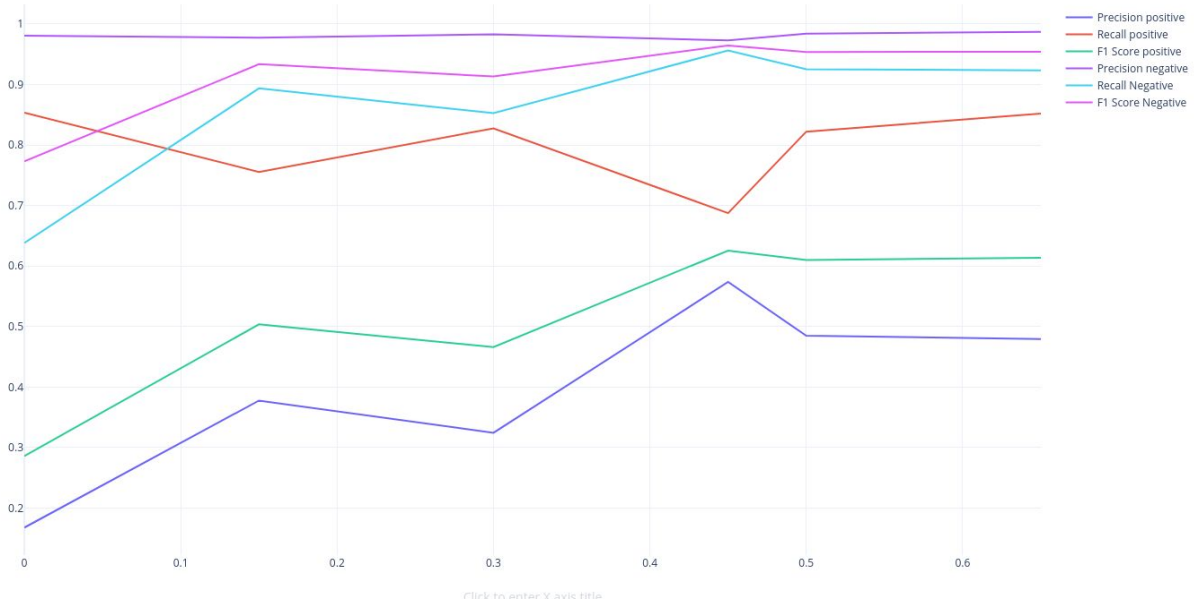
(x: Threshold Value, y : Recall for Negative relations)



(x: Threshold Value, y : F1 Score for Negative relations)



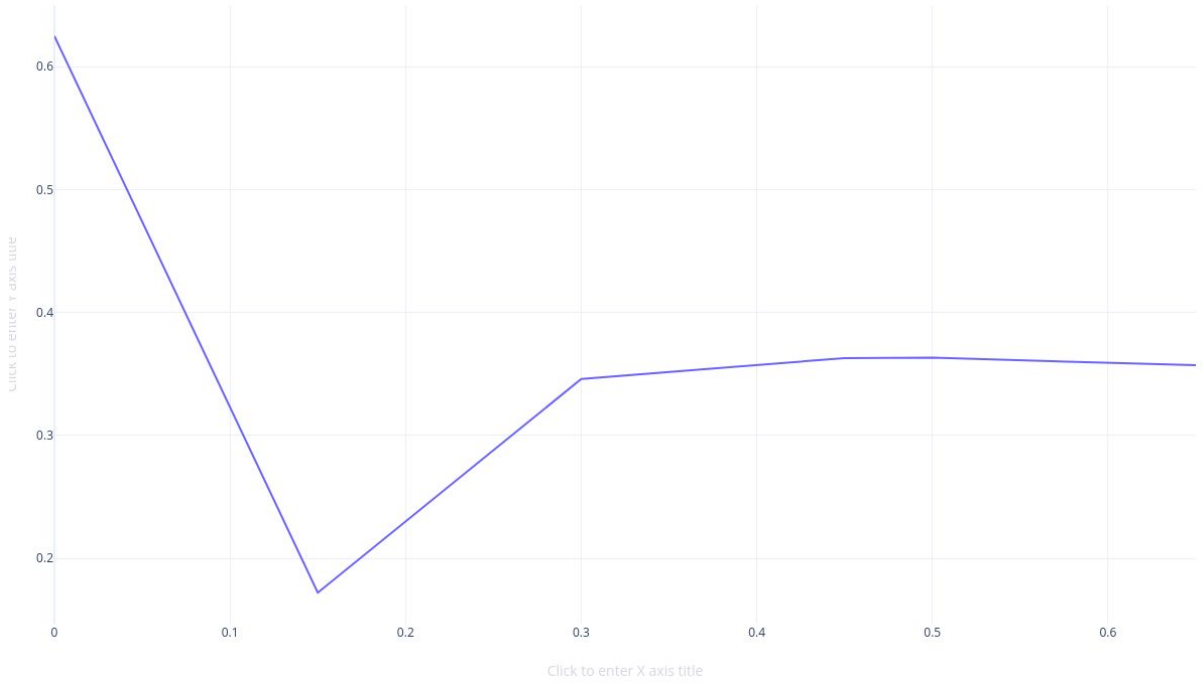
Όλα μαζί:



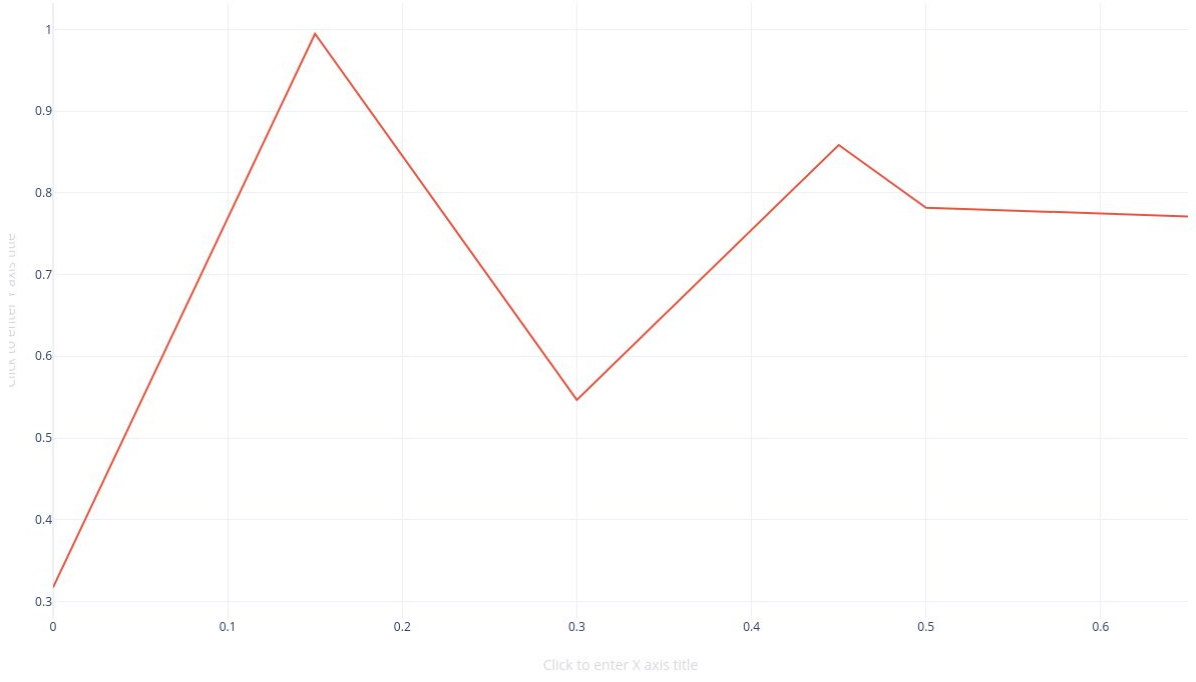
Εκτέλεση προγράμματος με την εντολή:

./main -f sigmod_large_labelled_dataset.csv -n on -v concat -b tf-idf -m 1000

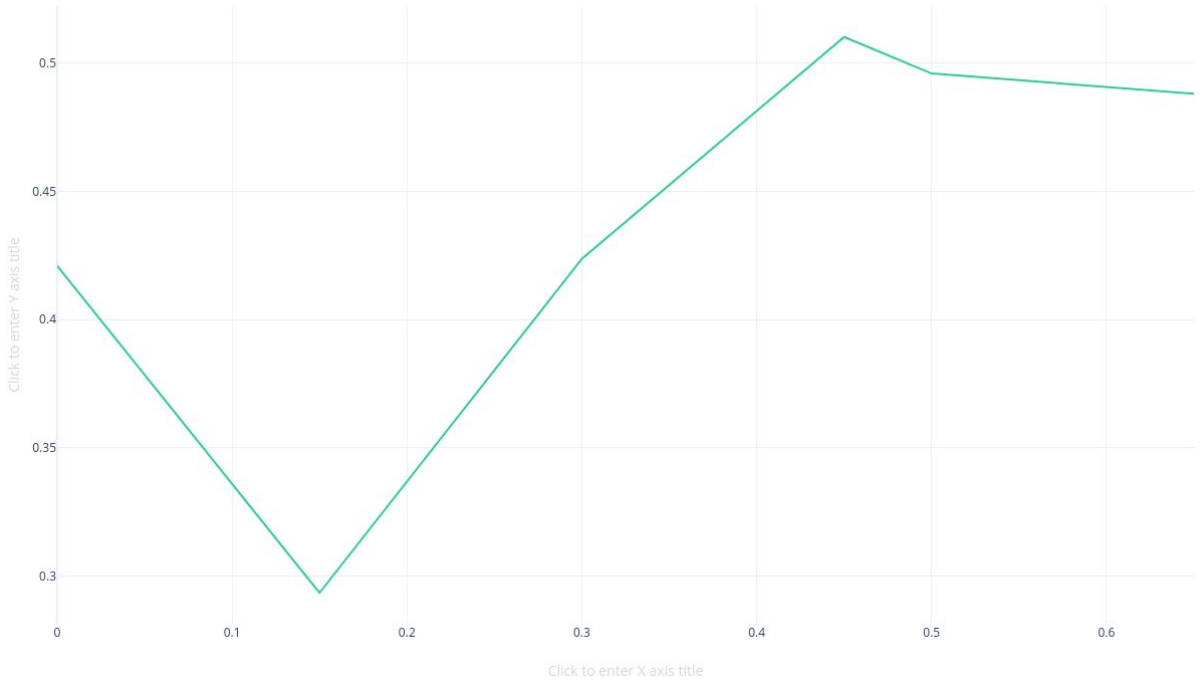
(x : Threshold Value, y : Precision for Positive relations)



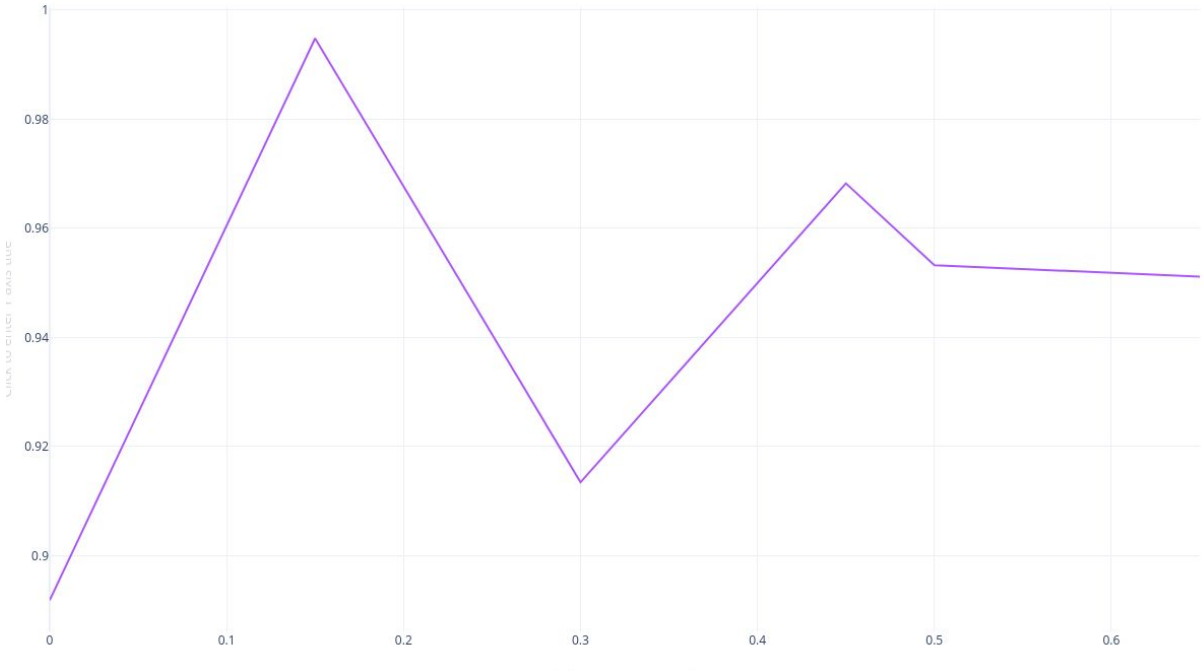
(x : Threshold Value, y : Recall for Positive relations)



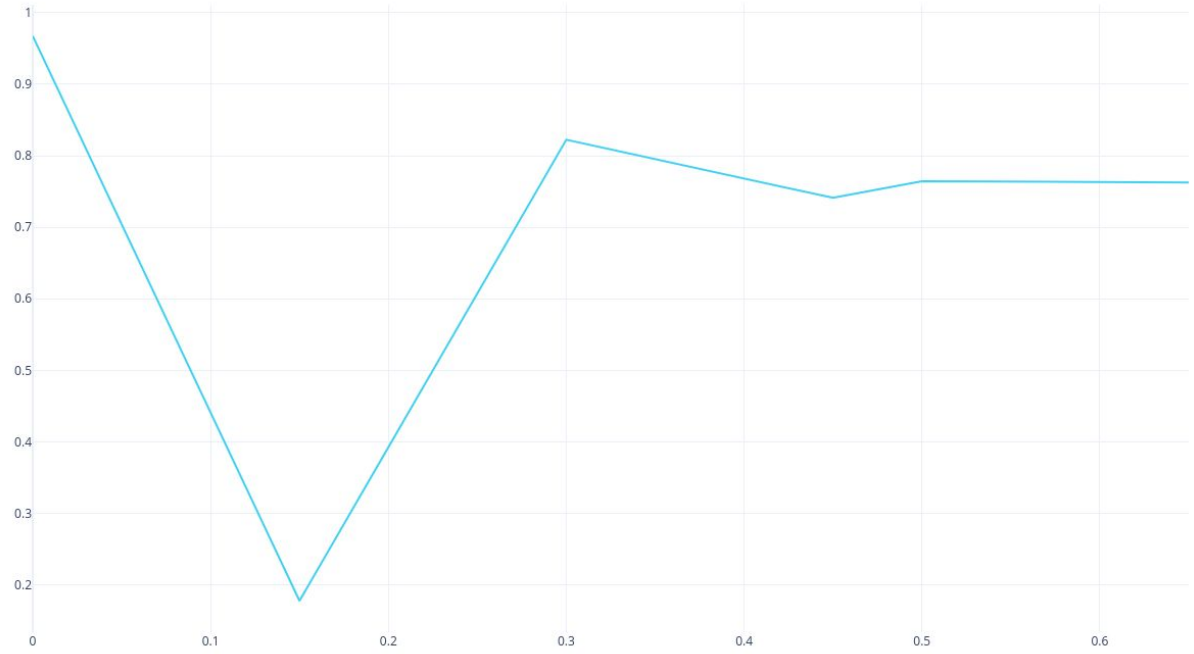
(x : Threshold Value, y : F1 score for Positive relations)



(x: Threshold Value, y : Precision for Negative relations)

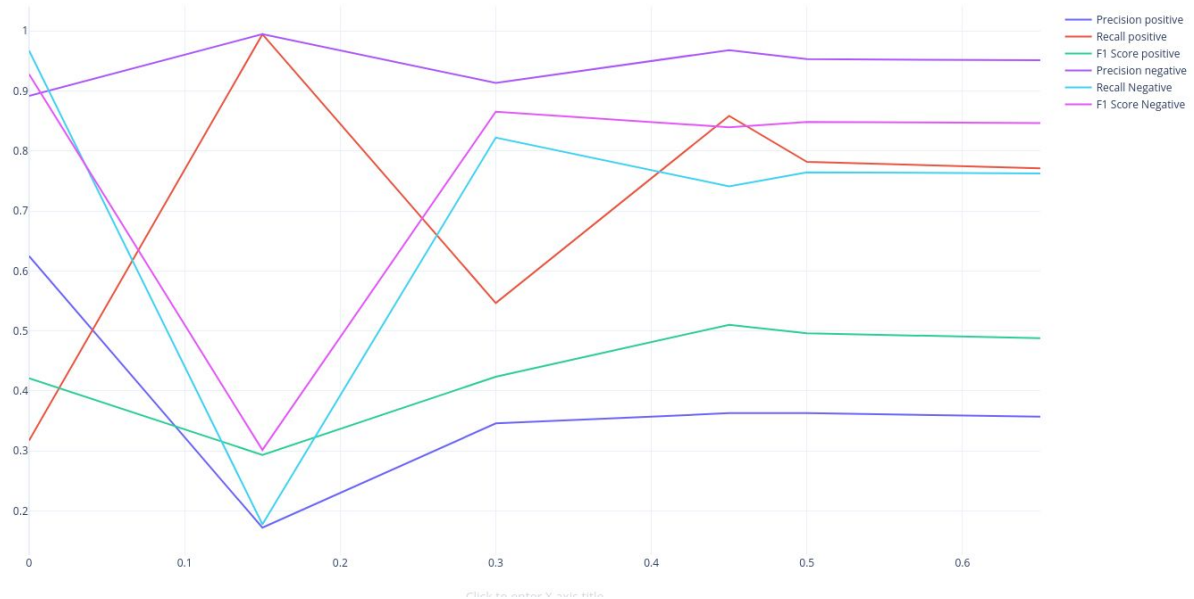


(x: Threshold Value, y : Recall for Negative relations)



(x: Threshold Value, y : F1 Score for Negative relations)

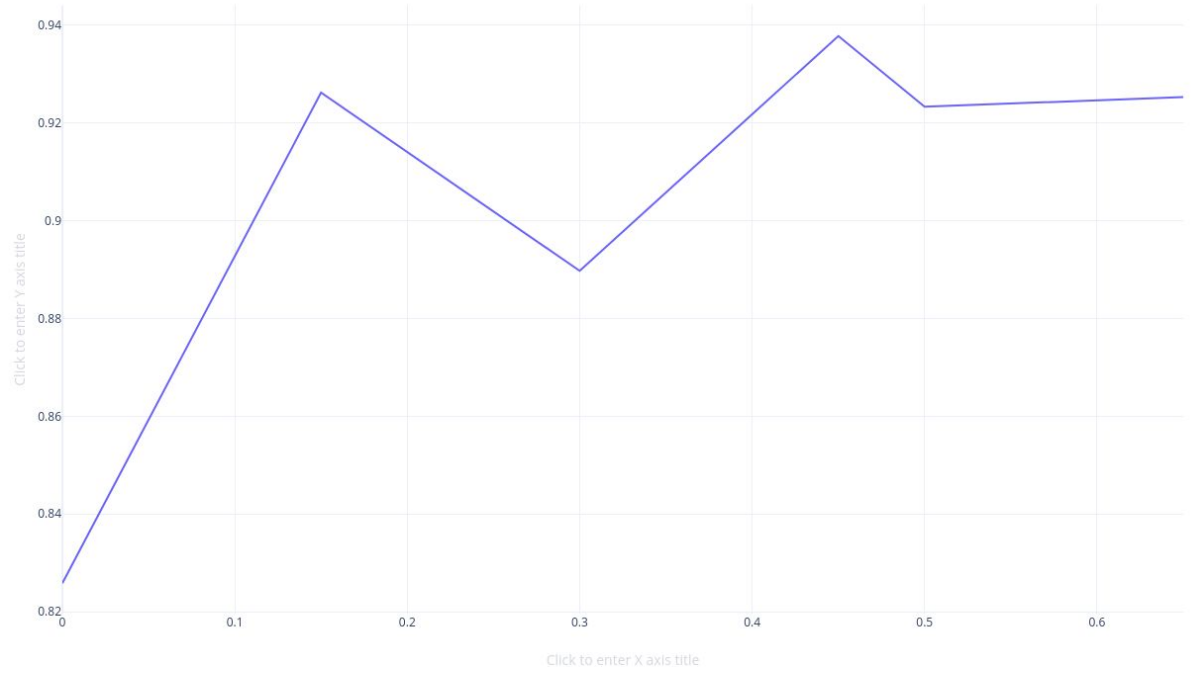
Όλα μαζί:



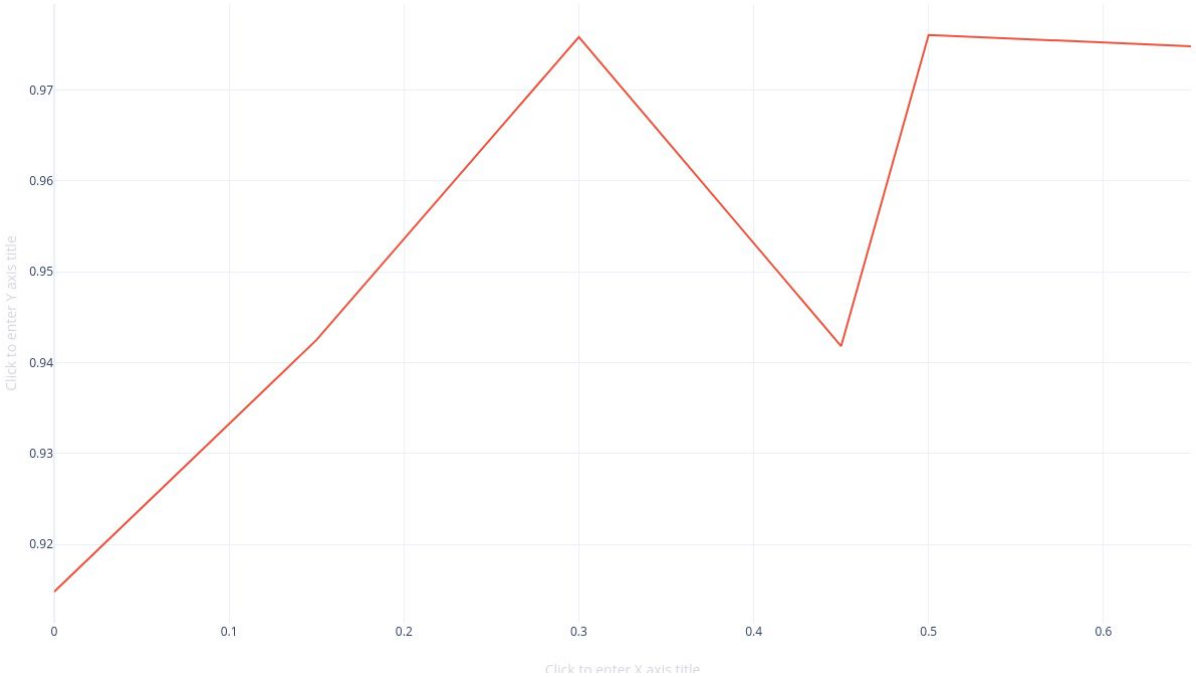
Εκτέλεση του προγράμματος με την εντολή:

./main -f sigmod_large_labelled_dataset.csv -v abs -b bow -m 1000 -n on

(x : Threshold Value, y : Precision for Positive relations)

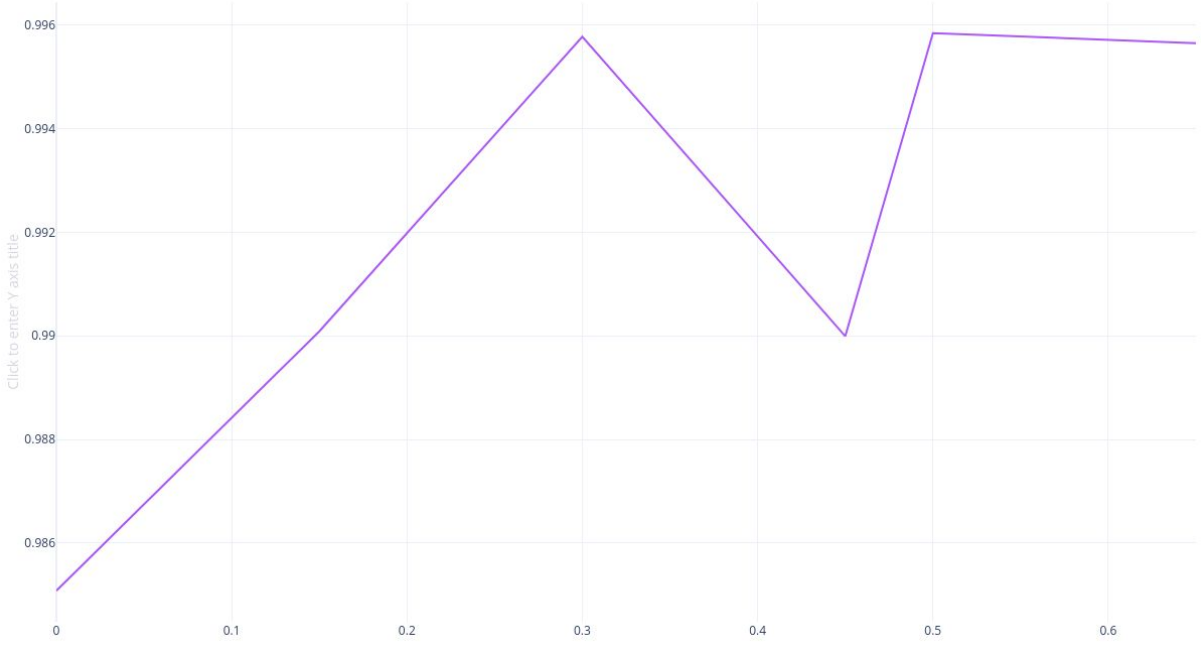


(x : Threshold Value, y : Recall for Positive relations)

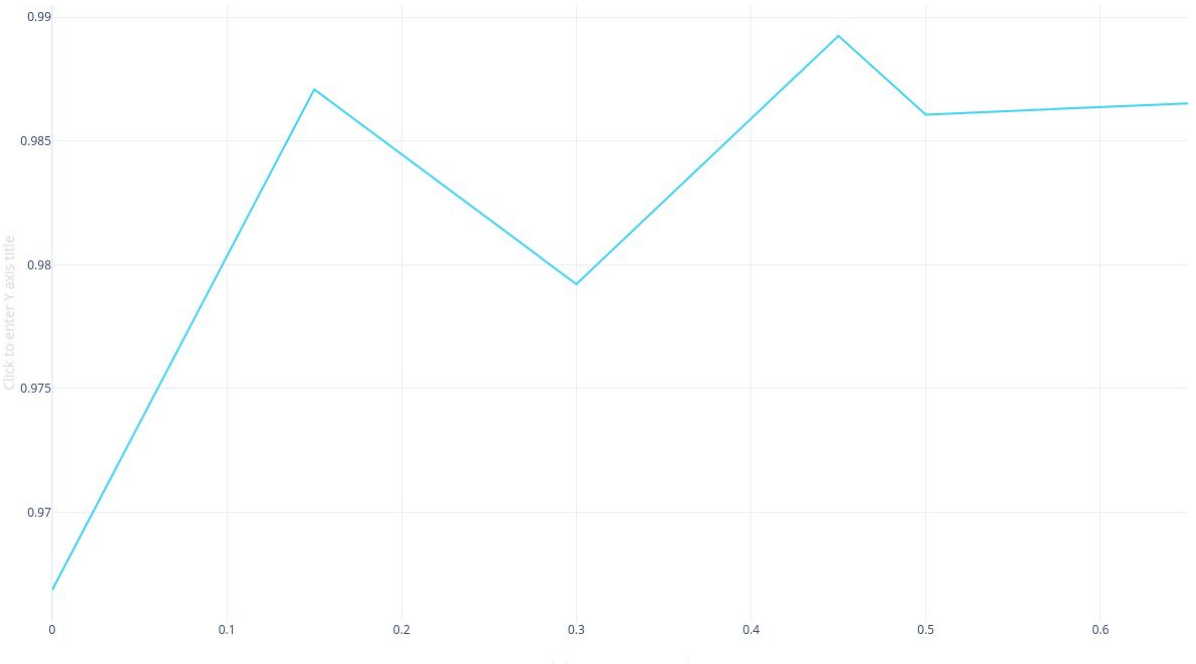


(x : Threshold Value, y : F1 score for Positive relations)

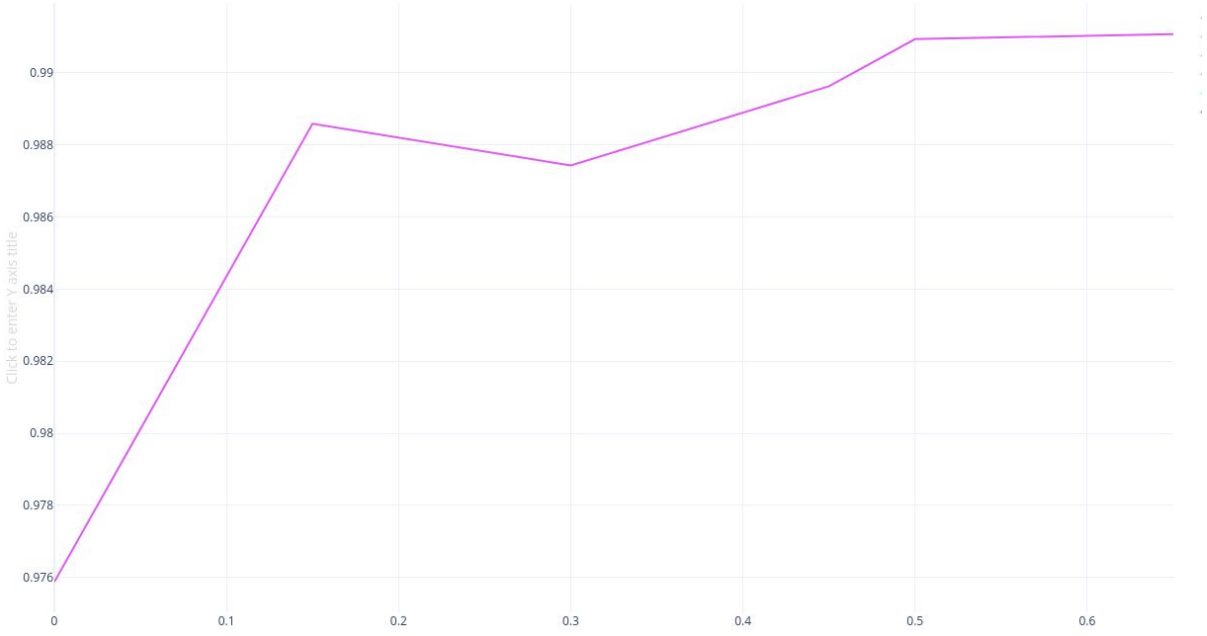
(x: Threshold Value, y : Precision for Negative relations)



(x: Threshold Value, y : Recall for Negative relations)



(x: Threshold Value, y : F1 Score for Negative relations)

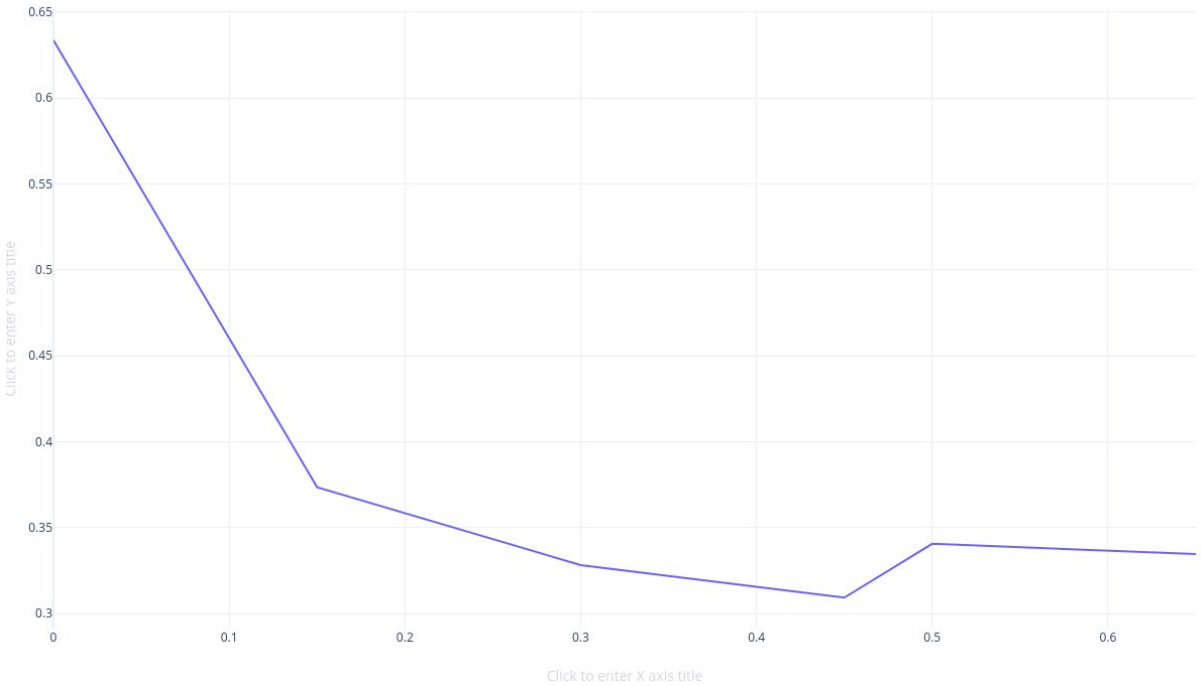


Όλα μαζί:

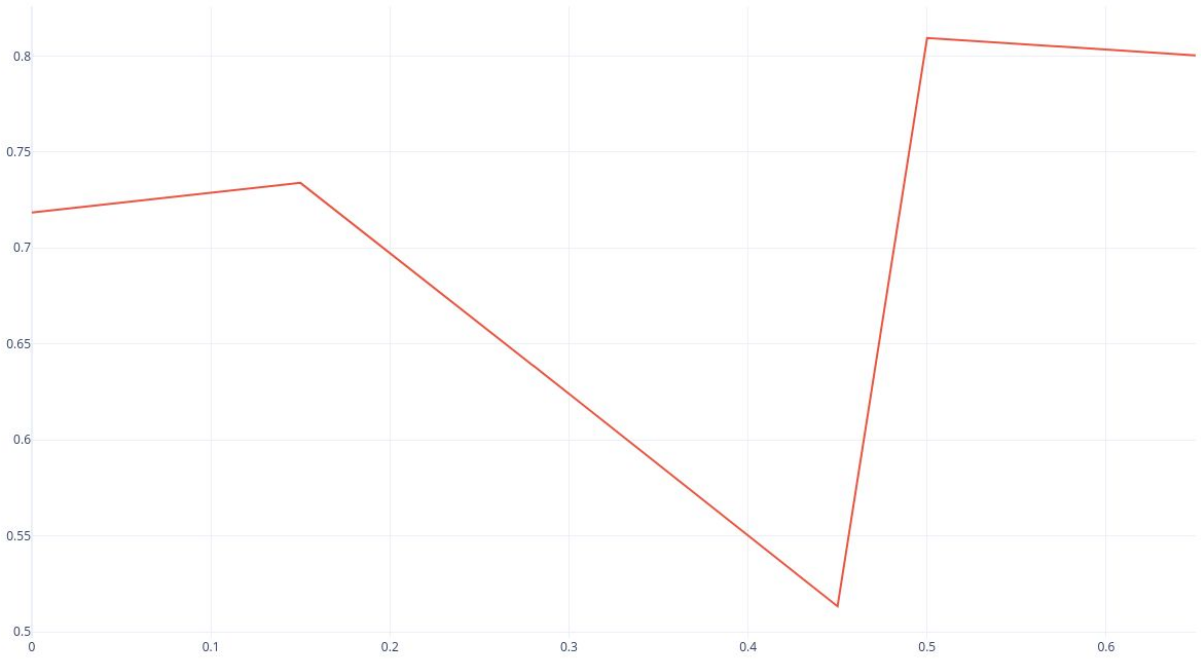
Εκτέλεση του προγράμματος με την εντολή:

./main -f sigmod_large_labelled_dataset.csv -v concat -b bow -m 1000 -n on

(x : Threshold Value, y : Precision for Positive relations)

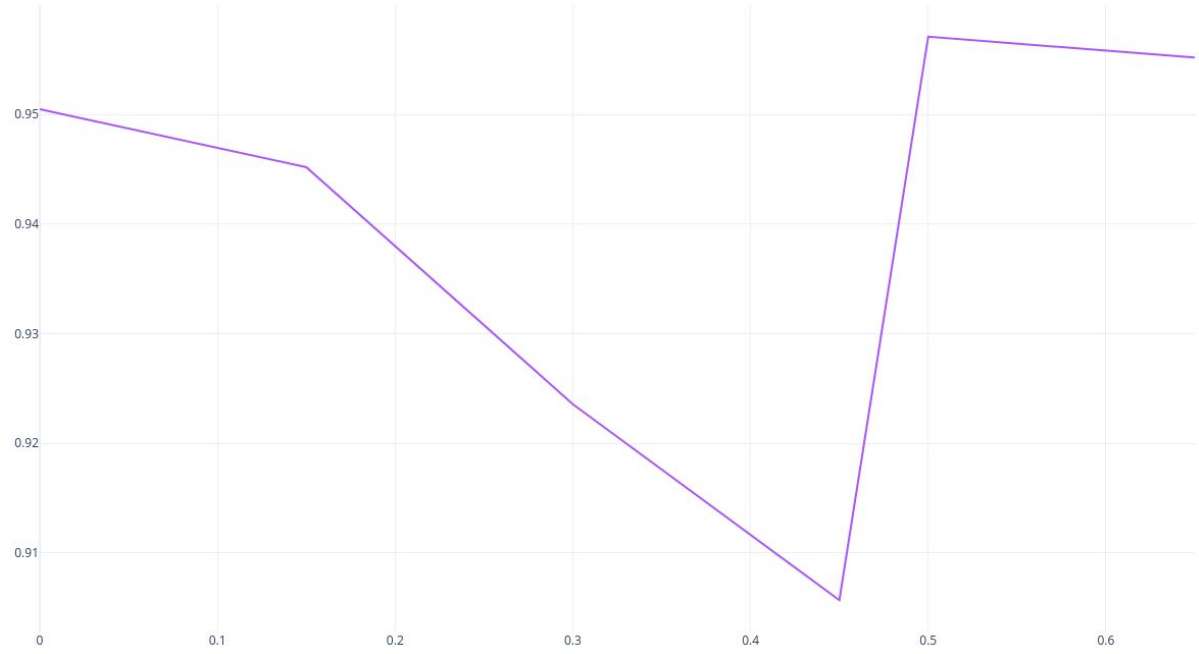


(x : Threshold Value, y : Recall for Positive relations)

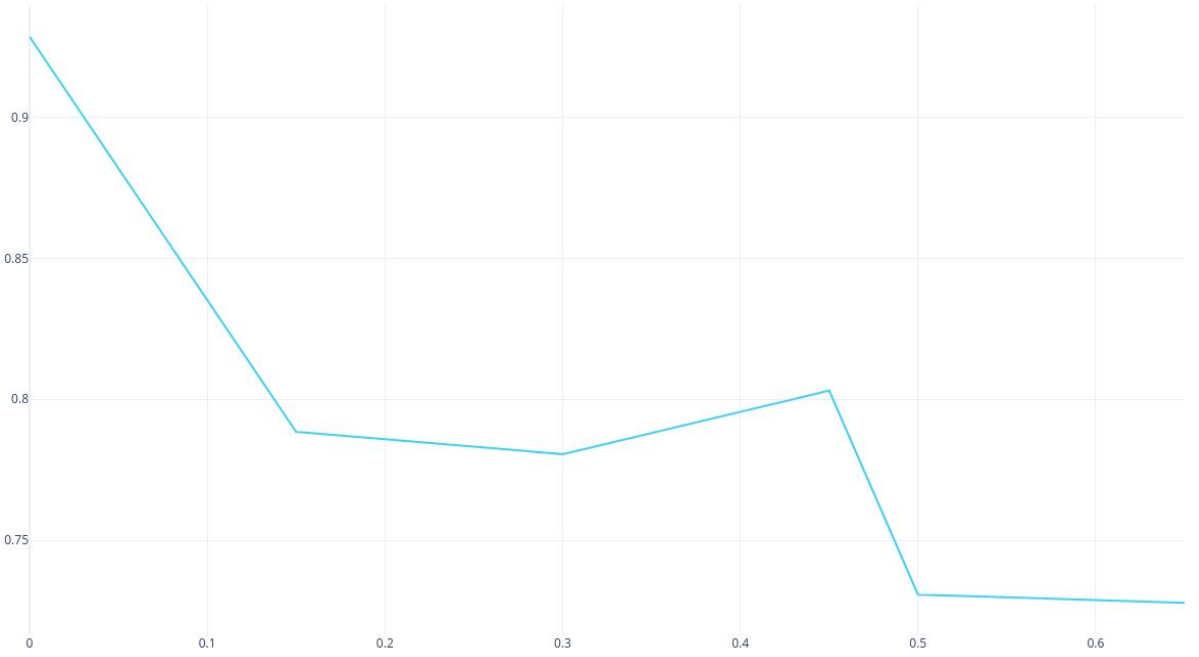


(x : Threshold Value, y : F1 score for Positive relations)

(x: Threshold Value, y : Precision for Negative relations)

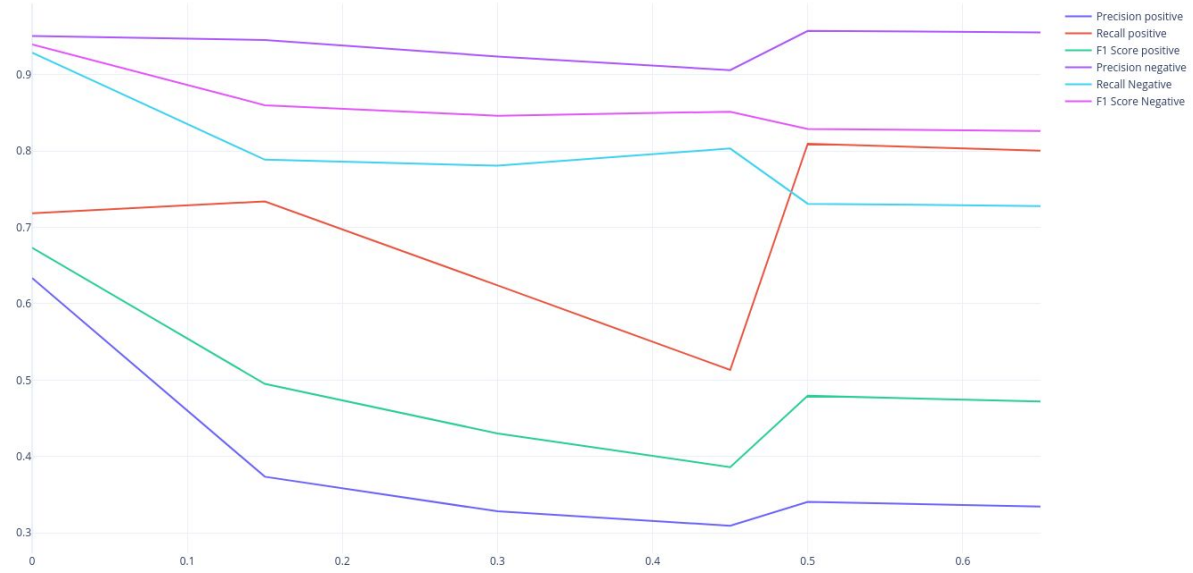


(x: Threshold Value, y : Recall for Negative relations)

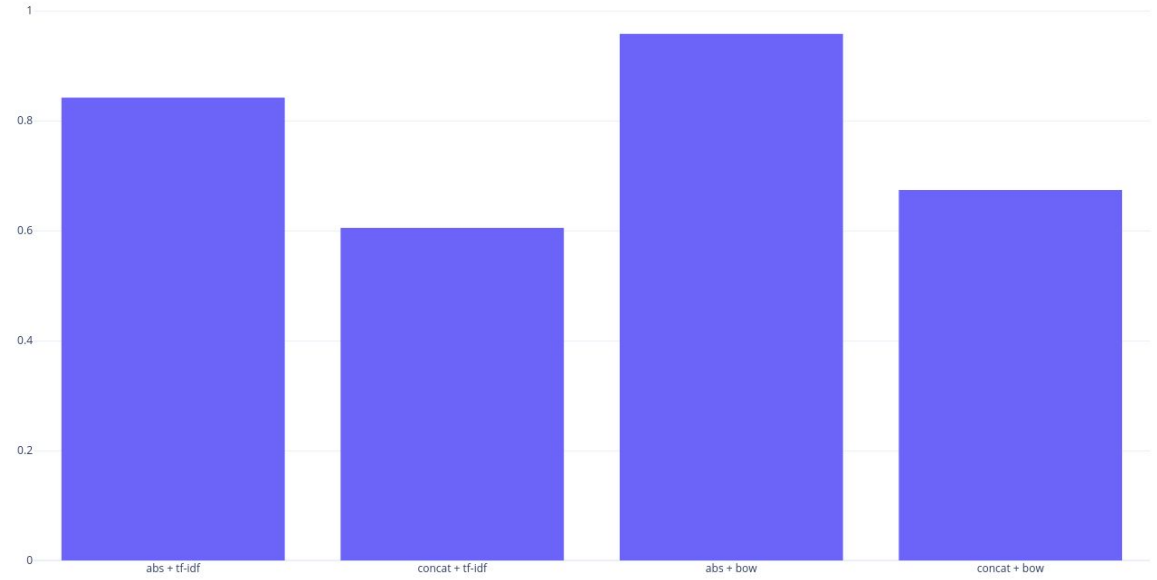


(x: Threshold Value, y : F1 Score for Negative relations)

Όλα μαζί:



Καταγράφοντας τον Μέσο όρο των F1 Score για τις διαφορετικές εκτελέσεις έχουμε ότι:



οπότε παρατηρείται ότι η καλύτερη κλήση για την εκτέλεση της main είναι :

./main -f sigmod_large_labelled_dataset.csv -v abs -b bow -m 1000 -n on

Τέλος ο Valgrind για την εκτέλεση με την εντολή :

./main -f sigmod_large_labelled_dataset.csv -n on -v abs -b tf-idf -m 10

```
==28096==
==28096== HEAP SUMMARY:
==28096==     in use at exit: 0 bytes in 0 blocks
==28096==   total heap usage: 64,732,280 allocs, 64,732,280 frees, 8,086,378,728 bytes allocated
==28096==
==28096== All heap blocks were freed -- no leaks are possible
==28096==
==28096== For lists of detected and suppressed errors, rerun with: -s
==28096== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Δεν σημειώνεται κανένα σφάλμα

και για το:

./inference -c sigmod_medium_labelled_dataset.csv

```
==31795==
==31795== HEAP SUMMARY:
==31795==     in use at exit: 0 bytes in 0 blocks
==31795==   total heap usage: 5,083,028 allocs, 5,083,028 frees, 1,321,323,286 bytes allocated
==31795==
==31795== All heap blocks were freed -- no leaks are possible
==31795==
==31795== For lists of detected and suppressed errors, rerun with: -s
==31795== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Τελικές παρατηρήσεις και σημειώσεις

Αναλυτικότερες λεπτομέρειες πάνω στην υλοποίηση ξεχωριστά κάθε συνάρτησης σημειώνεται στο ReadMe.md, που περιέχεται στο github. Όπως και τρόποι compile και εκτέλεσης του κώδικα.