Οικονομικό Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής
Μάθημα: Προγραμματισμός Υπολογιστών με C++
Ακαδημαϊκό έτος: 2011–12
Διδάσκων: Ι. Ανδρουτσόπουλος

3η Εργασία

Εισαγωγή

Η εργασία αυτή συνεχίζει την κατασκευή του φίλτρου ανεπιθύμητων διαφημιστικών μηνυμάτων ηλεκτρονικού ταχυδρομείου.

Μέρος 1ο: Η τάξη KNNClassifier

Σκοπός αυτού του μέρους της εργασίας είναι να δημιουργηθεί μια τάξη *KNNClassifier*, κάθε αντικείμενο της οποίας θα είναι ένας ταξινομητής «*k* κοντινότερων γειτόνων». Κάθε ταξινομητής θα «εκπαιδεύεται» πρώτα σε μια συλλογή από χειρωνακτικά ταξινομημένα μηνύματα (ένα *InstancePool*, βλ. 2η εργασία) και στη συνέχεια θα μπορεί να ταξινομήσει νέα μηνύματα (νέα *Instance*) ως επιθυμητά ή ανεπιθύμητα, όπως περιγράφεται στις διαφάνειες της 13ης διάλεξης (χρησιμοποιώντας Ευκλείδεια απόσταση).

Η τάξη *KNNClassifier* πρέπει να παρέχει τουλάχιστον τις ακόλουθες δημόσιες μεθόδους:

KNNClassifier(unsigned short kIn = 5);
Κατασκευαστής, που δέχεται ως όρισμα την τιμή του *k*. Θεωρήστε ότι το *k* θα είναι πάντα περιττός αριθμός.

static float distance(const Instance& inst1, const Instance& inst2);
Στατική μέθοδος. Υπολογίζει την Ευκλείδεια απόσταση δύο αντικειμένων *Instance*, όπως στις διαφάνειες της 13ης διάλεξης.

void train(const InstancePool& trainingPool);
Αποθηκεύει στο εσωτερικό του ταξινομητή ένα αντίγραφο του *trainingPool*, το οποίο χρησιμοποιείται στη συνέχεια από τη μέθοδο *classify*. Αν η *train* του ταξινομητή έχει ξανακληθεί προηγουμένως, τα αποτελέσματα της προηγούμενης κλήσης της *train* αγνοούνται.

bool classify(const Instance& inst) const;
Μαντεύει την κατηγορία του *inst*, εντοπίζοντας τα *k* κοντινότερα (με Ευκλείδεια απόσταση) προς αυτό μηνύματα εκπαίδευσης και κατατάσσοντας το inst στην κατηγορία που πλειοψηφεί μεταξύ των *k* γειτόνων. (Η *classify* δεν καλεί τη μέθοδο *getCategory* του *inst*. Το *inst* παριστάνει ένα νέο μήνυμα, του οποίου δεν γνωρίζουμε την κατηγορία, οπότε ο κωδικός κατηγορίας που βρίσκεται μέσα στο *inst* θεωρείται τυχαίος.) Η *classify* επιστρέφει *true*, αν ο ταξινομητής πιστεύει πως το *inst* παριστάνει ανεπιθύμητο μήνυμα. Διαφορετικά επιστρέφει *false*. Αν δεν έχει κληθεί προηγουμένως η *train*, η *classify* κατατάσσει το *inst* ως επιθυμητό.

Μέρος 2ο: Η τάξη NaiveBayesClassifier

Σε αυτό το μέρος της εργασίας ζητείται να κατασκευάσετε μια τάξη *NaiveBayesClassifier*, κάθε αντικείμενο της οποίας θα είναι ένας αφελής ταξινομητής Bayes (πολυμεταβλητή μορφή Bernoulli). Κάθε ταξινομητής της *NaiveBayesClassifier* θα «εκπαιδεύεται» πρώτα σε μια συλλογή από χειρωνακτικά ταξινομημένα μηνύματα (ένα *InstancePool*) και στη συνέχεια θα μπορεί να ταξινομήσει νέα μηνύματα (νέα *Instance*) ως επιθυμητά ή ανεπιθύμητα, όπως περιγράφεται στις διαφάνειες της 13ης διάλεξης.

Η τάξη *NaiveBayesClassifier* πρέπει να παρέχει τουλάχιστον τις ακόλουθες δημόσιες μεθόδους:

void train(const InstancePool& trainingPool);

Υπολογίζει και αποθηκεύει τις πιθανότητες *P(spam)*, *P(ham)*, *P(t | spam)* και *P(t | ham)*, για κάθε λέξη-κλειδί *t*, σύμφωνα με τους τύπους της 13ης διάλεξης, εκτιμώντας τις πιθανότητες από τα δεδομένα του *trainingPool*. Προσοχή: στις *P(t | spam)* και *P(t | ham)* μας ενδιαφέρει σε πόσα μηνύματα του αντίστοιχου είδους εμφανίζεται η λέξη-κλειδί t, όχι πόσες φορές εμφανίζεται η t στα μηνύματα. Αν η *train* του ταξινομητή έχει ξανακληθεί προηγουμένως, τα αποτελέσματα της προηγούμενης κλήσης της *train* αγνοούνται.

bool classify(const Instance& inst) const;

Μαντεύει την κατηγορία του *inst*, χρησιμοποιώντας το σχετικό τύπο των διαφανειών της 13ης διάλεξης. (Όπως και στην περίπτωση της *KNNClassifier*, η *classify* της *NaiveBayesClassifier* δεν καλεί τη μέθοδο *getCategory* του *inst*.) Η *classify* επιστρέφει *true* αν ο ταξινομητής πιστεύει πως το *inst* παριστάνει ανεπιθύμητο μήνυμα. Διαφορετικά επιστρέφει *false*. Αν δεν έχει κληθεί προηγουμένως η *train*, η *classify* κατατάσσει το *inst* ως επιθυμητό.

Μέρος 3ο: Η τάξη BaselineClassifier

Στο μέρος αυτό, ζητείται να κατασκευάσετε μια τάξη *BaselineClassifier*. Κάθε αντικείμενο της τάξης αυτής θα είναι ένας απλοϊκός (baseline) ταξινομητής, ο οποίος θα κατατάσσει πάντα τα νεοεισερχόμενα μηνύματα (των οποίων δεν γνωρίζουμε την κατηγορία) στην κατηγορία που ήταν συχνότερη στα μηνύματα εκπαίδευσής του.

Η *BaselineClassifier* πρέπει να παρέχει τουλάχιστον τις ακόλουθες δημόσιες μεθόδους:

void train(const InstancePool& trainingPool);

Εκπαιδεύει τον απλοϊκό ταξινομητή στο *trainingPool*. Απλά μετρά πόσα από τα *Instance* του *trainingPool* ανήκουν σε κάθε κατηγορία και αποθηκεύει αυτή την πληροφορία. Αν η *train* του ταξινομητή έχει ξανακληθεί προηγουμένως, τα αποτελέσματα της προηγούμενης κλήσης της *train* αγνοούνται.

bool classify(const Instance& inst) const;

Κατατάσσει το *inst* στην κατηγορία που ήταν πιο συχνή στο *trainingPool* της *train*. Αν δεν έχει κληθεί προηγουμένως η *train*, η *classify* κατατάσσει το *inst* ως επιθυμητό, δηλαδή επιστρέφει *false*.

Μέρος 4ο: Οι τάξεις Classifier και ClassifierEvaluator
Οι τάξεις *KNNClassifier*, *NaiveBayesClassifier* και *BaselineClassifier* πρέπει να είναι παράγωγες της τάξης *Classifier*. Η *Classifier* πρέπει να είναι αφηρημένη και να παρέχει τουλάχιστον τις εξής δημόσιες μεθόδους:

virtual ~Classifier();
Εικονικός καταστροφέας.

virtual void train(const InstancePool& trainingPool) = 0;
virtual bool classify(const Instance& inst) const = 0;
Καθαρά εικονικές μέθοδοι.

Ζητείται, επίσης, να δημιουργήσετε μια τάξη *ClassifierEvaluator*. Κάθε αντικείμενο αυτής της τάξης θα είναι ένα αξιολογητής της συμπεριφοράς ενός ταξινομητή (ενός αντικειμένου *Classifier*). Ο αξιολογητής εκπαιδεύει πρώτα τον ταξινομητή με παραδείγματα εκπαίδευσης, δηλαδή με μηνύματα των οποίων οι κατηγορίες είναι γνωστές. Κατόπιν αξιολογεί τον ταξινομητή με μηνύματα των οποίων οι κατηγορίες είναι άγνωστες στον ταξινομητή αλλά γνωστές στον αξιολογητή.

Η τάξη *ClassifierEvaluator* πρέπει να παρέχει τουλάχιστον τις εξής δημόσιες μεθόδους:

ClassifierEvaluator(Classifier& classifier,
const InstancePool& trainingPool, const InstancePool& testPool);
Κατασκευαστής. Το πρώτο όρισμα είναι ο ταξινομητής που θέλουμε να αξιολογηθεί. Το δεύτερο όρισμα καθορίζει το *InstancePool* που περιέχει τα παραδείγματα εκπαίδευσης. Το τρίτο όρισμα καθορίζει ένα *InstancePool* που χρησιμοποιείται για την αξιολόγηση του ταξινομητή. Τα *Instance* μέσα στο *testPool* περιέχουν τη σωστή κατηγορία (δηλαδή καλώντας την *getCategory* ενός *Instance* του *testPool* μπορούμε να δούμε την κατηγορία στην οποία πραγματικά ανήκει το *Instance*), αλλά αυτή την πληροφορία τη χρησιμοποιεί μόνο ο *ClassifierEvaluator* για να ελέγξει τις απαντήσεις του ταξινομητή. Ο ίδιος ο ταξινομητής δεν καλεί ποτέ την *getCategory* των *Instance* του *testPool*. Η εκπαίδευση και η αξιολόγηση του ταξινομητή γίνονται κατά τη διάρκεια της εκτέλεσης του κατασκευαστή της *ClassifierEvaluator*.

float getAccuracy() const;
Επιστρέφει το λόγο *(TruePositives + TrueNegatives) / Total*, όπου *TruePositives* είναι το πλήθος των *Instance* του *testPool* που ο ταξινομητής κατέταξε σωστά ως ανεπιθύμητα, *TrueNegatives* είναι το πλήθος των *Instance* του *testPool* που ο ταξινομητής κατέταξε σωστά ως επιθυμητά και *Total* είναι το συνολικό πλήθος των *Instance* του *testPool*.

float getPrecision() const;
Επιστρέφει το λόγο *TruePositives / (TruePositives + FalsePositives)*, όπου *TruePositives* όπως παραπάνω, ενώ *FalsePositives* είναι το πλήθος των *Instance* του *testPool* που ο ταξινομητής κατέταξε ως ανεπιθύμητα, ενώ στην πραγματικότητα ήταν επιθυμητά. Στην περίπτωση που ο παρανομαστής του λόγου είναι 0, η μέθοδος να επιστρέφει 0.

float getRecall() const;
Επιστρέφει το λόγο *TruePositives / (TruePositives + FalseNegatives)*, όπου *TruePositives* όπως παραπάνω, ενώ *FalseNegatives* είναι το πλήθος των *Instance* που ο ταξινομητής κατέταξε ως επιθυμητά, ενώ στην πραγματικότητα ήταν ανεπιθύμητα. (*TruePositives + FalseNegatives* είναι το πλήθος των ανεπιθύμητων *Instance* του *testPool*.)

Προσοχή: Διαβάστε οπωσδήποτε το έγγραφο «Γενικές πληροφορίες για τις εργασίες του μαθήματος» (αρχείο *cpp_assignments_general_info.pdf*), που βρίσκεται στα έγγραφα του μαθήματος στο e-class. Μεταξύ άλλων, περιλαμβάνει οδηγίες για τον τρόπο παράδοσης και εξέτασης των εργασιών.

Συμπληρωματικές οδηγίες:
- **Χωρίστε τον πηγαίο κώδικα που θα γράψετε** στα αρχεία *knnclassifier.h*, *knnclassifier.cpp*, *nbclassifier.h, nbclassifier.cpp, baselineclassifier.h*, *baselineclassifier.cpp*, *classifier.h*, *classifier.cpp*, *classifierevaluator.h*, *classifierevaluator.cpp*. Κάποια από τα αρχεία .cpp ενδέχεται να μη χρειάζονται (να είναι κενά).
- **Χρησιμοποιήστε** τα αρχεία *instance.h*, *instance.cpp*, *instancepool.h* και *instancepool.cpp* **της 2ης εργασίας**. Αν δεν κάνατε τη 2η εργασία ή αν το πρόγραμμά σας της 2ης εργασίας δεν δουλεύει σωστά, μπορείτε να χρησιμοποιήσετε τον κώδικα της 2ης εργασίας οποιασδήποτε άλλης ομάδας (αρκεί να δουλεύει σωστά).
- **Δοκιμάστε** την ορθότητα του κώδικά σας χρησιμοποιώντας ένα αρχείο όπως το *experiment.cpp*, που συνοδεύει αυτή την εκφώνηση, όπου το *training_data.txt* θα έχει δημιουργηθεί με το πρόγραμμα της 1ης εργασίας από τα *ling.tar.gz* και *spam.tar.gz* (βλ. αρχεία 1ης εργασίας) και το *test_data.txt* θα έχει δημιουργηθεί με το πρόγραμμα της 1ης εργασίας από τα *test_ling.tar.gz* και *test_spam.tar.gz*, που επίσης συνοδεύουν αυτή την εκφώνηση. Αν δεν κάνατε την 1η εργασία ή αν το πρόγραμμά σας της 1ης εργασίας δεν δουλεύει σωστά, μπορείτε να χρησιμοποιήσετε το πρόγραμμα της 1ης εργασίας οποιασδήποτε άλλης ομάδας (αρκεί να δουλεύει σωστά). **Μην περιλάβετε το *experiment.cpp* (ή άλλο αρχείο που περιέχει συνάρτηση *main*) στον κώδικα που θα παραδώσετε.** Οι βαθμολογητές θα προσθέσουν αυτόματα στον κώδικά σας ένα αρχείο που θα περιέχει κατάλληλη συνάρτηση *main*.