ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ ΜΕ ΧΡΗΣΗ Ν-ΓΡΑΜ ΓΡΑΦΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΛΗΣ Κ. ΕΛΛΗΝΑΣ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π

Αθήνα, Ιούνιος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ ΜΕ ΧΡΗΣΗ Ν-ΓΡΑΜ ΓΡΑΦΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΧΑΛΗΣ Κ. ΕΛΛΗΝΑΣ

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την $27^{η}$ Ιουνίου 2016.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Βασίλειος Λούμος	Εμμανουήλ Βαρβαρίγος
Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2016

.....

Μιχάλης Κ. Έλληνας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.

ΠΕΡΙΛΗΨΗ

Η ανάλυση συναισθήματος είναι η επιστήμη που προσπαθεί να επιλύσει το πρόβλημα της κατανόησης της γνώμης του συγγραφέα ενός κειμένου. Μια από τις ισχυρότερες μεθόδους ανάλυσης συναισθήματος είναι οι ν-γραμ γράφοι. Τα τελευταία χρόνια η ανάπτυξη των κοινωνικών δικτύων είναι ραγδαία. Καθημερινά παράγονται δεδομένα τεράστιου μεγέθους τα λεγόμενα Big data. Η ανάλυση συναισθήματος έχει αναγεννηθεί με την εμφάνιση των Big data. Όμως όταν αναφερόμαστε σε εφαρμογές βασισμένες σε τεράστιο όγκο δεδομένων η ταχύτητα και η κατανάλωση πόρων είναι λέξεις κλειδιά για την επιτυχία τους.

Στην παρούσα εργασία μελετώνται διάφορες τεχνικές της διαδικασίας ανάλυσης συναισθήματος πάνω σε τεράστιο όγκο δεδομένων για την βελτιστοποίησή της, με απώτερο σκοπό την εφαρμογή της σε πραγματικό χρόνο.

Λέξεις κλειδιά: μηχανική μάθηση, κατηγοριοποίηση, κοινωνικά δίκτυα, μεγάλα δεδομένα, βελτιστοποίηση, εξόρυξη γνώμης, ν-γραμ, γράφοι

ABSTRACT

Sentiment Analysis is the science that tackles the problem of understanding the author's opinion of a text. One of the most powerful methods for sentiment analysis is using n-gram graphs. In recent years the development of social networks is rapidly increasing. Every day data of large volumes are generated, the so called Big Data. Sentiment Analysis has been born again with the emergence of Big Data. However when researching applications regarding Big Data, speed and resource consumption are the key ingredients to success.

In this diploma thesis we review and test several methods for sentiment analysis over a large dataset for optimizing the process. Our ultimate goal is the implementation of a real time processing system.

Keywords: machine learning, classification, social networks, big data, optimization, opinion mining, n-grams, graphs

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα αρχικά να ευχαριστήσω την Καθηγήτρια Ε.Μ.Π κυρία Θεοδώρα Βαρβαρίγου για την εμπιστοσύνη που μου έδειξε δίνοντας μου την ευκαιρία να ασχοληθώ με ένα τόσο επίκαιρο και ενδιαφέρον θέμα. Επίσης θα ήθελα να ευχαριστήσω τους Κωνσταντίνο Τσερπέ και Ευάγγελο Ψωμακέλη για τον χρόνο που αφιέρωσαν και τις συμβουλές τους κατά την εκπόνηση αυτής της διπλωματικής εργασίας. Θα ήθελα να ευχαριστήσω τους καθηγητές μου, που όλα αυτά τα χρόνια με εφοδίασαν με τις απαραίτητες γνώσεις έτσι ώστε να πετύχω στη καριέρα μου και γενικότερα στη ζωή μου. Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου που ήταν πάντα δίπλα μου κατά τη διάρκεια των σπουδών μου στο Ε.Μ.Π.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟς ΣΧΗΜΑΤΩΝ

Μέρος Ι

ΕΙΣΑΓΩΓΗ

1

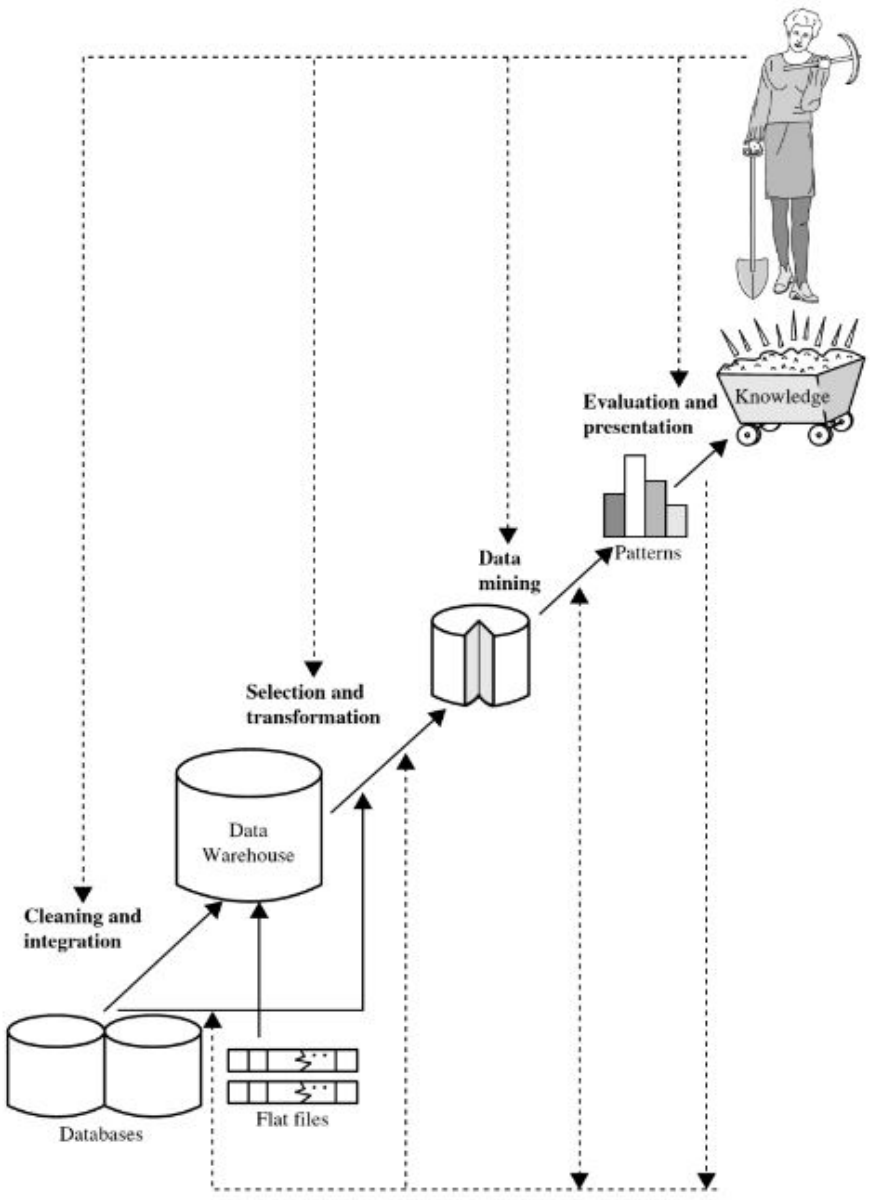
ΕΙΣΑΓΩΓΗ

1.1 *Big data*

Με την ανάπτυξη του διαδικτύου και την ευκολία δημιουργίας περιεχομένου, καθημερινά περισσότεροι χρήστες μοιράζονται δεδομένα και πληροφορίες τεραστίων διαστάσεων. Ωστόσο, οι περισσότερες από τις πληροφορίες είναι σε ακατέργαστη μορφή, τα λεγόμενα ανεπεξέργαστα δεδομένα(raw data). Ταυτόχρονα η ποσότητα τους αυξάνεται ανεξέλεγκτα αφού η επέκταση χώρου είναι πλέον πολύ φθηνή λύση. Αυτή η ραγδαία αύξηση της δημιουργίας περιεχομένου έχει οδηγήσει στην γένεση εννοιών όπως τα Big data και Data Mining. Αυτό έχει προσελκύσει το ενδιαφέρον πολλών ερευνητών για του σκοπούς της αξιοποίησης αυτών των δεδομένων. Μέσα σε αυτά όλα τα δεδομένα κρύβονται μακροσκοπικά πολύ σημαντικές πληροφορίες που αφορούν την ζωή μας. Όπως για παράδειγμα ποιες οι ανάγκες της αγοράς για κάποιο προϊόν, οι γνώμες των ανθρώπων για πράγματα, πρόσωπα ή ιδέες, η τάση για ξεσπάσματα ασθενειών. Ακόμη και η ποσοτική ανάλυση των εγκλημάτων σε ολόκληρη τη χώρα για να εξαχθούν συμπεράσματα για τους λόγους που γίνονται αυτά.

1.2 Εξόρυξη Δεδομένων *(Data Mining)*

Είναι η υπολογιστική διαδικασία ανακάλυψης μοτίβων σε μεγάλα σύνολα δεδομένων. Χρησιμοποιεί μεθόδους της τεχνητής νοημοσύνης, μηχανικής μάθησης, στατιστικής, και συστημάτων βάσεων δεδομένων. Ο γενικός στόχος της διαδικασίας εξόρυξης δεδομένων είναι να συλλεγούν πληροφορίες από ένα σύνολο δεδομένων και να μετατραπούν σε μια κατανοητή δομή για περαιτέρω χρήση. Ο όρος εξόρυξη δεδομένων είναι παραπλανητικός γιατί δεν αναφέρεται την εξόρυξη των δεδομένων των ιδίων αλλά την εξαγωγή μοτίβων και γνώσης για αυτά. Στη πραγματικότητα είναι η αυτόματη ή ημιαυτόματη ανάλυση σε Big data για να εξαχθούν για παράδειγμα ομάδες αρχείων δεδομένων (cluster analysis), ασυνήθιστα αρχεία (anomaly detection), και εξαρτήσεις (association rule mining). Αυτά τα μοτίβα μπορούν στη συνέχεια να θεωρηθούν ως ένα είδος περίληψης των δεδομένων εισόδου και μπορεί να χρησιμοποιηθούν σε περαιτέρω ανάλυση ή, για παράδειγμα, στην μηχανική μάθηση και predictive analytics.



Σχήμα 1.1: Data Mining as a step in the process of knowledge discovery [24]

Η εξόρυξη δεδομένων περιλαμβάνει έξι συνήθεις κατηγορίες εργασιών:

Εντοπισμό ανωμαλιών (Outlier/change/deviation detection) - Ο προσδιορισμός των ασυνήθιστων αρχείων δεδομένων, που μπορεί να είναι ενδιαφέρον ή τα λάθη στα δεδομένα που απαιτούν περαιτέρω διερεύνηση.

Association rule learning (Μοντελοποίηση εξαρτήσεων) - Ψάχνει για τις σχέσεις μεταξύ των μεταβλητών.

Summarization - παρέχοντας μια πιο συμπαγή αναπαράσταση του συνόλου δεδομένων, περιλαμβανομένης της απεικόνισης και δημιουργίας αναφορών.

Clustering - είναι η ανακάλυψη ομάδων και δομών δεδομένων που είναι κατά κάποιο τρόπο όμοιες, χωρίς τη χρήση γνωστών δομών στα δεδομένα.

Classification - είναι η εξαγωγή γενικεύσεων για γνωστές δομές με σκοπό να εφαρμοστούν σε νέα δεδομένα. Για παράδειγμα, ένα πρόγραμμα ηλεκτρονικού ταχυδρομείου ενδέχεται να προσπαθήσει να κατηγοριοποιήσει ένα e-mail ως νόμιμο ή ως spam .

Regression - προσπαθεί να βρει μια συνάρτηση που μοντελοποιεί τα δεδομένα με το ελάχιστο σφάλμα.

Στις τρείς τελευτές θα αναφερθούμε μετέπειτα γιατί αφορούν άμεσα την ανάλυση συναισθήματος.

1.3 Κοινωνικά δίκτυα

Τα κοινωνικά δίκτυα είναι πλεόν μέρος της ζωής εκατομμυρίων ανθρώπων ανά το παγκόσμιο. Οι χρήστες του Internet δεν είναι πλέον παθητικοί αποδέκτες πληροφοριών αλλά συμμετέχοντας σε κοινωνικά δίκτυα έχουν τη δυνατότητα να συζητήσουν με άλλους χρήστες, να ανταλλάξουν απόψεις και ιδέες. Απομακρύνονται έτσι από τις πιο παραδοσιακές υπηρεσίες όπως τα e-mails. Υπάρχουν πολλές πλατφόρμες κοινωνικής δικτύωσης με διάφορα χαρακτηριστικά ώστε να προσελκύσουν χρήστες με σκοπό την δημιουργία και διάδωση πληροφορίας όπως φωτογραφίες, βίντεο και κείμενο προς τους φίλους τους ή δημόσια. Οι πιο διαδομένες αυτών είναι Google+, Twitter και Facebook.

Συγκεκριμένα το Twitter έχει μηνιαία 316 εκατομμύρια ενεργούς χρήστες [8]. Ένας από τους λόγους που είναι τόσο διαδεδομένο είναι οτι παρέχει την δυνατότητα δημοσιεύσεων, τα λεγόμενα Tweets, τα οποία έχουν όριο 140 χαρακτήρων. Πολλοί το θεωρούν ως το καλύτερο μέσο ενημέρωσης αφού οι χρήστες, που μπορεί να είναι εταιρείες, οργανισμοί ή άτομα, παρέχουν το περιεχόμενο της σελίδας τους σε όλους και μπορεί οποιοσδήποτε ακόλουθος(follower) να τα δει. Επίσης το Twitter παρέχει κατάλληλο API για την δημιουργία συνόλων δεδομένων. Πράγμα που το καθιστά μια πολύ καλή πηγή δεδομένων για τους σκοπούς της ανάλυσης συναισθήματος. Τα τελευταία χρόνια οι περισσότερες εργασίες πάνω σε ανάλυση συναισθήματος εφαρμόζονται σε δεδομένα από το Twitter και αποτελούν σημείο ενδιαφέροντος για πολλές επιχειρήσεις.

1.4 Ανάλυση Συναισθήματος

Η ανάλυση συναισθήματος (sentiment analysis) ή εξόρυξη απόψεων(opinion mining) είναι ένας τομέας της ανάλυσης φυσικής γλώσσας ο οποίος έχει σκοπό να εντοπίσει την γνώμη/συναίσθημα που εκφράζεται σε κείμενα. Οι δύο αυτοί όροι εμφανίστηκαν παράλληλα και αναφέρονται στο ίδιο πρόβλημα.[36] Συσχετίζεται με τα πεδία της Ανάκτησης Πληροφορίας, της

Εξόρυξης Δεδομένων και της Μηχανικής Μάθησης. Έχει κινήσει το ενδιαφέρον της ερευνητικής κοινότητας πάνω του με την ανάπτυξη του διαδικτύου αφού ο όγκος δεδομένων παραγόμενων από χρήστες καθημερινά είναι πλέον τεράστιος. Μπορεί να χρησιμοποιηθεί για την επεξεργασία και ανάλυση της γνώμης εκατομμυρίων ανθρώπων σχετικά με κάποιο θέμα. Για αυτό και αποτελεί αντικείμενο ενδιαφέροντος για πολλές εταιρίες και οργανισμούς που θα μπορούν να πάρουν μια γενική εικόνα των απόψεων για την πολιτική[23], την οικονομία και για συγκεκριμένα προϊόντα ή πρόσωπα. Αντιθέτως όμως με την κατηγοριοποίηση κειμένων ως προς το θέμα τους (topic categorization) η ανάλυση συναισθήματος δεν εξαρτάται από το θέμα του κειμένου.[37] Η ανάλυση συναισθήματος μελετάται από πολύ πιο παλιά αλλά η ανάπτυξη του διαδικτύου με τα κοινωνικά δίκτυα και τα blogs είναι που έστρεψαν το ενδιαφέρον πάνω της.

Οι παράγοντες που επηρεάζουν την ανάλυση συναισθήματος είναι πολλοί και πολλές φορές δεν υπάρχει μια καλύτερη μέθοδος επίλυσης του προβλήματος. Μια από τις πρώτες μεθόδους που εφαρμόστηκαν ήταν ο προσδιορισμός της πολικότητας να γίνεται εξετάζοντας λέξεις κλειδιά. Δηλαδή να υπολογίζεται η πολικότητα των λέξεων κλειδιών του κειμένου και να εξάγεται από αυτά η συνολική πολικότητα. Όμως αυτή η μέθοδος δεν προσδίδει καλά αποτελέσματα. Αυτό συμβαίνει γιατί το συναίσθημα σε ένα κείμενο μπορεί να εκφρασθεί με έμμεσο τρόπο αντί με τη χρήση έντονων συναισθηματικά λέξεων.

Ταυτόχρονα μπορεί η αντίληψη της πολικότητας να μην εξαρτάται από το θέμα συζήτησης αλλά σίγουρα εξαρτάται από το σημασιολογικό πλαίσιο στο οποίο τοποθετείται. Επίσης η σειρά των λέξεων και φράσεων στο κείμενο δηλώνει πολύ διαφορετικό συναίσθημα σε κάθε περίπτωση.[36]

Όταν η πηγή κειμένων είναι μικροιστολόγια (π.χ. Twitter) προσθέτονται νέοι παράγοντες που μπορούν να επηρεάσουν την κατηγοριοποίηση. Για παράδειγμα το όριο 140 χαρακτήρων του Tweet αναγκάζει τον συγγραφέα να χρησιμοποιήσει λιγότερες λέξεις ή συντομογραφίες αυτών. Το λεξιλόγιο που χρησιμοποιείται είναι στην καθομιλουμένη με όρους slang, νεολογισμούς και παραλλαγές λέξεων, πράγμα που κάνει λεξικο-βασιζόμενες μεθόδους μη αποτελεσματικές. Επίσης τα Tweets περιέχουν θόρυβο σε μορφή emoticons, υπερσυνδέσμων ή γενικά λανθασμένη χρήση σύνταξης και γραμματικής.

Η ανάλυση συναισθήματος μπορεί να γίνει με πολλές μεθόδους. Οι κύριες μέθοδοι για την εξαγωγή του συναισθήματος από τα δεδομένα γίνονται με τη χρήση λεξικών και χρήση αλγορίθμων κατηγοριοποίησης, οι οποίοι κατατάσσουν αυτόματα τα κείμενα σε κατηγορίες. Μπορούμε να ομαδοποιήσουμε τις μεθόδους με βάση τα δομικά στοιχεία που χρησιμοποιούν σε τέσσερεις κύριες κατηγορίες: keyword spotting, lexical affinity, statistical methods, και concept-based techniques.[29]

Keyword spotting. Αποτελεί αφελής μέθοδο που όμως λόγω της ευκολίας και οικονομίας της είναι αρκετά δημοφιλής. Κατηγοριοποιεί το κείμενο με χρήση των ξεκάθαρης σημασίας λέξεων όπως χαρούμενος, λυπημένος, φο-

βισμένος, και βαρεμένος. Η μέθοδος αυτή όμως παρουσιάζει αδυναμία στην εύρεση λέξεων που δηλώνουν το αντίθετο των αντίστοιχων keywords. Επίσης όταν το κείμενο αποτελείται από ουδέτερης σημασίας λέξεις, που όμως σαν σύνολο δηλώνουν συναίσθημα, δεν υπάρχει δυνατότητα ορθής κατηγοριοποίησης.

Lexical affinity. Αυτή η μέθοδος βελτιώνει της προηγούμενης αναθέτοντας πιθανοτική κατηγοριοποίηση των μη ξεκάθαρης σημασίας λέξεων. Χρησιμοποιεί γνωστά γλωσσικά σώματα δεδομένων για να εκπαιδεύσει την πιθανοτική ερμηνεία. Όμως παρουσιάζει και πάλι το πρόβλημα ανίχνευσης αντιθετικών εκφράσεων και η εκπαίδευση από τα γλωσσικά σώματα εξαρτάται πλήρως από αυτά με αποτέλεσμα την προκατειλημμένη εκτίμηση.

Statistical methods. Αυτή η προσέγγιση, η οποία περιλαμβάνει Μπαϋεζιανή επαγωγή και support vector machines, είναι δημοφιλής στην ταξινόμηση κειμένου. Το σύστημα εκπαιδεύοντας αλγόριθμους μηχανικής μάθησης δεν ανιχνεύει μόνο το συναισθηματικό σθένος για λέξεις κλειδιά, αλλά και για ουδέτερες λέξεις, σημεία στίξης και αριθμό εμφάνισης λέξεων. Όμως αυτές οι μέθοδοι δείχνουν αδυναμία ανίχνευσης του συναισθήματος σε μικρά κείμενα και χρειάζονται μεγάλο όγκο δεδομένων εκπαίδευσης.

Concept-based techniques. Οι μέθοδοι αυτές βασίζονται σε τρόπους ανίχνευσης των εννοιών των κειμένων χρησιμοποιώντας σημασιολογικά δίκτυα και οντολογίες ιστού για να επιτύχουν τη σημασιολογική ανάλυση των κειμένων. Αγνοούν λέξεις κλειδιά και συντακτική ανάλυση. Χρησιμοποιούν μεγάλες βάσεις σημασιολογικής γνώσης για να ανιχνεύσουν την εννοιολογική πληροφορία που παράγεται από τη χρήση φυσικής γλώσσας. Όμως και πάλι βασίζονται πολύ στην ποιότητα της βάσης γνώσης τους ώστε αν αποφέρουν καλά αποτελέσματα.

Στην ουσία το πρόβλημα της ανάλυση συναισθήματος είναι η εύρεση μια συνάρτησης που λαμβάνει ως είσοδο ένα κείμενο και δίνει ως έξοδο την πολικότητα συναισθήματος του(θετικό, αρνητικό ή ουδέτερο). Αυτό τυπικά γίνεται μοντελοποιώντας το κείμενο με τη χρήση μεθόδων Επεξεργασίας Φυσική Γλώσσας. Από το μοντέλο χρησιμοποιείται κατάλληλη υπολογιστική μέθοδος ώστε να ευρεθεί η συνάρτηση μεταφοράς. Όταν όμως το μοντέλο είναι πολύ σύνθετο δεν είναι εύκολη η εξαγωγή αυτής της συνάρτησης. Για αυτό το λόγο οι περισσότεροι επιστήμονες χρησιμοποιούν την μηχανική μάθηση ώστε να πάρουν μια προσέγγιση αυτής της εξίσωσης.

Στα πλαίσια αυτής της εργασίας θα ασχοληθούμε με τους ν-γραμ γράφους για να πάρουμε τα χαρακτηριστικά(features) και στη συνέχεια θα εφαρμόσουμε αλγόριθμους μηχανικής μάθησης(machine learning). Οι αλγόριθμοι μηχανικής μάθησης εκπαιδεύονται-μαθαίνουν παίρνοντας τα (features) και την αντίστοιχη κατηγορία συναισθήματός και στη συνέχεια δοκιμάζονται με νέα δεδομένα για να εξάγουν την κατηγορία συναισθήματος αυτών. Αυτό ονομάζεται κατηγοριοποίηση(classification) .Έτσι υπολογίζεται η ακρίβεια αυτής της μεθόδου, πράγμα που αποτελεί και το κύριο χαρακτηριστικό της εφαρμογής μας. Είναι προφανώς επιθυμητή όσο μεγαλύτερη ακρίβεια γίνεται. Επίσης ένας σημαντικός παράγοντας είναι τα δεδομένα

εκπαίδευσης να είναι όσο περισσότερα γίνεται ώστε αν επιτύχουμε τα καλύτερα δυνατά αποτελέσματα και συμπεράσματα για τις μεθόδους και τα δεδομένα ελέγχου μας. Όπως όμως προφανώς καταλαβαίνουμε αφού η εφαρμογή έχει σκοπό να εφαρμοστεί σε τεράστιο όγκο δεδομένων τότε κλασικοί τύποι αλγορίθμων δεν αρκούν. Η ταχύτητα και η κατανάλωση πόρων αποτελούν πλέον ένα νέο πρόβλημα που πρέπει να επιλύσει ο επιστήμονας. Άλλωστε αν η κατηγοριοποίηση χρειάζεται τεράστιο χρόνο να επιλυθεί τότε θα είναι πλέον ξεπερασμένη η παραγόμενη πληροφορία.

1.5 Σκοπός της παρούσας διπλωματικής εργασίας

Μέχρι στιγμής έχουν γίνει τεράστιες προσπάθειες από την ερευνητική κοινότητα για επιλυση του προβλήματος της κατηγοριοποίησης με χρήση n-gram graphs. Σκοπός μας είναι η εύρεση βελτιστοποιήσεων της διαδιακασίας ώστε να αποφέρει το καλύτερα αποτελέσματα, να εκτελείται όσο το δυνατό γρηγορότερα και να καταναλώνει τους ελάχιστους δυνατούς πώρους. Θα δοκιμάσουμε να προτείνουμε λύσεις με σκοπό την αποδοτικότητα, συγκρίνοντας όλους του πιθανούς τρόπους.

Μέρος II

ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ

2

ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ(NATURAL LANGUAGE PROCESSING)

Είναι ένα σύνολο υπολογιστικών τεχνικών, βασισμένων στη θεωρία, για την αυτόματη ανάλυση και αναπαράσταση της ανθρώπινης γλώσσας.[28] Έχουν αναπτυχθεί διάφορες μέθοδοι ΕΦΓ για ανάλυση συναισθήματος όπως Bag of Words, N-grams και N-grams graphs. Από αυτές τις μεθόδους χρειάζεται κατάλληλη μορφή επεξεργασίας για εξαγωγή των χαρακτηριστικών που θα δοθούν στους αλγόριθμους μηχανικής μάθησης.

2.1 *Bag of Words*

Με τη μέθοδο αυτή κάθε έγγραφο-document χωρίζεται σε συλλογή από λέξεις. Δεν καταγράφονται άλλα δεδομένα για την σχέση των λέξεων αυτών εκτός από τον αριθμό εμφανίσεών τους. Αυτό εφαρμόζεται σε όλες τις κατηγορίες εγγράφων μας, θετικά, αρνητικά και ουδέτερα. Έτσι κάθε λέξη παίρνει τρία συνολικά σκορ αντίστοιχα. Τέλος από αυτά τα σκορ υπολογίζεται με τη χρήση κάποιου κατωφλίου σε ποια κατηγορία από τις τρείς ανήκει. Στη συνέχει μπορεί αυτή η συλλογή να συγκριθεί με κάποιο νέο έγγραφο και με τα συνολικά σκορ των λέξεων της να καταταχτεί σε κάποια κατηγορία.

Στη βιβλιογραφία επίσης έχουν χρησιμοποιηθεί και άλλα είδη χαρακτηριστικών που λαμβάνονται από αυτή τη μέθοδο. Ο πιο συνηθισμένος τύπος είναι η συχνότητα των όρων(term frequency) που περιγράψαμε. Όμως δεν αποτελεί και την καλύτερη αναπαράσταση. Συνηθισμένες λέξεις, όπως 'ο', 'ένα' και 'να' σχεδόν πάντα έχουν την υψηλότερη συχνότητα όρου στο κείμενο. Καταλαβαίνουμε έτσι πως έχοντας ένα υψηλό αριθμό ακατέργαστης καταμέτρησης δεν σημαίνει κατά ανάγκη ότι η αντίστοιχη λέξη είναι πιο σημαντική. Για να αντιμετωπιστεί αυτό το πρόβλημα, ένας από τους πιο δημοφιλής τρόπους εξομάλυνσης των συχνοτήτων όρου είναι να δοθεί βάρος από το αντίστροφο της συχνότητας εγγράφου, γνωστό ως TF-IDF. Άλλη μέθοδος είναι να λαμβάνεται υπόψη η πολικότητα του κειμένου από τους κατηγοριοποιητές. Τέλος μπορεί να χρησιμοποιηθεί αντί της συχνότητας η απλή παρουσία(δυαδική μορφή παρουσία/απουσία) ως βάρος, όπως παρέχει σαν δυνατότητα η βιβλιοθήκη Weka που θα παρουσιάσουμε σε επόμενο κεφάλαιο.

2.1.1 *TF-IDF* για αναπαράσταση κειμένων

Το TF-IDF έχει εξελιχθεί από IDF με την ευρετική διαίσθηση ότι ένας όρος ερώτημα που εμφανίζεται σε πολλά έγγραφα δεν αποτελεί μια καλή διάκριση, και θα πρέπει να δοθεί λιγότερο βάρος από εκείνο που παρουσιάζεται σε λιγότερα έγγραφα.

$$w_{i,j} = tf_{ij} \cdot log(\frac{N}{df_i}) \tag{2.1}$$

είναι η κλασική εξίσωση TF-IDF που χρησιμοποιείται για το βάρος όρου. Το w_{ij} είναι το βάρος του όρου i στο κείμενο j, N είναι ο αριθμός των κειμένων στη συλλογή, tf_{ij} είναι η συχνότητα όρου και df_i είναι η συχνότητα του i στα κείμενα της συλλογής.

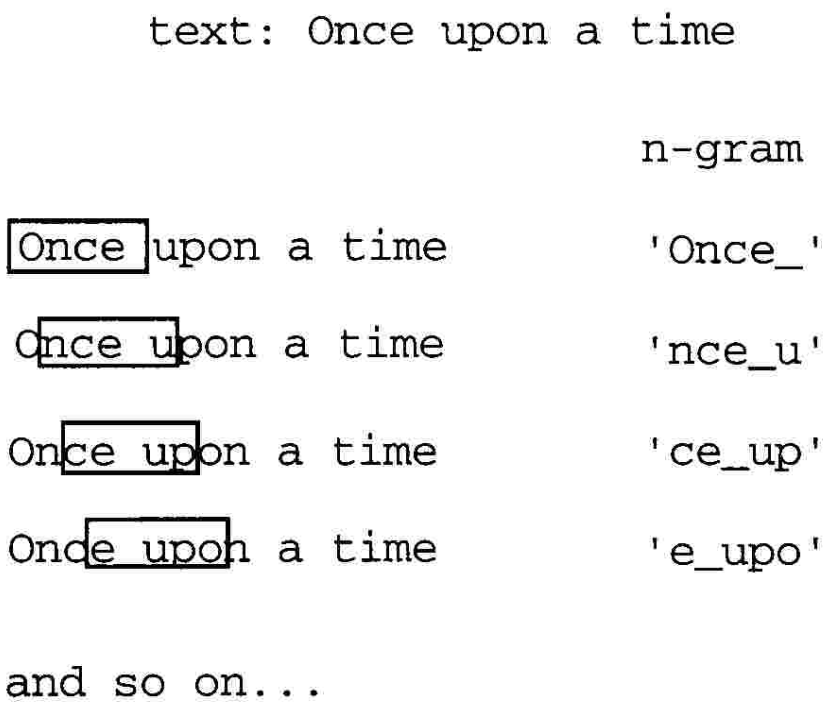
Η βασική ιδέα της TF-IDF είναι από τη θεωρία της μοντελοποίησης γλώσσας που οι όροι σε ένα συγκεκριμένο έγγραφο μπορούν να χωριστούν σε δύο κατηγορίες: τις λέξεις με eliteness και τις λέξεις χωρίς eliteness, δηλαδή, εάν ένας όρος είναι ή όχι σχετικός με το θέμα ενός συγκεκριμένου εγγράφου. Περαιτέρω, η eliteness ενός όρου για ένα δεδομένο έγγραφο μπορεί να αξιολογηθεί με TF και IDF και TF-IDF μοντελοποίηση και χρησιμοποιείται για τη μέτρηση της σημασίας ενός όρου στη συλλογή εγγράφων.

Ωστόσο, υπάρχουν ορισμένες επικρίσεις στη χρήση TF-IDF για την αναπαράσταση κείμενου. Η πρώτη είναι ότι η TF-IDF είναι πολύ ad hoc, διότι δεν προέρχεται άμεσα από ένα μαθηματικό μοντέλο, αν και συνήθως αυτό εξηγείται από τη θεωρία πληροφοριών του Shannon. Η δεύτερη κριτική προέρχεται από το ότι η διάσταση (μέγεθος του συνόλου χαρακτηριστικών) σε TF-IDF για τα δεδομένα κειμένου είναι το μέγεθος του λεξιλογίου σε όλο το σύνολο δεδομένων, με αποτέλεσμα να χρειάζεται τεράστιο υπολογισμό για το βάρος όλων αυτών των όρων. [44]

2.2 *N-grams*

Τα ν-γραμς είναι ψευδολέξεις ν χαρακτήρων στις οποίες χωρίζεται το έγγραφο-document. Ακολούθως αυτά τοποθετούνται σε μια συλλογή όμοια με το Bag of Words μαζί με τους αριθμούς εμφάνισης τους σε θετικά, αρνητικά και ουδέτερα έγγραφα. Ακολούθως ο μεγαλύτερος των τριών αυτών αριθμών αποτελεί το χαρακτηριστικό τους συγκεκριμένου ν-γραμ ως εξής. Εάν είναι περισσότερο θετικός τότε παίρνει χαρακτηριστικό το θετικό αυτό σκορ. Αν είναι περισσότερο αρνητικός παίρνει το σκορ με αρνητικό πρόσημο και τέλος αν είναι ουδέτερο παίρνει μηδενικό σκορ. Τότε μπορεί να χρησιμοποιηθεί κάποιος αλγόριθμος μηχανικής μάθησης όπως logistic regression και να αποφέρει αρκετά καλά αποτελέσματα. Αυτό συμβαίνει διότι ως πληροφορία στην ανάλυση δεν εισάγονται μόνο λέξεις αλλά

ο συνδυασμός τους. Το μοντέλο αυτό βελτιώνει πάνω στο προηγούμενο διότι αποτελεί τεχνική ανεξάρτητη της γλώσσας που είναι ανθεκτική σε θόρυβο όπως ορθογραφικά λάθη. Ωστόσο, η απόδοση του περιορίζεται από το γεγονός ότι αγνοεί εντελώς την αλληλουχία του ν-γραμ μέσα σε μια φράση.



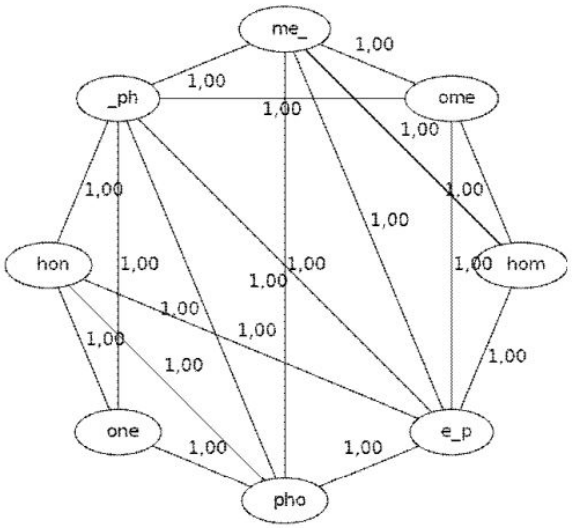
Σχήμα 2.1: N-γραμ

2.3 *N-grams graphs*

Στους ν-γραμ γράφους χρησιμοποιούνται τα ν-γραμς που περιεγράφηκαν πιο πάνω. Η κύρια διαφορά σε αυτή τη μέθοδο είναι το ότι λαμβάνουμε υπόψη τις μεταξύ τους σχέσεις. Έτσι φτιάχνουμε ένα γράφο με κόμβους τα διάφορα ν-γραμ και ακμές τις συσχετίσεις τους με βάρος τον αριθμό εμφανίσεων της συσχέτισης. Αυτοί οι γράφοι ενοποιούνται με τον γράφο χρυσού κανόνα που αντιστοιχεί στην πολικότητα του αντίστοιχου εγγράφου. Τελικά κάθε νέος γράφος συγκρίνεται με τους τρείς γράφους χρυσού κανόνα και τα σκόρ χρησιμοποιούνται σε αλγόριθμο μηχανικής μάθησης. Για να συγκρίνουμε αυτούς τους γράφους χρειαζόμαστε κάποια μετρική ομοιότητας από αυτές που θα αναλύσουμε παρακάτω. [38]

Το μόνο μειονέκτημα του μοντέλου αυτού είναι ο χρόνος που καταναλώνεται κατά τη δημιουργία των γράφων χρυσού κανόνα και τον υπολογισμό της ομοιότητας γράφων. Αυτός ο χρόνος εξαρτάται άμεσα από το μέγεθος του N και του συνόλου δεδομένων εισόδου. Όμως, ανάλογα με την

επιλεχθείσα μετρική ομοιότητας μπορεί αυτή η μέθοδος επεξεργασίας φυσικής γλώσσας να εφαρμοστεί αποδοτικότερα από τις προαναφερθείσες μεθόδους. Η ομοιότητα γράφων μπορεί να συγκρίνει ολόκληρους γράφους μεταξύ τους, (π.χ. κόμβο προς κόμβο) ή μπορεί να αναφερθεί σε συγκεκριμένα χαρακτηριστικά των δύο γράφων(π.χ. συσχέτιση με κοντινούς κόμβους). Προτέρημα αυτής της αναπαράστασης είναι η μεγαλύτερη εμφάνιση ν-γραμ σε σχέση με ολόκληρες λέξεις που σημαίνει την ενθυλάκωση περισσότερης πληροφορίας για το κείμενο. Ταυτόχρονα παρατηρείται μείωση τους αριθμού των χαρακτηριστικών που προκύπτουν σε σχέση με τις προηγούμενες αναπαραστάσεις, αφού εξαρτάται από τον αριθμό κλάσεων αντί του μεγέθους του λεξιλογίου των δεδομένων εισόδου.



Σχήμα 2.2: N-γραμ γράφος για την φράση 'home_phone'[21]

Μέρος III

ΟΜΟΙΟΤΗΤΑ ΓΡΑΦΩΝ

3

ΟΜΟΙΟΤΗΤΑ ΓΡΑΦΩΝ

Το πρόβλημα της ομοιότητας γράφων προκύπτει όταν καλούμαστε να δώσουμε μια μετρική ομοιότητας, δηλαδή ένα πραγματικό αριθμό μεταξύ 0 και 1, μεταξύ δύο γράφων G1 και G2 με πιθανόν διαφορετικούς κόμβους, ακμές και βάρος.[32]

Προτείνονται διάφορες μετρικές για λύση αυτού του προβλήματος:

3.1 *Containment Similarity*

Εκφράζει ποσοστό ακμών του γράφου G1 που ανήκουν επίσης και στον γράφο G2 ως προς το μικρότερο των μεγεθών των δύο αυτών γράφων.

$$CS(G1, G2) = \frac{\text{αριθμός κοινών ακμών}}{min(|G1|, |G2|)} \tag{3.1}$$

3.2 *Value Similarity*

Επίσης λαμβάνει υπόψη το βάρος των ακμών. Εκφράζει για κάθε κοινή ακμή e με αντίστοιχο βάρος w^{G1}, w^{G2} :

Λόγος τιμών:

$$VR(e) = \frac{min(w^{G1}(e), w^{G2}(e))}{max(w^{G1}(e), w^{G2}(e))}) \tag{3.2}$$

Ομοιότητα τιμών:

$$VS = \frac{\sum_{e \epsilon G1,G2} VR(e)}{max(|G1|, |G2|)} \tag{3.3}$$

Μια κύρια πτυχή είναι να αναγνωρίζει το μέγιστο κοινό υπογράφημα, όπως το Rascal [39], η οποία βασίζεται στη μέγιστη κλίκα μεταξύ δύο γράφων. Μια άλλη πτυχή βασίζεται στην ομοιότητα μεταξύ των κόμβων των γράφων[34].

3.3 *Normalized Value Similarity*

Αποσυνδέει το VS από την επίδραση του μεγέθους του μεγαλύτερου γράφου.

$$NVS = VS/SS \tag{3.4}$$

$$SS = \frac{min(|\mathrm{G1}|, |\mathrm{G2}|)}{max(|G1|, |\mathrm{G2}|)} \tag{3.5}$$

[22]

Επίσης μπορούν να χρησιμοποιηθούν συναρτήσεις όπως η Ευκλείδεια απόσταση και η Συνημιτονική ομοιότητα. Για την χρήση αυτών των αποστάσεων παίρνουμε δύο διανύσματα με τα αντίστοιχα βάροι για κάθε κοινή ακμή e, V(e) και W(e).

3.4 Ευκλίδεια απόσταση

Μετράει την συνήθη(ευκλίδεια) απόσταση μεταξύ δύο σημείων στον επίπεδο ν-διάστατο χώρο κάνοντα επανειλημμένη χρήση του Πυθαγορείου θεωρήματος. [19]

$$ED(V, W) = \sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \ldots + (v_n + w_n)^2} \tag{3.6}$$

3.5 Συνημιτονική ομοιότητα

Είναι ένα μέτρο ομοιότητας μεταξύ δύο διανυσμάτων σε χώρο εσωτερικού γινομένου που μετρά το συνημίτονο της γωνιάς μεταξύ τους. Χρησιμοποιείται κυρίως στο θετικό χώρο όπου το αποτέλεσμα ανήκει μόνο στο πεδίο τιμών [0,1]. [20]

$$CS = \frac{V \cdot W}{\|V\| \cdot \|W\|} = \frac{\sum_{i=1}^{n} V_i * W_i}{\sqrt{\sum_{i=1}^{n} V_i^2} * \sqrt{\sum_{i=1}^{n} W_i^2}} \tag{3.7}$$

Μέρος IV

ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ

4

ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

4.1 Επιβλεπόμενη μηχανική μάθηση

Στην πράξη η ευρύτερα χρησιμοποιούμενη μέθοδος μηχανική μάθησης. Πρόκειται για την διαδικασία εκπαίδευσης ενός αλγορίθμου ώστε να μάθει την χαρακτηριστική εξίσωση που αντιστοιχίζει δεδομένα γνωστής εισόδου και εξόδου. Ο σκοπός είναι να προσεγγίσει την συνάρτηση όσο καλύτερα μπορεί ώστε να μπορεί να προβλέψει την έξοδο για νέα δεδομένα εισόδου.

Ονομάζεται επιβλεπόμενη μηχανική μάθηση επειδή η διαδικασία εκπαίδευσης του αλγορίθμου μπορεί να παρομοιαστεί με έναν επιβλέπον δάσκαλο. Εφόσον είναι γνωστές οι απαντήσεις, ο αλγόριθμος κάνει επαναληπτικά προβλέψεις σχετικά με τα δεδομένα εκπαίδευσης και διορθώνεται από το δάσκαλο. Η μάθηση σταματά όταν ο αλγόριθμος επιτυγχάνει ένα αποδεκτό επίπεδο απόδοσης.

Τα προβλήματα επιβλεπόμενης μάθησης μπορούν να ομαδοποιηθούν περαιτέρω σε προβλήματα παλινδρόμησης(**regression**) και κατηγοριοποίησης(**classification**). Σε αυτά θα επικεντρωθούμε στη συνέχεια.

4.2 Μη επιβλεπόμενη μηχανική μάθηση

Στα προβλήματα μη επιβλεπόμενης μάθησης έχουμε μόνο δεδομένα εισόδου χωρίς την αντίστοιχη έξοδο. Ο στόχος της μη επιβλεπόμενης μάθησης είναι η μοντελοποίηση την υποκείμενης δομής ή της κατανομής των δεδομένων, προκειμένου να μάθουμε περισσότερα σχετικά με αυτά.

Ονομάζεται μη επιβλεπόμενη διότι αντίστοιχα δεν είναι γνωστές οι απαντήσεις και δεν υπάρχει επιβλέπον δάσκαλος. Η μηχανή αφήνεται να επινοήσει μόνη της και να ανακαλύψει τη σημαντική πληροφορία από τα δεδομένα.

Τα προβλήματα μη επιβλεπόμενης μάθησης μπορούν να ομαδοποιηθούν περαιτέρω σε προβλήματα ομαδοποίησης(**clustering**) και σύνδεσης(**association**).

4.3 Ημι-Επιβλεπόμενη Μηχανική Μάθηση

Τα προβλήματα όπου έχουμε ένα μεγάλο μέγεθος δεδομένων εισόδου, αλλά μόνο για μερικά από αυτά είναι γνωστή η απάντηση, ονομάζονται ημι-επιβλεπόμενα μαθησιακά προβλήματα.

Ένα καλό παράδειγμα είναι ένα φωτογραφικό αρχείο, όπου μόνο μερικές από τις εικόνες έχουν επισημανθεί, (π.χ. σκύλος, γάτα, πρόσωπο) και η πλειοψηφία είναι μη επισημασμένη.

Πολλά προβλήματα στη πραγματικότητα εμπίπτουν σε αυτή την περιοχή. Αυτό οφείλεται στο γεγονός ότι είναι ακριβή η επισήμανση των δεδομένων, δεδομένου ότι μπορεί να απαιτεί την γνώμη εμπειρογνωμόνων του αντίστοιχου θέματος. Αντιθέτως τα μη επισημασμένα δεδομένα είναι φτηνά και εύκολο να συλλέγουν και να αποθηκευτούν.

Μπορούν να χρησιμοποιηθούν τεχνικές επιβλεπόμενης μάθησης για να κάνουν την καλύτερη πρόβλεψη για τα μη επισημασμένα δεδομένα. Μετά μπορούν αυτά να ανατροφοδοτηθούν στον αλγόριθμο επιβλεπόμενης μάθησης ως δεδομένα εκπαίδευσης για τη δημιουργία ενός μοντέλου πρόβλεψης σε νέα δεδομένα.

5

ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗΣ(CLASSIFICATION ALGORITHMS

Στα πεδία της μηχανικής μάθησης και της στατιστικής, κατηγοριοποίηση είναι το πρόβλημα του εντοπισμού σε ποια από το σύνολο κατηγοριών ανήκουν τα δεδομένα προς παρατήρηση. Αυτό γίνεται αρχικά με την εκπαίδευση του αλγορίθμου με ένα σύνολο εκπαίδευσης από δεδομένα με την αντίστοιχη κατηγορία(έξοδο) να είναι γνωστή. Η κατηγοριοποίηση ανήκει στον τομέα της επιβλεπόμενης μάθησης. Το αντίστοιχο πρόβλημα για την μη-επιβλεπόμενη μάθηση ονομάζεται ομαδοποίηση(clustering).

Συνήθως τα δεδομένα προς εξέταση αναλύονται σε ένα σύνολο μετρήσιμων ιδιοτήτων τα λεγόμενα features. Αυτές οι ιδιότητες μπορεί ποικιλοτρόπως να είναι κατηγορικές (π.χ. 'A', 'B', 'AB' ή 'O', για τον τύπο του αίματος), ταξικές (π.χ. 'μεγάλα', 'μέσο' ή 'μικρό'), ακέραιων-τιμών (π.χ. ο αριθμός των εμφανίσεων μιας λέξης σε ένα κείμενο) ή πραγματικών τιμών (π.χ. μέτρηση της πίεσης του αίματος). Άλλοι κατηγοριοποιητές λειτουργούν με σύγκριση παρατηρήσεων(clustering) σε προηγούμενες παρατηρήσεις μέσω μιας συνάρτησης ομοιότητας ή την απόσταση.

[1]

Τύποι αλγορίθμων κατηγοριοποίησης

5.1 *Naive Bayes*

Είναι το πιο διαδεδομένο μοντέλο για κατηγοριοποίηση. Υποθέτει ότι τα αριθμητικά γνωρίσματα(attributes) δημιουργούνται από μια μοναδική Γκαουσιανή κατανομή. Έστω C η τυχαία μεταβλητή για την κλάση ενός παραδείγματος και X ένα διάνυσμα με τυχαίες μεταβλητές που δηλώνουν τις παρατηρούμενες τιμές του χαρακτηριστικού. Επίσης έστω c μια συγκεκριμένη τιμή κλάσεως και x ένα συγκεκριμένο διάνυσμα χαρακτηριστικών. Δεδομένου ότι το x είναι το διάνυσμα χαρακτηριστικών γνωστού συμβάντος υπολογίζεται η πιθανότητα κάθε κλάσης για το παρατηρούμενο παράδειγμα με τον γνωστό κανόνα Bayes.

$$P(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)} \tag{5.1}$$

όπου X=x είναι η περίπτωση $X = x_1 \wedge x_2 \wedge \ldots \wedge x_n$.

Εδώ καταλαβαίνουμε ότι για συνεχείς κατανομές ή πολύ μεγάλο ν θα ήταν πολύ δύσκολο να υπολογιστεί. Ο αλγόριθμος Naive Bayes κάνει την υπόθεση ότι όλα τα χαρακτηριστικά (features) είναι ανεξάρτητα από κάθε άλλο χαρακτηριστικό δεδομένης της κλάσης. Άρα:

$$P(X = x|C = c) = \prod_i p(X = x_i|C = c) \tag{5.2}$$

και τελικά για να βρούμε την κλάση με την υψηλότερη πιθανότητα:

$$y = argmax_{c\epsilon C} P(C = c) \prod_{i=1}^{n} P(X = x_i|C = c) \tag{5.3}$$

[31, 33]

5.2 Λογιστική Παλινδρόμηση(*Logistic Regression*)

Η λογιστική παλινδρόμηση ταξινομεί τα δεδομένα στην αντίστοιχη τους κλάση μοντελοποιώντας την πιθανότητα P(C=1 | X=x) ως συνάρτηση του χ. Ως C θεωρείται η μεταβλητή κλάσης και X το διάνυσμα χαρακτηριστικών. Η λογιστική παλινδρόμηση αποτελεί το μοντέλο όπου εκφράζεται ο λογιστικός μετασχηματισμός της πιθανότητας ως γραμμική σχέση του χ:

$$Logit(x) = \frac{log(P(X))}{(1 - P(X))} = \beta_0 + x \cdot \beta \tag{5.4}$$

,όπου β_0 σταθερά και β διάνυσμα τελεστών.

Επιλύοντας ως προς P(X) :

$$p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-\beta_0 - x \cdot \beta}} \tag{5.5}$$

Παρατηρείται μεγαλύτερη ευκολία στην κατανόηση του μοντέλου στην μετασχηματισμένη μορφή από τη μη μετασχηματισμένη. Για ελαχιστοποίηση του ποσοστού λανθάνουσας αποτίμησης της μεθόδου πρέπει για $p >= 0.5$ να προβλέπεται $C = 1$ και για $p < 0.5$ $C = 0$. Άρα για αρνητικές τιμές του $\beta_0 + x \cdot \beta$ να προβλέπεται 0 και 1 σε κάθε άλλη περίπτωση. Τελικά η λογιστική παλινδρόμηση αποτελεί γραμμικό κατηγοριοποιητή. Το σύνορο απόφασης που διαχωρίζει τις δύο κλάσεις προκύπτει από την επίλυση του $\beta_0 + x \cdot \beta = 0$. Η σταθερά β_0 και οι συντελεστές του διανύσματος β προσδιορίζονται αναζητώντας τις τιμές που μεγιστοποιούν την πιθανότητα στο σύνολο εκπαίδευσης.

Για το πρόβλημα περισσότερων των δύο κλάσεων προκύπτει η Πολυωνυμική Λογιστική Παλινδρόμηση (Multinomial Logistic Regression ή Maximum Entropy). Αντί των β_0 και β κάθε κλάση c θα έχει αντίστοιχο β_0^c και β^c και οι πιθανότητες θα είναι:

$$P(C=c \mid \overrightarrow{X}=x)=\frac{e^{\beta_0^c+x\cdot\beta^c}}{\sum_c e^{\beta_0^c+x\cdot\beta^c}} \tag{5.6}$$

[33, 42]

5.3 Δέντρα αποφάσεων

Τα δέντρα αποφάσεων είναι από τους πιο απλούς, αλλά ταυτόχρονα ισχυρούς αλγόριθμους μάθησης. Είναι εύκολοι στην υλοποίηση και αποτελούν καλή εισαγωγή στην επαγωγική μάθηση. Αρχικά τοποθετούνται όλες οι παρατηρήσεις σε ένα αρχικό κόμβο. Στη συνέχεια επαναλαμβανόμενα γίνεται προσπάθεια εύρεσης του χαρακτηριστικού με βάση τις τιμές του οποίου θα διαιρεθεί ο κόμβος ώστε να ελαχιστοποιηθεί το βάθος του δέντρου.

Παράδειγμα δέντρου αποφάσεων:

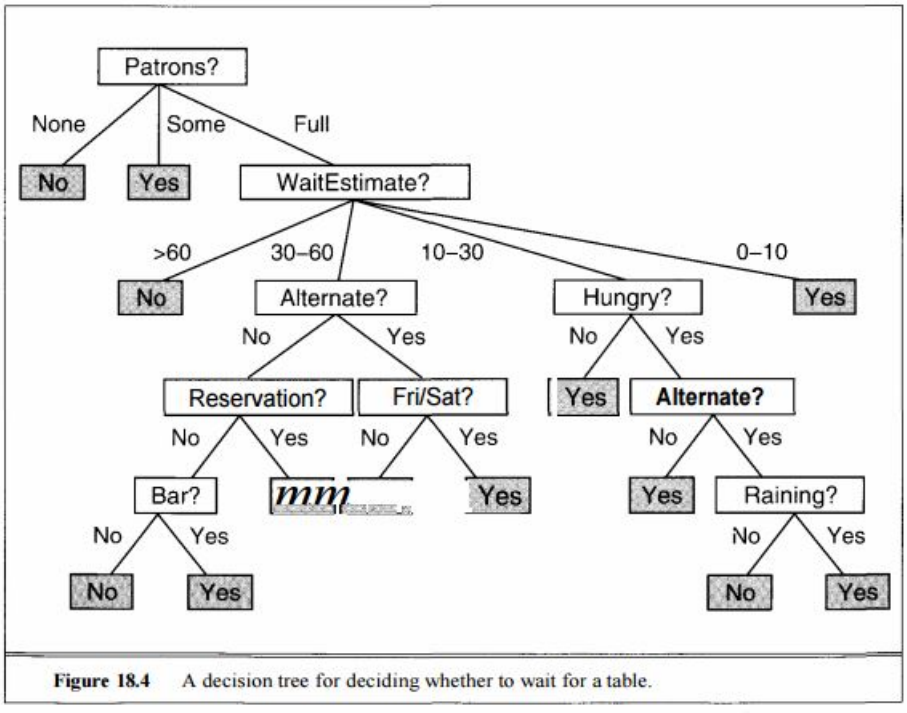


Figure 18.4 A decision tree for deciding whether to wait for a table.

[41]

5.3.1 *C4.5*:

Ο C4.5 είναι ένας αλγόριθμος δημιουργίας δέντρου αποφάσεων που αναπτύχθηκε από τον Ross Quinlan. Αποτελεί επέκταση του αλγορίθμου ID3 και είναι από τους πλέον γνωστούς αλγόριθμους στον τομέα της Εξόρυξης Δεδομένων. Ο C4.5 δημιουργεί τα δέντρο αποφάσεων από το σύνολο εκπαίδευσης με τη χρήση της έννοιας της εντροπίας πληροφοριών. Τα δεδομένα εκπαίδευσης είναι ένα σύνολο S με ήδη ταξινομημένες παρατηρήσεις. Κάθε s_i παρατήρηση αποτελείται από την κατηγορία στη οποία ανήκει και ένα διάνυσμα χαρακτηριστικών(features) (x_j). Σε κάθε κόμβο του δέντρου τα δεδομένα χωρίζονται, όπως αναφέραμε πιο πάνω, με τρόπο ώστε να διαιρούνται καλύτερα. Αυτό ο C4.5 το επιτυγχάνει με το κριτήριο κανονικοποιημένου κέρδους πληροφορίας(διαφορά εντροπίας). Αυτό επαναλαμβάνεται στους υπόλοιπους κόμβους.

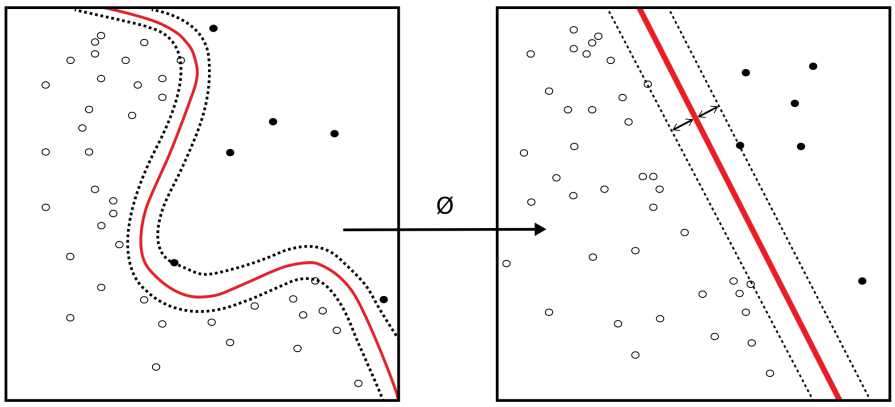
Ψευδοκώδικας:

1. Έλεγχος για βασικές περιπτώσεις.
2. Για κάθε χαρακτηριστικό Α
   α′) Βρίσκεται η κανονικοποιημένη αναλογία κέρδους πληροφορίας από διάσπαση για το Α
3. Αν το Αι είναι το καλύτερο
4. Δημιουργείται ένας κόμβος απόφασης που χωρίζεται για το Αι
5. Επανάληψη στους παραγόμενους κόμβους

[2]

5.4 *Support Vector Machines (SVM)*

Είναι αλγόριθμος ιδιαίτερα αποτελεσματικός στην κατηγοριοποίηση κειμένων που γενικά ξεπερνά τον Naive Bayes. Σε αντίθεση με τον Naive Bayes όμως δεν είναι πιθανοτικός, αλλά μεγάλου περιθωρίου. Προορίζεται για εφαρμογή σε παρατηρήσεις με διανύσματα πολλών χαρακτηριστικών. Στη περίπτωση δύο κλάσεων η βασική ιδέα είναι να βρεθεί ένα υπερεπίπεδο, που αντιπροσωπεύεται από ένα διάνυσμα Ω, το οποίο χωρίζει τα διανύσματα του εγγράφου στη μια κλάση από αυτά στην άλλη. Οι παρατηρήσεις τοποθετούνται σε ένα πολυδιάστατο χώρο διανυσμάτων των χαρακτηριστικών. Τα επίπεδα διαχωρισμού είναι άπειρα και αναζητάται ανάμεσα σε αυτά ο μέγιστος δυνατός διαχωρισμός. Αυτό το επίπεδο ονομάζεται maximum marginal hyperplane (MMH).Στους SVM δεν υπάρχει ο κίνδυνος υπερκάλυψης(overfitting) αφού η αναζήτηση αυτή αντιστοιχεί σε ένα περιορισμένο πρόβλημα βελτιστοποίησης. Έστω $c_j \epsilon \{1. - 1\}$ είναι η σωστή κλάση του εγγράφου d_j, η λύση γράφεται



Σχήμα 5.1: Kernel machines are used to compute a non-linearly separable functions into a higher dimension linearly separable function.

$\overrightarrow{w} := \sum_j a_j c_j \overrightarrow{d_j}$, $a_j \geq 0$

Όπου τα a_j λαμβάνονται λύνοντας ένα πρόβλημα διπλής βελτιστοποίησης. Τα d_j που επιτυγχάνουν τα a_j να είναι μεγαλύτερα του μηδέν ονομάζονται **support vectors**, αφού αποτελούν τα μόνα που αντιστοιχούν στο w. Η κατηγοριοποίηση τότε γίνεται απλά με την απόφαση σε ποια πλευρά του υπερεπιπέδου βρίσκεται το κάθε νέο δείγμα.

[37, 3]

5.5 Πλεονεκτήματα και μειονεκτήματα των πιο πάνω τύπων αλγορίθμων

Naive Bayes	Πολύ απλός, απλά κάνει διάφορες μετρήσεις. Αν η υπόθεση ανεξαρτησίας NB πράγματι ισχύει, ο Naive Bayes θα συγκλίνει ταχύτερα από διακριτικά μοντέλα όπως τη λογιστική παλινδρόμηση, ώστε να χρειαστεί λιγότερα δεδομένα εκπαίδευσης. Και ακόμα κι αν η υπόθεση NB δεν ισχύει, ένας κατηγοριοποιτής NB συχνά δίνει σπουδαία αποτελέσματα στην πράξη. Αποτελεί ισχυρό και γρήγορο εργαλείο. Το βασικό του μειονέκτημα είναι ότι δεν μπορεί να μάθει τις αλληλεπιδράσεις μεταξύ των χαρακτηριστικών.
Λογιστική παλιν-δρόμηση	Δεν υπάρχει η μεγάλη ανησυχία για την συσχέτιση των χαρακτηριστικών, όπως στον Naive Bayes. Επίσης έχουν μια ωραία πιθανοτική ερμηνεία, σε αντίθεση με τα δέντρα απόφασης ή τα SVMs, και μπορεί εύκολα να ενημερώνεται το μοντέλο με νέα δεδομένα (χρησιμοποιώντας μια online gradient descent method). Χρησιμοποιείται καλύτερα όταν χρειάζεται ένα πιθανοτικό πλαίσιο και όταν αναμένεται η λήψη περισσότερων δεδομένων εκπαίδευσης.
Δέντρα Α-ποφάσεων	Εύκολα στην ερμηνεία και κατανόησή τους. Μπορούν εύκολα να χειριστούν αλληλεπιδράσεις χαρακτηριστικών και είναι μη παραμετρικά, έτσι ώστε να μην επιφέρουν ανησυχίες για εσφαλμένα αποτελέσματα ή αν τα δεδομένα είναι γραμμικά διαχωρίσιμα. Ένα μειονέκτημα είναι ότι δεν υποστηρίζουν την μάθηση με δυνατότητα ενημέρωσης και πρέπει να ξαναχτιστούν από την αρχή για νέα δεδομένα. Ένα άλλο μειονέκτημα είναι ότι κάνουν εύκολα overfitting, αλλά αυτό επιλύνεται με βελτιωμένες μεθόδους όπως τον C4.5. Πλέον, όμως αποτελούν την καλύτερη επιλογή σε πολλές περιπτώσεις, συχνά ξεπερνώντας τα SVMs, είσαι γρήγορα και δεν χρειάζονται συντονισμό των παραμέτρων όπως τα SVMs.
SVM	Υψηλή ακρίβεια, ωραίες θεωρητικές εγγυήσεις σχετικά με το overfitting, και με ένα κατάλληλο πυρήνα(kernel) μπορεί να λειτουργήσει καλά, ακόμη και αν τα δεδομένα δεν είναι γραμμικά διαχωρίσιμα στο βασικό χώρο χαρακτηριστικών. Ιδιαίτερα δημοφιλής σε προβλήματα ταξινόμησης κειμένου που συνήθως αφορούν χώρους πολύ υψηλών διαστάσεων. Όμως είναι δύσκολα στην ερμηνεία, έχουν μεγάλη κατανάλωση μνήμης, και απαιτούν συνεχώς ρύθμιση για να επιφέρουν τα καλύτερα αποτελέσματα

Μέρος V

ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

6

ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ

6.1 Ανάλυση Συναισθήματος και ν-γραμ γράφοι

Σε αυτό το κεφάλαιο θα περιγράψουμε σχετικές δημοσιεύσεις που ασχολούνται με θέματα Ανάλυσης Συναισθήματος ή/και ανάλυση ν-γραμ γράφων:

6.1.1 *Thumbs up? Sentiment Classification using Machine Learning Techniques*[37]

Οι Bo Pang, Lillian Lee και Shivakumar Vaithyanathan εξετάζουν διάφορες μεθόδους εξαγωγής χαρακτηριστικών για εκπαίδευση αλγορίθμων μηχανικής μάθησης. Εκτελούν τα πειράματα τους σε ένα σύνολο δεδομένων από κριτικές ταινιών από την ιστοσελίδα IMDb. Εξετάζουν την επίδοση μηχανικών μοντέλων και μοντέλων εξαγωγής χαρακτηριστικών επιλεγμένων από ανθρώπους. Αποδεικνύουν την υπεροχή των μηχανικών μοντέλων και συνεχίζουν εξετάζοντας την μέθοδο Bag Of Words που αναλύσαμε σε προηγούμενο κεφάλαιο. Εφαρμόζουν διάφορων ειδών χαρακτηριστικά, με unigrams και bigrams. Τα οποία τροφοδοτούν στους αλγόριθμους μηχανικής μάθησης Naïve Bayes, maximum entropy και SVM. Τελικά έδειξαν ότι οι τρείς αυτοί αλγόριθμοι δίνουν καλή επίδοση με τις μηχανικές μεθόδους εξαγωγής χαρακτηριστικών. Καλύτερη ήταν η μέθοδος όπου για χαρακτηριστικό χρησιμοποιήθηκε η παρουσία ή όχι των unigrams.

Η συγκεκριμένη εργασία αποτελεί μια από τις πιο σημαντικές εργασίες που χρησιμοποίησε μηχανική μάθηση για σκοπούς ανάλυσης συναισθήματος και αποτελεί σημείο αναφοράς πλέον για τους υπόλοιπους ερευνητές.

6.1.2 *Twitter as a Corpus for Sentiment Analysis and Opinion Mining* [35]

Οι Pak Alexander και Paroubek Patrick μελετούν το γενικό πρόβλημα κατηγοριοποίησης με τρείς κλάσεις. Συλλέγουν τριακόσες χιλιάδες tweets με θετικά και αρνητικά emoticons και άλλα από λογαριασμούς εφημερίδων ως ουδέτερα. Εκτελούν ένα στάδιο προεπεξεργασίας όπου για παράδειγμα αφαιρούν τα ονόματα χρηστών, υπερσυνδέσμους, emoticons κ.α. Χρησιμοποιούν unigrams,bigrams και trigrams και δημιουργούν ένα κατηγοριοποιητή εκπαιδεύοντας δύο παραλλαγές του Naïve Bayes. Η μια χρησιμοποιεί ως χαρακτηριστικά την παρουσία ενός ν-γραμ και η άλλη χρησιμο-

ποιεί την κατανομή των μερών του λόγου(part-of-speech) για να υπολογίσει την εκ των υστέρων πιθανότητα του μοντέλου NB. Θεωρώντας αυτές τις κατανομές ανεξάρτητες υπολογίζεται ο λογάριθμος πιθανοφάνιας του κάθε συναισθήματος. Δοκιμάστηκαν και κατηγοριοποιητές όπως SVM και Conditional Random Forests(CRF) αλλά έδωσαν χαμηλότερο αποτέλεσμα από τον προηγούμενο. Στα πειράματα τους η χρήση bigrams έδωσε τα καλύτερα αποτελέσματα το οποίο εξήγησαν ως την καλή ισορροπία μεταξύ κάλυψης εύρους και ικανότητας αναγνώρισης συναισθηματικών μοτίβων.

Αυτή η εργασία επιβεβαίωσε την καταλληλόλητα των μικροιστολογίων όπως το Twitter ως πηγές δεδομένων για σκοπούς ανάλυσης συναισθήματος.

6.1.3 *Twitter sentiment classification using distant supervision*[30]

Οι Go et al. υπήρξαν από τους πρώτους που μελέτησαν την ανάλυση συναισθήματος σε δεδομένα από το Twitter. Ασχολούνται με το πρόβλημα κατηγοριοποίησης δύο κλάσεων (αρνητικά, θετικά). Λόγων της έλλειψης σύνολο δεδομένων χειροκίνητα κατηγοριοποιημένων, εφαρμόζουν την τεχνική της εξ αποστάσεως επίβλεψης για να εκπαιδεύσουν κάποιο κατηγοριοποιητή. Παίρνουν tweets τα οποία ταξινομούν σε κλάση συναισθήματος με βάση τα emoticons, τα οποία στη συνέχεια διαγράφουν. Έτσι προκύπτει ένα σύνολο 1.6 εκατομμυρίων tweets ισοκατανεμημένων σε αρνητικά και θετικά. Εφαρμόζουν ένα στάδιο προεπεξεργασίας για μείωση του θορύβου. Για εξαγωγή χαρακτηριστικών χρησιμοποιούν bigrams, unigrams, συνδυασμό αυτών και POS tags. Για κατηγοριοποιητές χρησιμοποιούν τους Naive Bayes, maximum entropy και SVM. Ο SVM με χρήση unigrams δίνει την καλύτερη επίδοση 82,9%. Παρατηρούν πως η προσθήκη των bigrams στο διάνυσμα χαρακτηριστικών βελτιώνει την επίδοση των Naive Bayes και maximum entropy αλλά όχι των SVM. Τέλος, καταλήγουν στο συμπέρασμα ότι προσθέτοντας την άρνηση ως ξεχωριστό χαρακτηριστικό καθώς και τα POS tags δεν παρατηρείται βελτίωση ενώ η χρήση μόνο των bigrams οδηγεί σε χειρότερα αποτελέσματα εξαιτίας του αραιού χώρου χαρακτηριστικών (feature-space).

6.1.4 *Comparing methods for twitter sentiment analysis* [38]

Οι Ευάγγελος Ψωμακέλης, Κωνσταντίνος Τσερπές και Δημοσθένης Αναγνωστόπουλος ασχολούνται με το πρόβλημα στης ανάλυσης συναισθήματος σε σύνολο δεδομένων παρμένων από το Twitter. Εξέταζουν την απόδοση που έχουν μέθοδοι αναπαράστασης των δεδομένων για τροφοδότηση μηχανισμών της ανάλυσης συναισθήματος, όπως bag of words, n-grams και n-gram graphs. Τους εφαρμόζουν για να εξετάσουν την απόδοση λεξικο-βασισμένων ή βασισμένων στη μάθηση μεθόδων κατηγοριοποίησης και συνδυασμών τους. Τα αποτελέσματα δείχνουν ότι οι

μέθοδοι με χρήση n-grams και n-gram graphs δίνουν τα καλύτερα αποτελέσματα. Ειδικά στην περίπτωση των συνδυασμένων μεθόδων με χρήση n-grams φτάνουν μέχρι και 83,15% ακρίβεια πρόβλεψης, ενώ οι μέθοδοι με χρήση n-gram graphs φτάνουν μέχρι και 94,52%. Για την εκτέλεση των δοκιμών τους χρησιμοποίησαν ένα σύνολο 4451 χειροκίνητα σχολιασμένων tweets. Αρχικά εφάρμοσαν τεχνικές προεπεξεργασίας των δεδομένων για την μείωση του θορύβου που προκαλείται από νεολογισμούς, αναφορές σε διευθύνσεις ιστοσελίδων(URLs) και τη χρήση εικονιδίων emoticons. Οι λεξικο-βασισμένες μεθόδοι που χρησιμοποίησαν αποτελούν κατηγοριοποιητές οι οποίοι προσπαθούν να ανιχνεύσουν το συναίσθημα ενός κειμένου υπολογίζοντας το συναίσθημα των λέξεων του με χρήση βαθμονομημένου λεξικού. Για τις τεχνικές μηχανικής μάθησης χρησιμοποίησαν γνωστούς κατηγοριοποιητές όπως Support Vector Machines (SVMs), Naïve Bayesian Networks, C4.5, Functional Trees, Best-First Trees, Multilayer Perceptrons και Logistic Regression.

Ακολουθεί η γραφική παράσταση με τα αποτελέσματα της εν λόγω δημοσίευσης:

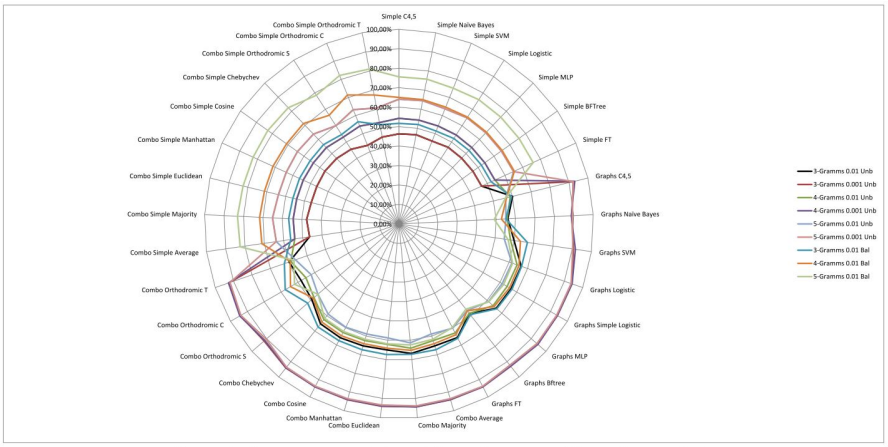


Figure 1: *Summary presentation of the performance of the various methods*

Σχήμα 6.1: Αποτελέσματα της εργασίας Comparing methods for twitter sentiment analysis

[38]

Αυτό αποτέλεσε την έμπνευση για την διπλωματική μου εργασία, με σκοπό να εξεταστεί αν μπορούν τέτοιες μέθοδοι να εφαρμοστούν γρήγορα ώστε να εκτελούνται σε πραγματικό χρόνο και με κατεύθυνε στην εξέταση μόνο της αποδοτικότερης μεθόδου αυτών, τους ν-γραμ γράφους. Ένα σημαντικό πρόβλημα που παρουσιάζεται κατα την εργασία αυτή είναι ποιά θα ήταν η καταλληλότερη αναπαράσταση των γράφων που προκύπτουν ώστε να είναι δυνατή τη επεξεργασία τους ανεξαρτήτως του μεγέθους.

6.1.5 *Efficient in-memory data structures for n-grams indexing.*[40]

Οι Daniel Robenek, Jan Platos, και Vaclav Snasel εξετάζουν δομές αναπαράστασης των n-grams ώστε να επιτύχουν τη μέγιστη δυνατή αποδοτικότητα ταχύτητας και κατανάλωσης πόρων. Οι δομές προς εξέταση είναι οι πίνακες κατακερματισμού(hash tables), B+ δέντρα, ternary AVL δέντρα, υβριδικά AVL δέντρα και διπλά ternary search AVL δέντρα.

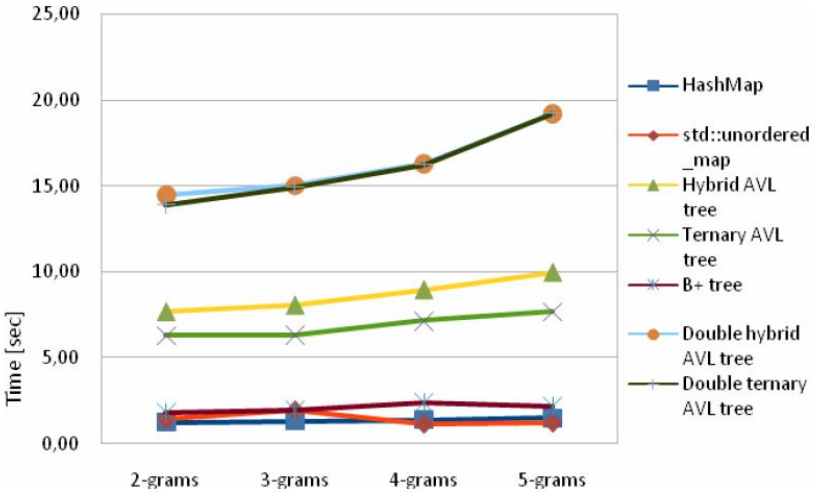


Fig. 1. Insert time comparison

Σχήμα 6.2: Αποτελέσματα της εργασίας Efficient in-memory data structures for n-grams indexing.1

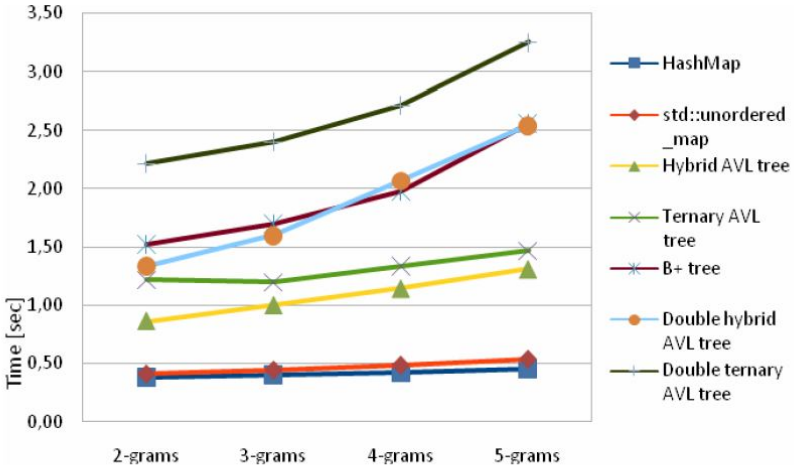


Fig. 2. Comparison of search time

Σχήμα 6.3: Αποτελέσματα της εργασίας Efficient in-memory data structures for n-grams indexing.2

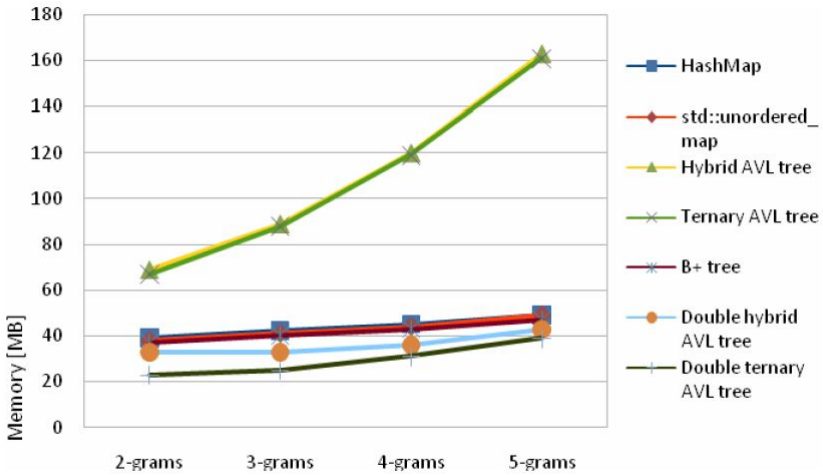


Fig. 3. Comparison of memory consumption

Σχήμα 6.4: Αποτελέσματα της εργασίας Efficient in-memory data structures for n-grams indexing.3

Εδώ βλέπουμε τα αποτελέσματα της έρευνας όπου οι διάφορες δομές εμφανίζουν πλεονεκτήματα και μειονεκτήματα στις τρείς μετρικές απόδοσης. Φαινομενικά η δομή HashTable δίνει και στις τρείς αυτές μετρικές από τις καλύτερες αποδόσεις. Το μόνο πρόβλημα που πιθανόν να εμφανιστεί είναι σε περίπτωση πολύ μεγάλου όγκου δεδομένων όπου θα παρατηρηθεί υπερχείλιση στον κατακερματισμό με αποτέλεσμα η αναζήτηση και εισαγωγή στοιχείου να τείνει σε γραμμική πολυπλοκότητα. Σε αυτή την περίπτωση προτείνεται μια λύση εφαρμογής HashTable σαν πρώτο επίπεδο και στη συνέχεια χρήση κάποιου ternary tree για την διαχείριση της υπερχείλισης. Προφανώς η αποδοτικότητα κάθε τέτοιου αλγορίθμου εξαρτάται από την υλοποίηση και το μέγεθος των δεδομένων. Αν τα δεδομένα είναι αδύνατον να αποθηκευτούν στη κύρια μνήμη η επιλογή κατάλληλης δομής αποθήκευσης σε αρχείο αποτελεί ένα νέο πρόβλημα.

6.1.6 *Content vs. Context for Sentiment Analysis: a Comparative Analysis over Microblogs:*[21]

Οι Φώτης Αίσοπος, Γιώργος Παπαδάκης και Κωνσταντίνος Τσερπές εξετάζουν μεθόδους βασισμένες στο περιεχόμενο και μεθόδους βασισμένες στα συμφραζόμενα για μεγιστοποίηση της αποδοτικότητας της κατηγοριοποίησης. Συγκεκριμένα για τις context-based χρησιμοποιούν n-gram graphs και προτείνουν μια νέα προσέγγιση επιλογής χαρακτηριστικών για χρήση στους κατηγοριοποιητές. Αφού λάβουν τις τρείς μετρικές απόστασής, που αναλύσαμε σε προηγούμενο κεφάλαιο, Containment, Value και Normalized Value Similarity εφαρμόζουν μια μέθοδο διακριτοποίησης τους συγκρίνοντας σε ζευγάρια τις ίδιες μετρικές για τις διαφορετικές κλάσεις αντίστοιχα. Τελικά τα αποτελέσματα που προκύπτουν είναι πάρα

πολύ καλά φτάνοντας μέχρι και 98,76% ευκρίνεια πρόβλεψης για 4-gram graphs με τον κατηγοριοποιητή C4.5.

Στην εργασία αυτή δεν αναφέρεται η κατανάλωση χρόνου και μνήμης για τα διάφορα πειράματα. Θα προσπαθήσουμε να εξετάσουμε κατά πόσον εφαρμογή λιγότερων χαρακτηριστικών και κατάλληλου κατηγοριοποιητή μπορεί να δώσει σε λιγότερο χρόνο και μνήμη μικρή μείωση της ευκρίνειας.

6.2 Ροές Δεδομένων

Εδώ θα παρουσιάσουμε έρευνες που αναφέρονται σε επεξεργασία ροών δεδομένων(data streams) και πως αυτή επηρεάζει την ανάλυση συναισθήματος:

6.2.1 *Employing traditional machine learning algorithms for big data streams analysis: The case of object trajectory prediction*

Αρχικά να αναφερθούμε στην εργασία των Valsamis et al. [43] οι οποίοι εξετάζουν την μοντελοποίηση της τροχιάς θαλάσσιων σκαφών. Πρόκειται για ένα πρόβλημα το οποίο πρέπει να επιλυθεί λαμβάνοντας τεράστιο όγκο δεδομένων σε μορφή ροής. Αυτό συνεπάγεται ότι για τον σκοπό ανάπτυξης ενός συστήματος που θα επεξεργάζεται τα δεδομένα σε πραγματικό χρόνο είναι αναγκαία η διερεύνηση των παραχωρήσεων μεταξύ ακρίβειας, απόδοσης και αξιοποίησης των πόρων. Επικεντρωνόμαστε στην ανάλυση τους για τις μεθόδους επεξεργασίας ροών όπου αναφέρουν ότι:

Τα εισερχόμενα δεδομένα θα πρέπει να υποβάλλονται σε επεξεργασία αμέσως μόλις καταστούν διαθέσιμα, αφού η υψηλή ταχύτητα και ο όγκος τους καθιστά αδύνατη την αποθήκευση σε μονάδες αποθήκευσης (όπως μια συμβατική βάση δεδομένων) και την επεξεργασία τους όταν τα χρειαστούμε.

Υπάρχουν δύο λογικές μέθοδοι επεξεργασίας ροών δεδομένων:

Περίληψη ροής: Διατήρηση μόνο ενός μικρού αριθμού από τα στοιχεία που χαρακτηρίζουν τη ροή δεδομένων (και συχνά την εφαρμογή προς αντιμετώπιση). Έτσι απαιτείται πολύ μικρότερος χώρος και χρόνος επεξεργασίας.

Επεξεργασία παραθύρων σταθερού μήκους: Σταθερά τμήματα των δεδομένων φορτώνονται σε μια buffer μνήμη και εφαρμόζονται αλγόριθμοι μηχανική μάθησης μόνο σε αυτά, λαμβάνοντας, ίσως, υπόψη τις προηγούμενες επιδόσεις για σκοπούς βελτιστοποίησης.

Από την άποψη της αρχιτεκτονικής, και στις δύο παραπάνω περιπτώσεις, ο επεξεργαστής ρεύματος διατηρεί στη μνήμη δεδομένα μόνο διαχειρίσιμου μεγέθους, κάτι που καθορίζεται από τις απαιτήσεις της εφαρμογής, από τους χρονικούς περιορισμούς, την αναμενόμενη ακρίβεια του αποτελέσματος, καθώς και τους διαθέσιμων πόρους.

6.2.2 *Sentiment Knowledge Discovery in Twitter Streaming Data*[25]

Οι Albert Bifet και Eibe Frank εξετάζουν τις ροές δεδομένων(data streams) που παρέχει το Twitter, τα προβλήματα που αυτές επιφέρουν στην κατηγοριοποίηση και σε εφαρμογές εξόρυξης γνώμης και ανάλυσης συναισθήματος.

Το Twitter χρησιμοποιεί το μοντέλο ροής δεδομένων. Αυτό έχει ως αποτέλεσμα, λόγο της μεγάλης ταχύτητας δεδομένων, οι αλγόριθμοι εξόρυξης δεδομένων να πρέπει να έχουν την δυνατότητα να προβλέψουν σε πραγματικό χρόνο και υπό αυστηρούς περιορισμούς χώρου και χρόνου. Επιπλέον, θα πρέπει να είναι σε θέση να χειριστούν δεδομένα των οποίων η φύση ή η κατανομή παρουσιάζει αλλαγές με τη πάροδο του χρόνου.

Εξετάζουν τον τρόπο λήψης ροής δεδομένων με το Twitter Streaming API και στη συνέχεια αναφέρουν τις μεθόδους κατηγοριοποίησης των Tweets(π.χ. από τα emoticons) ώστε να χρησιμοποιηθούν ως δεδομένα στη μάθηση.

Αναφέρουν την πιο συνηθισμένη μετρική για εξόρυξη ροής δεδομένων, prequential accuracy, η οποία υπολογίζεται σταδιακά με κάθε νέο παράδειγμα να εξετάζεται από το μοντέλο μάθησης πριν την χρήση του στην εκπαίδευση. Υποστηρίζουν ότι η μετρική αυτή είναι κατάλληλη μόνο όταν όλες οι κλάσεις είναι ισορροπημένες και έχουν σχεδόν τον ίδιο αριθμό παραδειγμάτων. Ακολούθως προτείνουν την στατιστική Kappa κυλιόμενου παραθύρου ως πιο ευαίσθητο μέτρο για την ποσοτικοποίηση της απόδοσης πρόβλεψης κατηγοριοποιητών ροής.

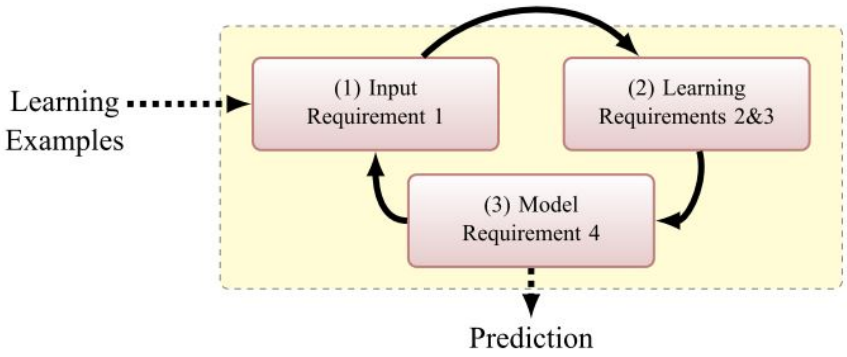
Για τα πειράματα τους χρησιμοποιούν τρεις γρήγορες σταδιακές μεθόδους: πολύωνυμικό Naive Bayes, κάθοδο στοχαστικής κλίσης(stochastic gradient descent-SGD) και δέντρα Hoeffding. Για τα δεδομένα χρησιμοποίησαν τα εργαλεία Weka και MOA. Τα σύνολα δεδομένων τους είναι το Stanford Twitter Sentiment των Go et al. και το Edinburgh Twitter Corpus των Petrovic et al. Κατά τα πειράματα τους οι αλγόριθμοι φτάνουν αποτελέσματα μέχρι και 82.45 % στο πρώτο dataset με χρήση Naive Bayes και 86.26 % στο δεύτερο σύνολο με χρήση του SGD. Παρατηρούν πως οι Naive Bayes και SGD παρουσιάζουν παρόμοια ποσοστά ακρίβειας σε αντίθεση με τα δένδρα Hoeffding τα οποία υστερούν συστηματικά. Αυτό συμβαίνει λόγω του ότι η υλοποίηση τους δεν χρησιμοποιεί αραιά Instances. Έτσι υποστηρίζουν πως οι δεντρικοί κατηγοριοποιητές υστερούν σε τέτοιες εφαρμογές.

Ταυτόχρονα υποδεικνύουν την αναγκαιότητα της στατιστικής Kappa σε περιπτώσεις όπου η συνήθης ακρίβεια δεν αντιλαμβανόταν τις μεταβολές των κατανομών. Επίσης προτείνουν ως καλύτερο αλγόριθμο κατηγοριοποίησης για ροές δεδομένων τον SGD, εφόσον έχει επιλεχθεί κατάλληλα ο ρυθμός μάθησης.

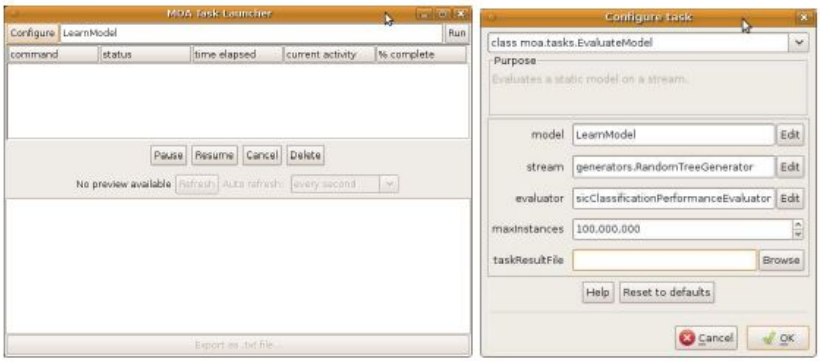
6.2.3 *MOA: Massive Online Analysis*[26]

Οι Bifet et al. σε αυτή τους τη δημοσίευση παρουσιάζουν το εργαλείο MOA: Massive Online Analysis. Πρόκειται για ένα περιβάλλον λογισμικού για την υλοποίηση αλγορίθμων και την εκτέλεση πειράματων για την online μάθηση από εξελισσόμενες ροές δεδομένων. Το MOA περιλαμβάνει μια συλλογή από online και offline μεθόδους, καθώς και εργαλεία για αξιολόγηση. Ειδικότερα, θέτει σε εφαρμογή ενίσχυση, ενσάκιση, και δέντρα Hoeffding, όλα με και χωρίς Naive Bayes ταξινομητή στα φύλλα. Επίσης το MOA υποστηρίζει αμφίδρομη αλληλεπίδραση με το εργαλείο WEKA, και είναι ελεύθερο λογισμικό υπό την άδεια GNU GPL.

Στο λεγόμενο green computing οι αλγόριθμοι χαρακτηρίζονται από την αποδοτικότητά τους. Στην περίπτωση των αλγόριθμων που χειρίζονται ροές δεδομένων η επεξεργασία πρέπει να γίνεται με πολύ αυστηρούς περιορισμούς χώρου και χρόνου. Οι διαφορές μεταξύ επεξεργασίας δέσμης και ροής δεδομένων είναι ότι τα παραδείγματα επεξεργάζονται μόνο μια φορά, η διαθέσιμη μνήμη και χρόνος είναι περιορισμένα και μπορεί να κάνει προβλέψεις ανά πάσα στιγμή.



Σχήμα 6.5: The data stream classification cycle.



Σχήμα 6.6: MOA GUI.

Στη συνέχεια αναλύουν το πρόβλημα που παρουσιάζουν οι μέθοδοι επεξεργασίας ροών. Είναι αναγκαία η εύρεση μια μετρικής απόδοσης στο χρόνο. Οι δύο καθιερωμένες μέθοδοι είναι η holdout και η Interleaved Test-Then-Train or Prequential. Η πρώτη χρησιμοποιείται όταν η διαίρεση των δεδομένων σε εκπαίδευσης και αξιολόγησης είναι ξεκάθαρη και προκαθορισμένη. Εφαρμόζεται έλεγχος της αποδοτικότητας σε ένα μόνο σύνολο holdout ώστε να μπορούν να συγκριθούν διάφορες μέθοδοι ασχέτως, όσο γίνεται, της ροής εισόδου. Η δεύτερη κάνει έλεγχο κάθε παραδείγματος καθώς αυτό εισέρχεται, πριν το προσθέσει στην εκπαίδευση του μοντέλου. Έτσι αξιοποιείται όλο το σύνολο δεδομένων και εμφανίζεται μια ομαλή παράσταση της αποδοτικότητας στο χρόνο.

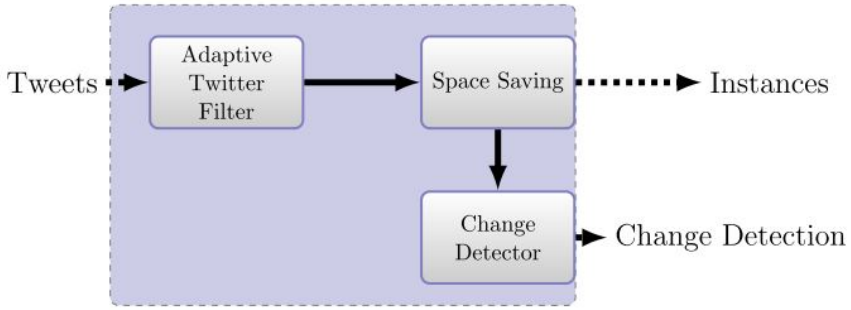
Όμως οι κλασικές μέθοδοι δεν επιτρέπουν την διαχείριση μεγάλων ροών δεδομένων. Αντιθέτως το εργαλείο MOA προδιαγράφει την ικανότητα διαχείρισης της τάξης δεκάδων εκατομμυρίων παραδειγμάτων ανάλογα με την διαθέσιμη μνήμη.

Τέλος το MOA μπορεί να εκτελεστεί σε πολλές πλατφόρμες λόγω της χρήσης JVM και παρέχει γραφικό περιβάλλον και περιβάλλον γραμμής εντολών. Παρέχει πλειάδα γεννητριών δεδομένων με τις οποίες κάποιος μπορεί να ενώνει πολλές ροές δεδομένων. Επίσης παρέχει μεγάλη γκάμα κατηγοριοποιητών(π.χ. Naive Bayes, Hoeffding Tree κ.α.). Επίσης υπάρχει πλούσια συλλογή παραδειγμάτων τεκμηρίωσης του ανοιχτού κώδικα ώστε να μπορεί κάποιος να το χρησιμοποιήσει εύκολα και να γράψει νέους κατηγοριοποιητές.

6.2.4 *Detecting Sentiment Change in Twitter Streaming Data*[27]

Οι Bifet et al. συνεχίζουν την ανάπτυξη του λογισμικού MOA με την παρουσίαση της επέκτασης MOA TweetReader. Πρόκειται για ένα σύστημα που διαβάζει tweets σε πραγματικό χρόνο, για την ανίχνευση όρων των οποίων η συχνότητα αλλάζει. Παρουσιάζουν τα χαρακτηριστικά του Twitter, που το καθιστούν μοναδικό στο είδος του και ιδανική πλατφόρμα για εξόρυξη δεδομένων αφού παρέχει πληροφορίες για το τι σκέφτονται και τι κάνουν οι άνθρωποι καθημερινά.

Το Twitter παρέχει τη ροή δεδομένων Firehose στους προγραμματιστές. Έτσι οι Bifet et al. έχουν σκοπό την ανάπτυξη ενός συστήματος που θα επεξεργάζεται τη ροή αυτή, να κατηγοριοποιεί τα μηνύματα και να αντιλαμβάνεται τις αλλαγές σε πραγματικό χρόνο.



Σχήμα 6.7: Διάγραμμα λειτουργίας του MOA TweetReader.

Στη συνέχεια παρουσιάζουν το Twitter Streaming API, το οποίο είναι πολύ απλό στη χρήση αφού βασίζεται σε ερωτήματα HTML και επιστρέφει τα δεδομένα σε αρχεία XML και JSON. Έτσι το MOA TweetReader μπορεί να διαβάζει ζωντανά τα Tweets, να τα προ επεξεργάζεται και να τα μετατρέπει με ένα TF-IDF φίλτρο σε διανύσματα ιδιοτήτων για σκοπούς μηχανικής μάθησης. Το δεύτερο συστατικό του συστήματος είναι ένας frequent item miner που αποθηκεύει τη συχνότητα των πιο συχνών όρων. Τέλος, ένας ανιχνευτής αλλαγών παρακολουθεί τις αλλαγές στις συχνότητες των όρων.

Αυτά εκτελούνται φυσικά έχωντας υπόψη την διαθέσιμη μνήμη, αποθηκεύοντας μόνο τα K στοιχεία που μπορούν να χωρέσουν με τη χρήση μια ουράς με τα K πιο συχνά στοιχεία. Έτσι το σφάλμα εκτίμησης των συχνοτήτων στοιχείου είναι το πολύ N / K, όπου N είναι ο αριθμός των στοιχείων στη ροή.

Για να εξετάσουν την αποτελεσματικότητα του προτεινόμενου συστήματος σε πραγματικά γεγονότα αποφάσισαν να αναφερθούν στην κρίση της Toyota του 2010. Στην περίοδο αυτή η εταιρεία δεν μπόρεσε έγγαιρα να αντιληφθεί το πρόβλημα που προκλήθηκε από τα ελαττωματικά πετάλια επιτάχυνσης(accelerator pedals). Με τη χρήση του συνόλου δεδομένων των Petrovic et al. από το πανεπιστήμιο του Εδιμβούργου που συλλέθηκε στην συγκεκριμένη χρονική περίοδο, φίλτραραν τα tweets που αναφέρονταν στην Toyota. Έτσι με το MOA TweetReader ανίχνευσαν τις αλλαγές στις συχνότητες σημαντικών όρων. Στη συνέχεια εκπαίδευσαν ένα δέντρο Hoeffding, που χρησιμοποιεί μια στρατηγική προ-κλαδέματος με βάση το όριο Hoeffding ώστε σταδιακά να αναπτυχθεί ένα δέντρο απόφασης. Ένας κόμβος επεκτείνεται με διάσπαση μόλις υπάρξουν επαρκή στατιστικά στοιχεία, με βάση το ανεξάρτητο της κατανομής όριο Hoeffding.

Τελικά φαίνεται ξεκάθαρα πως τα θετικά σχόλια για την Toyota μειώθηκαν κατά 50% και οι πιο γνωστοί όροι κατά την μεταβολή αυτή ήταν οι “gas” και “pedal”.

6.3 Εργαλεία

Σε αυτό σημείο θα αναλύσουμε εργαλεία που χρησιμοποιούνται από επιστήμονες για εκτέλεση αλγορίθμων πάνω σε κατανεμημένα συστήματα. Τα κατανεμημένα συστήματα χρησιμεύουν όταν ένα υπολογιστικό πρόβλημα μπορεί να χωριστεί σε πολλά καθήκοντα, καθένα από τα οποία λύνεται με έναν ή περισσότερους υπολογιστές, που επικοινωνούν μεταξύ τους με ανταλλαγή μηνυμάτων. Ένα από τα πιο γνωστά συστήματα κατανεμημένης επεξεργασίας είναι το Apache Hadoop. Για τους σκοπούς της ανάλυσης συναισθήματος υπάρχουν επίσης πολλά εργαλεία που βασίζονται στο Hadoop framework όπως τα Apache Spark, Mahout και Storm. Αυτά θα μπορούσαν να υποβοηθήσουν την αναγωγή των κλασσικών αλγόριθμων σε αλγόριθμους για κατανεμημένα συστήματα.

6.3.1 *Apache Hadoop*

Το Hadoop είναι ένα framework ανοιχτού κώδικα που αναπτύσσεται από το Apache Software Foundation και υποστηρίζει κατανεμημένη επεξεργασία μεγάλου όγκου δεδομένων. Η ανάπτυξη του ξεκίνησε το 2005 ως μια εναλλακτική επιλογή ανοιχτού κώδικα του Google Map Reduce framework και το Google File System (GFS) στα οποία και βασίστηκε και έκτοτε αποτελεί ένα από τα καλύτερα κατανεμημένα συστήματα για batch processing.

Το Apache Hadoop framework αποτελείται από τα εξής τμήματα:

Hadoop Common: Περιέχει βιβλιοθήκες και εργαλεία απαραίτητα για τη λειτουργία των υπόλοιπων τμημάτων.

Hadoop Distributed File System (HDFS): Κατανεμημένο, κλιμακώσιμο (scalable) σύστημα αρχείων υψηλής απόδοσης.

Hadoop YARN: Πλατφόρμα υπεύθυνη για τη διαχείριση των υπολογιστικών πόρων και υπεύθυνη για τη χρονοδρομολόγηση των εργασιών του χρήστη. Ενσωματώθηκε στο Hadoop framework στη έκδοση του Hadoop 2.0.

Hadoop MapReduce: Προγραμματιστικό μοντέλο για την παράλληλη επεξεργασία μεγάλου όγκου δεδομένων.

Ένας από τους κυριότερους λόγους της επιτυχίας αυτού του framework είναι το γεγονός ότι σχεδιάστηκε να τρέχει σε συστάδες από απλούς υπολογιστές (clusters) χαμηλού κόστους όπου όμως η συχνότητα βλαβών είναι υψηλή. Οι βλάβες αντιμετωπίζονται αυτόματα από το framework εξασφαλίζοντας τη σταθερότητα του συστήματος, ενώ παράλληλα η διαθεσιμότητα (availability) των δεδομένων από το HDFS δεν επηρεάζεται.

Το HDFS επιτυγχάνει να είναι αξιόπιστο, αποθηκεύοντας τα δεδομένα και αντίγραφα αυτών σε περισσότερους από ένα κόμβους. Συγκεκριμένα, ο

χώρος των αρχείων είναι ενιαίος για όλο το cluster με τα αρχεία να διασπώνται σε blocks με κάθε block να αντιγράφεται σε πολλαπλούς κόμβους δεδομένων. Μεταξύ των κόμβων υπάρχει επικοινωνία για εξισορρόπηση των δεδομένων, αντιγραφή ή μετακίνηση τους, έτσι ώστε να παραμείνει ψηλός ο λόγος αντιγραφής (replication). Μοναδικό σημείο αποτυχίας του HDFS (single point of failure) αποτελεί ο κεντρικός κόμβος, ο name node, για τον οποίο όμως υπάρχει αντίγραφο στον secondary namenode, ώστε να είναι δυνατή η επαναφορά σε κάθε περίπτωση βλάβης.

[9]

6.3.2 *Apache Mahout*

Είναι ένα εργαλείο για παραγωγή ελεύθερων υλοποιήσεων κατανεμημένων ή γενικά κλιμακώσιμων αλγόριθμων μηχανικής μάθησης κυρίως στους τομείς του συνεργατικού φιλτραρίσματος(collaborative filtering), ομαδοποίηση και κατηγοριοποίηση. Πολλές από τις εφαρμογές χρησιμοποιούν την πλατφόρμα Apache Hadoop. Επίσης παρέχει βιβλιοθήκες Java για συνήθης μαθηματικές πράξεις και πρωτόγονες συλλογές Java. Το Mahout είναι ένα έργο που βρίσκεται ακόμη υπό ανάπτυξη.

Ενώ οι βασικοί αλγόριθμοι για ομαδοποίηση, κατηγοριοποίηση και συνεργατικού φιλτραρίσματος υλοποιούνται πάνω στο Apache Hadoop χρησιμοποιώντας το μοντέλο map/reduce, δεν είναι αναγκαίο. Συνεισφορές που τρέχουν σε ένα ενιαίο κόμβο ή σε μη Hadoop cluster είναι επίσης ευπρόσδεκτες.

[10]

6.3.3 *Apache Kafka*

Το Apache Kafka αποτελεί ένα "message broker". Είναι ένα project ανοιχτού κώδικα το οποίο σχεδιάστηκε από την LinkedIn ως ένα κατανεμημένο σύστημα μηνυμάτων (messaging) για συλλογή και μεταφορά Log αρχείων με ελάχιστη καθυστέρηση(low latency) και high-throughput σε πραγματικό χρόνο. Πλέον έχει αναλάβει την ανάπτυξή του η Apache. Το Apache Kafka βρίσκει χρήση στο messaging καθώς χαρακτηριστικά όπως high throughput, τεμάχιση αρχείων (built-in partitioning), διατήρηση αντιγράφων (replication) και η ανοχή σε σφάλματα (fault-tolerance) το κάνουν ιδανική λύση σε εφαρμογές επεξεργασίας μηνυμάτων (message processing applications) μεγάλης κλίμακας. Συχνά χρησιμοποιείται για παρακολούθηση (monitoring) δεδομένων συγκεντρώνοντας στατιστικά και δεδομένα χρήσης από κατανεμημένες εφαρμογές. Μέσω του Kafka είναι δυνατό οι λεπτομέρειες που αφορούν τα αρχεία των log να αφαιρεθούν και να αποσταλεί μια πιο "καθαρή" εκδοχή των log ή των event data ως ρεύμα (stream) μηνυμάτων. Άλλες εφαρμογές του Kafka

είναι η επεξεργασία ρευμάτων (stream processing), event sourcing, παρακολούθηση δραστηριότητας web εφαρμογών κτλ.

[11, 12]

6.3.4 *Apache Spark*

Το Apache Spark Είναι ένα cluster computing framework ανοιχτού κώδικα. Αρχικά αναπτύχθηκε στο Πανεπιστήμιο της Καλιφόρνια, Berkeley AMPLab, και αργότερα δωρίστηκε στην Apache Software Foundation. Το Spark παρέχει μια διεπαφή για τον προγραμματισμό ολόκληρων συστάδων με σιωπηρό παραλληλισμό δεδομένων και ανοχή σε σφάλματα.

Παρέχει στους προγραμματιστές μια διεπαφή προγραμματισμού εφαρμογών που επικεντρώνεται σε μια δομή δεδομένων που ονομάζεται το ελαστικό κατανεμημένο σύνολο δεδομένων (resilient distributed dataset-RDD), ένα μόνο αναγνώσιμο πολυσύνολο από στοιχεία δεδομένων που διανέμονται σε μια συστάδα μηχανών, η οποία διατηρείται με ανοχή βλάβης. Αναπτύχθηκε ως απάντηση στους περιορισμούς του παραδείγματος cluster MapReduce, η οποία αναγκάζει μια συγκεκριμένη δομή γραμμικής ροής δεδομένων επί των κατανεμημένων προγραμμάτων. Τα RDDs λειτουργούν ως ένα σύνολο εργασιών για κατανεμημένα προγράμματα που προσφέρουν μια περιορισμένη μορφή κατανεμημένης κοινής μνήμης.

Το Spark βελτιώνει την απόδοση υλοποίησης επαναλαμβανόμενων αλγορίθμων, όπως για παράδειγμα οι αλγόριθμοι εκπαίδευσης στη μηχανική μάθηση, σε σχέση με υλοποιήσεις MapReduce όπως το Hadoop με αρκετές τάξεις μεγέθους.

Τέλος το Spark απαιτεί ένα διαχειριστή συστάδας και ένα κατανεμημένο σύστημα αποθήκευσης. Για τη διαχείριση της συστάδας, υποστηρίζει ένα αυτόνομο σύστημα(native Spark cluster), Hadoop YARN, ή Apache Mesos. Για το κατανεμημένο σύστημα αποθήκευσης, το Spark μπορεί να συνδεθεί με ένα ευρύ φάσμα, όπως Hadoop Distributed File System (HDFS), Cassandra, HBase, Hive, Tachyon, ή οποιαδήποτε πηγή δεδομένων Hadoop.

[13, 14]

6.3.5 *Apache Spark Streaming*

Το Spark Streaming αξιοποιεί τη γρήγορη ικανότητα προγραμματισμού του Spark Core για να εκτελέσει stream analytics. Λαμβάνει δεδομένα σε mini-batches και εκτελεί μετασχηματισμούς RDD σε αυτά. Αυτό επιτρέπει στο ίδιο σύνολο από κώδικα εφαρμογών που γράφτηκε για batch analytics να χρησιμοποιηθεί σε stream analytics, διευκολύνοντας έτσι την εφαρμογή της αρχιτεκτονικής λάμδα. Ωστόσο, αυτή η ευκολία έρχεται

με την ποινή μιας καθυστέρησης ίσης με τη διάρκεια του mini-batch. Αντιθέτως άλλα συστήματα δεδομένων ροής που επεξεργάζονται κάθε συμβάν με τη σειρά και όχι σε mini-batches, όπως το Storm και το Flink. Ταυτόχρονα χωρίς την ανάγκη επιπλέον κώδικα επαναφέρει την χαμένη δουλειά και την κατάσταση των φορέων(π.χ. συρόμενο παράθυρο).

[13, 15]

6.3.6 *Apache Storm*

Το Apache Storm είναι ένα κατανεμημένο σύστημα που επιτρέπει την ανάλυση και επεξεργασία δεδομένων σε πραγματικό χρόνο. Σχεδιάστηκε αρχικά από την Twitter Inc. ενώ στη συνέχεια ανέλαβε την ανάπτυξη του η Apache. Έχει υψηλή ανοχή σε σφάλματα (high fault-tolerance) και είναι κατακόρυφα κλιμακώσιμο (horizontal scaling) στην ποσότητα δεδομένων εισόδου.

Το Storm για να κάνει υπολογισμούς σε πραγματικό χρόνο υλοποιεί τοπολογίες (topologies). Κάθε κόμβος σε μια τοπολογία επεξεργάζεται μερικώς τα δεδομένα, ενώ συσχετίσεις μεταξύ των κόμβων καθορίζουν σε ποιους κόμβους και με ποια σειρά θα μεταβούν τα δεδομένα ώστε να ολοκληρωθεί η επεξεργασία. Πρόκειται δηλαδή για ένα μοντέλο ροής δεδομένων.

Δεδομένα από εξωτερικές πηγές φτάνουν στις τοπολογίες του Storm μέσω των spouts, που δημιουργούν ροές δεδομένων. Γενικά, τα spouts διαβάζουν πλειάδες (tuples) από τις εξωτερικές πηγές και τις κάνουν emit σε μια τοπολογία.

Μια ροή δεδομένων καλείται stream και αποτελείται από μια απέραντη σειρά πλειάδων (tuples). Η πλειάδα είναι σαν μια δομή που μπορεί να αντιπροσωπεύσει τυποποιημένους τύπους δεδομένων (όπως ints, floats και byte arrays) ή τύπους που καθορίζονται από το χρήστη. Κάθε stream ορίζεται από ένα μοναδικό αναγνωριστικό που μπορεί να χρησιμοποιηθεί για την κατασκευή τοπολογιών με spouts και bolts.

Τα Bolts εφαρμόζουν ένα ενιαίο μετασχηματισμό για κάθε stream σε μια τοπολογία. Μπορούν να εφαρμόσουν λειτουργίες αντίστοιχες του MapReduce ή ακόμα και πιο σύνθετες δράσεις που περιορίζονται σε ένα επίπεδο (single-step functions), όπως το φιλτράρισμα, ομαδοποιήσεις ή επικοινωνία με εξωτερικούς φορείς, όπως μια βάση δεδομένων. Μια τυπική τοπολογία Storm εκτελεί πολλαπλούς μετασχηματισμούς και ως εκ τούτου, απαιτεί πολλαπλά bolts.

Ένα bolt είναι δυνατόν να μεταφέρει δεδομένα σε πολλαπλά bolts καθώς επίσης μπορεί να δεχτεί δεδομένα από πολλαπλές πηγές.

Ένα από τα πιο ενδιαφέροντα χαρακτηριστικά στην αρχιτεκτονική του Storm είναι η εγγυημένη επεξεργασία μηνυμάτων. Το Storm μπορεί να

εγγυηθεί ότι κάθε πλειάδα που γίνεται **emit** από ένα **spout** θα υποβληθεί σε επεξεργασία.

Χρησιμοποιείται κυρίως για εφαρμογές **machine learning**, μελέτη τάσεων (**trends**) μέσα από κοινωνικά δίκτυα και συνεχείς υπολογισμούς σε δεδομένα που λαμβάνονται σε πραγματικό χρόνο.

[16, 17]

Μέρος VI

ΥΛΟΠΟΙΗΣΗ

7

ΥΛΟΠΟΙΗΣΗ

7.1 Προσέγγιση:

Εφόσον ο σκοπός της διπλωματικής αυτής είναι να εξετάσουμε την αποδοτικότητα των διαφόρων κομματιών στη διαδικασία ανάλυσης συναισθήματος η δική μου προσέγγιση στο πρόβλημα είναι η εξής:

1) Εύρεση ενός κατάλληλου συνόλου δεδομένων παρμένων από το Twitter μεγέθους 1,6 εκατομμυρίων tweets χειροκίνητα κατηγοριοποιημένων στην κλάση συναισθήματος και υλοποίηση ενός parser για ανάγνωση των tweets και του αντίστοιχου συναισθήματος.

2) Για κάθε tweet δημιουργούμε τις n-gram ακμές του γράφου τις οποίες τοποθετούμε σε μια δομή HashMap(η οποία αποτελεί την προτεινόμενη υλοποίηση πίνακα κατακερματισμού από την Oracle)[4]. Αυτό επιτυγχάνεται προσπελάζοντας το κείμενο του tweet σειριακά . Για κάθε νέο n-gram που προκύπτει προσπελάζοντας με κατάλληλου μεγέθους παράθυρο τα n-grams που το ακολουθούν λαμβάνεται η συσχέτιση τους που αποτελεί μια ακμή του γράφου. Ως κλειδί της HashMap για κάθε ακμή χρησιμοποιήθηκε το String της μορφής:

$N-GRAM1+"->"+N-GRAM2$, όπου η τιμή κάθε ακμής αποτελεί τον αριθμό εμφάνισης της εν λόγω ακμής στον γράφο.

Όταν παρουσιασθεί ήδη υπάρχουσα ακμή σε ένα γράφο απλά αυξάνεται ο αριθμός εμφάνισης της εν λόγω ακμής. Στη συνέχεια κάθε γράφος θετικής πολικότητας συγχωνεύεται σε ένα θετικό γράφο χρυσού κανόνα και κάθε γράφος αρνητικής πολικότητας συγχωνεύεται σε ένα αρνητικό γράφο χρυσού κανόνα. Ομοίως με πριν μια ακμή του νέου γράφου, που υπάρχει ήδη στο γράφο χρυσού κανόνα απλά προσθέτει τον αριθμό εμφάνισης της στην αντίστοιχη.

3) Πλέον από αυτά τα δύο γραφήματα Golden rule μπορούμε να πάρουμε τα features που επιθυμούμε για την εκπαίδευση και έλεγχο κάποιου κατηγοριοποιητή. Αυτό επιτυγχάνεται χρησιμοποιώντας κάποια μετρική ομοιότητας γράφων όπως αυτές που αναλύσαμε σε προηγούμενο κεφάλαιο, με την οποία θα συγκριθούν οι γράφοι των tweets που προορίζονται για εκπαίδευση και έλεγχο αντίστοιχα.

Για τους διάφορους κατηγοριοποιητές χρησιμοποιήθηκε το εργαλείο Weka [5] του πανεπιστημίου του Waikato που αποτελεί μια πολύ ισχυρή συλλογή αλγορίθμων μηχανικής μάθησης για σκοπούς εξόρυξης δεδομένων. Επίσης παρέχει ένα καλά ενημερωμένο και εύχρηστο API για χρήση της βιβλιοθήκης σε προγράμματα Java.

7.1.1 Σύνολο δεδομένων

Το σύνολο δεδομένων που χρησιμοποιήσαμε για τα πειράματα μας αποτελεί μια συλλογή από Tweets που συλλέχθηκαν από το πανεπιστήμιο του Stanford [6]. Περιέχει 1.6 εκατομμύρια Tweets προεπεξεργασμένα ώστε να μην περιέχει εικονίδια emoticons και είναι σε κανονική μορφή CSV. Έχει έξι πεδία σε κάθε γραμμή με την ακόλουθη δομή:

1. Η πολικότητα του tweet(0=αρνητικό,4=θετικό)
2. Τον αριθμό ταυτότητας του(π.χ. 1467811184)
3. Η ημερομηνία δημιουργίας του(Sat May 16 23:58:44 UTC 2009)
4. Ερώτηση. Το πεδίο αυτό στο συγκεκριμένο σύνολο δεν περιέχει κάποια τιμή για αυτό είναι NO_QUERY
5. Το όνομα χρήστη που το έγραψε(π.χ. joy_wolf)
6. Το κείμενο του tweet.

Τέλος να αναφέρουμε ότι στο συγκεκριμένο σύνολο δεδομένων τα tweets δεν αναφέρονται σε κάποιο συγκεκριμένο θέμα, δηλαδή είναι γενικού περιεχομένου.

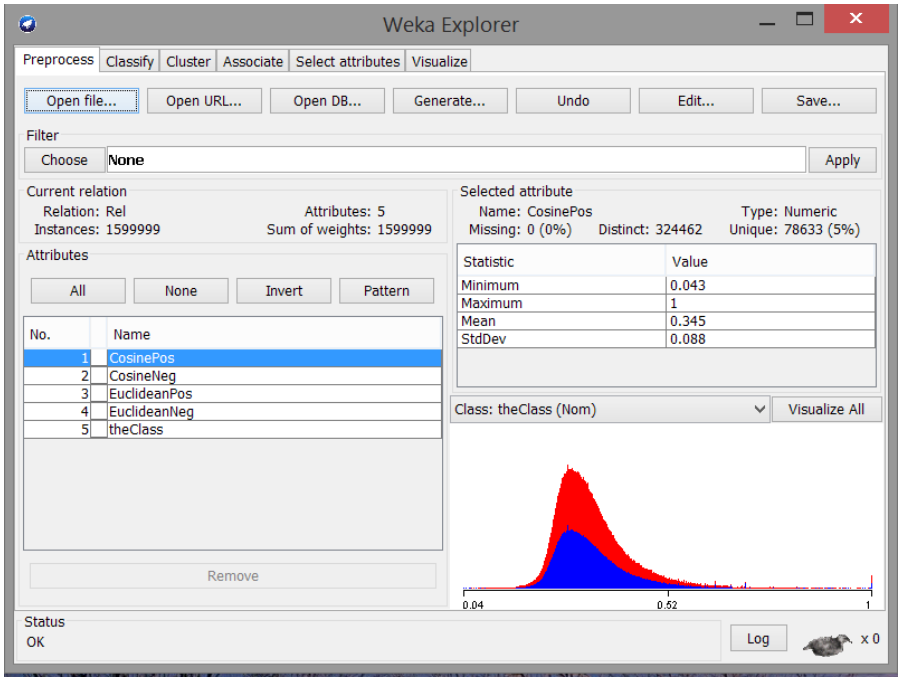
7.1.2 Δημιουργία γράφων

Η επιλογή της δομής HashMap έγινε επειδή είναι επιθυμητή μια απλή δομή, με την ελάχιστη δυνατή κατανάλωση μνήμης και τον καλύτερο συνδυασμό ταχύτητας δημιουργίας ενός γράφου(στην ουσία χρόνο εισαγωγής στοιχείων) και χρόνων εύρεσης στοιχείων κατά την σύγκριση δύο γράφων. Υπάρχουν διάφορες υλοποιήσεις δομών πίνακα στη Java αλλά η Oracle προτείνει σε περίπτωση υλοποιήσεων ενός νήματος την κλάση HashMap. Προδιαγράφει την πολυπλοκότητα εισόδου και εύρεσης στοιχείων ως O(1) με την προϋπόθεση ότι ο κατακερματισμός θα γίνει επιτυχώς ώστε να χωρέσουν όλα τα στοιχεία μέσα στον πίνακα. Σε αντίθετη περίπτωση χρειάζεται επιπλέον χρόνος επανακατακερματισμού του πίνακα για αύξηση του μεγέθους. Για αυτό και θεωρείται σκόπιμο να δηλωθεί σωστά το αρχικό μέγεθός του ανάλογα με το συνολικό μέγεθος του dataset.[4] Επίσης λόγω απώλειας πλεονάζουσας πληροφορίας αποτελεί πολύ οικονομική δομή σε κατανάλωση μνήμης. Για τους σκοπούς της εργασίας αυτής θεωρήθηκε σκόπιμο όπως η δομή είναι δυνατόν να αποθηκευτεί εξολοκλήρου στη μνήμη για αποφυγή καθυστερήσεων προσπέλασης στον σκληρό δίσκο. Για αυτό και δεν εξετάστηκαν λύσεις αποθήκευσης σε αρχεία.

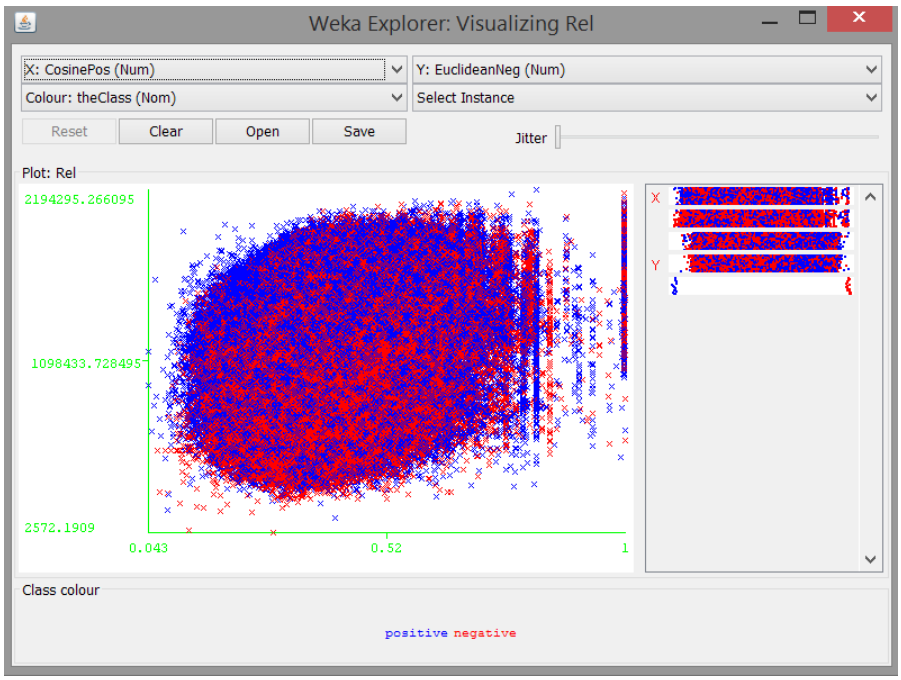
7.1.3 Διαδικασία κατηγοριοποίησης

Η βιβλιοθήκη Weka παρέχει υλοποίηση πολλών κατηγοριοποιητών και κατάλληλη δομή παραμέτρων ώστε να τους ρυθμίσουμε όπως θέλουμε. Το

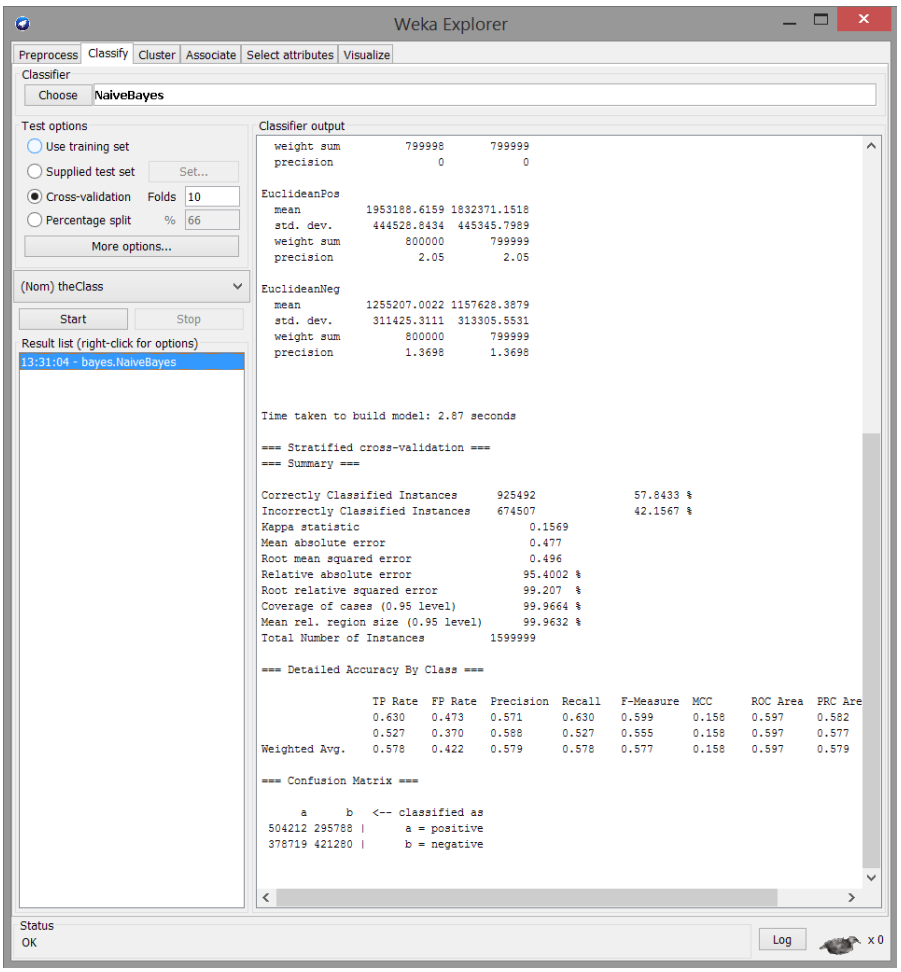
documentation που παρέχεται δίνει ευνόητα παραδείγματα για χρήση της βιβλιοθήκης σε προγράμματα Java. Επίσης παρέχει ένα εύχρηστο γραφικό περιβάλλον για εφαρμογή φίλτρων στα δεδομένα, γραφικές παραστάσεις και τέλος κατηγοριοποίηση των δεδομένων που επιστρέφει αποτελέσματα με όλες τις χρήσιμες στατιστικές μετρικές.



Σχήμα 7.1: Γραφικό περιβάλλον Weka



Σχήμα 7.2: Γραφικό περιβάλλον Weka 2



Σχήμα 7.3: Γραφικό περιβάλλον Weka 3

7.2 Επιλογή γλώσσας προγραμματισμού και περιβάλλον ανάπτυξης

Για την ανάπτυξη αυτού του πολύπλοκου συστήματος πολλών σταδίων επεξεργασίας επιλέγηκε η γλώσσα Java με το περιβάλλον ανάπτυξης Eclipse. Η επιλογή της γλώσσας Java έγινε με σκοπό τη χρήση του αντικειμενοστραφούς μοντέλου προγραμματισμού ώστε να οργανωθεί πιο εύκολα η σχεδίαση και υλοποίηση του συστήματος.

Κάθε στοιχειώδες συστατικό που χρησιμοποιήθηκε γράφτηκε ως ξεχωριστή οντότητα ώστε να μπορεί να χρησιμοποιηθεί σε οποιαδήποτε αλλαγή του συνολικού συστήματος. Εφόσον πρόκειται για να ένα σύστημα πολλών σταδίων με σκοπό την δοκιμή διαφόρων δομών δεδομένων και αλγορίθμων θεώρησα σκόπιμο τα διάφορα συστατικά του να έχουν την λειτουργία plug-and-play. Έτσι για παράδειγμα μου δόθηκε η δυνατότητα δοκιμής διαφόρων δομών αποθήκευσης των n-gram graphs όπως ArrayList, HashMap, Graphs κτλ.

Επίσης η Java προσφέρει τεράστια βιβλιοθήκη έτοιμων κλάσεων για την χρήση διαφόρων λειτουργιών που σε άλλη περίπτωση θα ήταν αναγκαία η

εκ νέου υλοποίηση αχρείαστα πολύπλοκων προγραμμάτων για τους σκοπούς αυτής της εργασίας. Για παράδειγμα η ανάγνωση και εγγραφή σε αρχεία χρειάζεται μηδαμινή προσπάθεια συγχώνευσης στο σύστημα με τα διάφορα έτυμα εργαλεία και κλάσεις της. Πιθανώς το πιο σημαντικό χαρακτηριστικό της εν λόγω γλώσσας προγραμματισμού είναι η ύπαρξη πολυάριθμων κοινοτήτων στο διαδίκτυο ώστε να επιλυθούν όλες οι απορίες που τυχόν να προκύψουν κατά την εκπόνηση της εργασίας.

Ταυτόχρονα όπως έχουμε αναφέρει και για την βιβλιοθήκη λογισμικού Weka, η χρήση της γλώσσας Java έκανε την χρήση των αλγορίθμων κατηγοριοποίησης, των διαφόρων φίλτρων προεπεξεργασίας και των λειτουργιών δοκιμής τους πολύ πιο εύκολη. Με τη χρήση της κλάσης Instances της Weka δημιουργήθηκε το τελικό σύνολο χαρακτηριστικών με το οποίο μπορεί κανείς πλέον να αποθηκεύσει τα δεδομένα σε αρχεία τύπου 'arff' για χρήση στο γραφικό περιβάλλον της Weka. Ταυτόχρονα μπορεί κάποιος να χρησιμοποιήσει τα Instances αυτά για κατευθείαν εφαρμογή κατηγοριοποιητών. Προσφέρεται υλοποίηση τυχαίας κατανομής των συμβάντων στα σύνολα εκπαίδευσης και δοκιμής με διάφορες προδιαγραφές. Για τη δική μου υλοποίηση χρησιμοποίησα κατά περίπτωση τον διαχωρισμό 90% για δεδομένα εκπαίδευσης και 10% για δοκιμή ή αντίστοιχα την γνωστή μέθοδο ten-fold cross-validation.

Στη μέθοδο ten-fold cross-validation το σύνολο από Instances μοιράζεται σε 10 υποσύνολα για δοκιμή. Το υπόλοιπο του συνόλου χρησιμοποιείται σε κάθε περίπτωση για εκπαίδευση εκ νέου του κατηγοριοποιητή και έτσι η διαδικασία κατηγοριοποίησης επαναλαμβάνεται 10 φορές ώστε τα αποτελέσματα να παρουσιαστούν ως ο μέσος όρος των επαναλήψεων. Αυτό το εφαρμόζουμε ώστε να αποδείξουμε την ανεξαρτησία των δεδομένων εισόδου από την μέθοδό μας. Κατά τη διάρκεια όμως της εκτέλεσης τα αποτελέσματα αυτής της διαδικασίας δεν έδειχναν μεγάλες αποκλίσεις μεταξύ τους με αποτέλεσμα να μην είναι αναγκαία η χρονοβόρα χρήση αυτής της μεθόδου.

Για την εύκολή διαχείριση όλων αυτών των προγραμμάτων και βιβλιοθηκών χρειάστηκε η χρήση ενός κατάλληλου γραφικού περιβάλλοντος ανάπτυξης(Integrated development environment-IDE). Επιλέξαμε το περιβάλλον Eclipse για την ευκολία του και μεγάλη γκάμα εργαλείων που προσφέρει σε ένα κατά τα άλλα ελαφρύ πακέτο. Παρέχει ότι χρειάζεται ένας προγραμματιστής Java, όπως και πολλά άλλα δημοφιλή IDEs. Ένα σημαντικό εργαλείο του που βοήθησε στην οργάνωση και ευκολία ανάπτυξης του συστήματος είναι η συμβατότητα με το version control tool Git. Αυτό έκανε τη επανεξέταση και οργάνωση των προγραμμάτων ευκολότερη με την παρουσίαση αλλαγών που εφάρμοζα κατά την διάρκεια της ανάπτυξης ώστε να λυθούν τα προβλήματα που προκύπταν. Ταυτόχρονα όλο το project φορτωνόταν στο νέφος(Cloud) για την ευπρόσιτη υποστήριξη από το διδακτικό προσωπικό.

Μέρος VII

ΠΕΙΡΑΜΑΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ

8

ΠΕΙΡΑΜΑΤΙΚΗ ΔΙΑΔΙΚΑΣΙΑ

Εδώ θα αναφέρουμε τα πειράματα που διεξάχθηκαν στα πλαίσια της παρούσας εργασίας με σκοπό την εξαγωγή συμπερασμάτων. Επίσης θα κάνουμε ανάλυση μελλοντικής εργασίας που μπορεί να γίνει ως επέκταση για περεταίρω βελτιστοποίηση της διαδικασίας κατηγοριοποίησης.

1. Εξέταση της περίπτωσης διαγραφής των κοινών ακμών μεταξύ των γράφων χρυσού κανόνα για εξοικονόμηση μνήμης.
2. Ανάλυση της υλοποίησης με HashMap και εξέταση των χρόνων και κατανάλωσης μνήμης για την δημιουργία της δομής.
3. Ανάλυση της εξαγωγής χαρακτηριστικών και εξέταση των μετρικών απόδοσης.
4. Ανάλυση κατηγοριοποιητών και εξέτασης μετρικών απόδοσης.

Όλα τα πειράματα έγιναν σε υπολογιστή με τετραπύρηνο επεξεργαστή Intel i5 με 8GB μνήμης. Στην εικονική μηχανή της Java(JVM) δόθηκε η παράμετρος για μέγεθος σωρού 5GB.

Για τους κατηγοριοποιητές χρησιμοποιήθηκαν οι εξής παράμετροι:

weka.classifiers.bayes.NaiveBayes:

1. batchSize: 100
2. debug: False
3. displayModelInOldFormat: False
4. doNotCheckCapabilities: False
5. numDecimalPlaces: 2
6. useKernelEstimator: False
7. useSupervisedDiscretization: False

weka.classifiers.functions.Logistic:

1. batchSize: 100
2. debug: False
3. doNotCheckCapabilities: False
4. maxIts: -1
5. numDecimalPlaces: 2
6. ridge: 1.0E-8
7. useConjugateGradientDescent: False

weka.classifiers.functions.SMO:

1. batchSize: 100
2. buildLogisticModels: False
3. c: 1.0
4. checkTurnedOff: False
5. debug: False
6. doNotCheckCapabilities: False
7. epsilon: 1.0E-12
8. filterType: Normalize training data
9. kernel: PolyKernel –E 1.0 –C 250007
10. numDecimalPlaces: 2
11. numFolds: -1
12. randomSeed: 1
13. toleranceParameter: 0.001

weka.classifiers.trees.J48:

1. batchSize: 100
2. binarySplits: False
3. collapseTree: True
4. confidenceFactor: 0.25
5. debug: False
6. doNotCheckCapabilities: False
7. doNotMakeSplitPointActualValue: False
8. minNumObj: 2
9. numDecimalPlaces: 2
10. numFolds: 3
11. reducedErrorPruning: False
12. saveInstanceData: False
13. seed: 1
14. subtreeRaising: True
15. unpruned: False
16. useLaplace: False
17. useMDLcorrection: True

8.1 Σύνολο πειραμάτων 1

8.1.1 Η μέθοδος διαγραφής κοινών ακμών

Αρχικά η δημιουργία των γράφων χρυσού κανόνα στην υλοποίησή μου συγχώνευε όλους τους γράφους που παράγονταν από το **dataset**. Μια πιθανή βελτίωση που μπορούμε να εξετάσουμε είναι η παράλειψη εισόδου ή και διαγραφή των ακμών που ανήκουν και στους δύο γράφους χρυσού κανόνα. Σε αυτή τη σειρά πειραμάτων θα συγκρίνουμε την εκτέλεση, για όμοιων παραμέτρων δοκιμές, με και χωρίς τις κοινές ακμές.

Ο πρώτος πίνακα παρουσιάζει τις εξής πληροφορίες για τα πειράματα: μέγεθος **n-gram**, αριθμός **tweets**, ελεύθερη μνήμη στο τέλος της δημιουργίας δομών χρυσού κανόνα, χρόνος δημιουργίας δομών χρυσού κανόνα, ελεύθερη μνήμη στο τέλος της σύγκρισης όλων των **n-gram graphs** με τους γράφους χρυσού κανόνα και αντίστοιχος χρόνος εκτέλεσης. Στο δεύτερο πίνακα παρουσιάζεται για τα πειράματα αυτά η απόδοση των κατηγοριοποιητών **NaiveBayes, LogisticRegression, J48, SMO** που αποτελούν τις υλοποιήσεις της **Weka** για τους αλγόριθμους κατηγοριοποίησης που αναλύσαμε σε προηγούμενο κεφάλαιο.

n-gram	Δεδομένα	ΕΜΧΚ(Mb)	ΧΧΚ(sec.)	ΕΜΣ(Mb)	ΧΣ(sec.)
Χωρίς αφαίρεση κοινών ακμών					
3	1600000	2,895	208	1,748	270
4	1600000	1,403	412	313	446
5	200000	2,250	96	2,046	67
5	400000	1,196	202	1,094	173
5	600000				
Με αφαίρεση κοινών ακμών					
3	1600000	3,691	285	2,550	287
4	1600000	2,299	543	946	370
5	200000	2,859	91	2,405	60
5	400000	1,551	210	1,599	114
5	600000	986	344	1,253	200

Πίνακας 8.1: Χρόνοι και κατανάλωση μνήμης πειραμάτων 1

Εδώ παρατηρούμε ότι η αφαίρεση των κοινών ακμών προκαλεί μια μικρή αύξηση κατά μέσο όρο στον χρόνο εκτέλεσης τον πειραμάτων. Αντιθέτως όμως προκαλεί βελτίωση στην κατανάλωση μνήμης. Πράγμα που φαίνεται καλύτερα στο γεγονός ότι το πείραμα για μέγεθος **5-gram** και 600000 **tweets** δεν μπορούσε να ολοκληρωθεί κανονικά, ενώ όταν εφαρμόσαμε αφαίρεση κοινών ακμών ολοκληρώθηκε.

Αποτελέσματα κατηγοριοποίησης:

n-gram	Δεδομένα	NB(%)	ΛΠ(%)	J48(%)	SVM(%)
Χωρίς αφαίρεση κοινών ακμών					
3	1600000	64,8	79,6	65,5	68,8
4	1600000	85,4	94,2	86,2	93,1
5	200000	98,5	98,9	99,1	98,2
5	400000	98,4	98,7	98,9	97,9
5	600000				
Με αφαίρεση κοινών ακμών					
3	1600000	66,9	69,8	69,7	
4	1600000	76,9	78,5	78.5	
5	200000	94,0	94,1	94,1	94,0
5	400000	91,6	91,9	92.1	91,9
5	600000	90.4	90.6	90.7	90,62

Πίνακας 8.2: Κατηγοριοποίηση Πειραμάτων 1

Εδώ παρατηρούμε τη γνωστή στη βιβλιογραφία θεωρία πως η κατηγοριοποίηση γίνεται αποδοτικότερη με την αύξηση του μεγέθους n-gram. Τα αποτελέσματα δείχνουν μια μείωση στην απόδοση με την μέθοδο αφαίρεσης των κοινών ακμών. Αυτό δηλώνει πως η παρουσία κοινών ακμών στου δύο γράφους χρυσού κανόνα αποτελεί πληροφορία που επηρεάζει την κατηγοριοποίηση. Έτσι καταλαβαίνουμε ότι η μέθοδος αυτή, παρόλο που μας επιτρέπει να εκτελέσουμε τη διαδικασία για πιο πολλά 5-grams τα οποία σε κάποιες περιπτώσεις δίνουν σχεδόν τέλεια απόδοση, δίνει χειρότερη απόδοση ως προς τον χρόνο εκτέλεσης και κατανάλωση μνήμης σε σχέση με τη κλασική μέθοδο με χρήση 4-grams. Θα μπορούσε να εξετασθεί για χρήση σε περίπτωση ανάγκης εξοικονόμησης περεταίρω μνήμης ακόμα και αν προκαλεί μια μικρή μείωση στην απόδοση. Από τώρα και στο εξής επιλέγουμε να μην εφαρμόσουμε τη μέθοδο αφαίρεσης κοινών ακμών στα υπόλοιπα πειράματα.

8.1.2 Η δομή αποθήκευσης των *n-gram* ακμών *HashMap*

Ας εξετάσουμε τώρα τα αποτελέσματα των πινάκων των πειραμάτων 1 χωρίς αφαίρεση κοινών ακμών. Βλέπουμε ότι η προσέγγιση αυτή αποτελεί αρκετά γρήγορη μέθοδο που ολοκληρώνει τη δημιουργία των γράφων χρυσού κανόνα για ένα αρκετά μεγάλο dataset για 3-grams και 4-grams. Ταυτόχρονα παρουσιάζει αρκετά μικρή κατανάλωση μνήμης και στις δύο περιπτώσεις κάτω από 4GB. Σε ένα υπολογιστή με περισσότερη ελεύθερη μνήμη θα μπορούσε εύκολα να περατώσει την διαδικασία και για 5-grams.

8.1.2.1 Σύγκριση Γράφων με τους Γράφους Χρυσού Κανόνα και εξαγωγή χαρακτηριστικών:

Σε αυτό το στάδιο εφαρμόζουμε μετρικές ομοιότητας γράφων. Συγκεκριμένα για τα πειράματα 1 χρησιμοποιήσαμε τα τρία ζευγάρια σύγκρισης των γράφων με του δύο γράφους χρυσού κανόνα, εφαρμόζοντας Containment Similarity, Value Similarity και Normalized Value Similarity. Οι τρείς αυτές μετρικές απαιτούν μια αρκετά εύκολη υλοποίηση χωρίς να αυξάνουν την πολυπλοκότητα του αλγορίθμου δημιουργίας των γράφων n-gram για κάθε tweet.

Παρατηρούμε χρόνο εκτέλεσης της εξαγωγής χαρακτηριστικών συγκρίσιμο με αυτό του προηγούμενου σταδίου και λίγο μεγαλύτερη κατανάλωση σε μνήμη αφού απαιτείται η παρουσία και των γράφων χρυσού κανόνα μαζί με τα Instances. Γενικά όμως η επιπλέον κατανάλωση σε μνήμη δεν είναι απαγορευτική σε σχέση με πριν.

8.1.2.2 Κατηγοριοποίηση των πειραμάτων 1:

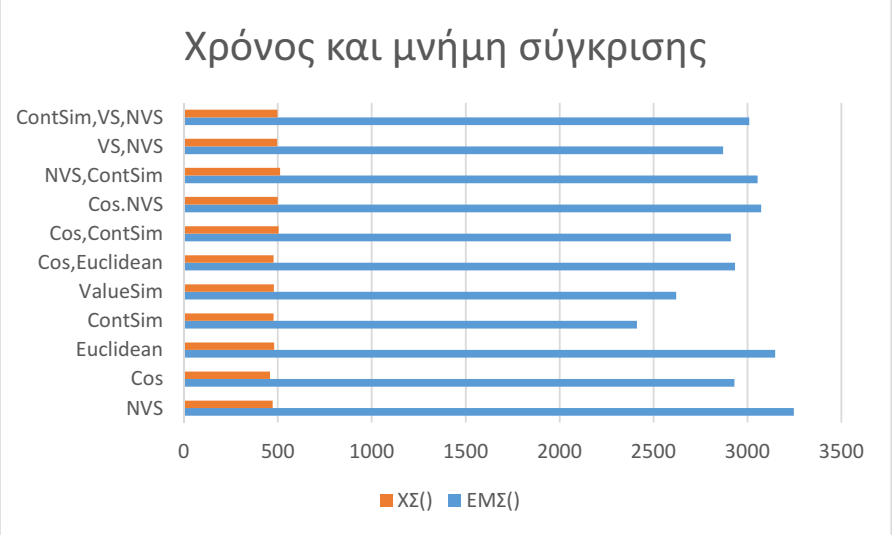
Τελικά παρατηρούμε πάρα πολύ καλά αποτελέσματα απόδοσης κατά την κατηγοριοποίηση με πρωταθλητή στα 3-grams και 4-grams την Λογιστική Παλινδρόμηση να δίνει ποσοστό επιτυχίας 79.6% και 94.2% αντίστοιχα. Ταυτόχρονα στα πειράματα με 5-grams, που έγιναν με μικρότερο σύνολο δεδομένων, η επιτυχία κατηγοριοποίησης έφτασε μέχρι και 99,1%. Το συμπέρασμα μας από αυτά είναι πώς η προσέγγιση μας δεν μειώνει την απόδοση της κατηγοριοποίησης αφού η σημαντική πληροφορία στους n-gram γράφους είναι η παρουσία των ακμών που τους απαρτίζουν.

8.2 Σύνολο πειραμάτων 2

Σε αυτή τη σειρά πειραμάτων έχουμε σκοπό να αναλύσουμε διάφορους συνδυασμούς μετρικών σύγκρισης γράφων και στη συνέχεια εξετάζουμε την απόδοση κατά τη κατηγοριοποίηση των παραγόμενων χαρακτηριστικών. Σε αυτό το σύνολο πειραμάτων εξετάσαμε τις περιπτώσεις υπολογισμού ενός ή ζευγαριού ή τριάδας χαρακτηριστικών και μετρήσαμε την κατανάλωση χρόνου και μνήμης κατά την εκτέλεση.

Χαρακτηριστικά	ΕΜΣ(Mb)	ΧΣ(sec.)
NVS	3246	472
Cos	2931	459
Euclidean	3147	481
ContSim	2411	477
ValueSim	2620	479
Cos,Euclidean	2933	478
Cos,ContSim	2911	505
Cos,NVS	3072	500
NVS,ContSim	3054	512
VS,NVS	2871	496
ContSim,VS,NVS	3009	498

Πίνακας 8.3: Πειράματα 2: Κατανάλωση μνήμης και χρόνου κατα τη σύγκριση.



Σχήμα 8.1: Πειράματα 2: Κατανάλωση μνήμης και χρόνου κατα τη σύγκριση.

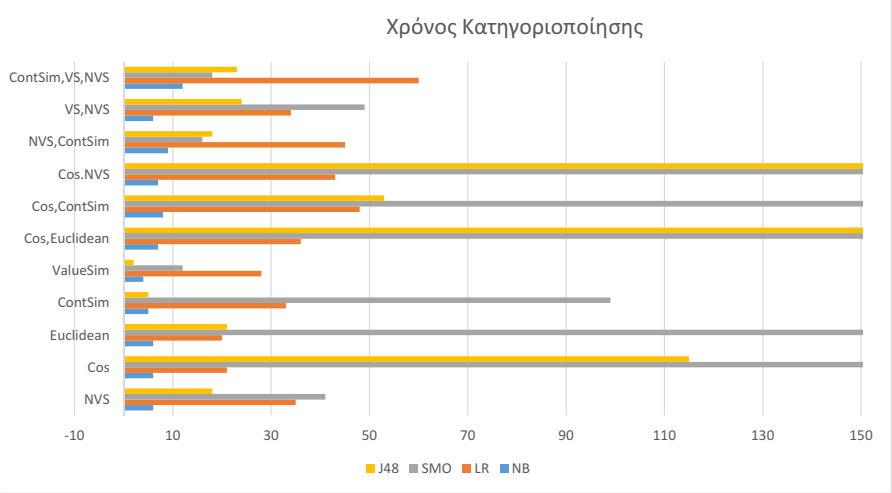
Γενικά παρατηρούμε όμοιους χρόνους και κατανάλωση μνήμης κατά την σύγκριση των γράφων και υπολογισμών των πιο πάνω μετρικών. Η αποδοτικότερη αυτών φαίνεται να είναι η Containment Similarity, η οποία καταναλώνει τους λιγότερους πόρους είτε στη περίπτωση που υπολογίζεται μόνη είτε σε συνδυασμούς με άλλες. Οι χειρότερες φαίνονται να είναι οι συνημιτονική, η ευκλείδεια και η κανονικοποιημένη ομοιότητα τιμών για αυτά τα κριτήρια.

Ακολούθως παρουσιάζουμε με τη σειρά τα αποτελέσματα της κατηγοριοποίησης μετρώντας χρόνο εκτέλεσης, κατανάλωση μνήμης και ακρίβεια κατηγοριοποίησης.

8.2.1 Σχολιασμός χρόνου εκτέλεσης κατηγοριοποιητών:

Χαρακτηριστικά	NBt(s)	LRt(s)	SMOt(s)	J48t(s)
NVS	6	35	41	18
Cos	6	21	256	115
Euclidean	6	20	980	21
ContSim	5	33	99	5
ValueSim	4	28	12	2
Cos,Euclidean	7	36	1571	380
Cos,ContSim	8	48	339	53
Cos,NVS	7	43	681	298
NVS,ContSim	9	45	16	18
VS,NVS	6	34	49	24
ContSim,VS,NVS	12	60	18	23
Μέσος Όρος	6.91	36.64	369.27	87.00
Τυπική απόκλιση	2.07	11.30	484.98	123.66

Πίνακας 8.4: Πειράματα 2: Χρόνος εκτέλεσης κατηγοριοποιητών.



Σχήμα 8.2: Πειράματα 2: Χρόνος εκτέλεσης κατηγοριοποιητών.

Παρατηρούμε ότι ο Naïve Bayes αποτελεί τον γρηγορότερο αλγόριθμο κατηγοριοποίησης ακολουθούμενος από τη Λογιστική Παλινδρόμηση, τον J48(υλοποίηση δέντρων απόφασης C4.5) και χείριστο τον SMO. Οι δύο τελευταίοι εκτός του ότι παρουσιάζουν χειρότερους μέσους όρους παρατηρούνται σε κάποιες περιπτώσεις να αποκλίνουν αρκετά από τη νόρμα παρουσιάζοντας απαγορευτικά μεγάλους χρόνους εκτέλεσης. Για καλύτερη παρουσίαση των αποτελεσμάτων επιλέξαμε να κόψουμε στην γραφική

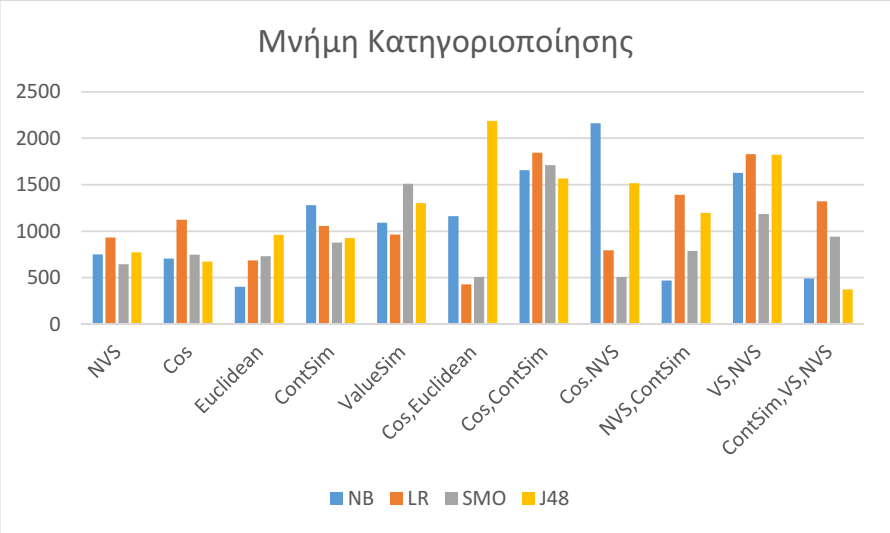
παράσταση για τις περιπτώσεις που χρειάστηκαν πέραν των 150 δευτερολέπτων. Αν θέλουμε να έχουμε γρήγορη εκτέλεση με συνεπής αποτελέσματα κατηγοριοποίησης προτείνεται η χρήση των NB και LR.

Μία δεύτερη παρατήρηση είναι, όπως φυσικά αναμενόταν, να αυξάνεται ο χρόνος εκτέλεσης με την αύξηση του αριθμού των χαρακτηριστικών. Παρόλα αυτά η ευκλείδεια απόσταση και η συνημιτονική ομοιότητα χρειάζονται μεγαλύτερους χρόνους κατηγοριοποίησης από τις υπόλοιπες μετρικές. Ειδικά η ευκλείδεια σε κάποιες περιπτώσεις ξεφεύγει από το όριο των 150 δευτερολέπτων που πιθανόν να οφείλεται στο μεγάλο πεδίο τιμών που έχει σε αντίθεση με τις υπόλοιπες οι οποίες ανήκουν μόνο στο διάστημα [0,1].

8.2.2 Σχολιασμός κατανάλωσης μνήμης κατηγοριοποιητών:

Χαρακτηριστικά	NBm(Mb)	LRm(Mb)	SMOm(Mb)	J48m(Mb)
NVS	749	931	645	771
Cos	704	1123	746	673
Euclidean	402	685	731	961
ContSim	1279	1055	877	926
ValueSim	1092	964	1510	1303
Cos,Euclidean	1162	427	506	2186
Cos,ContSim	1655	1843	1711	1568
Cos,NVS	2159	793	507	1516
NVS,ContSim	469	1391	787	1198
VS,NVS	1628	1828	1183	1822
ContSim,VS,NVS	490	1320	940	374
Μέσος Όρος	1071.73	1123.64	922.09	1208.91
Τυπική απόκλιση	527.10	444.75	393.25	536.69

Πίνακας 8.5: Πειράματα 2: Κατανάλωση μνήμης κατηγοριοποιητών.



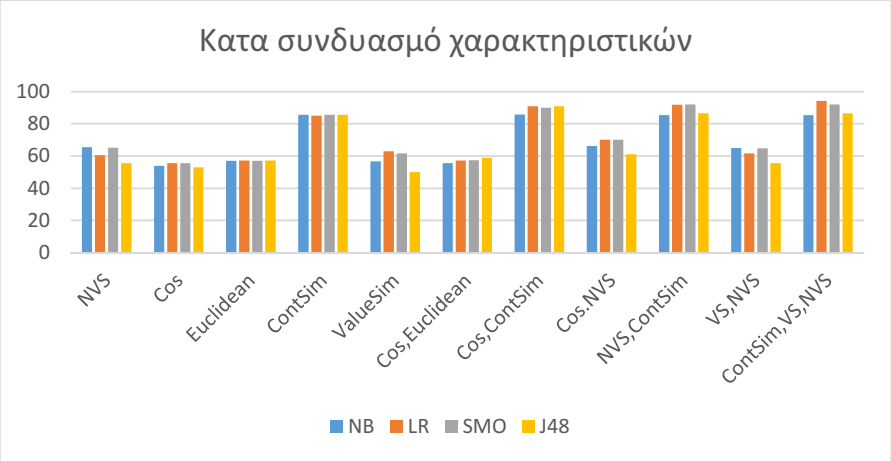
Σχήμα 8.3: Πειράματα 2: Κατανάλωση μνήμης κατηγοριοποιητών.

Η κατανάλωση μνήμης των κατηγοριοποιητών αποτελεί γενικά ομοιόμορφη μετρική και στους τέσσερις υπό εξέταση γύρο στο 1GB φτάνοντας το πολύ μέχρι τα 2GB. Η απόκλιση των πιο πάνω κατηγοριοποιητών από τη νόρμα δεν αποτελεί επίσης σε καμία των περιπτώσεων απαγορευτική αφού γενικά η διαδικασία κατηγοριοποίησης καταναλώνει λιγότερη μνήμη από τα προηγούμενα στάδια επεξεργασίας.

8.2.3 Σχολιασμός ευκρίνειας κατηγοριοποιητών:

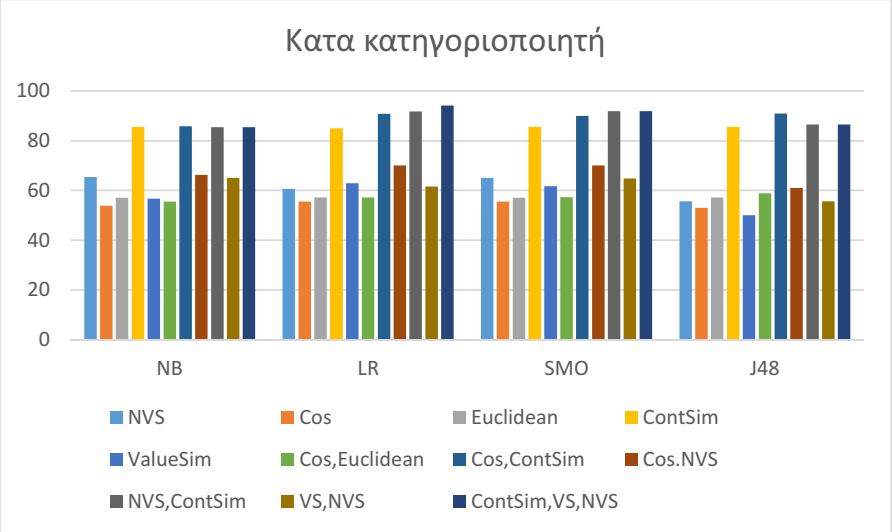
Χαρακτηριστικά	NBa(%)	LRa(%)	SMOa(%)	J48a(%)
NVS	65.45	60.63	65.06	55.65
Cos	53.85	55.51	55.51	53.03
Euclidean	57.03	57.14	57.02	57.20
ContSim	85.61	85.02	85.61	85.61
ValueSim	56.72	62.86	61.71	49.99
Cos,Euclidean	55.56	57.22	57.33	58.89
Cos,ContSim	85.83	90.82	89.98	90.93
Cos,NVS	66.29	70.02	70.10	61.05
NVS,ContSim	85.40	91.75	91.93	86.52
VS,NVS	65.00	61.61	64.82	55.63
ContSim,VS,NVS	85.41	94.21	91.94	86.52

Πίνακας 8.6: Πειράματα 2: Απόδοση κατηγοριοποιητών.



Σχήμα 8.4: Πειράματα 2: Απόδοση κατηγοριοποιητών.A

Εδώ θα αναλύσουμε την ευκρίνεια ως προς τα χαρακτηριστικά. Ανάμεσα στα διάφορα πειράματά μας ξεχωρίζει η Containment Similarity, ο συνδυασμός αυτής με την συνημιτονική και την Normalized Value Similarity και τέλος η τριάδα των Containment Similarity, Value Similarity και Normalized Value Similarity. Αυτά τα τέσσερα σύνολα χαρακτηριστικών παρουσιάζουν όμοια αποτελέσματα κοντά στο 90%, με την έκπληξη που παρουσιάζει η Containment Similarity ως μόνο χαρακτηριστικό να προσφέρει τουλάχιστον 85%. Αν όμως λάβουμε υπόψιν τη κατανάλωση πόρων που εμφανίστηκε καταλαβαίνουμε πώς ο συνδυασμός με τη συνημιτονική ομοιότητα είναι η χειρότερη επιλογή από αυτές. Οι υπόλοιποι τρείς συνδυασμοί παρουσιάζουν μια αύξηση στην ευκρίνεια με ταυτόχρονη αύξηση της κατανάλωσης πόρων καθώς αυξάνεται ο αριθμός χαρακτηριστικών. Αυτό δηλώνει πως η επιλογή μεταξύ αυτών αφήνεται στη κρίση του επιστήμονα ανάλογα με τις ανάγκες της εφαρμογής του.



Σχήμα 8.5: Πειράματα 2: Απόδοση κατηγοριοποιητών.B

Εδώ θα αναλύσουμε την ευκρίνεια ως προς τους κατηγοριοποιητές. Τα αποτελέσματα είναι αρκετά όμοια μεταξύ των κατηγοριοποιητών. Παρατηρούμε μια πολύ μικρή υπερίσχυση του SMO από τους υπολοίπους. Αν όμως λάβουμε υπόψιν τη κατανάλωση πόρων ο NB αποτελεί πολύ ισχυρό υποψήφιο αφού καταναλώνει τους ελάχιστούς πόρους με συνέπεια σε όλα τα πειράματά μας και προσφέρει ευκρίνεια ελάχιστα μικρότερη από τους υπολοίπους.

Μέρος VIII

ΣΥΜΠΕΡΑΣΜΑΤΑ

9

ΣΥΜΠΕΡΑΣΜΑΤΑ

9.1 Σύνοψη πειραματικών αποτελεσμάτων:

1. Στη σειρά πειραμάτων 1 εξετάσαμε τη σύγκριση μεταξύ εφαρμογής και μη της μεθόδου αφαίρεσης των κοινών ακμών. Καταλήξαμε ότι η μέθοδος έχει το πλεονέκτημα λιγότερης κατανάλωσης μνήμης αλλά κοστίζει περισσότερο σε χρόνο και ευκρίνεια πρόβλεψης. Ταυτόχρονα στα πειράματα αυτά εξετάσαμε την απόδοση της προσέγγισης μας για διάφορα μεγέθη ν-γραμ φτάνοντας μέχρι και σε τέλεια πρόβλεψη 99.1% στα 5-γραμ.
2. Στη σειρά πειραμάτων 2 εξετάσαμε σε διάφορους συνδυασμούς τις μετρικές ομοιότητας γράφων εξετάζοντας κατανάλωση μνήμης και χρόνου. Στη συνέχεια οδηγήσαμε αυτά τα χαρακτηριστικά σε τέσσερις κατηγοριοποιητές για να εξετάσουμε την απόδοση τους.
3. Στο πρώτο στάδιο παρατηρήθηκε καλύτερη συμπεριφορά από τις Containment και Value Similarity.
4. Στους χρόνους κατηγοριοποίησης δεν παρατηρήσαμε σοβαρές διαφορές μεταξύ των συνόλων χαρακτηριστικών εκτός της ευκλείδειας και της συνημιτονικής ομοιότητας που οδηγούν σε κάποιες περιπτώσεις σε απαγορευτικά μεγάλο χρόνο εκτέλεσης. Ταυτόχρονα επιβεβαιώθηκε η άποψη πως με την αύξηση του αριθμού χαρακτηριστικών αυξάνεται και ο χρόνος εκτέλεσης.
5. Για την κατανάλωση μνήμης κατά την κατηγοριοποίηση δεν παρατηρήσαμε κάποια σημαντική πληροφορία.
6. Τελικά παρατηρώντας την ευκρίνεια των μεθόδων αυτών κατά τη κατηγοριοποίηση και πάλι ξεχωρίζει η Containment Similarity, το ζευγάρι αυτής με την NVS, και η τριάδα CS,VS,NVS. Αυτοί οι συνδυασμοί παρουσιάζουν τα καλύτερα αποτελέσματα από όλους όσους εξετάστηκαν και αναφέρονται με τη σειρά αυξημένης ευκρίνειας και ταυτόχρονα αυξημένης κατανάλωσης πόρων.
7. Από τα αποτελέσματα αυτά ταυτόχρονα συμπεραίνουμε πώς ο SMO παρόλο που κάνει σε περιπτώσεις μεγάλη κατανάλωση πόρων προσφέρει τα καλύτερα αποτελέσματα σε ευκρίνεια. Ο NB καταναλώνει τους λιγότερους πόρους και δεν υστερεί δραματικά σε σχέση με τους υπολοίπους.

9.2 Επιπτώσεις:

Από αυτή την διπλωματική εργασία η ερευνητική κοινότητα μπορεί να πάρει ότι η χρήση της δομής HashMap αποτελεί οικονομική σε πόρους λύση που δεν υστερεί ούτε στα αποτελέσματα κατηγοριοποίησης. Επίσης τα πιο πάνω συμπεράσματα των πειραμάτων μου θα διευκολύνουν στην επιλογή των κατάλληλων συνδυασμών χαρακτηριστικών και κατηγοριοποιητή σε νέες έρευνες.

Επίσης η υλοποίησή μου υπάρχει σε git repository[7]. Έτσι μπορεί οποιοσδήποτε επιθυμεί να μελετήσει την προσέγγιση μου και να επεκτείνει τις δυνατότητες της να το κάνει εύκολα. Αφού εγκαταστήσει το λογισμικό Git στο υπολογιστή του, μπορεί στη συνέχεια να προμηθευτεί την τελευταία έκδοση του κώδικα μου με την εντολή:

```
1 git clone https://MichalisEllinas@bitbucket.org/ellinasm91/
      diplomatiki.git
```

9.3 Μελλοντική εργασία:

Κάθε δευτερόλεπτο, κατά μέσο όρο, περίπου 6.000 tweets δημοσιεύονται στο Twitter, το οποίο αντιστοιχεί σε πάνω από 350.000 tweets ανά λεπτό, 500 εκατομμύρια tweets ανά ημέρα ή περίπου 200 δισεκατομμύρια tweets ανά έτος.[18] Αυτό δηλώνει πως ο αντίστοιχος αριθμός 1.6 εκατομμυρίων tweets δημοσιεύεται μέσα σε 267 δευτερόλεπτα περίπου. Για την επεξεργασία αυτών των tweets ακόμα και στην πιο γρήγορη μέθοδο που εξετάσαμε με trigrams χρειάστηκε 208 δευτερόλεπτα για την δημιουργία των γράφων χρυσού κανόνα, 270 δευτερόλεπτα για την σύγκριση των γράφων και μεταξύ 1-10 δευτερόλεπτα για την κατηγοριοποίηση με Naive Bayes. Συμπεραίνουμε λοιπόν ότι επί του παρόντος η προσέγγισή μου δεν είναι ικανή να επεξεργαστεί τον μεγάλο όγκο δεδομένων του Twitter stream.

Στο μέλλον μπορεί να προχωρήσει η διερεύνηση για απόκτηση ενός συστήματος ανάλυσης συναισθήματος που θα λειτουργεί σε πραγματικό χρόνο. Η προσέγγισή μου μπορεί να επεκταθεί ώστε να δέχεται νέα δεδομένα (updateable) και να διερευνηθούν αποδόσεις μιας τέτοιας εφαρμογής. Ταυτόχρονα υπάρχουν πολυάριθμοι κατηγοριοποιητές τους οποίους δεν έχουμε εξετάσει σε αυτή την εργασία. Επίσης μπορεί να εξετασθεί κάποια νέα μετρική ομοιότητας γράφων για καλύτερη απόδοση σε χρόνο και μνήμη.

Το επόμενο φυσικό στάδιο είναι να εξεταστεί η προσέγγιση μου για το μοντέλο κατηγοριοποίησης ροής δεδομένων, σαν αυτό τους σχήματος 6.5 που χρησιμοποιούν οι Bifet et al. στο εργαλείο MOA. Δηλαδή να εξεταστεί κατά πόσον μια τέτοια προσέγγιση θα μπορεί να παραλλαχθεί ώστε να εκτελεί τους υπολογισμούς της με περιορισμένο χρόνο και μνήμη και θα είναι σε θέση να κατηγοριοποιεί ανα πάσα στιγμη.

Ταυτόχρονα θα μπορούσε να διερευνηθεί η εφαρμοσιμότητα της προσέγγισης μου σε ένα σύστημα κατανεμημένης επεξεργασίας. Δηλαδή για παράδειγμα να χρησιμοποιηθεί το Apache Spark για να παραλληλοποιηθούν τα στοιχειώδη κομάτια της διαδικασίας. Αντίστοιχα για την περίπτωση επεξεργασίας ροών δεδομένων με το Apache Storm θα μπορεί η προσέγγιση αυτή να χρησιμοποιήσει τα προτερήματα αυτού του συστήματος για να επιχηρήσει να εκτελέσει κατηγοριοποίηση του Twitter Stream σε πραγματικό χρόνο.

Μέρος IX

ΚΩΔΙΚΑΣ

10

ΚΩΔΙΚΑΣ

10.1 *NgramTester.java*

```
package myPackage;

import java.io.File;
import java.io.IOException;

import weka.core.converters.ArffSaver;

public class NgramTester {

  private static long goldTime;

  @SuppressWarnings("unused")
  public static void main(String[] args) {
    SystemInfo sysinfo = new SystemInfo();
    long startTime = System.currentTimeMillis();
    if (ProjectConstants.dataSize > 1600000) {
      System.out.println("Please use an input size smaller than
          1600000");
    } else {
      ReadCVS reader;
      NgramHash goldenP, goldenN;
      String csvFileP = ProjectConstants.myPath + "nghPositive" +
           ProjectConstants.dataSize + "_"
          + ProjectConstants.nGramSize + "gram.csv";
      String csvFileN = ProjectConstants.myPath + "nghNegative" +
           ProjectConstants.dataSize + "_"
          + ProjectConstants.nGramSize + "gram.csv";
      File fileP = new File(csvFileP);
      File fileN = new File(csvFileN);
      if ((fileP.length() == 0) || (fileN.length() == 0)) {
        System.out.println("No file found. Creating the golden
            roules normally.");
        // This reader with the method getNextTweet returns
            tweets from
        // the
        // input dataset in the form of an iterator.
        reader = new ReadCVS();
        // Create the two (pos,neg) Golden Roule Tables.
        GoldenHashTable myGHT = new GoldenHashTable(reader);
        goldenP = (myGHT).getPositiveGold(); // goldenP.
            normalizeNGH();
```

```
        goldenN = (myGHT).getNegativeGold();// goldenN.
            normalizeNGH();
        System.out.println("Success Building merged hash.P= " +
            goldenP.size() + " N=" + goldenN.size());
        // System.out.println(sysinfo.MemInfo());
        goldTime = System.currentTimeMillis();
        System.out.println("Building the golden roules took " + (
            goldTime - startTime) / 1000 + " s");
        System.out.println(sysinfo.MemInfo());
        // HashCSVWrite writes to csv file all the edges from the
             golden
        // rules
        HashCSVWrite hcwrite = new HashCSVWrite(goldenP, goldenN,
             csvFileP, csvFileN);
      } else {
        System.out.println("File found. Creating the golden
            roules from file.");
        // HashCSVReader reads from file all the edges for the
            golden
        // rules
        HashCSVReader hcread = new HashCSVReader(csvFileP,
            csvFileN);
        goldenP = hcread.getP();
        goldenN = hcread.getN();
        System.out.println(
            "Success reading the golden roules from file.P= " +
                goldenP.size() + " N=" + goldenN.size());
        goldTime = System.currentTimeMillis();
        System.out.println("Building the golden roules took " + (
            goldTime - startTime) / 1000 + " s");
        System.out.println(sysinfo.MemInfo());
      }

      goldTime = System.currentTimeMillis();
      //
      // Let's try comparing all the tweets with the merged
          hashsets.
      // We will use all of the dataset to compare each
          individual tweet
      // and feed the similarity to our machine learning
          algorithms.
      reader = new ReadCVS();

      NgramHashComparer a = new NgramHashComparer();
      // NgramHashComparer creates an Instances object
      wekaData wekadata = a.myComparer(goldenP, goldenN, reader);
      System.out.println("Weka Dataset size=" + wekadata.isAllSet
          .size());
      long endTime = System.currentTimeMillis();
      System.out.println("Comparing took " + (endTime - goldTime)
           / 1000 + " s");
      System.out.println(sysinfo.MemInfo());
```

```
      // ArffSaver is weka's class for writing Instances to arff
          files
74    ArffSaver saver = new ArffSaver();
      saver.setInstances(wekadata.isAllSet);
      try {

        saver.setFile(new File(ProjectConstants.myPath + "/test2-
            " + (ProjectConstants.dataSize / 1000) + "-"
79          + ProjectConstants.nGramSize + "gram-NVS2.arff"));
        saver.writeBatch();
      } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace(System.out);
84    }

      // wekadata.runTheClassifier();
      long classifyTime = System.currentTimeMillis();
      System.out.println("Classifier took " + (classifyTime -
          endTime) / 1000 + " s");
89    System.out.println(sysinfo.MemInfo());
      return;
    }
  }

94 }
```

10.2 *GoldenHashTable.java*

```
1  package myPackage;

   public class GoldenHashTable {

     private NgramHash positiveGold, negativeGold;
6    ReadCVS reader = null;
     public NgramHash getPositiveGold() {
       return positiveGold;
     }

11   public NgramHash getNegativeGold() {
       return negativeGold;
     }

     public GoldenHashTable(ReadCVS reader2) {
16     this.reader = reader2;


       // We use the parser to get the tweets in
       // the form of an iterator.
21     Tweet temp = reader.getnextTweet();
       int i = 0;
       //positiveGold=new NgramHash();
```

```
    //negativeGold=new NgramHash();
    // Then we create the NgramHash for each individual tweet and
         merge to
    // the golden NgramHash tables accordingly.
    positiveGold = new NgramHash("", ProjectConstants.nGramSize);
    negativeGold = new NgramHash("", ProjectConstants.nGramSize);
    if (temp.getOppinion().equals("0")) {
      negativeGold = new NgramHash(temp.getTweet(),
          ProjectConstants.nGramSize);
    } else if (temp.getOppinion().equals("4")) {
      positiveGold = new NgramHash(temp.getTweet(),
          ProjectConstants.nGramSize);
    } else {
      System.out.println("Error in oppinion parsing. Value= " +
          temp.getOppinion());
    }

    while (((temp = reader.getnextTweet()) != null)){
        //&& (i < 10000)) {
      if (temp.getOppinion().equals("0")) {
        negativeGold = negativeGold.hashMerge(new NgramHash(temp.
            getTweet(), ProjectConstants.nGramSize));

      } else if (temp.getOppinion().equals("4")) {
        positiveGold = positiveGold.hashMerge(new NgramHash(temp.
            getTweet(), ProjectConstants.nGramSize));
      } else {
        System.out.println("Error in oppinion parsing. Value= " +
             temp.getOppinion());
      }

      i++;

    }
    System.out.println("Number of tweets prosecced during Golden
        creation= "+i);

  }



}
```

10.3 *HashCSVReader.java*

```
package myPackage;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
import java.io.IOException;

public class HashCSVReader {
  /*
  String csvFileP = ProjectConstants.myPath + "nghPositive" +
      ProjectConstants.dataSize + "-"
      + ProjectConstants.nGramSize + "gram.csv";
  String csvFileN = ProjectConstants.myPath + "nghNegative" +
      ProjectConstants.dataSize + "-"
      + ProjectConstants.nGramSize + "gram.csv";
  */
  BufferedReader br = null;
  String line = "";
  String cvsSplitBy = "\",\"";
  private NgramHash p, n;

  public NgramHash getP() {
    return p;
  }

  public NgramHash getN() {
    return n;
  }

  public HashCSVReader(String csvFileP,String csvFileN) {
    p = new NgramHash("", ProjectConstants.nGramSize);
    n = new NgramHash("", ProjectConstants.nGramSize);
    HashRead(p, csvFileP);
    HashRead(n, csvFileN);
  }

  public void HashRead(NgramHash ngh, String inp) {
    try {

      br = new BufferedReader(new FileReader(inp));
      line=br.readLine();
      while ((line = br.readLine()) != null) {

        // use comma as separator
        String[] str = line.split(cvsSplitBy);
        ngh.put(quoteCleaner(str[0]), Integer.parseInt(
            quoteCleaner(str[1])));
      }

    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (br != null) {
        try {
          br.close();
```

```
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
  }
  public String quoteCleaner(String a){
    String temp=a;
    if (temp.charAt(temp.length()-1)=='\"'){
      temp=temp.substring(0,temp.length()-1);

    }
    if (temp.charAt(0)=='\"'){
      temp=temp.substring(1,temp.length());

    }
    return temp;
  }
}
```

10.4 *HashCSVWrite.java*

```
package myPackage;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Map.Entry;

public class HashCSVWrite {
  public HashCSVWrite(NgramHash p, NgramHash n,String csvFileP,
      String csvFileN) {
    generateCsvFile(p, csvFileP);
    generateCsvFile(n, csvFileN);
  }

  private static void generateCsvFile(NgramHash ngh, String
      sFileName) {
    try {
      FileWriter writer = new FileWriter(sFileName);
      writer.append("nGramName");
      writer.append(',');
      writer.append("Weight");
      writer.append('\n');
      for (Entry<String, Integer> entry : ngh.entrySet()) {
        writer.append("\""+entry.getKey()+"\"");
        writer.append(',');
        writer.append("\""+entry.getValue().toString()+"\"");
        writer.append('\n');
      }

      // generate whatever data you want
```

```
      writer.flush();
      writer.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

10.5 *Ngram.java*

```
package myPackage;

public class Ngram {
  private String myName;

  public String getMyName() {
    return myName;
  }

  // Constructor
  public Ngram(String a) {
    this.myName = a;

  }

}
```

10.6 *NgramEdge.java*

```
package myPackage;

public class NgramEdge {
  Ngram a;
  Ngram b;
  Integer freq;

  public double getFreq() {
    return freq;
  }

  public void setFreq(Integer d) {
    this.freq = d;
  }

  public NgramEdge(Ngram a, Ngram b, int freq) {
    this.a = a;
    this.b = b;
```

```
    this.freq = freq;

  }

  public boolean equals(NgramEdge that) {
    if ((this.a.equals(that.a))&&(this.b.equals(that.b))
        &&(this.freq==(that.freq))){
      return true;
    }
    else return false;
  }

  public String myhashCode() {
    // TODO Auto-generated method stub
    return (a.getMyName()+"—>"+b.getMyName());
  }


}
```

10.7 *NgramHash.java*

```
package myPackage;

import java.util.HashMap;

public class NgramHash extends HashMap<String, Integer> {
  /**
   *
   */
  private static final long serialVersionUID = 1L;
  private int n;

  // HashMap<String, NgramEdge> edges;
  /*
   * public HashMap<String, NgramEdge> getEdges() { return edges;
       }
   */
  public NgramHash(String inputString, int nSize) {
    n = nSize;
    // edges = new HashMap<String, NgramEdge>();
    if (inputString.length() >= 1) {
      String[] myNgrams = nGramSplit(inputString);

      for (int i = 0; i < myNgrams.length; i++) {
        // Here we create the edges
        Ngram newNgram = new Ngram(myNgrams[i]);
        // System.out.println(myNgrams[i]);

        for (int j = i; (j < myNgrams.length) && (j < i + n); j
            ++) {
```

```
28            Ngram secNgram = new Ngram(myNgrams[j]);
              NgramEdge newEdge = new NgramEdge(newNgram, secNgram,
                  1);
              this.put(newEdge.myhashCode(), newEdge.freq);
            }
          }
33      }
    }

    public NgramHash(int i) {
      // TODO Auto-generated constructor stub
38    super(i);
    }

    public NgramHash() {
      // TODO Auto-generated constructor stub
43    super();
    }

    private String[] nGramSplit(String inputString) {
      int count = 0;
48    String[] myArray = new String[inputString.length() - n];
      while (inputString.length() > (count + n)) {

        myArray[count] = inputString.substring(count, count + n);
        count++;
53    }
      return myArray;

    }

58  ////////////////////////////////////////////////////
    // We overwrite toString for easy printing.
    String a = "";

    public String toString() {
63
      for (java.util.Map.Entry<String, Integer> myentry : this.
          entrySet()) {
        a += (myentry.getKey()) + " " + myentry.getValue();
        a += "\n";
      }
68
      return a;
    }
    ////////////////////////////////////////////////////

73  public NgramHash hashMerge(NgramHash smallOne) {
      // We will merge all edges from the small table to this one.
          If an
      // edge
      // already exists we add the two frequencies.
```

```
    for (java.util.Map.Entry<String, Integer> smallentry :
        smallOne.entrySet()) {
      String tempkey = smallentry.getKey();
      if (this.containsKey(tempkey)) {
        Integer temp = this.get(tempkey) + smallentry.getValue();
        this.put(tempkey, temp);
      } else {

        this.put(tempkey, smallentry.getValue());
      }
    }
    return this;

  }

}
```

10.8 *ReadCVS.java*

```
package myPackage;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Iterator;

public class ReadCVS {

  String csvFile = ProjectConstants.myPath + "/training.1600000.
      processed.noemoticon.csv";

  BufferedReader br = null;
  String line = "";
  String cvsSplitBy = "\",\"";
  Long lineNo;
  HashSet<Tweet> myTable = new HashSet<Tweet>();
  Iterator<Tweet> myIterator;

  public Tweet getnextTweet() {
    if (myIterator.hasNext()) {
      return myIterator.next();
    }
    return null;
  }

  public ReadCVS() {
    try {
      lineNo=0L;
      br = new BufferedReader(new FileReader(csvFile));
```

```
      if (lineNo < (800000-(ProjectConstants.dataSize/2))){
      while (((line = br.readLine()) != null)&& (lineNo <
          (800000-(ProjectConstants.dataSize/2)))) {
        lineNo++;
      }}
      lineNo=0L;

      String[] country;
      while (((line = br.readLine()) != null)&& (lineNo<
          ProjectConstants.dataSize)) {
        // use comma as separator
        country = line.split(cvsSplitBy);
        String sent=quoteCleaner(country[5]);

        String twe=quoteCleaner(country[0]);

        //System.out.println(sent +"   "+twe);
        myTable.add(new Tweet(sent, twe));
        lineNo++;
      }


    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (br != null) {
        try {
          br.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }

    myIterator = myTable.iterator();
    //System.out.println("CvsReader lineNO: "+lineNo);
  }
  public String quoteCleaner(String a){
    String temp=a;
    if (temp.charAt(temp.length()-1)=='\"'){
      temp=temp.substring(0,temp.length()-1);

    }
    if (temp.charAt(0)=='\"'){
      temp=temp.substring(1,temp.length());

    }
    return temp;
  }

}
```

10.9 *NgramHashComparer.java*

```
package myPackage;

import java.util.Map.Entry;

public class NgramHashComparer {
  public NgramHashComparer() {
  }

  @SuppressWarnings("unused")
  public wekaData myComparer(NgramHash goldenP, NgramHash goldenN
      , ReadCVS reader) {
    wekaData wekadata = new wekaData(1);
    Tweet temp;

    // We create two sets of vectors one for comparing with the
        negative and
    // one with the positive golden sets.
    Integer[] bigArrayP = new Integer[10000];
    Integer[] bigArrayN = new Integer[10000];
    Integer[] smallArrayP = new Integer[10000];
    Integer[] smallArrayN = new Integer[10000];
    int counter = 0;
    while (((temp = reader.getnextTweet()) != null)) {
      // for (int p = 0; p < 10000; p++) {
      int i = 0, j = 0;
      counter++;
      // if ((temp = reader.getnextTweet()) != null) {

      NgramHash newnh = new NgramHash(temp.getTweet(),
          ProjectConstants.nGramSize);
      double valueRatioP = 0, valueRatioN = 0;
      for (Entry<String, Integer> entry : newnh.entrySet()) {
        String tempkey = entry.getKey();
        if (goldenP.containsKey(tempkey)) {
          bigArrayP[i] = goldenP.get(tempkey);
          smallArrayP[i] = entry.getValue();
           valueRatioP += min(bigArrayP[i], smallArrayP[i]) / max
               (bigArrayP[i], smallArrayP[i]);
          i++;
        }
        if (goldenN.containsKey(tempkey)) {
          bigArrayN[j] = goldenN.get(tempkey);
          smallArrayN[j] = entry.getValue();
           valueRatioN += min(bigArrayN[j], smallArrayN[j]) /max(
               bigArrayN[j], smallArrayN[j]);
          j++;
        }
```

```
43      }

        double containmentSimilarityP = 0, containmentSimilarityN =
             0, valueSimilarityP = 0, valueSimilarityN = 0,
            normalizedValueSimilarityP = 0,
                normalizedValueSimilarityN = 0;

48      if ((goldenP.size() == 0) || (goldenN.size() == 0) || newnh
            .size() == 0) {
          System.out.println(goldenP.size() + ", " + goldenN.size()
               + ", " + newnh.size());
        } else {
          // Containment similarity is the proportion of edges that
               are
          // shared
53
          //containmentSimilarityP = i / min(goldenP.size(), newnh.
              size());
          //containmentSimilarityN = j / min(goldenN.size(),newnh.
              size());

          // Value Similarity considers the weights.
58         //valueSimilarityP = valueRatioP / max(newnh.size(),
               goldenP.size());
          // valueSimilarityN = valueRatioN / max(newnh.size(),
              goldenN.size());

          // Normalized VS decouples from the largest graph's size.
          // NVS=VS/(min()/max()) or :
63         normalizedValueSimilarityP = valueRatioP / min(newnh.
              size(), goldenP.size());
           normalizedValueSimilarityN = valueRatioN / min(newnh.
              size(), goldenN.size());
        }

        //double CSP = cosineSimilarity(bigArrayP, smallArrayP, i);
68      //double eP = euclideanDistance(bigArrayP, smallArrayP,i);
        //double CSN = cosineSimilarity(bigArrayN, smallArrayN, j);
        //double eN = euclideanDistance(bigArrayN, smallArrayN,j);

        //Depending on the number of features we choose the
            appropriate variation of addInstance
73      /*
         * wekadata.addInstance(containmentSimilarityP,
         * containmentSimilarityN, valueSimilarityP,
             valueSimilarityN,
         * normalizedValueSimilarityP, normalizedValueSimilarityN,
         * toSentiment(temp.getOppinion()));
78       */
        wekadata.addInstance(normalizedValueSimilarityP,
            normalizedValueSimilarityN,toSentiment(temp.getOppinion
            ()));
    }
```

```
    System.out.println("Number of tweet read during comparing= "
        + counter);
    return wekadata;

  }

  private double min(Integer bigArrayP, Integer smallArrayP) {
    // TODO Auto-generated method stub
    if (bigArrayP > smallArrayP) {
      return smallArrayP;
    } else
      return bigArrayP;
  }

  private double max(Integer bigArrayP, Integer smallArrayP) {
    // TODO Auto-generated method stub
    if (bigArrayP < smallArrayP) {
      return smallArrayP;
    } else
      return bigArrayP;
  }

  public static double cosineSimilarity(Integer[] bigArrayP,
      Integer[] smallArrayP, int length) {
    double dotProduct = 0.0;
    double normA = 0.0;
    double normB = 0.0;
    for (int j = 0; j < length; j++) {
      dotProduct += bigArrayP[j] * smallArrayP[j];
      normA += Math.pow(bigArrayP[j], 2);
      normB += Math.pow(smallArrayP[j], 2);
    }
    return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
  }

  public double euclideanDistance(Integer[] bigArrayP, Integer[]
      smallArrayP,int alength) {
    double Sum = 0.0;
    for (int i = 0; i < alength; i++) {
      Sum = Sum + (bigArrayP[i] - smallArrayP[i])*(bigArrayP[i] -
           smallArrayP[i]);
    }
    return Math.sqrt(Sum);
  }

  public String toSentiment(String a) {
    if (a.equals("0"))
      return ("negative");
    else if (a.equals("4"))
      return ("positive");
    else
      return null;
```

```
  }

}
```

10.10 *Tweet.java*

```
package myPackage;

public class Tweet {
  private String myTweet;
  private String oppinion;
  public Tweet(String myTweet, String oppinion) {
    this.myTweet = preProcessTweet(myTweet);
    this.oppinion = oppinion;
    //System.out.println(myTweet+"---->"+oppinion);
  }
  public  String preProcessTweet(String tweet){
    String temp;
    //Convert to lower case
    temp = tweet.toLowerCase();
      //Convert www.* or https?://* to URL
    String regex = "(https?|ftp|file)://[-a-zA-Z0-9+&@#/%?=~_
        |!:,.;]*[-a-zA-Z0-9+&@#/%=~_|]";
    temp=temp.replaceAll(regex, "URL");

      //Convert @username to AT_USER
    temp = temp.replaceAll("@[-a-zA-Z0-9+&@#/%?=~_|!:,.;]*[-a-zA-
        Z0-9+&@#/%=~_|]","AT_USER");
      //Remove additional white spaces
    temp = temp.replaceAll("\\s\\s", " ");
      //Replace #word with word
    temp = temp.replaceAll("#(?i)(-a-zA-Z0-9+&@#/%=~_|)", "$1");
      return temp;

  }
  public String getTweet() {
    return myTweet;
  }
  public void setTweet(String myTweet) {
    this.myTweet = myTweet;
  }
  public String getOppinion() {
    return oppinion;
  }
  public void setOppinion(String oppinion) {
    this.oppinion = oppinion;
  }

}
```

10.11 *WekaExecute.java*

```
package myPackage;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.*;
import weka.classifiers.functions.Logistic;
import weka.classifiers.functions.SMO;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.core.converters.ArffLoader.ArffReader;

public class WekaExecute {

  public static void main(String[] args) throws IOException {
    PrintWriter writer = new PrintWriter(ProjectConstants.myPath
        + "/test3-output.txt", "UTF-8");

    // TODO Auto-generated method stub
    SystemInfo sysinfo = new SystemInfo();
    BufferedReader reader;
    Instances data; // contains the full dataset we wann create
        train/test
            // sets from
    int seed = 4; // the seed for randomizing the data
    int folds = 2; // the number of folds to generate, >=2
    /*
    String[] filenames = { "gram-NVS.arff", "gram-Cos.arff", "
        gram-Eu.arff", "gram-CS.arff", "gram-VS.arff",
        "gram-CosEU.arff", "gram-CosCon.arff", "gram-CosNVS.arff
            ", "gram-ContNVS.arff", "gram-VSNVS.arff" };
    */
    String[] filenames ={"gram-NVS2.arff","gram-VS2.arff","gram-
        Cont2.arff",
        "gram-Eu2.arff","gram-Cos2.arff"};
    for (String filename : filenames) {
      writer.println();
      writer.println(filename);
      writer.println();

      reader = new BufferedReader(new FileReader(ProjectConstants
          .myPath + "/test2-"
```

```
44          + (ProjectConstants.dataSize / 1000) + "_" +
                ProjectConstants.nGramSize + filename));
        ArffReader arff = new ArffReader(reader);
        data = arff.getData();
        data.setClassIndex(data.numAttributes() - 1);
        Instances ranData = new Instances(data);
49      Random rand = new Random(seed);
        ranData.randomize(rand);
        double[] corClassified = new double[10];

        Classifier[] cModelarray = { (Classifier) new NaiveBayes(),
             (Classifier) new Logistic(),
54          (Classifier) new SMO(), (Classifier) new J48() };
        for (Classifier cModel : cModelarray) {
          double corClassSum = 0;
          long startTime = System.currentTimeMillis();
          ranData.randomize(rand);
59        for (int n = 0; n < 2; n++) {

            Instances train = ranData.trainCV(folds, n);
            Instances test = ranData.testCV(folds, n);

64          // Classifier cModel = (Classifier) new NaiveBayes();
            Evaluation eTest = null;
            // further processing, classification, etc.
            try {

69            cModel.buildClassifier(train);
              eTest = new Evaluation(test);
              eTest.evaluateModel(cModel, test);
              corClassified[n] = eTest.pctCorrect();
              corClassSum += corClassified[n];
74            // writer.println("ERROR RATE: "+eTest.errorRate());
              // writer.println("Correctly Classified:
              // "+eTest.pctCorrect());
              // writer.println(strSummary);
            } catch (Exception e) {
79            // TODO: handle exception
              e.printStackTrace(writer);
            }
            /*
             * // Get the confusion matrix double[][] cmMatrix =
84           * eTest.confusionMatrix(); for (int i = 0; i <
             * cmMatrix.length; i++) { for (int j = 0; j <
             * cmMatrix[i].length; j++) {
             * writer.print(cmMatrix[i][j] + " "); }
             * writer.println(""); } writer.println(cmMatrix);
89           */
          }
          writer.println(cModel.getClass());
          NumberFormat formatter = new DecimalFormat("#0.00");
```

```
94          writer.println("Average ratio of correctly classified: "
                + formatter.format(corClassSum / 2) + "%");
            long goldTime = System.currentTimeMillis();
            writer.println("Building the golden roules took " + (
                goldTime - startTime) / 1000 + " s");
            writer.println(sysinfo.MemInfo());
          }
99
        }
        writer.close();
      }
    }
```

10.12 *wekaData.java*

```
    package myPackage;
2
    import java.util.ArrayList;

    import weka.classifiers.Classifier;
    import weka.classifiers.Evaluation;
7   import weka.classifiers.bayes.BayesNet;
    import weka.core.Attribute;
    import weka.core.DenseInstance;
    import weka.core.Instance;
    import weka.core.Instances;
12
    public class wekaData{
      Instances isTrainingSet, isTestingSet, isAllSet;
      ArrayList<Attribute> fvWekaAttributes;
      // Create a naive bayes classifier
17    Classifier cModel = (Classifier) new BayesNet();
      Evaluation eTest = null;
      int folds = 10;

      public wekaData   (int numOfFeatures) {
22      if (numOfFeatures == 1) {
          // Declare two numeric attributes
          Attribute Attribute1 = new Attribute("CosinePos");
          Attribute Attribute2 = new Attribute("CosineNeg");

27        // Declare the class attribute along with its values
          ArrayList<String> fvClassVal = new ArrayList<String>();
          fvClassVal.add("positive");
          fvClassVal.add("negative");
          Attribute ClassAttribute = new Attribute("theClass",
              fvClassVal);
32
          // Declare the feature vector
          fvWekaAttributes = new ArrayList<Attribute>();
          fvWekaAttributes.add(Attribute1);
```

```
        fvWekaAttributes.add(Attribute2);
        fvWekaAttributes.add(ClassAttribute);

        // Create an empty training set
        // (name, attributes list, size)
        this.isAllSet = new Instances("Rel", fvWekaAttributes, 2);
        this.isTrainingSet = new Instances("Rel", fvWekaAttributes,
             2);
        this.isTestingSet = new Instances("Rel", fvWekaAttributes,
            2);
        // Set class index
        this.isTrainingSet.setClassIndex(2);
        this.isTestingSet.setClassIndex(2);
        this.isAllSet.setClassIndex(2);
      } else if (numOfFeatures == 2) {
        // Declare two numeric attributes
        Attribute Attribute1 = new Attribute("CosinePos");
        Attribute Attribute2 = new Attribute("CosineNeg");
        Attribute Attribute3 = new Attribute("EuclideanPos");
        Attribute Attribute4 = new Attribute("EuclideanNeg");
        // Declare the class attribute along with its values
        ArrayList<String> fvClassVal = new ArrayList<String>();
        fvClassVal.add("positive");
        fvClassVal.add("negative");
        Attribute ClassAttribute = new Attribute("theClass",
            fvClassVal);

        // Declare the feature vector
        fvWekaAttributes = new ArrayList<Attribute>();
        fvWekaAttributes.add(Attribute1);
        fvWekaAttributes.add(Attribute2);
        fvWekaAttributes.add(Attribute3);
        fvWekaAttributes.add(Attribute4);
        fvWekaAttributes.add(ClassAttribute);

        // Create an empty training set
        // (name, attributes list, size)
        this.isAllSet = new Instances("Rel", fvWekaAttributes, 4);
        this.isTrainingSet = new Instances("Rel", fvWekaAttributes,
             4);
        this.isTestingSet = new Instances("Rel", fvWekaAttributes,
            4);
        // Set class index
        this.isTrainingSet.setClassIndex(4);
        this.isTestingSet.setClassIndex(4);
        this.isAllSet.setClassIndex(4);
      } else if (numOfFeatures == 3) {
        // Declare two numeric attributes
        Attribute Attribute1 = new Attribute("1Pos");
        Attribute Attribute2 = new Attribute("1Neg");
        Attribute Attribute3 = new Attribute("2Pos");
        Attribute Attribute4 = new Attribute("2Neg");
        Attribute Attribute5 = new Attribute("3Pos");
```

```
    Attribute Attribute6 = new Attribute("3Neg");
    // Declare the class attribute along with its values
    ArrayList<String> fvClassVal = new ArrayList<String>();
    fvClassVal.add("positive");
    fvClassVal.add("negative");
    Attribute ClassAttribute = new Attribute("theClass",
        fvClassVal);

    // Declare the feature vector
    fvWekaAttributes = new ArrayList<Attribute>();
    fvWekaAttributes.add(Attribute1);
    fvWekaAttributes.add(Attribute2);
    fvWekaAttributes.add(Attribute3);
    fvWekaAttributes.add(Attribute4);
    fvWekaAttributes.add(Attribute5);
    fvWekaAttributes.add(Attribute6);
    fvWekaAttributes.add(ClassAttribute);

    // Create an empty training set
    // (name, attributes list, size)
    this.isAllSet = new Instances("Rel", fvWekaAttributes, 6);
    this.isTrainingSet = new Instances("Rel", fvWekaAttributes,
         6);
    this.isTestingSet = new Instances("Rel", fvWekaAttributes,
        6);
    // Set class index
    this.isTrainingSet.setClassIndex(6);
    this.isTestingSet.setClassIndex(6);
    this.isAllSet.setClassIndex(6);
  } else {
    System.out.println("Please use 1-3 types of features");
  }
}

public void addInstance(double posSimilarity, double
    negSimilarity, String sentiment) {
  Instance iExample = new DenseInstance(4);
  iExample.setValue((Attribute) fvWekaAttributes.get(0),
      posSimilarity);
  iExample.setValue((Attribute) fvWekaAttributes.get(1),
      negSimilarity);
  iExample.setValue((Attribute) fvWekaAttributes.get(2),
      sentiment);

  this.isAllSet.add(iExample);
}

public void addInstance(double posSimilarity, double
    negSimilarity, double posEu, double negEu, String sentiment
    ) {
  Instance iExample = new DenseInstance(6);
  iExample.setValue((Attribute) fvWekaAttributes.get(0),
      posSimilarity);
```

```
127     iExample.setValue((Attribute) fvWekaAttributes.get(1),
            negSimilarity);
        iExample.setValue((Attribute) fvWekaAttributes.get(2), posEu)
            ;
        iExample.setValue((Attribute) fvWekaAttributes.get(3), negEu)
            ;
        iExample.setValue((Attribute) fvWekaAttributes.get(4),
            sentiment);

132     this.isAllSet.add(iExample);
    }
    public void addInstance(double Pos1, double Neg1,double Pos2,
        double Neg2, double Pos3, double Neg3, String sentiment) {
    Instance iExample = new DenseInstance(8);
    iExample.setValue((Attribute) fvWekaAttributes.get(0), Pos1);
137 iExample.setValue((Attribute) fvWekaAttributes.get(1), Neg1);
    iExample.setValue((Attribute) fvWekaAttributes.get(2), Pos2);
    iExample.setValue((Attribute) fvWekaAttributes.get(3), Neg2);
    iExample.setValue((Attribute) fvWekaAttributes.get(4), Pos3);
    iExample.setValue((Attribute) fvWekaAttributes.get(5), Neg3);
142 iExample.setValue((Attribute) fvWekaAttributes.get(6),
        sentiment);

    this.isAllSet.add(iExample);
    }
/*
147 public void addInstanceTraining(double posSimilarity, double
        negSimilarity, String sentiment) {
    Instance iExample = new DenseInstance(4);
    iExample.setValue((Attribute) fvWekaAttributes.get(0),
        posSimilarity);
    iExample.setValue((Attribute) fvWekaAttributes.get(1),
        negSimilarity);
    iExample.setValue((Attribute) fvWekaAttributes.get(2),
        sentiment);
152
    this.isTrainingSet.add(iExample);
    }

    public void addInstanceTesting(double posSimilarity, double
        negSimilarity, String sentiment) {
157 Instance iExample = new DenseInstance(4);
    iExample.setValue((Attribute) fvWekaAttributes.get(0),
        posSimilarity);
    iExample.setValue((Attribute) fvWekaAttributes.get(1),
        negSimilarity);
    iExample.setValue((Attribute) fvWekaAttributes.get(2),
        sentiment);

162 this.isTestingSet.add(iExample);
    }

    public void runTheClassifier() {
```

```
    for (int n = 0; n < folds; n++) {
      isTrainingSet = isAllSet.trainCV(folds, n);
      isTestingSet = isAllSet.testCV(folds, n);

      // further processing, classification, etc.
      try {
        cModel.buildClassifier(isTrainingSet);
        eTest = new Evaluation(isTrainingSet);
        eTest.evaluateModel(cModel, isTestingSet);
        String strSummary = eTest.toSummaryString();
        System.out.println(strSummary);
      } catch (Exception e) {
        // TODO: handle exception
        e.printStackTrace(System.out);
      }

      // Get the confusion matrix
      double[][] cmMatrix = eTest.confusionMatrix();
      for (int i = 0; i < cmMatrix.length; i++) {
        for (int j = 0; j < cmMatrix[i].length; j++) {
          System.out.print(cmMatrix[i][j] + " ");
        }
        System.out.println("");
      }
      System.out.println(cmMatrix);
    }

  }
  */
}
```

10.13 *ProjectConstants.java*

```
package myPackage;

public class ProjectConstants {

  public static final String myPath= "C:/Users/Mixalis/Documents/
      Jinsect/JinsectTest";
  public static final int nGramSize=4,dataSize= 1600000;
}
```

10.14 *SystemInfo.java*

```
package myPackage;
import java.io.File;
import java.text.NumberFormat;
```

```
public class SystemInfo {

    private Runtime runtime = Runtime.getRuntime();

    public String Info() {
        StringBuilder sb = new StringBuilder();
        sb.append(this.OsInfo());
        sb.append(this.MemInfo());
        sb.append(this.DiskInfo());
        return sb.toString();
    }

    public String OSname() {
        return System.getProperty("os.name");
    }

    public String OSversion() {
        return System.getProperty("os.version");
    }

    public String OsArch() {
        return System.getProperty("os.arch");
    }

    public long totalMem() {
        return Runtime.getRuntime().totalMemory();
    }

    public long usedMem() {
        return Runtime.getRuntime().totalMemory() - Runtime.
            getRuntime().freeMemory();
    }

    public String MemInfo() {
        NumberFormat format = NumberFormat.getInstance();
        StringBuilder sb = new StringBuilder();
        long maxMemory = runtime.maxMemory();
        long allocatedMemory = runtime.totalMemory();
        long freeMemory = runtime.freeMemory();
        /*
        sb.append("Free memory: ");
        sb.append(format.format(freeMemory / 1048576));
        sb.append("     ");
        sb.append("Allocated memory: ");
        sb.append(format.format(allocatedMemory / 1048576));
        sb.append("     ");
        sb.append("Max memory: ");
        sb.append(format.format(maxMemory / 1048576));
        sb.append("     ");
        sb.append("Total free memory: ");
        sb.append(format.format((freeMemory + (maxMemory -
            allocatedMemory)) / 1048576));
        sb.append("     ");
```

```
        */
        sb.append("Used memory: ");
        sb.append(format.format((maxMemory-(freeMemory + (
            maxMemory - allocatedMemory))) / 1048576));
        sb.append("     ");
        return sb.toString();

    }

    public String OsInfo() {
        StringBuilder sb = new StringBuilder();
        sb.append("OS: ");
        sb.append(this.OSname());
        sb.append("<br/>");
        sb.append("Version: ");
        sb.append(this.OSversion());
        sb.append("<br/>");
        sb.append(": ");
        sb.append(this.OsArch());
        sb.append("<br/>");
        sb.append("Available processors (cores): ");
        sb.append(runtime.availableProcessors());
        sb.append("<br/>");
        return sb.toString();
    }

    public String DiskInfo() {
        /* Get a list of all filesystem roots on this system */
        File[] roots = File.listRoots();
        StringBuilder sb = new StringBuilder();

        /* For each filesystem root, print some info */
        for (File root : roots) {
            sb.append("File system root: ");
            sb.append(root.getAbsolutePath());
            sb.append("<br/>");
            sb.append("Total space (bytes): ");
            sb.append(root.getTotalSpace());
            sb.append("<br/>");
            sb.append("Free space (bytes): ");
            sb.append(root.getFreeSpace());
            sb.append("<br/>");
            sb.append("Usable space (bytes): ");
            sb.append(root.getUsableSpace());
            sb.append("<br/>");
        }
        return sb.toString();
    }
}
```

BIBLIOGRAPHY

[1] URL https://en.wikipedia.org/wiki/Statistical_classification. (Cited on page 23.)

[2] URL https://en.wikipedia.org/wiki/C4.5_algorithm. (Cited on page 26.)

[3] URL https://en.wikipedia.org/wiki/Support_vector_machine. (Cited on page 27.)

[4] URL https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html. (Cited on pages 49 and 50.)

[5] URL http://www.cs.waikato.ac.nz/ml/weka/. (Cited on page 49.)

[6] URL http://help.sentiment140.com/for-students/. (Cited on page 50.)

[7] URL https://MichalisEllinas@bitbucket.org/ellinasm91/diplomatiki.git. (Cited on page 72.)

[8] URL http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. (Cited on page 5.)

[9] URL https://hadoop.apache.org/. (Cited on page 42.)

[10] URL https://mahout.apache.org/. (Cited on page 42.)

[11] URL https://en.wikipedia.org/wiki/Apache_Kafka. (Cited on page 43.)

[12] URL https://kafka.apache.org/documentation.html. (Cited on page 43.)

[13] URL http://zdatainc.com/2014/08/real-time-streaming-apache-spark-streaming/. (Cited on pages 43 and 44.)

[14] URL https://spark.apache.org/. (Cited on page 43.)

[15] URL https://spark.apache.org/streaming/. (Cited on page 44.)

[16] URL https://storm.apache.org/. (Cited on page 45.)

[17] URL http://zdatainc.com/2014/07/real-time-streaming-apache-storm-apache-kafka/. (Cited on page 45.)

[18] URL http://www.internetlivestats.com/twitter-statistics/. (Cited on page 72.)

[19] URL https://en.wikipedia.org/wiki/Euclidean_distance. (Cited on page 18.)

[20] URL https://en.wikipedia.org/wiki/Cosine_similarity. (Cited on page 18.)

[21] Fotis Aisopos, George Papadakis, Konstantinos Tserpes, and Theodora Varvarigou. Content vs. context for sentiment analysis: a comparative analysis over microblogs. *ICCS*, . (Cited on pages 10, 12, 14, and 35.)

[22] Fotis Aisopos, George Papadakis, Konstantinos Tserpes, and Theodora Varvarigou. Textual and contextual patterns for sentiment analysis over microblogs. *ICCS*, . (Cited on page 18.)

[23] Tumasjan Andranik, Sprenger Timm, O., Sandner Philipp, G., and Welpe Isabell, M. Predicting elections with twitter: What 140 characters reveal about political sentiment. *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*, 2010. (Cited on page 6.)

[24] Michael J Berry and Gordon Linoff. *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc., 1997. (Cited on page 4.)

[25] Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. In *Proceedings of the 13th International Conference on Discovery Science*, DS'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16183-9, 978-3-642-16183-4. URL http://dl.acm.org/citation.cfm?id=1927300.1927301. (Cited on pages 10 and 37.)

[26] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010. URL http://portal.acm.org/citation.cfm?id=1859903. (Cited on pages 10 and 38.)

[27] Albert Carles Bifet Figuerol, Geoffrey Holmes, Bernhard Pfahringer, and Ricard Gavaldà Mestre. Detecting sentiment change in twitter streaming data. In *Journal of Machine Learning Research: Workshop and Conference Proceedings Series*, pages 5–11, 2011. (Cited on pages 10 and 39.)

[28] E. Cambria and B. White. Jumping nlp curves: A review of natural language processing research [review article]. *IEEE Computational Intelligence Magazine*, 9(2):48–57, May 2014. ISSN 1556-603X. doi: 10.1109/MCI.2014.2307227. (Cited on page 11.)

[29] Erik Cambria, Bjorn Schuller, Yunqing Xia, and Catherine Havasi. New avenues in opinion mining and sentiment analysis. *IEEE Intelligent Systems*, 2013. (Cited on page 6.)

[30] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1:12, 2009. (Cited on pages 10 and 32.)

[31] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. *In proceedings of the Eleventh Conference of Uncertainty in Artificial Intelligence*. (Cited on page 24.)

[32] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. 2011. (Cited on page 17.)

[33] T.M. Mitchell. *Machine Learning*. McGraw Hill, 2016. (Cited on pages 24 and 25.)

[34] Mladen Nikolic. Measuring similarity of graph nodes by neighbor matching. *Intell. Data Anal.*, 16(6):865–878, 2012. ISSN 1088-467X. doi: 10.3233/IDA-2012-00556. URL http://dx.doi.org/10.3233/IDA-2012-00556. (Cited on page 17.)

[35] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326, 2010. (Cited on pages 10 and 31.)

[36] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2008. (Cited on pages 5 and 6.)

[37] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. *Appears in Proc. 2002 Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, 2002. (Cited on pages 10, 6, 27, and 31.)

[38] Evangelos Psomakelis, Konstantinos Tserpes, Dimosthenis Anagnostopoulos, and Theodora Varvarigou. Comparing methods for twitter sentiment analysis. (Cited on pages 10, 13, 32, and 33.)

[39] J.W. Raymond, E.J. Gardiner, and P. Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *THE COMPUTER JOURNAL*, 45(6), 2002. (Cited on page 17.)

[40] Daniel Robenek, Jan Platos, and Vaclav Snasel. Efficient in-memory data structures for n-grams indexing. (Cited on pages 10 and 34.)

[41] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Alan Apt, 1995. (Cited on page 25.)

[42] Cosma Rohilla Shalizi. *Advanced Data Analysis from an Elementary Point of View*. Cambridge University Press. (Cited on page 25.)

[43] Angelos Valsamis, Konstantinos Tserpes, Dimitrios Zissis, Dimosthenis Anagnostopoulos, and Theodora Varvarigou. Employing traditional machine learning algorithms for big

data streams analysis: The case of object trajectory prediction. *Journal of Systems and Software*, pages –, 2016. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2016.06.016. URL http://www.sciencedirect.com/science/article/pii/S016412121630084X. (Cited on page 36.)

[44] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. A comparative study of tf*idf, {LSI} and multi-words for text classification. *Expert Systems with Applications*, 38(3):2758 – 2765, 2011. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2010.08.066. URL http://www.sciencedirect.com/science/article/pii/S0957417410008626. (Cited on page 12.)