*Ηλίας Σέττας 3150156*

*Χρήστος Γκουρνέλος 3140033*

*Πέτρος Δημητρακόπουλος 3150034*

**Τεχνητή Νοημοσύνη Εργασία 3: Μηχανική Μάθηση**

**Άσκηση 1 – Bayes**

Τα δεδομένα που χρησιμοποιήσαμε είναι e-mails, τα οποία βρήκαμε από το συγκεκριμένο link:[**http://nlp.cs.aueb.gr/software\_and\_datasets/Enron-Spam/preprocessed/enron3.tar.gz**](https://l.facebook.com/l.php?u=http%3A%2F%2Fnlp.cs.aueb.gr%2Fsoftware_and_datasets%2FEnron-Spam%2Fpreprocessed%2Fenron3.tar.gz&h=ATMWI5c3f69OggmiwYQ40Z7tOka3tno5vdORE0b4zh_J3hcoXX7NFlDnpZddp7nNKiAQ90t4Xdlj8y23nIbjV6yu6z0ptBDU_QIBMp_x6xy-QtT6FgB4px4aDbD0VGab-gtieG9n3sbm)

Σας παρέχουμε το αρχείο με τα δεδομένα στο φάκελο της εργασίας. Πρέπει όμως για τις ασκήσεις 1 και 3 να τον μετακινήσετε μέσα στο φάκελο τους. Η άσκηση 3 χρησιμοποιεί τα ίδια δεδομένα με την άσκηση 1.

**ΠΡΟΣΟΧΗ: ΑΝ ΤΟ ΤΡΕΞΕΤΕ ΕΤΣΙ ΟΠΩΣ ΤΟ ΣΤΕΙΛΑΜΕ ΧΩΡΙΣ ΤΑ DATA ΘΑ ΠΕΤΑΞΕΙ ERROR!**  
Τα e-mails χωρίζονται σε ham και spam και τα αναπαριστά η κλάση **Email**. Κάθε αντικείμενο Email περιέχει ένα HashSet<String> το οποίο περιέχει όλες τις λέξεις αυτού του e-mail χωρίς διπλότυπα. Λόγω της υλοποίησής μας, δεν χρειαζόταν να μετράμε το πόσες φορές εμφανιζόταν μια λέξη και έτσι χρησιμοποιήσαμε το HashSet. H μεταβλητή type (Boolean) συμβολίζει αν ένα e-mail είναι spam ή ham. Πιο συγκεκριμένα η τιμή false δείχνει ότι είναι spam και η τιμή true ότι είναι ham.

Επίσης, υπάρχει η κλάση με το όνομα wordEnt η οποία αναπαριστά μια λέξη, το IG της, καθώς και 4 άλλες double μεταβλητές, τις οποίες θα αναλύσουμε παρακάτω. Η κάθε μια από αυτές τις μεταβλητές αναπαριστά τις δεσμευμένες πιθανότητες .Για παράδειγμα η c0x0 αναπαριστά το P(C=0|X=0). Αυτές οι μεταβλητές θα πάρουν τιμή μόνο για τις ιδιότητες τις οποίες θα κρατήσουμε ως τις πιο σημαντικές.

H βάση δεδομένων αποτελείται από ένα ArrayList το οποίο περιέχει όλα τα mails, ένα HashSet το οποίο περιέχει όλες τις λέξεις που έχουν χρησιμοποιηθεί σε όλα τα mails, έναν πίνακα που περιέχει όλες τις λέξεις που έχουν χρησιμοποιηθεί στα mails, το IG τους, και τέλος περιέχει πόσες και ποιές από αυτές τις λέξεις διαλέξαμε ως τις «σημαντικές ιδιότητες» που θα χρησιμοποιήσουμε στον Bayes. Κάθε φορά που κάνουμε add ένα e-mail στη βάση μας, γίνονται αυτόματα add και οι λέξεις του στο HashSet που περιέχει όλες τις λέξεις.

Όπως είπαμε και πριν, ο αλγόριθμος αρχικά βρίσκει το **Information Gain** όλων των λέξεων που περιέχονται στα mails και διαλέγει αυτές που είναι οι πιο σημαντικές (το πόσες θα διαλέξει εξαρτάται από τη μεταβλητή importantThings). Αυτό που κάνουν οι συναρτήσεις SumPxHc\_x, IG, Hc\_x και Hc είναι να υλοποιούν τους τύπους που περιέχονται στο pdf της διάλεξης 15. Η επιλογή των ιδιοτήτων αυτών γίνεται μέσω της συνάρτησης train.  
Αρχικά αυτή η συνάρτηση βρίσκει το IG για όλες τις λέξεις που περιέχονται μέσα στα e-mails .Έπειτα κρατάει τις πιο σημαντικές και τις προσθέτει στο HashSet important-properties αφού πρώτα υπολογίσει τις δεσμευμένες πιθανότητες P(C=0|X=0) P(C=1|X=0) κοκ.

Παρακάτω ακολουθούν οι συναρτήσεις με τις οποίες υπολογίζουμε διάφορες πιθανότητες και επειδή τα ονόματά τους είναι αρκετά αντιπροσωπευτικά ως προς το τι κάνουν δεν θα τις αναλύσουμε εκτός αν χρειαστεί.

**countEmails**: μετράει πόσα e-mail έχουμε (τύπου Spam ή Ham).

**typeChance**: υπολογίζει την πιθανότητα να είναι Spam/Ham ένα mail στην βάση μας.

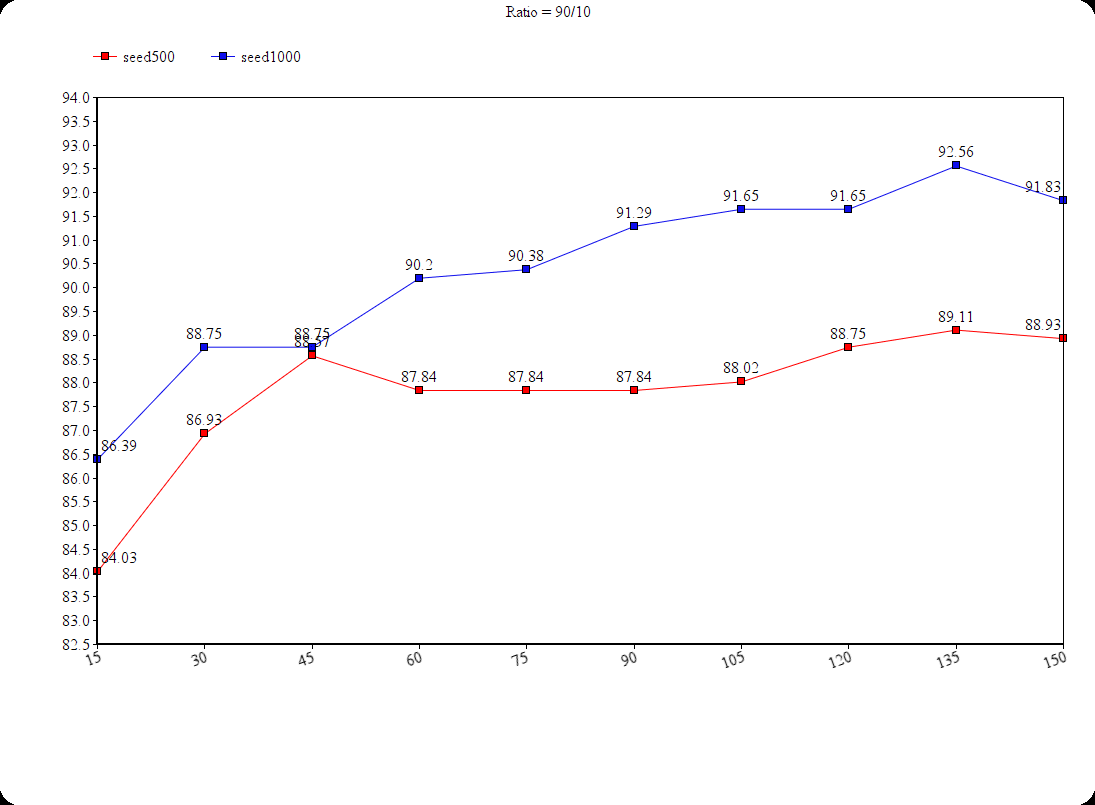
**wordChance3**: υπολογίζει την πιθανότητα μια συγκεκριμένη λέξη να εμφανίζεται ή όχι (κρίνεται από την Boolean μεταβλητή appears).

**wordChance**: υπολογίζει την πιθανότητα μιας λέξης να εμφανίζεται ή όχι δεδομένου του τύπου του mail (P(X=x|C=c)). Επίσης σε αυτή τη μέθοδο χρησιμοποιούμε και τη γεννήτρια LaPlace και για αυτό στο πλήθος των counter μας προσθέτουμε +1 και +2.

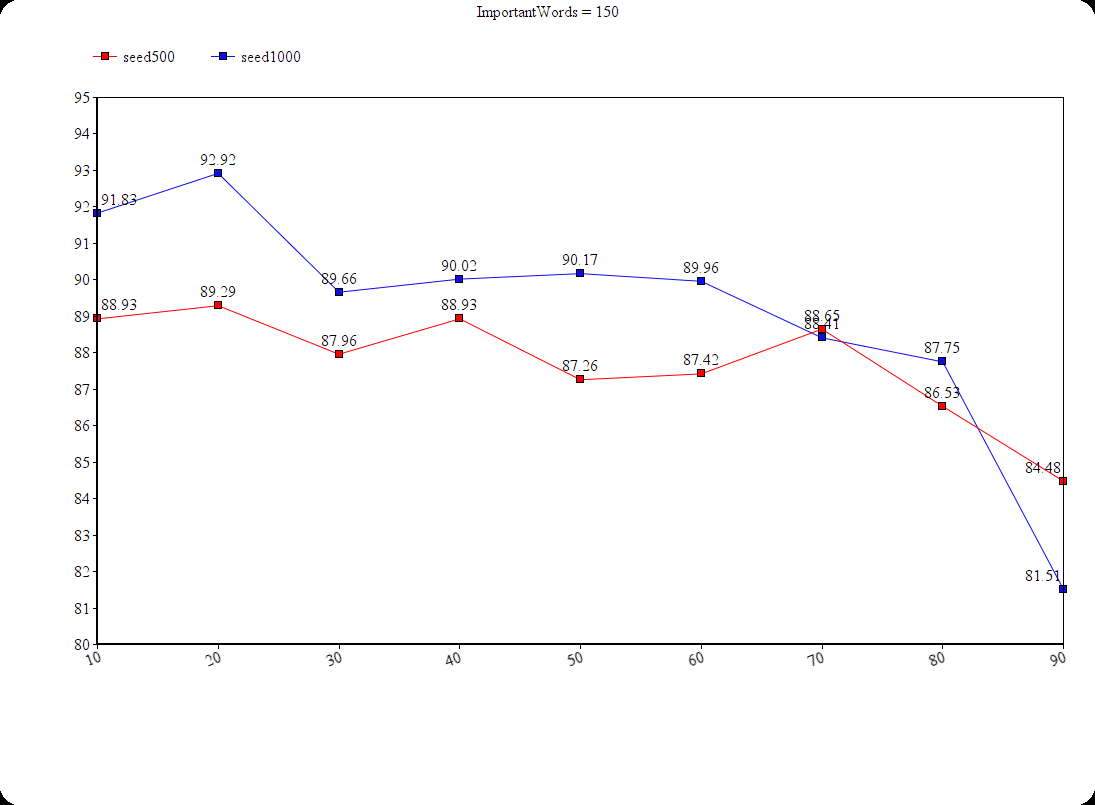
Τέλος, η συνάρτηση **findChance** είναι αυτή που υπολογίζει τη πιθανότητα ενός e-mail να είναι spam ή ham. Αυτό που κάνει είναι να υλοποιεί τους τύπους στις διαφάνειες της διάλεξης 16, σελίδα 6. Η συνάρτηση **sort,** αφού έχει υπολογίσει τη πιθανότητα ένα mail να είναι spam/ham μέσω της συνάρτησης findChance με τα ανάλογα ορίσματα, συγκρίνει αυτές τις 2 πιθανότητες και επιστρέφει τον τύπο με την μεγαλύτερη.

Όσον αφορά τα test data μας στην **main** υπάρχει μια μεταβλητή που λέγεται percentage η οποία δείχνει το πόσο % από τα δεδομένα μας θα είναι test. Η επιλογή τους γίνεται με καθαρά τυχαίο τρόπο. Αρχικά διαβάζουμε όλα τα mails με την σειρά, πρώτα τα ham μετά τα spam και τα αποθηκεύουμε στον ίδιο πίνακα. Έπειτα, μέσω της γεννήτριας τυχαίων αριθμών διαλέγουμε το ανάλογο ποσοστό από αυτά τα mails. Η κατανομή τους γίνεται αρκετά καλά και η πιθανότητα πχ να έχουμε μόνο spam ή ham στο δείγμα μας είναι σχεδόν μηδενική. Αφού διαλέξουμε τα δεδομένα, ο αλγόριθμος τρέχει τον naive Bayes για κάθε ένα mail από αυτά ξεχωριστά, έτσι ώστε να δούμε σε ποια κατηγορία ανήκει.

Στο 1ο τεστ από όλα τα e-mails το 10% τα χρησιμοποιήσαμε ως test και τα υπόλοιπα ως training. Χρησιμοποιώντας 2 διαφορετικά seed (500 και 1000) για να τα χωρίσουμε τυχαία και κάθε φορά αυξάναμε το importantThings κατά 15 (ξεκινώντας από την τιμή 15). Αυτά είναι τα αποτελέσματά:



Αυτό που παρατηρήσαμε είναι ότι όσο αυξάνεται ο αριθμός των importantThings τόσο πιο πολύ αυξανόταν και το ποσοστό επιτυχίας του αλγορίθμου μας.  
Στην 2η εικόνα χρησιμοποιούμε τα ίδια seeds αλλά αυτή τη φορά το Important\_things το κρατάμε σταθερό στο 150 και παρατηρήσαμε πως λειτουργεί ο αλγόριθμος με διαφορετικό ποσοστό test-training data:  
  
Όσο αυξάναμε τα test data και μειώναμε τα training data το ποσοστό επιτυχίας μειωνότανε, το οποίο είναι και φυσιολογικό.



**Άσκηση 2 – ID3**

Η υλοποίηση χωρίζεται σε 4 βασικές κλάσεις και μία που καλεί ουσιαστικά τον αλγόριθμο. Στην **Main** γίνεται και η ανάγνωση των δεδομένων. Συγκεκριμένα η κλάση **TreeData** αναπαριστά τον κόμβο στο δέντρο που φτιάχνουμε, η κλάση **Attribute** αναπαριστά ένα γνώρισμα στα δεδομένα μας και η κλάση **Entropy** αναπαριστά και υπολογίζει την Εντροπία κάθε κόμβου. Στην κλάση **TrainData** περιλαμβάνεται το μεγαλύτερο μέρος του αλγορίθμου. Η κλάση, αφού διαβάσει τα δεδομένα, φτιάχνει το δέντρο απόφασης (μέθοδοι createTree και treeBuilder), υπολογίζει τις Εντροπίες και το Info Gain και τέλος υπολογίζει τα διάφορα στατιστικά (ποσοστά ακρίβειας) για τα δεδομένα εκπαίδευης και τα δεδομένα δοκιμής (κλάσεις **calculateTrainDataMetrics** και **calculateTestDataMetrics**).

Η μορφή δεδομένων που διαβάζει το πρόγραμμα είναι αρχεία κειμένου που περιέχουν γραμμές όπου κάθε γραμμή μπορεί να περιλαμβάνει περισσότερα από ένα γνωρίσματα που λαμβάνουν ακέραιες τιμές. Οι τιμές των γνωρισμάτων χωρίζονται μεταξύ τους με τον κενό χαρακτήρα. Στην πρώτη γραμμή του αρχείου (κεφαλίδα) αναφέρεται το όνομα κάθε γνωρίσματος καθώς και το πλήθος των διαφορετικών τιμών που μπορεί να λάβει το γνώρισμα (ακολουθεί μετά από το όνομα του γνωρίσματος χωρισμένο με κενό από αυτό).

π.χ:

age 49 year 12 nodes 31

Ο τελευταίος αριθμός σε κάθε γραμμή αναφέρει την κλάση / κατηγορία στην οποία ανήκει η συγκεκριμένη εγγραφή.

π.χ:

age 49 year 12 nodes 31

30 64 1 1

30 62 3 1

Στο παραπάνω παράδειγμα λοιπόν η πρώτη γραμμή μετά την κεφαλίδα μας δείχνει ότι η εγγραφή με age = 30, year = 64 και nodes = 1 ανήκει στην κλάση / κατηγορία 1.

Το πρόγραμμα διαβάζει όλα τα δεδομένα από το αρχείο και κρατάει ένα ποσοστό από αυτά για εκπαίδευση (90%) και τα υπόλοιπα για δοκιμή. Τα δεδομένα επιλέγονται τυχαία και απλά φροντίζουμε πάντα το πλήθος τους να έχει την σχέση 90 - 10. Στο πρόγραμμα δοκιμάσαμε 2 datasets (αφού τα μορφοποιήσαμε έτσι ώστε να έχουν την μορφή που διαβάζει το πρόγραμμά μας). Τα 2 datasets βρίσκονται στα παρακάτω links:

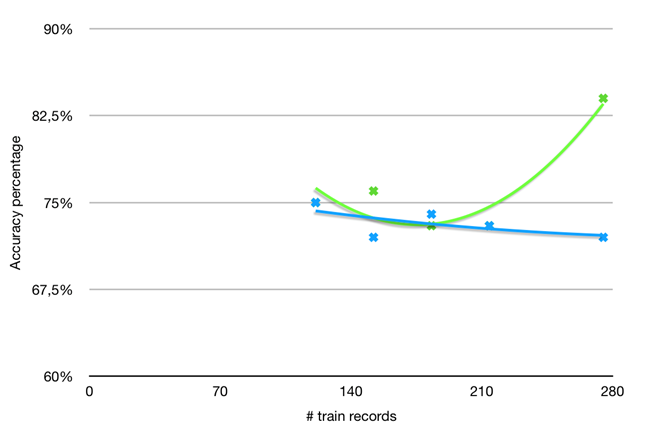
[**http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29**](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29)

**(breastCancer.dat)**

[**https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival**](https://archive.ics.uci.edu/ml/datasets/Haberman%27s+Survival)

**(haberman.dat)**

Μετά από διαδοχικές εκτελέσεις του προγράμματος οι καμπύλες μάθησης που προέκυψαν είναι οι παρακάτω:



**Άσκηση 3 – AdaBoost**

Αρχικά για αυτό το ερώτημα χρησιμοποιούμε την κλάση **wordEnt**  από το προηγούμενο ερώτημα χωρίς κάποια αλλαγή, καθώς και την κλάση Email με την μόνη αλλαγή ότι της προσθέσαμε μια μεταβλητή που δείχνει το βάρος κάθε e-mail. Επίσης δημιουργήσαμε μια νέα κλάση, την **miniID3** η οποία αναπαριστά ένα δεντράκι ID3 το οποίο έχει βάθος 1. Η κλάση «φιλτράρει» τα emails που περιέχονται στον πίνακα βάσει μιας λέξης που της δίνουμε ως όρισμα και αναλόγως με το πόσα spam/ham έχουμε στη περίπτωση που υπάρχει η λέξη και που δεν υπάρχει πράττω και αναλόγως. Να σημειωθεί ότι αν ένα mail πχ έχει την λέξη και είναι spam, δεν θα προσθέσουμε +1 στον μετρητή, αλλά θα προσθέσουμε το βάρος του mail.

Τώρα, εφόσον τελειώσαμε την ανάλυση των βασικών κλάσεων θα αναλύσουμε πως διαλέγουμε ποια λέξη θα έχουμε στο μικρό δέντρο ID3 που θα χρησιμοποιήσουμε για τον αλγόριθμο adaboost. Αρχικά χρησιμοποιούμε την κλάση Data για να γίνει αυτή η λειτουργία. Η κλάση αυτή περιέχει 2 array list, ένα για να κρατάει τα emails και ένα άλλο για να αποθηκεύει λέξεις και εντροπία. Επίσης υπάρχει και ένα hashSet που περιέχει όλες τις λέξεις που έχουν χρησιμοποιηθεί στα e-mails καθώς και ένα miniID3 δεντράκι. Τέλος, περιέχεται και μια double μεταβλητή η οποία δείχνει το βάρος ψήφου του συγκεκριμένου δέντρου. Από μεθόδους υπάρχουν οι μέθοδοι που χρησιμοποιήσαμε στην άσκηση 1 για να βρίσκουμε την εντροπία και το IG με την μόνη διαφορά ότι πλέον το πως υπολογίζουμε τις πιθανότητες έχει αλλάξει ελαφρώς. Προηγουμένως για να βρούμε τη πιθανότητα πχ των e-mail που ήτανε spam μετρούσαμε το πόσα spam είχαμε και διαιρούσαμε αυτό το νούμερο με το συνολικό πλήθος των e-mails. Πλέον όμως εφόσον κάθε e-mail έχει βάρος θα χρησιμοποιήσουμε αυτά. Έτσι η πιθανότητα να είναι ένα e-mail spam θα είναι το άθροισμα των βαρών όλων των e-mail που είναι spam προς το συνολικό άθροισμα όλων των βαρών των e-mails (το οποίο ισούται με 1). Έτσι, η πιθανότητα θα ισούται μόνο με το άθροισμα των βαρών των αντικειμένων που μας ενδιαφέρουν ανάλογα την περίπτωση. Με το ίδιο σκεπτικό αλλάζουμε όλες τις συναρτήσεις που υπολογίζουνε την εντροπία και το IG.  
Τώρα όσον αφορά την «εκπαίδευση» του αλγορίθμου μας έχουμε 2 συναρτήσεις. Η πρώτη που ονομάζεται **train2** χρησιμοποιείται μόνο μια φορά στην αρχή του adaboost και αυτό που κάνει είναι να διαλέξει έναν αριθμό λέξεων που δέχεται ως όρισμα, να τις βάλει σε ένα hashSet<String> και να επιστρέψει αυτό το hashSet. Οι λέξεις που διαλέγει είναι αυτές με το μεγαλύτερο information gain. Αυτές οι λέξεις που επιστρέφει θα είναι και οι λέξεις οι οποίες θα χρησιμοποιηθούν από τα επόμενα id3 trees ως οι λέξεις στη ρίζα ,αν βάζαμε τον αλγόριθμο να ψάχνει όλες τις λέξεις που περιέχονται στα mails θα έπρεπε να υπολογίζει σε κάθε επανάληψη το IG για ~65000 λέξεις, πράγμα που θα έκανε την εκτέλεση πολύ αργή (για 500 επαναλήψεις ήθελε κοντά 2 ώρες).  
Εκτός όμως από την train2 υπάρχει και η **train** η οποία είναι και η συνάρτηση που φτιάχνει το δέντρο μας. Αυτό που κάνει αυτή η συνάρτηση είναι να υπολογίζει το κέρδος πληροφορίας της κάθε λέξης που περιέχεται στον πίνακα με τις σημαντικές λέξεις, έπειτα δημιουργεί ένα mindID3 object που έχει ως λέξη στη ρίζα του την λέξη με το μεγαλύτερο IG.

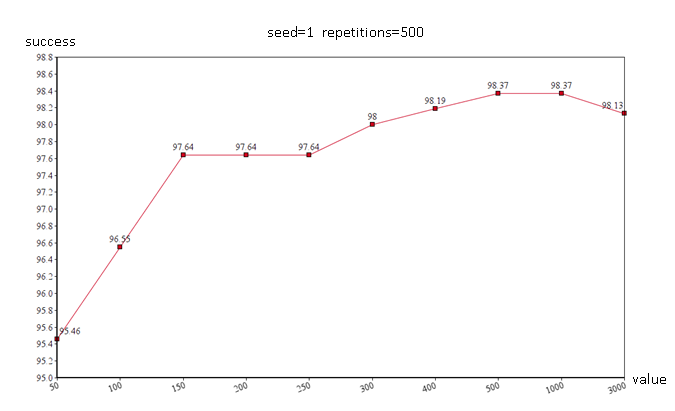
Εφόσον αναλύσαμε όλες τις έξτρα συναρτήσεις ήρθε η ώρα να αναλύσουμε και το πως λειτουργεί ο **adaboost**. Αυτό που κάνει ο αλγόριθμος adaboost στη συγκεκριμένη περίπτωση είναι να χρησιμοποιεί πολλά αδύναμα δέντρα ID3 (με ποσοστό επιτυχίας λίγο πιο πάνω από 50%) και αναλόγως με το πόσο καλά συμπεριφέρεται το κάθε ένα σε συγκεκριμένα παραδείγματα να έχει και ένα βάρος ψήφου. Όταν θέλουμε να αξιολογήσουμε ένα mail όλα αυτά τα δέντρα ψηφίζουν με βάρος το βάρος ψήφου τους και βάσει του τι έβγαλε η πλειοψηφία ταξινομούμε αναλόγως.  
Η κλάση αυτή περιέχει ένα arrayList με τα e-mails μας και 2 hashSet, το 1 περιέχει όλες τις λέξεις που έχουν χρησιμοποιηθεί σε ένα e-mail και το άλλο περιέχει τις λέξεις που βρήκαμε ότι είναι οι πιο σημαντικές. Εκτός αυτών έχουμε 1 μεταβλητή που μας λέει πόσες επαναλήψεις να κάνει ο αλγόριθμος μας (άρα και το πόσα δέντρα να δημιουργήσει), μια μεταβλητή που λέει πόσο % από τα δεδομένα θα πάρουμε ως test data, έναν πίνακα που περιέχει τα testData και 1 αριθμό που μας λέει πόσες μεταβλητές να διαλέξω ως σημαντικές.  
Από μεθόδους αυτές που αξίζει να συζητήσουμε είναι οι εξής :  
 **update :** Αυτή η μέθοδος γεμίζει τον πίνακα με τα test data, έπειτα γεμίζει τον πίνακα που περιέχει όσες λέξεις έχουν χρησιμοποιηθεί, αρχικοποιεί το βάρος του κάθε mail και το κάνει 1/n και στο τέλος βρίσκοντας το IG σε όλες τις λέξεις που έχουν χρησιμοποιηθεί και διαλέγει αυτές με το μεγαλύτερο IG.  
  
**normalize :** Αυτή η μέθοδος φροντίζει ώστε το άθροισμα των βαρών του κάθε e-mail να ισούται με 1. Η απόδειξη είναι αρκετά απλή:  
Έστω w1,w2,w3,…,wn  
  
Το (W­1+W2+…+Wn) μας κάνει sum  
To x(W­1+W2+…+Wn) μας κάνει 1  
  
Με απλή μέθοδο των τριών βγαίνει ότι   
~~(W­~~~~1~~~~+W~~~~2~~~~+…+W~~~~n~~~~)~~1= x~~(W­~~~~1~~~~+W~~~~2~~~~+…+W~~~~n~~~~)~~sum  
1=x\*sum  
x=1/sum

Άρα αν διαιρέσουμε το βάρος του κάθε mail με sum το άθροισμα του βάρους τους θα είναι ίσο με 1

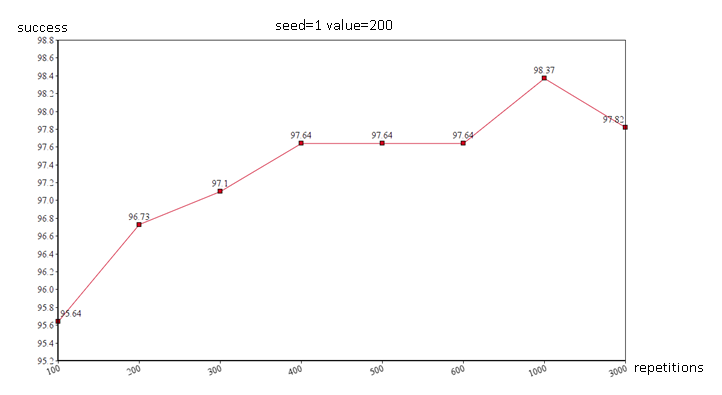
**Run\_ada :** Αυτός είναι ο αλγόριθμος που περιέχεται στις διαφάνειες υλοποιημένος. Σε κάθε επανάληψη δημιουργείται και ένα καινούριο δέντρο. Έπειτα αξιολογούμε αυτό το δέντρο στα παραδείγματά μας και έπειτα αλλάζουμε τα βάρη όσων e-mail βρήκε σωστά το δέντρο (το μειώνουμε) και θέτουμε το βάρος ψήφου του δέντρου. Να σημειωθεί ότι το βάρος ψήφου και το πόσο θα αλλάξει το βάρος των e-mails εξαρτάται καθαρά από το πόσα λάθη έκανε το δέντρο κατά την εκτέλεση του συγκεκριμένου δέντρου στα παραδείγματα.

**Test :** Έχουμε μια μεταβλητή που λέγεται result. Αυτή θα μας βοηθήσει στο να κατατάξουμε το e-mail. Με κάθε ένα από τα δέντρα που δημιουργήσαμε θα κάνουμε evaluate το mail προς sort. Αν το δέντρο βγάλει ότι το e-mail είναι ham, θα προσθέσουμε το βάρος ψήφου του δέντρου στο result, αν είναι spam θα το αφαιρέσουμε. Όταν έχουμε τρέξει τον αλγόριθμο για όλα τα δέντρα, θα δούμε το πρόσημο του result. Αν είναι θετικό τότε το e-mail το έχουμε κατανείμει ως ham, αν είναι αρνητικό ως spam.

Για έλεγχο αρχικά είδαμε πόσο μπορεί να επηρεάσει το ποσοστό επιτυχίας ο αριθμός των properties που έχουμε ως σημαντικές. Έτσι για seed=1 τρέξαμε το πρόγραμμα με διάφορες τιμές του value:



Παρατηρούμε ότι αν είναι πολύ χαμηλός ο αριθμός είναι και χαμηλότερο το ποσοστό επιτυχίας. Όσο ανεβαίνει όμως το νούμερο το ποσοστό ανεβαίνει. Επίσης κάτι άλλο που παρατηρήσαμε είναι ότι αν από το 1000 ανεβεί στο 3000, υπάρχει μια πολύ μικρή πτώση η οποία βεβαίως μπορεί να οφείλεται στα συγκεκριμένα παραδείγματα (ότι δηλαδή επιλέγονται λέξεις κατά το τέλος οι οποίες δεν παίζουν σημαντικό ρόλο στο διαχωρισμό). Έπειτα για το seed=1 και με value=200 κάναμε διάφορα test για να δούμε πόσο επηρεάζει ο αριθμός των επαναλήψεων το αποτέλεσμα:

  
Πάλι παρατηρούμε ότι με χαμηλό αριθμό επαναλήψεων το ποσοστό είναι πιο χαμηλό συγκριτικά, αλλά όσο αυξάνονται οι επαναλήψεις αυξάνεται και το ποσοστό επιτυχίας. Κάτι άλλο που είναι σημαντικό να αναφέρουμε είναι ότι πάλι όταν κάναμε αρκετές επαναλήψεις το ποσοστό επιτυχίας έπεσε συγκριτικά με το όταν η τιμή του ήταν 1000.  
Τέλος παραθέτουμε και ένα ακόμα σχήμα που δείχνει το ποσοστό επιτυχίας για διάφορες αναλογίες test-train data ξανά με seed=1, values=200 και 400 επαναλήψεις.

