ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ

Ανάπτυξη Λογισμικού Για Πληροφοριακά Συστήματα
Χειμερινό Εξάμηνο 2020 – 2021
Καθηγητής Ι. Ιωαννίδης

Βαρώνος Διονύσης A.M:1115201600017
Βουρτζούμη Ουρανία A.M:1115201600024
Κοσμάς Αλέξανδρος A.M:1115201700299

ΠΕΡΙΕΧΟΜΕΝΑ

Παρουσίαση της Εργασίας

Το θέμα με το οποίο ασχοληθήκαμε σε αυτή την εργασία ήταν η δημιουργία δομών για την αποθήκευση και διαχείρηση δεδομέων και η ανάπτυξη ενός μοντέλου μηχανικής μάθησης για την αξιολόγηση τους.

Αρχικός στόχος μας ήταν να διαμορφώσουμε και να υλοποιήσουμε τις κατάλληλες δομές που θα καθιστούν δυνατή, εύκολη και γρήγορη την αρχειοθέτηση ενός μεγάλου όγκου δεδομένων, την διαχείρησή τους, καθώς επίσης και μια στοιχειώδη ανάλυση και εξαγωγή πληροφοριών από αυτά. Πιό συγκεκριμένα, τη δημιουργία συσχετίσεων μεταξύ τους και την οργάνωσή τους βάσει αυτών.

Αυτή η διαδικασία χωρίζεται σε τρία βασικά στάδια.

Στο πρώτο στάδιο, αρχικά, κληθήκαμε να διαχειριστούμε έναν μεγάλο όγκο πληροφοριών σχετικά με φωτογραφικές μηχανές από διάφορους ιστιτόπους. Μέρος αυτής της διαχείρισης ήταν η επεξεργασία των αρχείων με τα δεδομένα κάθε μηχανής και η μετατροπή τους σε μορφές κατάλληλες για την μετέπειτα επεξεργασία τους. Έπειτα, σχεδιάσαμε δομές ικανές να αναπαραστήσουν τον γράφο με κλίκες συσχετίσεων ο οποίος προέκυψε από την παραπάνω επεξεργασία των δεδομένων με βάση τις μεταξύ τους θετικές συσχετίσεις.

Στο επόμενο στάδιο, η υλοποίησή μας συμπληρώθηκε από την εισαγωγή και των αρνητικών συσχετίσεων μεταξύ των εγγραφων στις ήδη διαμορφομένες δομές με τις θετικές συσχετίσεις. Η διαδικασία αυτή απαιτούσε υλοποίηση ξεχωριστών δομών για την αποθήκευση των επιπλέν πληροφοριών και αναδιάταξη και επενασχεδιασμό των δομών που είχαμε μέχρι εκείνο το σημείο.

Πέραν αυτού και αφού είχαμε συγκεντρωμένες τις δομές και τα πλήρη δεδομένα με τα οποία θα δουλεύαμε, ξεκινήσαμε την ανάπτυξη του μοντέλου μηχαινκής μάθησης. Πιο συγκεκριμένα, πρόκειται για έναν δυαδικό ταξινομητή που παράγει προβλέψεις-αποτελέσματα για κάποια στιγμιότυπα και τα ταξινομεί σε δύο κλάσεις. Η χρήση του στην συγκεκριμένη εφαρμογή είναι η αξιολόγηση ζευγών προϊόντων ως προς το αν ανήκουν ή όχι στην ίδια κλίκα (θετικών συσχετίσεων).

Τέλος, στο τελευταίο στάδιο, το βασικό ζήτημα ήταν η εισαγωγή πολυνηματισμού με σκοπό την εξοικονόμηση χρόνου. Επιπλέον, εφόσον η επεξεργασία των δεδομένων γίνεται με αυότν τον τρόπο πιό γρήγορη, μας επιτρέπεται να υλοποιήσουμε και πιο πολύπλοκες και , ενδεχομένως, πιο

αποτελεσματικές λειτουργίες που πριν ήταν ανέφυκτες καθώς απαιτούσαν πολύ χρόνο.

Παρουσίαση Δεδομένων

Τα δεδομένα με τα οποία ασχοληθήκαμε ήταν ένα dataset μεγέθους περίπου 30.000 κωδικών προϊόντων, που αντιστοιχούν σε φωτογραφικών μηχανών, τα οποία προέρχονταν από 24 διαφορετικούς ιστοτόπους (ebay, alibaba, walmart κ.α.) JSON μορφής. Κάθε JSON αρχείο περιέχει το όνομα της σελίδας η οποία αντιστοιχεί σε κάθε κάμερα και κατηγορίες με πληροφορίες για τα χαρακτηριστικά της. Το πεδίο “page title” είναι κοινό. Πέραν αυτού, κάθε κάμερα μπορεί να περιέχει διαφορετικές κατηγορίες, ή διαφορετική διατύπωση για την ίδια κατηγορία, με τρόπο συνήθως κοινό από ιστότοπο σε ιστότοπο.

Πέρα από αυτό το dataset, είχαμε διαθέσιμα και δύο ακόμα αρχεία .csv διαφορετικών μεγεθών (το sigmod_large_labelled_dataset.csv 297651 και το sigmod_medium_labelled_dataset 46665 καταχωρήσεων). Αυτά αποτελούνται από δύο στήλες κωδικών καμερών και σε μια τρίτη στήλη υπάρχει η πληροφορία αν το ζευγάρι της κάθε σειράς είναι όμοιο ή όχι (1 ή 0).

Για τα προϊόντα ισχύει η μεταβατική ιδιότητα, δηλαδή, αν κάποια κάμερα είναι ίδια με κάποια αλλη τότε αυτό ισχύει και για τις υπόλοιπες ίδιες κάμερες αυτών των δύο. Αντίστοιχα για τις αρνητικές συσχετίσεις.

Ακόμα, υπάρχουν και προϊόντα τα οποία δεν έχουν κάποια συσχέτιση μεταξύ τους, είτε θετική είτε αρνητική.

Προετοιμασία Δεδομένων

Για την ευκολότερη διαχείρηση των παραπάνω δεδομένων, ήταν απαραίτητη κάποια επεξεργασία τους.

Γίνεται προσπέλαση κάθε JSON και το περιεχόμενο του διαβάζεται γραμμή-γραμμή ανα 250 χαρακτήρες, τα οποία χωρίζονται σε token-λέξεις ανάλογα με τα σημεία στήξης. Οι ετικέτες-κατηγορίες κάθε προϊόντος παραλλείπονται, καθώς πολλές θα είναι κοινές μεταξύ των αρχείων αλλά δεν προσφέρουν χρήσιμες πληροφορίες όσον αφορά την ταυτότητα των προϊόντων. Χαρακτηριστικά, τα αποτελέσματα που πήραμε πριν αποφασίσουμε να μην περιλάβουμε τις κατηγορίες κάθε κάμερας ήταν πολύ λιγότερο ακριβή απότι τα τελικά μας αποτελέσματα, καθώς το μοντέλο έβρισκε θετικές συσχετίσεις μεταξύ προϊόντων που δεν ήταν όμοια λόγω των κοινών του πεδίων-κατηγοριών.

Ακόμα, κάθε λέξη η οποία προσμετράται ως περιεχόμενο του JSON ελέγχεται να έχει μέγεθος πάνω από 2 χαρακτήρες. Αυτό γίνεται για την παράλειψη αριθμών οι οπόιοι δεν αποτελούν αριθμούς μοντέλον (καθώς αυτοί ειναι μεγαλύτεροι σε μέγεθος), και κάποιων άρθρων ή/και κομματιών από από λέξεις ή γραμμάτων που δεν προσφέρουν καμία αξιόλογη πληροφορία. Επιπλέον, για την ομοιομορφία μεταξύ του περιεχομένου κάθε αρχείου, όλοι οι χαρακτήρες μετατρέπονται σε πεζά.

Τέλος, γίνεται σύκριση των υποψήφιων για περιεχόμενο λέξεων με το περιεχόμενο ενός αρχείου common.txt, το οποοίο περιέχει συχνούς συνδέσμους, άρθρα, ρηματικούς τύπους και άλλες λέξεις της αγγλικής γλώσσας που δεν προσφέρουν σημαντική πληροφορία για την ταυτότητα των προϊόντων.

Στο τέλος αυτής της διαδικασίας μένουμε με ένα σύνολο σημαντικών λέξεων για κάθε JSON αρχείο. Σε πρώτη φάση, αυτό αποθηκευόταν ως είχαν σε μια δομή για την αναπαράσταση κάθε κάμερας, αργότερα όμως, που προέκυψε η ανάγκη μετατροπής των προίόντων σε διανύσαματα και για οικονομία χώρου, επανασχεδιάσαμε την υλοποίησή μας έτσι ώστε να μην αποθηκεύεται το κείμενο σαν κείμενο αλλα με τον εξής τρόπο.

Κάθε νέα λέξη που διαβάζεται, εισάγεται σε έναν πίνκα κατακερματισμού, ο οποίος, στο τέλος, θα περιέχει όλο το λεξιλόγιο που προκύπτει από το αρχείο με τις διαθέσιμες κάμερες από όλους τους ιστοτόπους. Εκεί, περιέχεται ένας μετριτής που μετράει σε πόσα JSON αρχεία έχει βρεθεί κάθε μία λέξη.

Ταυτόχρονα, σε μία αλλη δομή αποθηκεύονται οι λέξεις που βρίσκονται σε κάθε JSON ξεχωριστά, με μετρητές για τηο πλήθος της κάθε λέξης μέσα στο κάθε αρχείο.

Μετά την ολοκλήρωση της ανάγνωσης κάθε JSON, από τις παραπάνω μετρήσεις υπολογίζεται για κάθε λέξη το tf (text frequency) της, δηλαδή, το κλάσμα εμφάνισης κάθε λέξης στο κείμενο προς τον συνολικό αριθμό λέξεων του κειμένου. Αυτός ο αριθμός στην σνέχεια αθροίζεται με το προηγούμενο tf της λέξης από τα προηγούμεν αρχεία και, ως εκ τούτου, τελικά έχουμε το συνολικό tf της κάθε λέξης για το σύνολο των δεδομένων μας.

Επόμενο στάδια στην προετοιμασία-μετατροπή των δεδομένων είναι ο υπολογισμός του μέσου tf-idf της κάθε λέξης, η απεικόνιση, δηλαδή, του λογάριθμου της συχνότητας εμφάνισης της λέξης στο σύνολο των αρχείων.

Τέλος, αυτά τα αποτελέσματα, για το συνολικό λεξιλόγιο, ταξινομούνται και κρατάμε τις 1000 σημαντικότερες λέξεις, αυτές με το μεγαλύτερο idf, αυτές που περέχουν σημαντικότερη πληροφορία για τα προϊόντα. Αυτό είναι το τελικό λεξιλόγιο που θα χρησιμοποιήσουμε για την αξιολόγηση στο μοντέλο μας.

Όσον αφορά στην αναπαράσταση του κειμένου κάθε κάμερας, μετά την διαμόρφωση του τελικού λεξιλογίου, γίνεται ή μετάφρασή του σε δίανυσμα. Κάθε λέξη του πίνακα λεξιλογίου κάθε JSON, σε περίπτωση που βρίσκεται στο τελικό λεξιλόγιο, εισάγεται σε μία νέα δομή, έναν ακόμα πίνακα κατακερματισμού. Δεν γίνεται εισαγωγή του κειμένου, αλλά της θέσης της λέξης στον πίνακα του λεξιλογίου (πληροφορία κοινή για κάθε λέξη σε για κάθε JSON), και η τιμή tf-idf της που υπήρχε στο λεξιλόγιο της συγκεκριμένης λέξης (μοναδική και ανεξάρτητη από τα άλλα αρχεία). Ο πίνακας αυτός είναι ένας sparse array, ο οποίος κρατάει πληροφορίες μόνο για τις λέξεις που υπάρχουν σε κάθε JSON, με αποτέλεσμα να είναι πιο “οικονομικός” από άποψη χώρου που καταλαμβάνει και χρόνου που χρειάζεται για την προσπέλασή του, καθώς δεν αποθηκεύεται άχρηστη πληροφορία (λέξεις που δεν ανήκουν στο κάθε αρχείο).

Αυτή η διαδικασία ακολουθείται σε κάθε αρχείο JSON και με την ολοκλήρωσή της διαγράφεται η προηγούμενη δομή που αποθηκευόταν το λεξιλόγιο του κάθε αρχείου ως κείμενο.

Στο τέλος, καταλήγουμε με έναν πίνακα χ θέσεων, που είναι η αναπαράσταση ενός διανύσματος 1000 θέσεων λεξιλογίου για κάθε αρχείο JSON των διαθέσιμων δεδομένων μας.

ΔΟΜΕΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

Η δομή Hash είναι η δομή που αποθηκευονται τα δεδομένα του κάθε json αρχείου. Είναι ένας δυναμικός πίνακας κατακερματισμου με bucket-list όπου ως κλειδί χρησιμοποιεί το id κάθε json (πχ www.ebay.com//567) και κάνει rehash κάθε φορά που φτάνει 80% πληρότητα. Για την υλοποίηση του bucket-list χρησιμοποιείται η δομή NList που σε κάθε κόμβο της αποθηκεύεται: το id του json στην μεταβλητή camera τύπου char* οι λέξεις του JSON αρχείου στην μεταβλητή spear που είναι

τύπου Whash* το αντίστοιχο vector του JSON που είναι τύπου Hvector* ένας δείκτης σε δομη CList που αντιστοιχεί στην κλικα την οποία ανήκει η camera. Η δομή CList είναι μια συνδεδεμένη λίστα που για την υλοποίηση των κλικων. Σε κάθε θέση της αποθηκεύει:

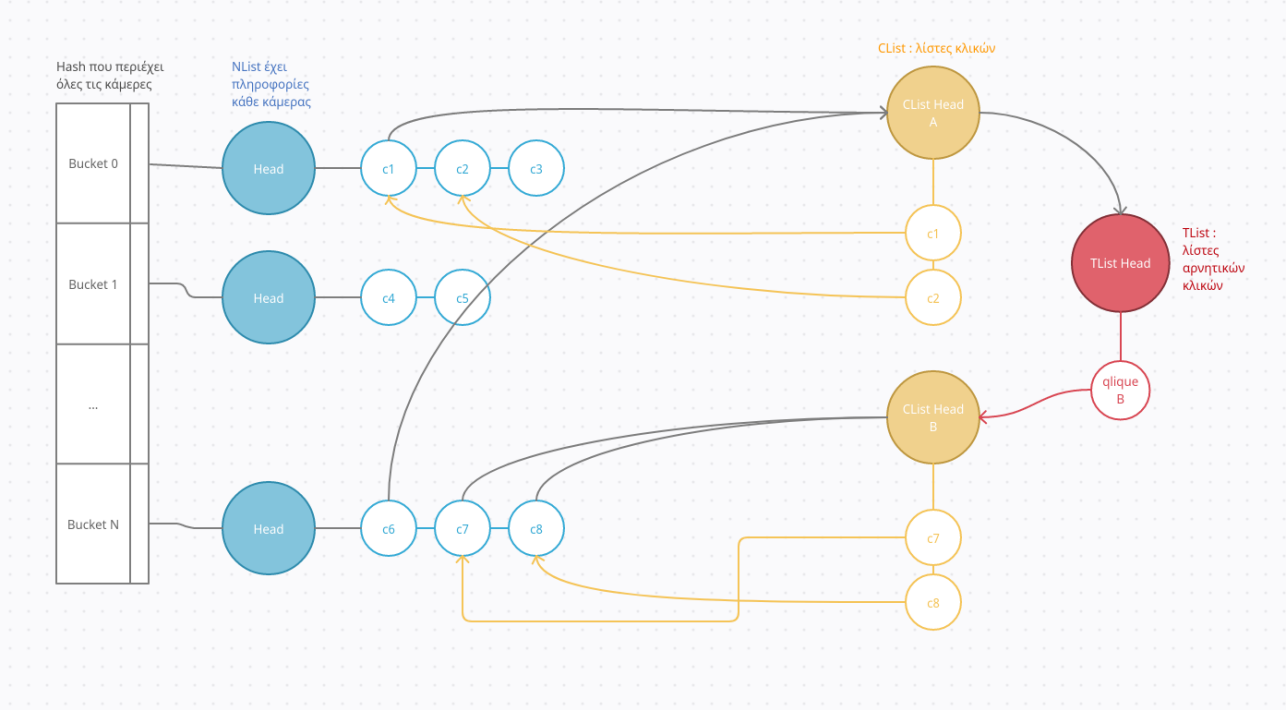
- το όνομα της κάμερας
- έναν δείκτη σε NList που αντιστοιχεί με τον κόμβο NList που είναι αποθηκευμένα τα στοιχεία της κάμερας.

Ο πρώτος κόμβος της κάθε CList δεν αποθηκεύει δεδομένα μιας κάμερας αλλά μια λίστα Tlist (συνδεδεμένη λίστα που αποθηκεύει δείκτες σε CList) που αποθηκεύει τις κλικες με τις οποίες δεν τεροιαζει η κλικα.

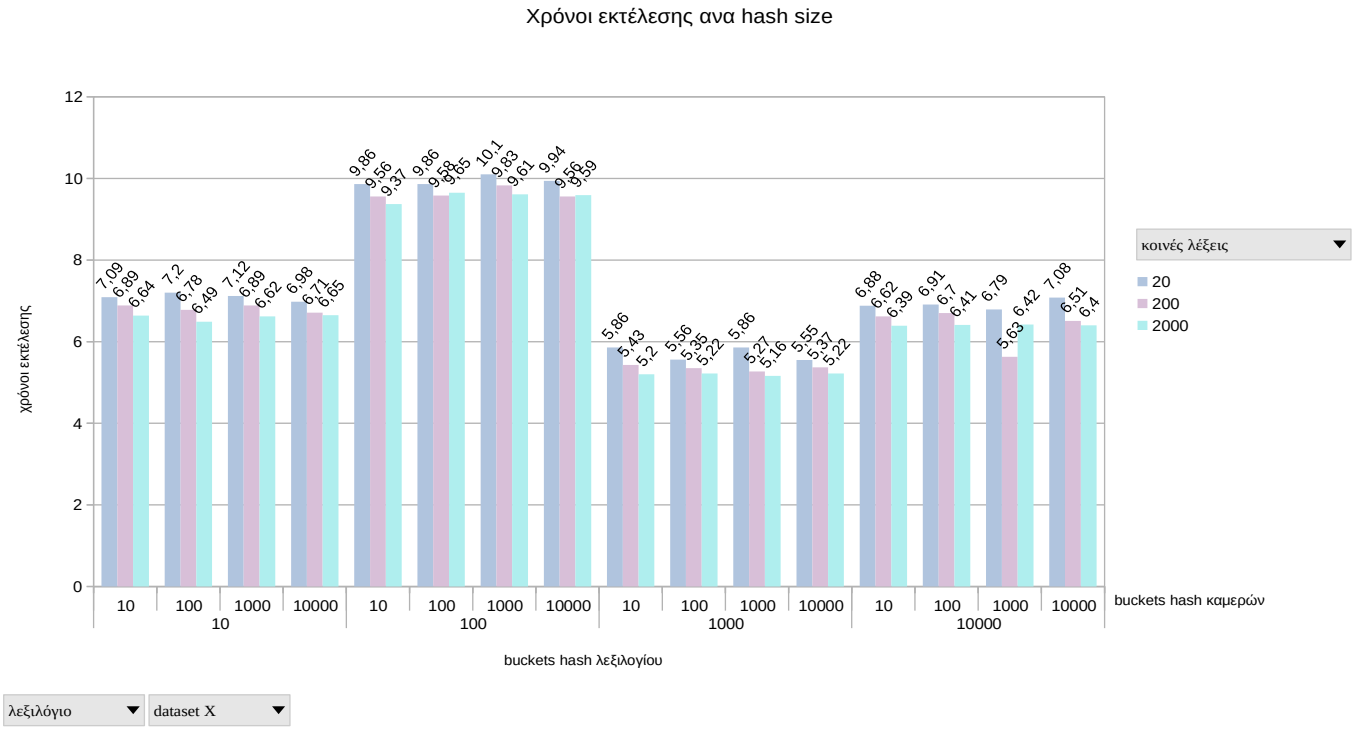
Η δομή WHash είναι μια δομή Πίνακα κατακερματισμου χωρίς bucket-list που αποφεύγει τα collision πηγαίνοντας στην επόμενη διαθέσιμη κενή θέση και κάνει rehash όταν έχει 80% πληρότητα. Η δομή αυτή χρησιμοποιείται για να αποθηκεύει τις λέξης του κάθε json αρχείου με κλειδί την κάθε λέξη. Σε κάθε bucket αποθηκεύει μια λέξη και το tf αυτής της λέξης για το συγκεκριμένο json.

Η δομή Hvector είναι μια δομή πίνακα κατακερματισμου χωρίς bucket-list που αποφεύγει τα collision πηγαίνοντας στην επόμενη διαθέσιμη κενή θέση και κάνει rehash όταν έχει 80% πληρότητα. Η δομή αυτή αναπαριστά το vector της κάθε κάμερας. Σε κάθε bucket αποθηκεύει την θέση και την tf*idf της αντίστοιχης θέσεις για κάθε λέξη απο τις 1000 πιο σημαντικές του λεξιλογίου ενα υπαρχει στην κάμερα.

Η δομή LHash είναι μια δομή Πίνακα κατακερματισμου χωρίς bucket-list που αποφεύγει τα collision πηγαίνοντας στην επόμενη διαθέσιμη κενή θέση και κάνει rehash όταν έχει 80% πληρότητα. Η δομή αυτή χρησιμοποιείται για την υλοποίηση του λεξιλογιου όλων των json με κλειδί την κάθε λέξη. Σε κάθε bucket αποθηκεύεται η κάθε λέξη το idf και το μέσω tf-idf της κάθε λέξης



Πειραματιστήκαμε με τα μεγέθη των πινάκων και πήραμε τα εξής αποτελέσματα, όσον αφορά τον χρόνο εκτέλεσης του προγράμματος :



Οι μικρότεροι χρόνοι εκτέλεσης έχουν προκύψει όταν ο hash που αποθηκεύει τις λέξεις του συνολικού λεξιλογίου έχει αριθμό bucket ίσο με 1000, ενώ οι χειρότεροι χρόνοι, όταν έχει 100. Οι διακυμάνσεις απο εκεί και πέρα είναι αρκετά μικρές, αλλά οι καλύτερες αποδόσεις υπάρχουν με hash αποθήκευσης των κοινών λέξεων μεταξύ του λεξιλογίου κάθε κάμερας και συνολικού λεξιλογίου μεγέθους 2000 buckets και hash αποθήκευσης καμερών μεγέθους 1000 bucket. Ο μικρότερος χρόνος που καταγράφηκε ήταν 5,16 sec.

Χωρισμός Train / Test / Validation Set

Ως προετοιμασία για την διαδικασία της μηχανικής μάθησης τα δεδομένα πρέπει να χωριστούν σε τρία διαφορετικά σετ.

Πρώτα, το train set, το οποίο θε αποτελείται από τα προϊόντα τα οποία θα αποτελέσουν την βάση εκπαίδευσης του μοντέλου, έπειτα, το test set, τα δεδομένα που θα χρησιμοποιηθούν για την αξιολόγηση της επίδοσης του μοντέλου και με βάση αυτά υπολογίζεται το ποσοτό επιτυχίας του μοντέλου μας, και , τέλος, το validation set, η ύπαρξη του οποίου χρησιμοποιείται γενικά για την σύγκριση μεταξύ διάφορων μοντέλων και υλοποιήσεωνκαι την επιλογή της καλύτερης μεταξύ αυτών. Σε αυτήν την εργασία, καθώς το είδος του αλγορίθμου ταξινόμησης είναι προκαθορισμένο, ενώ δημιουργείται ένα validation set, δεν χρησιμοποιείται σε κάποια αξιολόγηση, ούτε προσφέρει κάποια παραπάνω πληροφορία στην ανάπτυξη του μοντέλου.

Τα δεδομένα από τα οποία προέρχεται το περιεχόμενο αυτών των σετ είναι, ανάλογα με τον τρόπο που τρέχει κανείς το πρόγραμμα, ένα από τα δύο αρχεία με συσχετίσεις μεταξύ ζευγών καμερών, δηλαδή είτε το sigmod_large_labelled_dataset.csv είτε το sigmod_medium_labelled_dataset.

Ο χωρισμός των παραπάνω σετ έγινε αρχικά με τον εξής τρόπο.

Το σύνολο του αρχείου που έχει δωθεί σαν είσοδος στο πρόγραμμα χρησιμοποιείται για την δημιουργία κλικών θετικών και αρνητικών συσχετήσεων (σε δομές που θα συζητηθούν αργότερα). Από αυτές, επιλεγόταν το 60 (περίπου) τοις εκατό της κάθε κλίκας για το train set, το επόμενο 20 για το test και το τελευταίο 20 για το validation. Με αυτόν τον τρόπο εξασφαλιζόταν η ύπαρξη ομοιομορφίας στα δεδομένα εκπαίδευσης του μοντέλου, καθώς από κάθε υποσύνολο θετικά συσχετισμένων προϊόντων θα επιλεγόταν ανάλογος αριθμός με το πλήθος τους για να συμμετέχει στο train set.

Σε επόμενο στάδιο, μετά από οδηγία που δόθηκε στο μάθημα, η προσέγγισή μας άλλαξε.

Το αρχείο που δίνεται σαν είσοδος στο πρόγραμμα διαβάζεται μία μία γραμμή δύο φορές. Πρώτα για την μέτρηση των γραμμών και του αριθμού των θειικών ζευγών έτσι ώστε να υπολογιστύν σωστά των επι μέρους ποσοστά (60% των θετικών και 60% των αρνητικών συσχετήσεων) και, έπειτα, για την δημιουργία των προαναφερθέντων σετ. Από αυτά, το train set χρησιμοποιείται για την δημιουργία των κλικών με τις συσχετήσεις, ενώ το test set χρησιμοποιείται για την δημιουργία ενός νέου αρχείου Testing.csv για μετέπειτα χρήση.

Έπειτα, γίνεται προσπέλαση στις κλίκες που δημιουργήθηκαν και έτσι παράγονται δύο αρχεία, Sema.csv και Diffrend.csv, τα οποία περιέχουν όλα τα θετικά και όλα τα αρνητικά ζευγάρια που προέκυψαν. Η πληροφορία των δύο αυτών αρχείων δεν είναι ίδια αλλά συμπληρωματική του αρχικού αρχείου εισόδου, καθώς, με βάση αυτό, δημιουργήθηκαν και συσχετίσεις μεταξύ προϊόντων που πριν δεν αναφέρονταν καν ως ζευγάρια.

Υλοποίηση αλγορίθμου Μηχανικής Μάθησης

Ο αλγόριθμος που χρησιμοποιήθηκε για την ανάπτυξη του μοντέλου είναι ο αλγόριθμος της λογιστικής παλινδρόμισης (logistic regression), που, όπως προαναφέρθηκε χρησιμοποιείται για την ταξινόμηση δεδομένων σε δύο (εδώ) κλάσσεις.

Κάθε ζεύγος προϊόντων αναπαριστάται από ένα τελικό διάνυσμα, το οποίο προκύπτει από την συνένωση των αρχικών διανυσμάτων που αναπαριστούν κάθε προϊόν. Για την δημιουργία του γίνεται σύγκριση του κάθε διανύσματος και, ανάλογα με το μέγεθος του καθενός επιλέγεται πιο θα αποτελέσει το πρώτο και ποιό το δεύτερο κομμάτι του τελικού διανύσαμτος. Το μέγεθος εδώ αναφέρεται στο άθροισμα όλων των τιμών που αποτελούν το κάθε διάνυσμα.

Αυτό το διάνυσμα χρησιμοποιούνταν σε αρχικά στάδια υλοποίησης της εργασίας για τον υπολογισμό της πρόβλεψης του μοντέλου. Στην τελική μορφή της εργασίας,μόνο στο κομμάτι του testing όλου του συνόλου των δεδομένων μας (ολα-με-όλα), η πρόβλεψη γίνεται χωρίς να χρειαστεί να γίνει η παραπάνω διαδικασία. Πιο συγκεκριμένα, το “μέγεθος” του κάθε διανύσματος έχει υπολογιστεί από την στιγμή δημιουργίας του και αποθηκεύεται στα στοιχεία της δομής του κάθε διανύσματος (πεδίο sum της δομής HVector). Αυτό εξυπηρετεί καθώς, ενώ κάποιο προϊόν ανήκει σε πολλά ζεύγη, δεν χρειάζεται ο υπολογισμός να γίνεται κάθε φορα και η διαδικασία γίνεται πιο γρήγορη.

Αφού γίνει η σύγκριση, υπολογίζεται το γινόμενο του βάρους κάθε λέξης επί την τιμή της στο συγκεκριμένο διάνυσμα, για το συγκεκριμένο προϊόν. Τα γινόμενα για κάθε λέξη των δύο διανυσμάτων του ζεύγους αθροίζονται σε μία μεταβλητή sum. Τέλος, υπολογίζεται η σιγμοειδής συνάρτηση αυτής της τιμής ($p(\mathbf{x}) = \frac{1}{1+e^{-f(x)}}$) και αποτελεί την πρόγνωση του μοντέλου.

Έπειτα, αφαιρούμε από την πραγματική τιμή (1 αν το ζεύγος είχε θετική συσχέτιση, 0 αν είχε αρνητική) την πρόβλεψη και έτσι παίρνουμε την απόκλιση του μοντέλου. Η τιμή της απόκλησης χρησιμοποιείται, πολλαπλασιασμένη με έναν συντελεστή lerning rate για την διαμόρφωση του βάρους των λέξεων στο τέλος αυτής της διαδικασίας. Πιο συγκεκριμένα, από την παλιά τιμή του βάρους αφαιρείται η τιμή της απόκλισης επί το learning rate. Τέλος, ενημερώνουμε τον συντελεστή b.

Η παραπάνω διαδικασία γινόταν αρχικά για κάθε ζέυγος που υπάρχει στο training set τρείς φορές.

Στην τελική μορφή της εργασίας, το training γίνεται διαφορετικά.

Η συνάρτηση RepetitiveTraining δέχεται σαν είσοδο το αρχικό training set. Γίνεται η αρχικοποίηση του μοντέλου και καθορίζονται οι μεταβητές treshold και stepvalue. Γίνεται η διαδικασία εκπαίδευσης του μοντέλου με τον τρόπο που περιγράφηκε παραπάνω και, στην συνέχεια, καλέιται η συνάρτηση TestAndAdd.

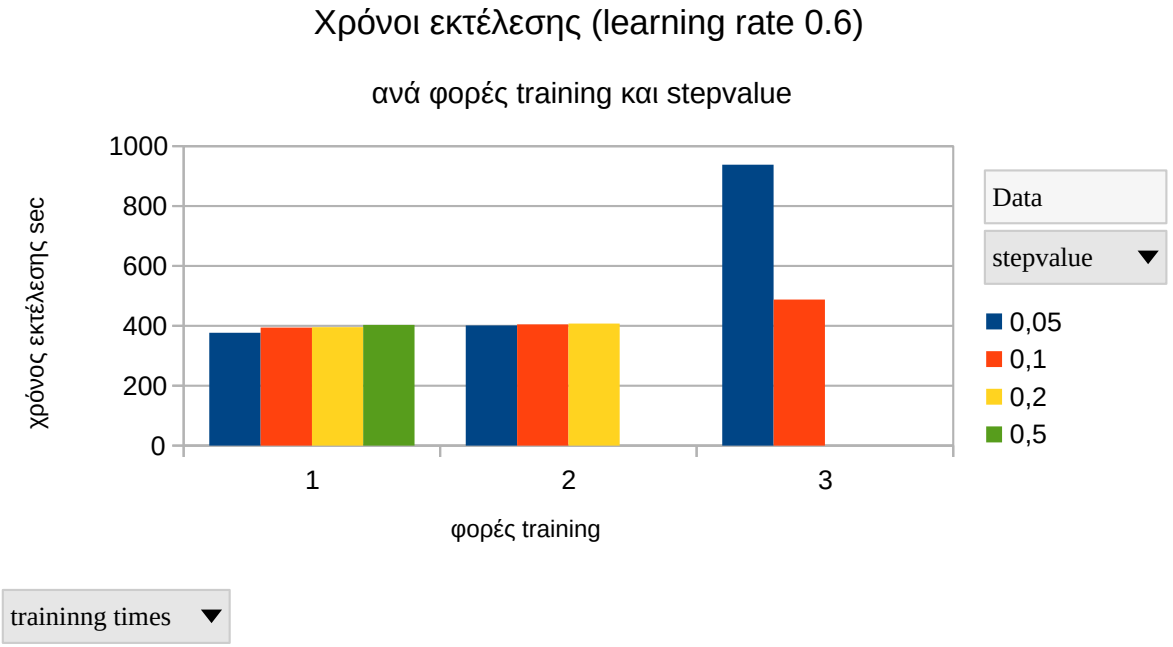
Η συνάρτηση αυτή προσπελαύνει τον πίνακα κατακερματισμού πουπεριέχει όλα τα προϊόντα που έχουμε στην διάθεση μας. Για κάθε δυνατό ζεύγος που προκύπτει, αρχικά, ελέγχεται αν υπάρχει ήδη κάποια συσχέτιση μεταξύ τους και, αν δεν υπάρχει, υπολογίζεται η πρόβλεψη του μοντέλου για αυτό. Αν η πρόβλεψη αυτή είναι αρκετά ισχυρή, είτε για αρνητική, είτε για θετική συσχέτιση, τότε γίνεται εισαγωγή του ζεύγους στο πίνακα κατακερματισμού και (καθώς τα προϊόντα υπάρχουν ήδη από πριν σαν οντότητες μέσα) οι κλίκες αναπροσαρμόζονται με βάση τα νέα δεδομένα όπως και στην αρχική εισαγωγή στον πίνακα.

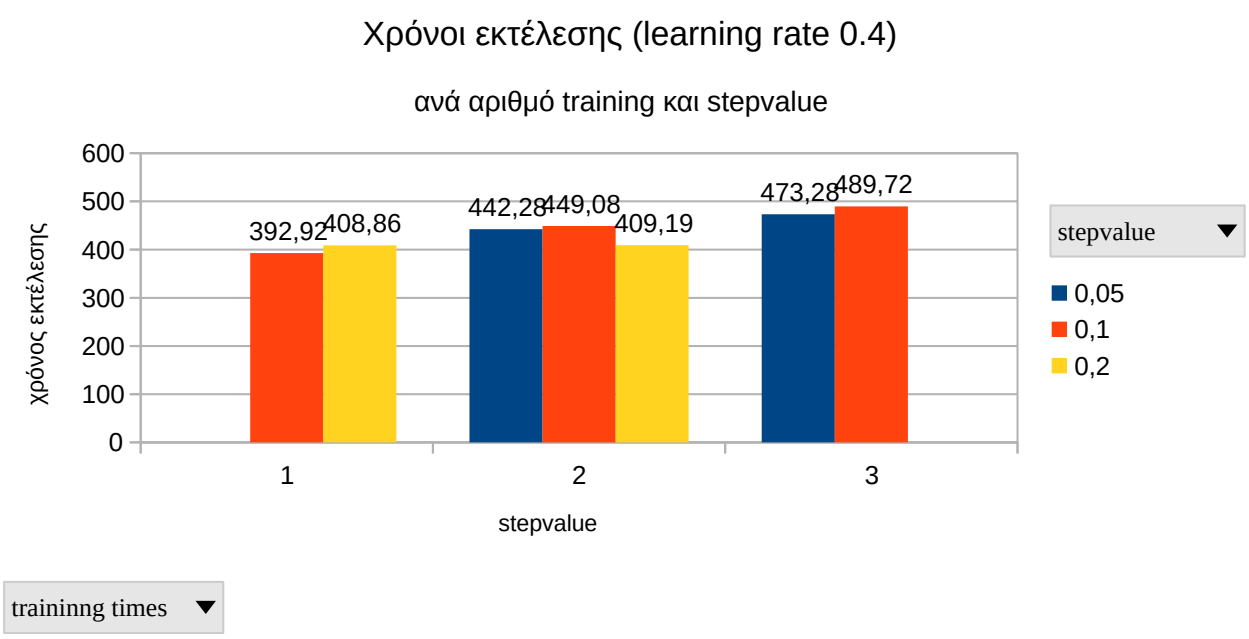
Το αν είναι αρκετά ισχυρή η πρόβλεψη καθορίζεται κάθε φορά από την τιμή του threshold. Για παράδειγμα, αν το threshold είναι 0.2, ισχυρή πρόβλεψη θεωρείται οποιαδήποτε τιμή που απέχει από το 1 ή από το 0 λιγότερο από 0.2., δηλαδή όποια τιμή είναι κάτω από 0.2 και πάνω από 0.8.

Η παραπάνω διαδικασία επαναλαμβάνεται έως ότου το threshold ξεπεράσει κάποιο προκαθωρισμένο όριο. Μετά από κάθε επανάληψη η τιμή του αυξάνεται ανάλογα με την τιμή της μεταβλητής stepvalue, η οποία είναι και αυτή προκαθωρισμένη.

Χρονικές μετρήσεις

Μετά από διάφορους πειραματισμούς προέκυψαν τα εξής αποτελέσματα :





Όλες οι παραπάνω μετρήσεις έχουν γίνει με αρχικό threshold ίσο με 0.1 και, στην περίπτωσητων 3 επαναλήψεων της διαδικασίας εκπαίδευσης και με stepvalue ίσο με 0.05, με αρχικό Threashold ίσο με 0,05.

Ο χρόνος διεκπεραίωσης του προγράμματος αυξάνεται δρματικά στις περιπτώσεις που η αρχική τιμή του threshold είναι αρκετά μικρή (π.χ. 0.05). Αυτό συμβαίνει καθώς, από τον αρχικό αριθμό δεδομένων που ελέγχονται, πολύ λιγότερα θεωρείται ότι έχουν ισχυρή πρόβλεψη από το μοντέλο έτσι ώστε να εισαχθούν στο training set. Έτσι, παραμένουν ακόμα πολλά ζεύγη για έλεγχο και οι πρώτες επαναλήψεις αργούν αρκετά.

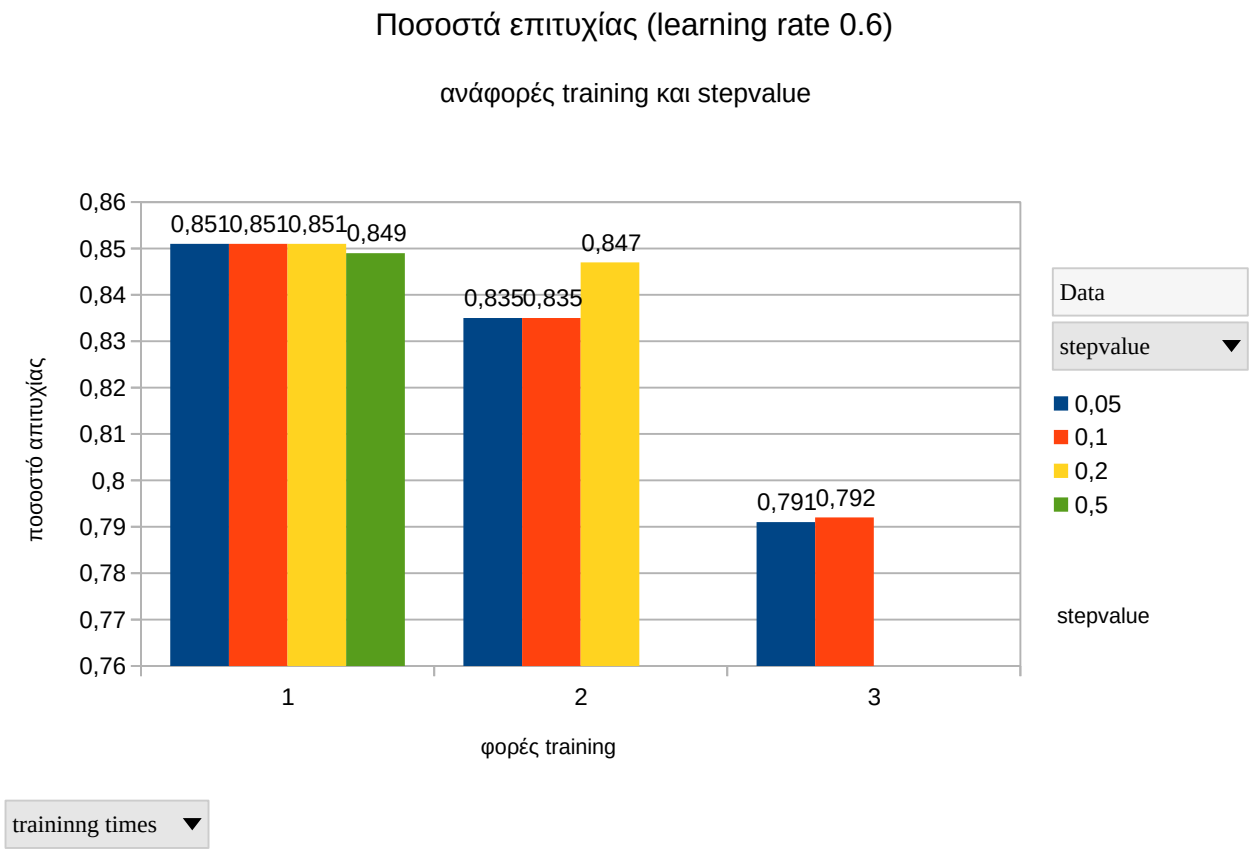
Testing

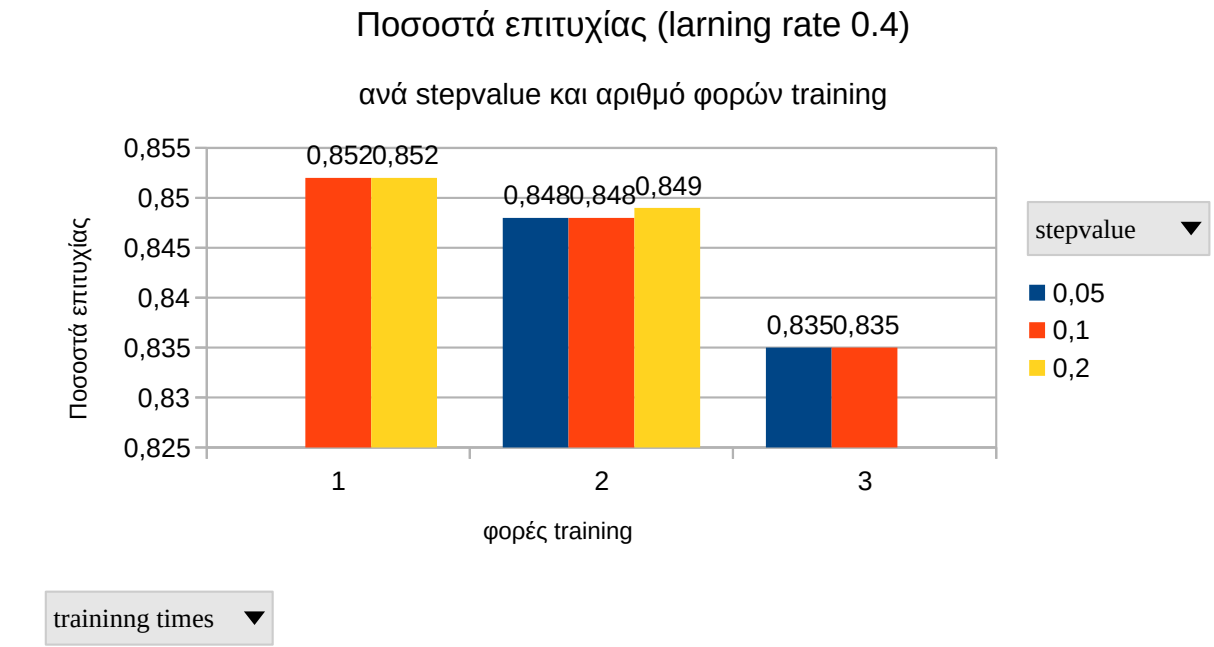
Η διαδικασία του testing δεν παρουσιαάζει κάποια ιδιεταιρότητα. Γίνεται προσπέλαση του αρχείου testing.csv, που δημιουργήθηκε σε προηγούμνεη φάση, γραμμή-γραμμή, και το κάθε ζεύγος καμερών προωθείται στο μοντέλο για πρόβλεψη. Τα αποτελέσματα της πρόβλεψης αξιολογούνται με παρόμοιο τρόπου όπως και στο training.

Θεωρούμε έναν αριθμό χ το όριο με βάση το οποίο κάθε πρόβλεψη θεωρείται θετική ή αρνητική συσχέτηση. Θετική συσχέτιση θεωρείται κάθε πρόβλεψη μεγαλύτερη από 1-χ και αρνητικές όλες οι υπόλοιπες.

Για τον υπολογισμό, λοιπόν, του ποσοστού επιτυχίας του μοντέλου, αρκεί να συγκρίνουμε την συσχέτιση που προκύπτει από την πρόβλεψη με την πραγματική συσχέτιση του ζευγαριού.

Μετά από πειράματα με διαφορετικές τιμές για το χ προέκυψαν τα εξής αποτελέσματα :


Ποσοστά επιτυχίας (learning rate 0.6)
ανάφορές training και stepvalue
0,86
0,85
0,84
0,83
0,82
0,81
0,8
0,79
0,78
0,77
0,76
ποσοστό απιτυχίας
0,8510,8510,8510,849
0,8350,835
0,847
0,7910,792
1
2
3
φορές training
Data
stepvalue
0,05
0,1
0,2
0,5
stepvalue
traininng times


Ποσοστά επιτυχίας (larning rate 0.4)
ανά stepvalue και αριθμό φορών training
0,855
0,85
0,845
0,84
0,835
0,83
0,825
Ποσοστά επιτυχίας
0,8520,852
0,8480,8480,849
0,8350,835
1
2
3
φορές training
stepvalue
0,05
0,1
0,2
traininng times

Threads

Για την επίτευξη καλύτερων χρόνων η υλοποίησή μας επανασχεδιάστηκε έτσι ώστε να χρησιμοποιεί threads στην διαδικασία εκπαίδευσης τυ μοντέλου. Σε προηγούμενα στάδια, χωρίς χρήση threads, το πρόγραμμα απαιτούσε πολύ μεγάλο χρόνο εκτέλεσης, γεγονός που καταστούσε πολύ δύσκολη την διαδικασία πειραματισμών με διάφορες τιμές και τον έλεγχο των αποτελεσμάτων μας αλλά και της χρήσης που γινόταν στην μνήμη.

Ο τρόπος με των οποίο αξιοποιούνται τα threads είναι ο εξής :

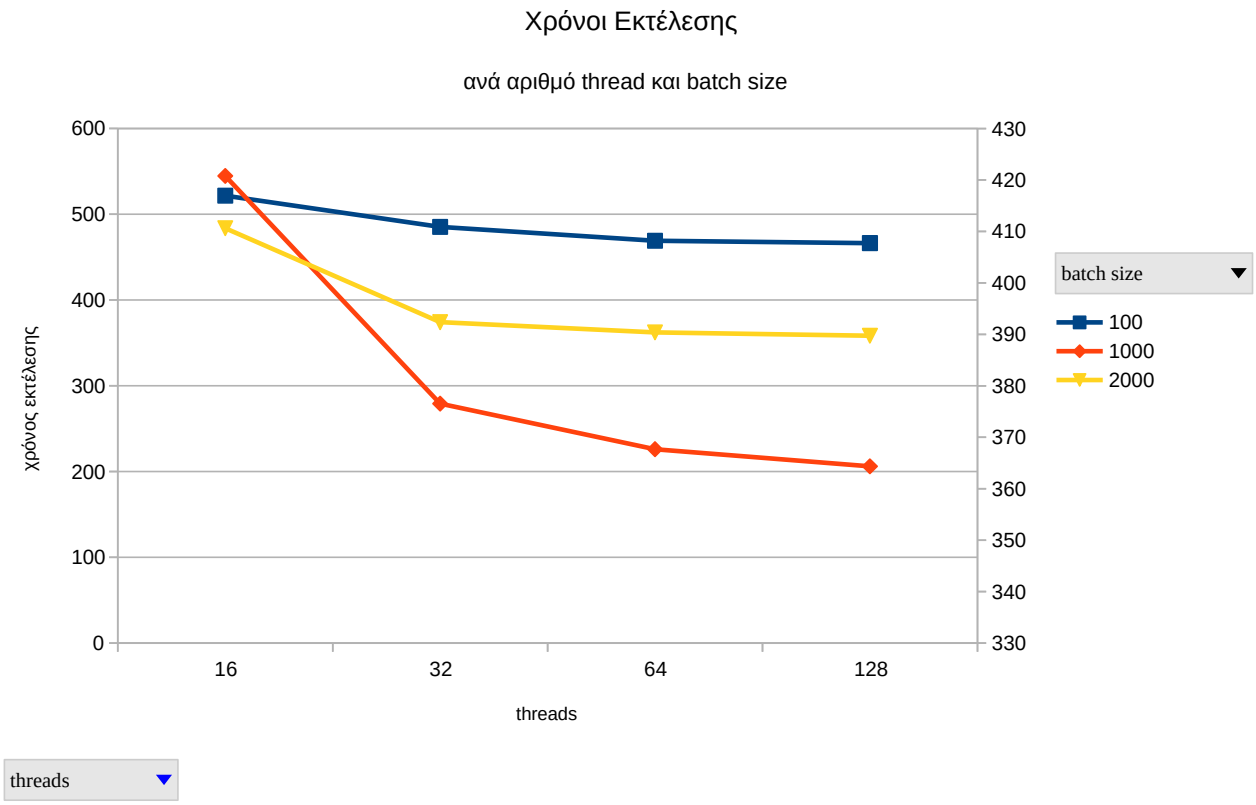
Η συνάρτηση training εκπαιδεύει το μοντέλο σε δέσμες των 1000 παρατηρήσεων(batch size). Κάθε φορά δίνονται τόσες δέσμες όσες είναι ο αριθμός των threads που μπορεί να εκτελέσει ο Job Scheduler.
Ο Job Scheduler εκτελεί οποιαδήποτε ρουτίνα η οποία δίνεται σε μορφη δομης job.
Στην συγκεκριμένη περίπτωση η ρουτίνα που εκτελίται είναι η Job training που λειτουργέι με τον τρόπο που αναπτύχθηκε παραπάνω.

Ο Sheduler αρχικοποιείται ωστε να εκτελεί 16 threads και η τιμή του batch size είναι 1000.

Η δομή Job scheduler κατά την αρχικοποίηση της της δέχεται μία τιμή η οποία δείχνει το πόσα threads θα μπορεί να υποστηρίζει. Αρχικοποιει τις τιμές της και δημιουργεί τόσα threads όσα ζητήθηκαν τα οποία τρέχουν τη ρουτίνα worker.

Η ρουτίνα worker είναι στην ουσία η καρδιά του scheduler και λειτουργεί ως εξής :
Τρέχει μία while η οποία τερματίζει μόνο όταν καταστραφεί ο scheduler και μέσα σε αυτή την while κάθε φορά περιμένει να εμφανιστεί ένα καινούργιο job. Μόλις γίνει αυτό, η νέα εργασία εκτελείται, και στη συνέχεια καταστρέφεται από τον scheduler, ο οποίος μετά περιμένει να πάρει κάποιο καινύριο job.

Μετά από πειράματα με διαφορετικά μεγέθη batch και αριθμό threads προέκυψαν τα εξής αποτελέσματα :



Το πρόγραμμα παράγει πιο γρήγορα αποτελέσματ για όσο μεγαλό αριθμό από threads απασχολεί και όσο μεγαλύτερο batch size έχει οριστεί.

	batch size		
threads	100	1000	2000
16	521,73	420,78	483,76
32	485,29	376,53	374,18
64	469,15	367,68	362,25
128	466,38	364,34	358,4