ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση Συναισθημάτων Κειμένου σε Hadoop & Spark

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Μήλας

Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης

Vasileios Mamalis
Digitally signed by Vasileios Mamalis
Date: 2021.03.13 00:28:48 +02'00'

Αθήνα, Μάρτιος 2021

Ανάλυση Συναισθημάτων Κειμένου σε Hadoop & Spark

Αλέξανδρος Μήλας

Επιβλέπων Καθηγητής: Βασίλειος Μάμαλης

Εξεταστική Επιτροπή Διπλωματικής Εργασίας:

Βασίλειος Μάμαλης	Γραμματή Πάντζιου	Χρήστος Σκουρλάς
Καθηγητής ΠΑ.Δ.Α	Καθηγήτρια ΠΑ.Δ.Α	Καθηγητής ΠΑ.Δ.Α

Αθήνα, Μάρτιος 2021

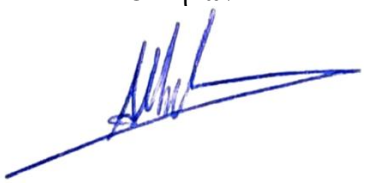
Δήλωση Συγγραφέα Διπλωματικής Εργασίας

Ο κάτωθι υπογεγραμμένος Αλέξανδρος Μήλας του Νικολάου, με αριθμό μητρώου 711141293 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών



Ευχαριστίες

Θα ήθελα πρωτίστως να ευχαριστήσω τον επιβλέποντα καθηγητή μου κύριο Βασίλειο Μάμαλη ο οποίος με την υπομονή και την πολύτιμη βοήθεια του καθώς και την εμπιστοσύνη που μου έδειξε κατά την εκπόνηση της παρούσας εργασίας, διευκόλυνε στον υπέρτατο βαθμό την πορεία της μελέτης ώστε να αντιμετωπιστούν τα εμπόδια ή οι τεχνικές δυσκολίες που προέκυψαν εν μέσω μιας έκτακτης και μεταβατικής περιόδου για όλους μας.

Επίσης θα ήθελα να ευχαριστήσω τους αξιότιμους καθηγητές κύριο Χρήστο Σκουρλά και κυρία Γραμματή Πάντζιου που αποτελούν μαζί με τον κύριο Μάμαλη την εξεταστική επιτροπή η οποία αφιέρωσε τον πολύτιμο χρόνο της πάνω στην αξιολόγηση αυτής της εργασίας, θέτωντας παράλληλα ιδιαίτερα ενδιαφέρουσες παρατηρήσεις και εύστοχες επισημάνσεις.

Περίληψη

Τα τελευταία χρόνια που ο όγκος των απαιτούμενων προς επεξεργασία δεδομένων σε (εμπορικά, ερευνητικά, και άλλα) συστήματα ολοένα και αυξάνεται, υπάρχει έντονο ενδιαφέρον σε ό,τι αφορά την τις τεχνικές διαχείρισης αυτού του όγκου με αποδοτικό τρόπο και ικανοποιητικά αποτελέσματα. Μία από τις τεχνικές που προτιμούνται για τέτοια συστήματα είναι η επεξεργασία όγκου δεδομένων στην κλίμακα των big data χρησιμοποιώντας κοινά διαθέσιμο εξοπλισμό που βρίσκεται εύκολα στο εμπόριο για να σχεδιαστεί και εφαρμοστεί μια δομή κατανεμημένου περιβάλλοντος. Σκοπός αυτής της προσπάθειας είναι να υποστηρίζεται η παράλληλη εκτέλεση εργασιών για διαφορετικά κομμάτια δεδομένων, ώστε να επιτευχθεί μία επιτάχυνση του χρόνου ολοκλήρωσης, καθώς και καλύτερη συμπεριφορά κλιμάκωσης ανάλογα με το μέγεθος των δεδομένων που δίνονται στην είσοδο, σε σχέση με κοινές σειριακές υλοποιήσεις. Ένα προγραμματιστικό μοντέλο στο οποίο βασίζονται συστήματα τέτοιου είδους είναι αυτό του MapReduce, το οποίο στηρίζεται σε αλληλουχίες δύο απλών λειτουργιών πάνω στα δεδομένα και μπορεί να εφαρμοστεί με ευκολία πάνω στις ανάγκες μια συγκεκριμένης επεξεργασίας, είτε μέσω μίας τοπικής υποδομής είτε κάνοντας χρήση υπηρεσιών απομακρυσμένων πόρων όπως γίνεται στον χώρο του cloud computing. Η ελεύθερα διατεθειμένη σαν λογισμικό ανοικτού κώδικα πλατφόρμα του Apache Hadoop είναι από τις πιο γνωστές πάνω στην υποστήριξη MapReduce εργασιών, δίνοντας με την δημοφιλία του το έναυσμα για την υλοποίηση χρήσιμων επεκτάσεων που είτε βασίζονται εξ ολοκλήρου στην δομή του είτε καινοτομούν πέρα αυτού για την βελτίωση της εκτέλεσης συγκεκριμένων εφαρμογών. Μία από τις τελευταίες αυτές επεκτάσεις είναι εκείνη του Apache Spark, όπου δίνεται προτεραιότητα στην επεξεργασία μέσα στην κύρια μνήμη έναντι του δίσκου και υποστηρίζεται μια πληθώρα χρήσιμων υλοποιήσεων στις βιβλιοθήκες του. Μία από τις εργασίες που μπορούν να βρουν εφαρμογή οι δύο εν λόγω πλατφόρμες είναι εκείνη της εξόρυξης κειμένου και πιο συγκεκριμένα της ανάλυσης συναισθημάτων κειμένου, όπου επιχειρείται να προσδιοριστεί με χρήση τεχνικών μηχανικής μάθησης το συναισθηματικό πρόσημο που χαρακτηρίζει κάθε έγγραφο μέσω ενός μοντέλου. Κύριος σκοπός της παρούσας εργασίας είναι η διερεύνηση και ανάπτυξη εφαρμογών που υλοποιούν τα μοντέλα αλγορίθμων κατηγοριοποίησης δειγμάτων (όπως είναι εκείνος των Naïve Bayes και Support Vector Machines) στις πλατφόρμες των Hadoop και Spark αλλά και η δοκιμή τροποποιημένων εκδοχών αυτών, ώστε να εξεταστούν τα αποτελέσματα τους σε πειραματικό περιβάλλον από άποψη αποτελεσματικότητας και παράλληλης εκτέλεσης και εν τέλει να αξιολογηθούν βάσει ενδεικτικών σεναρίων χρήσης.

Λέξεις-Κλειδιά

Apache Hadoop, Apache Spark, MapReduce, Εξόρυξη Κειμένου, Κατηγοριοποίηση Κειμένου, Ανάλυση Συναισθημάτων, Naïve Bayes, Support Vector Machines

Abstract

In recent years where the volume of data required to be processed in (commercial, research-based, and others) systems is increasing, there is a strong interest in terms of studying techniques in order to manage this volume efficiently and retrieve adequate results. One of the preferred techniques for such systems is the process of these large volumes of data in the scale of big data using commonly and commercially available equipment to design and implement it in a distributed environment structure. The purpose of this effort is to support the parallel execution of tasks for different chunks of data, in order to achieve an acceleration regarding the execution time, as well as better behavior in scalability depending on the size of the data given at the input, in relation to common serial programming implementations. One programming model in which such systems are based on is that of MapReduce, which consists of sequences of two simple types of functions to be used on the data and can be easily applied to the needs of a particular application, either through a local infrastructure or using remote resource services as in the cloud computing. The open source platform of Apache Hadoop is one of the best known for supporting MapReduce tasks, which thanks to its popularity gives the impetus for the implementation of useful extensions that are either entirely based on its very own structure or innovate beyond that for performance improvements of specific applications. One of these latter extensions is Apache Spark, which prioritizes processing in-memory over using disk storage and supports a plethora of useful implementations from its libraries. One of the task types that can be applied in both of the mentioned platforms is that of text classification and more specifically text sentiment analysis, where using machine learning techniques a model is created in an attempt to determine the sentiment that characterizes each text document. The main purpose of this thesis is to investigate and develop applications that implement the models of document classification algorithms (such as those of Naïve Bayes and Support Vector Machines) using the platforms of Hadoop and Spark, but also to test modified versions of them, in order to examine their results in an experimental environment in terms of efficiency and parallel execution and finally evaluate them on the basis of indicative usage scenarios.

Keywords

Apache Hadoop, Apache Spark, MapReduce, Text Mining, Text Classification, Sentiment Analysis, Naïve Bayes, Support Vector Machines

Περιεχόμενα

1. Θεωρητικό Υπόβαθρο

Με τα δεδομένα να εμφανίζονται σε όλο και πιο μεγάλες και πολύπλοκες κλίμακες, γίνεται ξεκάθαρο ότι η διαχείριση και οι επεξεργασίες πάνω σε αυτά δεν μπορούν να εφαρμοστούν στα παραδοσιακά συστήματα που υπήρχαν μέχρι πρότινος. Το φαινόμενο αυτό των **big data** χαρακτηρίζεται κυρίως από το αρκετά μεγάλο μέγεθος των δεδομένων, την πολυποικιλότητα των μορφών των δεδομένων μεταξύ τους, καθώς και την ταχύτητα με την οποία μπορούν αυτά να επεξεργαστούν σε ένα σύστημα.

Ένας από τους παράγοντες που μπορεί να θεωρηθεί ότι κατέστησε αυτό το φαινόμενο ένα σύγχρονο ζήτημα είναι τα κοινωνικά δίκτυα, αφού λόγω του ολοένα και αυξανόμενου αριθμού χρηστών μα και ρυθμού ποσού πληροφορίας (με ένα σημαντικό μέρος αυτής σε μορφή κειμένου) δεν άργησαν να προκύψουν ανάγκες για εξαγωγή πληροφορίας από μεγάλα σύνολα δεδομένων. Αυτές οι ποσότητες δεδομένων παρουσιάζουν ενδιαφέρουσες δυνατότητες αξιοποίησης και χρησιμοποιούνται στο πλαίσιο της έρευνας, της επιχειρηματικότητας και άλλων πεδίων. Μία από τις χρήσεις στις οποίες αξιοποιούνται αυτές οι ποσότητες είναι και η **ανάλυση των συναισθημάτων** ενός (συνήθως αδόμητου) σώματος κειμένου με σκοπό τον προσδιορισμό του συναισθήματος που αυτό αποτυπώνει για περαιτέρω ανάλυση μέσω μιας διαδικασίας **επεξεργασίας φυσικής γλώσσας** (natural language processing) και **μηχανικής μάθησης** (machine learning). Με αυτό τον τρόπο επιχειρείται η αξιοποίηση της συναισθηματικής πληροφορίας (δηλαδή γνώμες, αντιδράσεις, ή άλλου είδους συναισθηματικές καταστάσεις) που μπορεί να στεγάζεται στο Twitter, το Facebook, το Tumblr και άλλες πλατφόρμες κοινωνικής δικτύωσης για διάφορους σκοπούς όπως για παράδειγμα την αποτίμηση κριτικών καταναλωτών για ένα προϊόν ή την μελέτη δημοτικότητας ενός πολιτικού προσώπου σε ένα δείγμα εν δυνάμει ψηφοφόρων. Τα αποτελέσματα μιας τέτοιας ανάλυσης μπορούν να χρησιμοποιηθούν μετέπειτα σε μοντέλα για την αξιολόγηση ενδεχόμενων αποφάσεων που εξετάζονται να παρθούν ή στην περαιτέρω επεξεργασία για λογαριασμό μιας ευρύτερης μελέτης.

Παρόλα αυτά οι προκλήσεις των big data επηρεάζουν την ανάλυση συναισθημάτων υποχρεώνοντας κάθε εφαρμογή της τελευταίας να σχεδιάζεται βάσει των τριών συνιστωσών που χαρακτηρίζουν το φαινόμενο αυτό (όπως αναφέρθηκαν και στην αρχή). Αυτό το γεγονός έδωσε και την κινητήρια ώθηση για την χρήση παράλληλων προγραμματιστικών μοντέλων που μπορούν να εργάζονται στην κλίμακα των big data και να μεγιστοποιούν την αποδοτικότητα της εξαγωγής πληροφορίας, όπως το **MapReduce**. Το προγραμματιστικό παράδειγμα του MapReduce, επηρεασμένο από τον συναρτησιακό προγραμματισμό, βασίζεται στις δύο στοιχειώδεις μεθόδους Map και Reduce (τις μόνες που χρειάζεται να υλοποιήσει ο χρήστης για λογαριασμό μιας εφαρμογής) για την διαχείριση μικρού ή μεγάλου μεγέθους δεδομένων σε μορφή ζευγαριών κλειδιού τιμής με παράλληλο τρόπο, μέσω κόμβων που εκτελούν στιγμιότυπα των μεθόδων αυτών πάνω σε διαφορετικά δεδομένα κάθε φορά.

Η πιο γνωστή πλατφόρμα του μοντέλου MapReduce, συνοδευόμενη από ένα ολοκληρωμένο περιβάλλον κατανεμημένου χώρου αποθήκευσης και εκτέλεσης ενός παραμετροποιημένου προγράμματος από τον χρήστη, είναι το **Apache Hadoop**. Το Hadoop χαρακτηρίζεται από την αξιοπιστία και την κλιμακωσιμότητα (scalability) που προσφέρει έχοντας την ευχέρεια για δυναμική αντιμετώπιση μιας "έκρηξης" φόρτου, λειτουργώντας χωρίς πρόβλημα σε τοπολογίες μερικών έως χιλιάδων κόμβων. Επιπλέον υποστηρίζει ένα οικοσύστημα εργαλείων που λειτουργούν σαν επεκτάσεις για πιο εξειδικευμένες ανάγκες επεξεργασίας. Η γοητεία του έγκειται στον σχεδιασμό του που του επιτρέπει να εκτελεί κατανεμημένους παράλληλους υπολογισμούς πάνω σε συμβατικό εξοπλισμό, μειώνοντας έτσι σημαντικά το κόστος υποδομής και απόκτησης τεχνογνωσίας για τον χειρισμό ενός πιο εξειδικευμένου framework.

Μια πιο πρόσφατη υλοποίηση που ακροβατεί μεταξύ των χαρακτηριστικών που περιγράφουν το MapReduce και μιας πιο σύγχρονης αντίληψης για την επεξεργασία μεγάλων όγκων δεδομένων είναι αυτή του **Apache Spark**. Η διαφοροποίηση του από το Hadoop βασίζεται (μαζί με άλλες βελτιστοποιήσεις) στην επεξεργασία των δεδομένων στην άμεση μνήμη έναντι του σκληρού δίσκου, κάνοντας το Spark πολλαπλάσια ταχύτερο στην εκτέλεση μιας εφαρμογής. Αυτή η προσέγγιση σε συνδυασμό με τις πολύτιμες βιβλιοθήκες που διαθέτει εντός (π.χ. υποστήριξη εφαρμογών μηχανικής μάθησης, ευελιξία θεώρησης των δεδομένων σαν στοιχεία πινάκων βάσεων δεδομένων κ.α.) λειτουργεί σαν μεγάλο δέλεαρ για την χρήση της εν λόγω πλατφόρμας, με αποτέλεσμα να κατέχει σημαντική θέση στην επεξεργασία μεγάλων συλλογών δεδομένων σήμερα.

1.1 Εξόρυξη Δεδομένων

Με την ύπαρξη μεγάλων δεξαμενών πληροφοριών που γίνονται ολοένα και πιο αναγκαίες για αποθήκευση και συντήρηση δεδομένων από οργανισμούς και υπηρεσίες, είναι επιτακτική η ανάγκη για την ανακάλυψη χρήσιμης πληροφορίας μέσω αυτών. Αυτό τον ρόλο έχει αναλάβει η εξόρυξη δεδομένων, που παρέχει βαθιά διερεύνηση πληροφορίας πάνω σε μεγάλες συλλογές ή βάσεις δεδομένων καθώς και προσεγγιστικές προβλέψεις βάσει παρατηρήσεων πάνω στα δεδομένα. Η ανακάλυψη της χρήσιμης γνώσης γίνεται με χρήση συγκεκριμένων αλγορίθμων (π.χ. για συσταδοποίηση, κατηγοριοποίηση κλπ.) και της μηχανικής μάθησης, και ως αποτέλεσμα εξάγεται η πληροφορία σε μορφή κατανοητή από ανθρώπους, οι οποίοι καλούνται να την αξιοποιήσουν.

Η ποικιλία μορφών των δεδομένων που δίνονται ως είσοδος στην διαδικασία της εξόρυξης δεδομένων εξαναγκάζει την τελευταία να παρέχει μία φάση προεπεξεργασίας η οποία θα φέρει τα δεδομένα αυτά στην κατάλληλη δομή και μορφή για την επικείμενη μελέτη τους, κάτι που μπορεί να αποβεί χρονοβόρο για την συνολική διαδικασία μα είναι πέρα για πέρα αναγκαίο. Μετά την προεπεξεργασία καλό είναι να έχουν εξαλειφθεί στο μέγιστο βαθμό φαινόμενα διπλότυπων δεδομένων ή θορύβου και να έχουν επιλεγεί τα κατάλληλα χαρακτηριστικά για τους σκοπούς της εργασίας μας.

Έχοντας πάντα υπόψη την εφαρμογή της εξόρυξης δεδομένων στο πεδίο των big data (μιας και οι παραδοσιακές εφαρμογές επεξεργασίας δεδομένων δεν μοιάζουν τόσο ελκυστικές ή αποδοτικές λόγω πολυπλοκότητας ή υπολογιστικού μεγέθους), χρειάζεται μέριμνα για αρκετά σημεία που κάνουν ξεκάθαρους τους λόγους χρήσης της ανακάλυψης πληροφορίας με παράλληλο τρόπο. Τέτοια σημεία είναι η κλιμακωσιμότητα των αλγορίθμων εξόρυξης, η διαχείριση δεδομένων που αποτελούνται από πολλές (εκατοντάδες ή ακόμα και χιλιάδες) διαστάσεις (ή αλλιώς **χαρακτηριστικά**), και η διανομή των δεδομένων αν αυτά αποθηκεύονται κατανεμημένα (βάσει χώρου ή είδους πληροφορίας) έτσι ώστε ένας παράλληλος αλγόριθμος εξόρυξης να λειτουργεί με αποδοτικό και ασφαλή τρόπο πάνω στα στοιχεία εισόδου.

1.1.1 Εξόρυξη Κειμένου

Με το κείμενο να διατηρεί ισχυρή θέση σαν μέσο επικοινωνίας και ανταλλαγής πληροφοριών, είναι ξεκάθαρο πόσο πολύτιμη είναι η επεξεργασία κειμένων για την ανακάλυψη χρήσιμης γνώσης. Κοιτώντας πέρα από τα σύνολα δομημένων εγγράφων που υπάρχουν και μπορούν να επεξεργαστούν στις κοινές βάσεις δεδομένων, εκτιμάται ότι περίπου το 80% των δεδομένων που αφορούν εταιρείες και οργανισμούς βρίσκεται σε μη δομημένη μορφή (όπως τα μηνύματα ηλεκτρονικού ταχυδρομείου ή τα απλά αρχεία κειμένου). Αυτή η κατάσταση καλείται να αντιμετωπιστεί μέσω της εξόρυξης κειμένου,

ενός πεδίου που συνδυάζει την εξόρυξη δεδομένων με την ανάκτηση πληροφορίας και την επεξεργασία φυσικής γλώσσας. Μερικοί από τους τομείς που βρίσκει εφαρμογή αυτό το πεδίο είναι οι τηλεπικοινωνίες, τα μέσα ενημέρωσης, οι χρηματοπιστωτικές αγορές, το σύστημα υγείας, και η εθνική ασφάλεια.

Η εξόρυξη κειμένου μπορεί να εκφραστεί μέσα από δύο συγκεκριμένες φάσεις, αυτή της εκκαθάρισης κειμένου και εκείνη της εξαγωγής γνώσης. Στην πρώτη φάση τα αρχεία κειμένου μετατρέπονται σε μια ειδική ενδιάμεση (ημιδομημένη) μορφή την οποία χρησιμοποιεί η δεύτερη φάση για καταλήξει στην ανακάλυψη χρήσιμης γνώσης. Αυτή η ενδιάμεση μορφή μπορεί να είναι βασισμένη είτε σε έγγραφα (όπου εξάγονται οι σχέσεις μεταξύ ενός συνόλου εγγράφων π.χ. για κατηγοριοποίηση ή συσταδοποίηση των δεδομένων) είτε σε έννοιες (όπου εξάγονται οι σχέσεις μεταξύ διαφορετικών αντικειμένων ή εννοιών π.χ. για μοντελοποίηση προβλέψεων ή αναζήτηση συσχετίσεων στα δεδομένα). Η διαδικασία εξόρυξης κειμένου μπορεί να εκφραστεί πιο λεπτομερώς μέσα από έναν αριθμό βημάτων που εκτελούνται διαδοχικά μέχρι να φτάσουμε στα αποτελέσματα της, τα οποία παρουσιάζονται παρακάτω.

Βήμα Εξόρυξης Κειμένου	Περιγραφή
1. Προεπεξεργασία Κειμένου	Μετατροπή ενός εγγράφου σε ένα σύνολο χαρακτηριστικών που αναπαριστούν στοιχεία κειμένου όπως μία λέξη, φιλτράροντας τα από τα στοιχεία που κρίνονται ακατάλληλα.
2. Μετατροπή Κειμένου	Κάθε έγγραφο αναπαριστάται από τα χαρακτηριστικά τα οποία περιέχει καθώς και τον αριθμό εμφάνισης κάθε ενός χαρακτηριστικού.
3. Επιλογή Χαρακτηριστικών	Επιλογή των πιο σχετικών χαρακτηριστικών από την συλλογή για την δημιουργία του μοντέλου στο επόμενο βήμα χωρίς τα περιττά ή τετριμμένα χαρακτηριστικά να επηρεάζουν την διαδικασία.
4. Εξόρυξη Κειμένου	Δημιουργία και εφαρμογή του μοντέλου εξόρυξης γνώσης που επιλέχθηκε (π.χ. στο πλαίσιο της παρούσας εργασίας η κατηγοριοποίηση κειμένου).
5. Αξιολόγηση Αποτελεσμάτων	Ανάλυση επίδοσης και αποτελεσματικότητας της εξόρυξης μέσω των αποτελεσμάτων που λαμβάνει ο τελικός χρήστης με το πέρας της διαδικασίας.

Πίνακας 1.1 - Βήματα Εξόρυξης Κειμένου

Θα πρέπει βέβαια να υπάρχει προετοιμασία για την αντιμετώπιση ζητημάτων που ενδέχεται να προκύψουν όπως την δομή και τις ιδιοτροπίες της γλώσσας που είναι γραμμένο το κείμενο υπό μελέτη, αφού τα φαινόμενα ασάφειας (όπως για παράδειγμα ο σαρκασμός, οι λέξεις με πολλαπλές σημασίες, ή τα κοινωνικοπολιτικά συμφραζόμενα) κάνουν την διαδικασία εξόρυξης εξαιρετικά πολύπλοκη. Άλλα ζητήματα μπορούν να είναι η διαχείριση πολύγλωσσων κειμένων και η ανάγκη διαφορετικής ενδιάμεσης μορφής κειμένων για διαφορετική μέθοδο εξόρυξης κειμένου.

1.1.2 Κατηγοριοποίηση

Είναι χρήσιμο σε πολλές περιπτώσεις να μπορούν να διαχωρίζονται τα συστατικά ενός συνόλου σε μια σειρά **κατηγοριών** (ή αλλιώς κλάσεων), όπως για παράδειγμα με την διαδικασία κατηγοριοποίησης των ρούχων ανά τόνο χρώματος πριν αυτά πλυθούν στο πλυντήριο. Όταν έχουμε να κάνουμε με μεγάλο και σύνθετο όγκο δεδομένων με πολλά χαρακτηριστικά είναι θεμιτή και αναγκαία η αυτοματοποίηση μιας τέτοιας διαδικασίας. Αυτό το ρόλο παίζει η κατηγοριοποίηση (ή ταξινόμηση, όπως θα αναφέρεται εδώ συνώνυμα) δεδομένων η οποία εφαρμόζεται σε ένα σύνολο στιγμιοτύπων που εκφράζονται από τα χαρακτηριστικά τους και την ετικέτα κλάσης (δηλαδή ένα κατηγορηματικό χαρακτηριστικό) που ανήκει. Ένα αντιπροσωπευτικό παράδειγμα για την κατηγοριοποίηση θα μπορούσε να είναι αυτό της ταξινόμησης ενός συνόλου δανειοληπτών (βάσει χαρακτηριστικών όπως το ετήσιο εισόδημα, η ηλικία, η οικογενειακή κατάσταση, και άλλα) για να προβλεφθεί ποιοι από αυτούς δύνανται να αποπληρώσουν τις δόσεις στο ενδεχόμενο χορήγησης ενός δανείου (με ετικέτες κατηγορίας εδώ να είναι φυσικά δύο, που υποδηλώνουν την δυνατότητα ή μη αποπληρωμής των δόσεων).

Σχήμα 1.1 - Σχηματική Απεικόνιση Κατηγοριοποίησης

Η ταξινόμηση των στιγμιότυπων μιας συλλογής σε κλάσεις γίνεται με ένα μοντέλο κατηγοριοποίησης που αναπαριστά την σχέση των χαρακτηριστικών με την ετικέτα κατηγορίας. Το μοντέλο αυτό μπορεί να χρησιμοποιηθεί είτε ως προβλεπτικό μοντέλο για την κατηγοριοποίηση στιγμιοτύπων χωρίς ετικέτα κατηγορίας (όπως στο παράδειγμα των δανειοληπτών) είτε ως περιγραφικό μοντέλο για την διάκριση των χαρακτηριστικών που ξεχωρίζουν τα στιγμιότυπα ανά κλάση (όπου χρησιμοποιείται κυρίως για την πλήρη αιτιολόγηση και διαφάνεια των προβλέψεων ενός μοντέλου πάνω σε σημαντικές εφαρμογές που έχουν μικρά περιθώρια λάθους).

Η διαδικασία της κατηγοριοποίησης περιέχει επί της ουσίας δύο βήματα. Στο πρώτο βήμα σχηματίζεται ένα μοντέλο μέσω ενός συνόλου δειγμάτων με ενδεικτικά χαρακτηριστικά και ετικέτες κατηγοριών ώστε να χρησιμοποιηθούν σαν **στιγμιότυπα εκπαίδευσης** στον αλγόριθμο που θα εφαρμοστεί. Στο δεύτερο βήμα, το κατασκευασμένο μοντέλο εφαρμόζεται σε ένα άλλο σύνολο **στιγμιοτύπων ελέγχου** (που μπορεί να φέρουν ή όχι ετικέτα κατηγορίας) για την πρόβλεψη της κατηγορίας στην οποία ανήκει κάθε ένα από αυτά. Τα σύνολα εκπαίδευσης και ελέγχου φυσικά είναι ανεξάρτητα μεταξύ τους για να είναι οι προβλέψεις του μοντέλου αξιόπιστες, μα είναι σημαντικό να μελετάται η επίδοση της ταξινόμησης βάσει και των δύο αυτών συνόλων για να υπάρχει πλήρης εικόνα σχετικά με την αποτελεσματικότητα ολόκληρης της διαδικασίας. Η προσέγγιση αυτή εντάσσεται στο πεδίο της **μηχανικής μάθησης** (machine learning) **με επιβλεπόμενη μάθηση** (supervised learning) αφού υπάρχει σαφής καθορισμός της κλάσης κάθε στιγμιότυπου εκπαίδευσης ώστε να κατευθυνθεί το μοντέλο αναλόγως κατά τη κατασκευή.

Κάτι τέτοιο μπορεί να γίνει κατά κύριο λόγο έχοντας ένα σύνολο ελέγχου με στιγμιότυπα που φέρουν ήδη ετικέτα κατηγορίας, ούτως ώστε να συγκρίνουμε αυτές τις ετικέτες με τις προβλέψεις του

μοντέλου. Τα αποτελέσματα αυτής της σύγκρισης παρουσιάζονται σε έναν πίνακα που ονομάζεται **μήτρα σύγχυσης** (confusion matrix) με κάθε στοιχείο f_{ij} να υποδηλώνει το πλήθος των στιγμιότυπων που ανήκουν στην κλάση *i* και προβλέφθηκαν στην κλάση *j*.

		Προβλεφθείσα Κατηγορία	
		+	-
Πραγματική Κατηγορία	+	f_{++} (True Positive)	f_{+-} (False Negative)
	-	f_{-+} (False Positive)	f_{--} (True Negative)

Πίνακας 1.2 - Μήτρα Σύγχυσης Δυαδικής Κατηγοριοποίησης Θετικού-Αρνητικού

Έπειτα συνοψίζοντας τα δεδομένα της μήτρας σύγχυσης σε κάποιες μετρικές όπως για παράδειγμα την ορθότητα (δηλαδή το πλήθος των σωστών προβλέψεων προς το σύνολο των προβλέψεων) που εκφράζεται σαν ένα ποσοστό επί των δειγμάτων, υπάρχει η δυνατότητα απευθείας σύγκρισης στην επίδοση ενός ή περισσοτέρων μοντέλων παρατηρώντας απλώς αυτές τις ποσότητες.

Παρόλα αυτά η αποτελεσματικότητα της κατηγοριοποίησης δεν είναι αποκλειστική και αντίστοιχη μόνο της εκπαίδευσης του μοντέλου αλλά και στο ίδιο το σύνολο των στιγμιοτύπων. Δεν είναι διόλου σπάνια η ύπαρξη θορύβου στην μορφή ενός στιγμιότυπου που έχει ταξινομηθεί σε μία κλάση βάσει των χαρακτηριστικών του, τα οποία τυχαίνει να μην εμφανίζονται σε κανένα άλλο στιγμιότυπο άλλης κλάσης στο σύνολο εκπαίδευσης. Αυτό έχει σαν αποτέλεσμα την λανθασμένη κατηγοριοποίηση ενός στιγμιότυπου στο σύνολο ελέγχου σε μια συγκεκριμένη κλάση απλά και μόνο επειδή περιέχει χαρακτηριστικά ενός δείγματος-θορύβου από το σύνολο εκπαίδευσης. Αυτή η λανθασμένη γενίκευση βασισμένη στα θορυβώδη χαρακτηριστικά των δεδομένων εκπαίδευσης ονομάζεται **υπερπροσαρμογή** (overfitting), και συνεπάγεται ουσιαστικά την απομνημόνευση των δεδομένων εκπαίδευσης μαζί με τον θόρυβο τους από το μοντέλο με αποτέλεσμα την αύξηση του σφάλματος ταξινόμησης. Το πιο κοινό αίτιο της υπερπροσαρμογής κατά την κατηγοριοποίηση είναι το μικρό μέγεθος του συνόλου εκπαίδευσης (όταν δηλαδή υπάρχουν λίγα αντιπροσωπευτικά στιγμιότυπα για κάθε κλάση ή πολύς θόρυβος, ώστε ο τελευταίος να έχει πολύ μεγάλη πιθανότητα να γίνει η βάση των προβλέψεων του μοντέλου).

Έχοντας καλύψει τα κύρια σημεία της κατηγοριοποίησης, η μελέτη μπορεί να εστιαστεί στην εφαρμογή της πάνω σε δεδομένα κειμένου "μεταφράζοντας" τις ετικέτες κατηγοριών σε θεματικές περιοχές που μπορεί να αφορούν τα στιγμιότυπα κειμένου (π.χ. μια εφαρμογή κατηγοριοποίησης κειμένου σε ένα σύνολο ειδησεογραφικών άρθρων θα ξεχώριζε κάθε έγγραφο ανάλογα με την θεματολογία του στην κλάση των τοπικών νέων, των αθλητικών, των οικονομικών, και άλλων), αφού πρώτα περιγράψουμε παρακάτω δύο ενδεικτικές μεθόδους κατηγοριοποίησης που μπορούν να χρησιμοποιηθούν στο πλαίσιο της ταξινόμησης κειμένου.

1.1.2.1 Naïve Bayes

Για την κατηγοριοποίηση ενός εγγράφου κειμένου με χρήση της πιθανοτικής μεθόδου της πολυωνυμικής Naïve Bayes, ένα έγγραφο *x* ταξινομείται στην κλάση *y* μέσω της ακόλουθης σχέσης.

$$P(y|x) \propto P(y) \cdot \prod_{i=1}^{n_x} P(x_i|y)$$

Σχέση 1.1 - Σχέση Πιθανότητας Ταξινόμησης Εγγράφου *x* στην Κλάση *y* με Naïve Bayes

Όπου:

- Η πιθανότητα του αριστερού μέλους είναι ανάλογη με το αποτέλεσμα του δεξιού μέλους.
- Ως $P(y|x)$ ορίζεται η **υπό συνθήκη πιθανότητα** ώστε **το έγγραφο *x* να ανήκει στην κλάση *y***.
- Ως P(y) ορίζεται η **εκ των προτέρων πιθανότητα το έγγραφο να ανήκει στην κλάση *y*** (σχετίζεται ευθέως με το σύνολο εκπαίδευσης, αφού ισούται με τον αριθμό των στιγμιοτύπων εκπαίδευσης που ανήκουν στην εκάστοτε κλάση προς το σύνολο των στιγμιοτύπων).
- Ως $P(x_i|y)$ ορίζεται η **υπό συνθήκη πιθανότητα το χαρακτηριστικό x_i να ανήκει σε έγγραφο της κλάσης *y***, όπου υπολογίζεται ως το πλήθος των εμφανίσεων του χαρακτηριστικού στα εγγραφα κλάσης *y* προς το γινόμενο του πλήθους των όρων σε στιγμιότυπα κλάσης *y* με το πλήθος των χαρακτηριστικών στο λεξιλόγιο.
- Ως n_x ορίζεται το πλήθος των όρων του εγγράφου *x* που ανήκουν στο λεξιλόγιο χαρακτηριστικών του μοντέλου.

Η συγκεκριμένη σχέση υπολογίζεται για λογαριασμό κάθε εγγράφου προς έλεγχο και για κάθε ετικέτα κατηγορίας που έχει εκπαιδευτεί το μοντέλο, ταξινομώντας το εκάστοτε έγγραφο στην ετικέτα με τη μέγιστη ύστερη πιθανότητα (maximum a posteriori - map), όπως φαίνεται και παρακάτω.

$$y_{map} = argmax(P(y|x)) = argmax(P(y) \cdot \prod_{i=1}^{n_x} P(x_i|y))$$

Σχέση 1.2 - Σχέση Βέλτιστης Πιθανότητας Ταξινόμησης Εγγράφου *x* στην Κλάση *y*

Αυτή η διαφάνεια του τρόπου κατηγοριοποίησης είναι από τους βασικούς λόγους που αυτή η μέθοδος βρίσκει συχνή εφαρμογή σε προβλεπτικά και περιγραφικά μοντέλα. Ωστόσο με λίγη προσοχή γίνεται εμφανές το ζήτημα της αντιμετώπισης μηδενικών τιμών πιθανοτήτων υπό συνθήκη $P(x_i|y)$ για χαρακτηριστικά που εμφανίζονται μόνο σε κάποιες και όχι σε όλες τις ετικέτες κατηγορίας του συνόλου εκπαίδευσης. Η περίπτωση της μηδενικής πιθανότητας είναι μοιραίο αποτέλεσμα του γεγονότος πως το σύνολο εκπαίδευσης δεν μπορεί να είναι ποτέ αρκετά πλήρες για τις πιο σπάνιες περιπτώσεις χαρακτηριστικών κάθε ετικέτας. Για την επίλυση αυτού του ζητήματος προστίθεται μια μονάδα στην σχέση πιθανότητας υπό συνθήκη για κάθε χαρακτηριστικό. Αυτή η τεχνική ονομάζεται **εξομάλυνση Laplace** (Laplace smoothing) και εξασφαλίζει ότι καμία πιθανότητα υπό συνθήκη δεν θα ισούται με μηδέν επηρεάζοντας έτσι το γινόμενο της πιθανότητας ενός εγγράφου να ανήκει σε μία κλάση (όπως θα περιγραφεί με μεγαλύτερη λεπτομέρεια και στη συνέχεια κατά την περιγραφή ανάπτυξης των υλοποιήσεων).

Στα της χρονικής πολυπλοκότητας της μεθόδου Naïve Bayes, χρειάζεται να γίνει αρχικά η διάκριση σε αλγόριθμο εκπαίδευσης και ελέγχου, μιας και αναφέρονται οι αντίστοιχες φάσεις της μεθόδου στα δεδομένα εισόδου τους με διαφορετικό τρόπο. Για την εκπαίδευση, η πολυωνυμική Naïve Bayes επωμίζεται το χρέος του υπολογισμού των παραμέτρων για το μοντέλο που αποτελούνται από τις πρότερες ($\Theta(|C|)$, όπου $|C|$ πλήθος των κλάσεων) και τις υπό συνθήκη ($\Theta(|CV|)$, όπου $|V|$ το πλήθος του λεξιλογίου όρων) πιθανότητες, κάνοντας τουλάχιστον μία σάρωση του συνόλου εκπαίδευσης ($\Theta(|D|L_{ave})$, όπου $|D|$ το πλήθος των στιγμιοτύπων εκπαίδευσης και L_{ave} το εκτιμώμενο μέσο μήκος

κάθε στιγμιοτύπου), οπότε η χρονική πολυπλοκότητα θα είναι το άθροισμα αυτών των συντελεστών. Το σκέλος του ελέγχου με τη σειρά του επηρεάζεται από το πλήθος των κλάσεων του μοντέλου (*Θ(|C|)*) και τα πλήθος των στιγμιοτύπων (*Θ(|S|)* στο σύνολο ελέγχου, άρα η χρονική πολυπλοκότητα θα είναι και εδώ το γινόμενο αυτών των συντελεστών.

Λειτουργία	Χρονική Πολυπλοκότητα
Εκπαίδευση	$\Theta(\|D\|L_{ave} + \Theta(\|C\|\|V\|)$
Έλεγχος	$\Theta(\|C\|\|S\|)$

Πίνακας 1.3 - Χρονική Πολυπλοκότητα Λειτουργιών Naïve Bayes

Συμπεραίνουμε από τα παραπάνω ότι ο Naïve Bayes αλγόριθμος έχει γραμμική πολυπλοκότητα για την σάρωση των δεδομένων και στις δύο λειτουργίες του, κάτι ιδιαίτερα ικανοποιητικό για τον υπολογιστικό φόρτο ενός τέτοιου μοντέλου. Αυτός είναι και ένας από τους κύριους λόγους για την ευρεία χρήση του στην ταξινόμηση κειμένου από πλευρά χρόνου εκτέλεσης.

Για την εξέταση της εφαρμογής του Naïve Bayes σε ένα παράλληλο υπολογιστικό πρότυπο δεν χρειάζεται κάποια ιδιαίτερα πολύπλοκη διαδικασία αφού πρόκειται για μία πιθανοτική μέθοδο. Ο πιο απαιτητικός υπολογισμός για την κατασκευή του μοντέλου είναι αυτός των υπό συνθήκη πιθανοτήτων (χαρακτηριστικών και κλάσεων) επειδή εμπεριέχει την ανάγκη ανάκτησης του πλήθους εγγράφων για κάθε κλάση, το αναγνωριστικό ενός χαρακτηριστικού, και το πλήθος εμφάνισης κάθε χαρακτηριστικού για κάθε έγγραφο. Επιπλέον, γνωρίζοντας ότι όλες οι υπό συνθήκη πιθανότητες είναι ανεξάρτητες μεταξύ τους, δεν υπάρχει κανένα ζήτημα κατά την παραλληλοποίηση της διαδικασίας δίνοντας διαφορετικά υποσύνολα εγγράφων εκπαίδευσης σε κάθε υπολογιστικό κόμβο ή διεργασία. Έτσι κατά την εκπαίδευση ανακτώνται και επεξεργάζονται οι πληροφορίες κάθε εγγράφου για τον υπολογισμό των αναγκαίων πιθανοτήτων του μοντέλου, ενώ παράλληλα μπορεί να λειτουργήσει και η διαδικασία ελέγχου με κάθε διεργασία να δέχεται ως είσοδο διαφορετικά στιγμιότυπα ελέγχου, και στη συνεχεια να υπολογίζονται όλες οι πιθανότητες κατηγοριοποίησης για το σύνολο των κλάσεων και να επιλέγεται η μέγιστη αυτών.

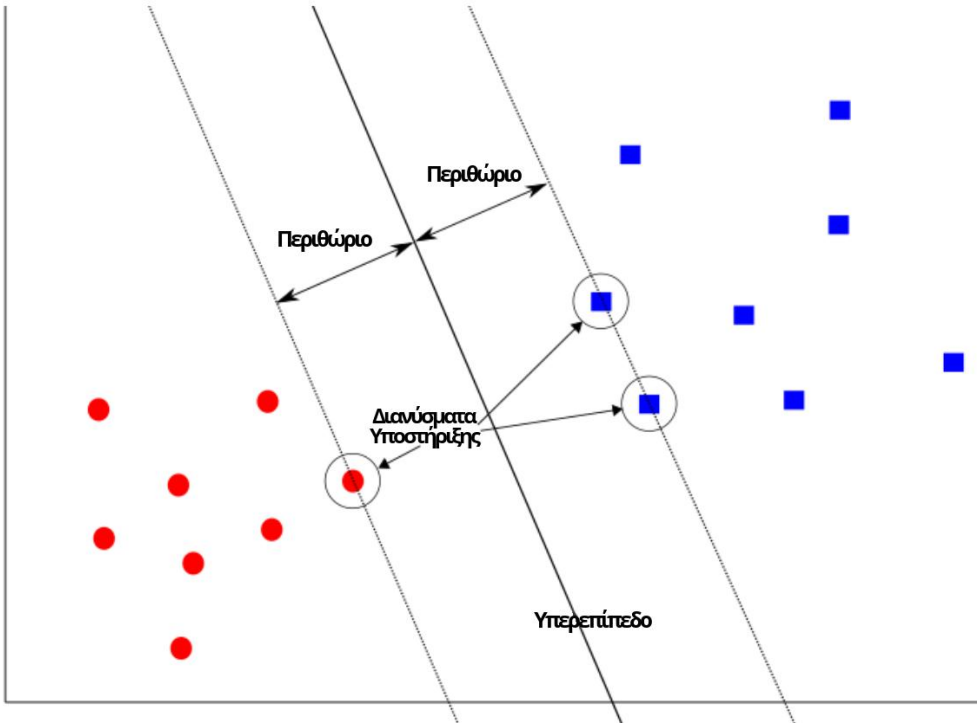
1.1.2.2 Support Vector Machines

Ένας άλλος τρόπος κατηγοριοποίησης είναι εκείνος που διαχωρίζει τα χαρακτηριστικά στον διανυσματικό χώρο κατά τέτοιο τρόπο ώστε να ταξινομούνται σε κατηγορίες. Κάτι τέτοιο αφορά και τις Μηχανές Διανυσμάτων Υποστήριξης (Support Vector Machines) που έχουν τη δυνατότητα να προσαρμόζουν την εκπαίδευση χωρίς φαινόμενα υπερπροσαρμογής (κάτι που τις κάνει ιδιαίτερα ελκυστικές για κατηγοριοποίηση κειμένου). Αυτή η μέθοδος αναπαριστά τα όρια απόφασης που υπολογίζει με χρήση μόνο ενός μέρους των στιγμιοτύπων εκπαίδευσης (γνωστά και ως **διανύσματα υποστήριξης**) που χρήζουν περισσότερης διερεύνησης για να ταξινομηθούν σωστά, άρα το μοντέλο βασίζεται μόνο στα στιγμιότυπα εκπαίδευσης που βρίσκονται κοντά στα όρια των κατηγοριών. Το διαχωριστικό υπερεπίπεδο εκφράζεται από την παρακάτω σχέση, όπου *w* (ως ανεστραμμένο διάνυσμα) και *b* είναι οι παράμετροι του υπερεπιπέδου και *x* είναι το σύνολο των χαρακτηριστικών.

$$w^T \cdot x + b = 0$$

Σχέση 1.3 - Γενική Εξίσωση Διαχωριστικού Υπερεπιπέδου SVM

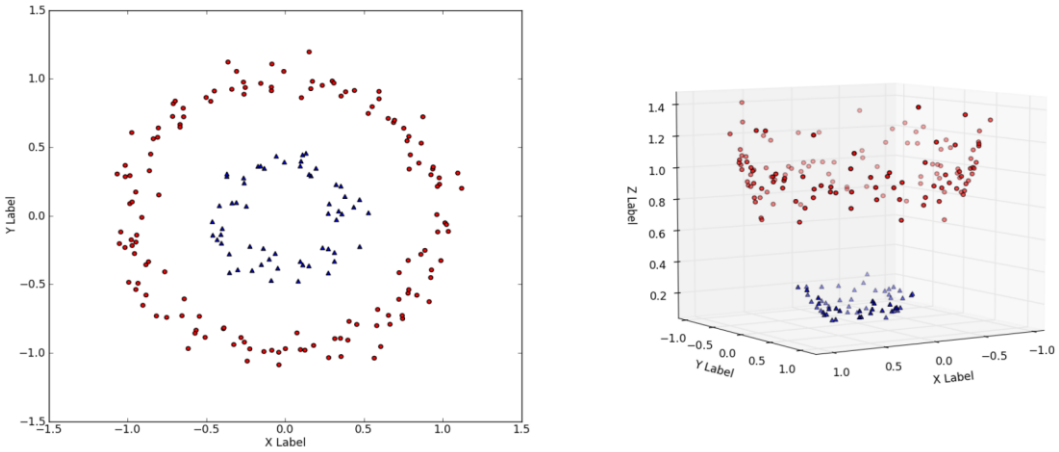
Κάθε στιγμιότυπο εκπαίδευσης x_i βρίσκεται σε μία από τις δύο πλευρές του υπερεπιπέδου, ανάλογα με το πρόσημο (για τιμές *-1* ή *+1*) της παραπάνω σχέσης, άρα με αυτή τη μέθοδο τα δεδομένα μπορούν να διαχωριστούν μόνο σε δύο ετικέτες κατηγορίας (αν και υπάρχουν προτάσεις που λειτουργούν και για κατηγοριοποίηση πολλών κατηγοριών, όπως η επεξεργασία κατηγοριών ανά ζεύγη με χρήση μιας αλληλουχίας SVM μοντέλων). Ο τελικός στόχος του μοντέλου αυτής της μορφής είναι να βρεθεί το υπερεπίπεδο που διαχωρίζει καλύτερα τα στιγμιότυπα. Εάν τουλάχιστον ένα τέτοιο υπερεπίπεδο υφίσταται τότε τα δεδομένα είναι **γραμμικά διαχωρίσιμα**.



Σχήμα 1.2 - Διανύσματα Υποστήριξης, Περιθώρια, και Υπερεπίπεδο SVM

Η επιλογή του βέλτιστου από αυτά τα υπερεπίπεδα γινεται με χρήση των **περιθωρίων** κάθε διαχωριστικού, που δεν είναι κάτι άλλο από την απόσταση μεταξύ ενός ζεύγους παράλληλων υπερεπιπέδων που ακουμπούν στα κοντινότερα στιγμιότυπα κάθε κατηγορίας. Αν επιλεγεί ένα διαχωριστικό με όσο το δυνατόν πιο μεγάλο περιθώριο, τότε έχουμε να κάνουμε με ένα **υπερεπίπεδο μέγιστου περιθωρίου** (και κατ' επέκταση με έναν **κατηγοριοποιητή μέγιστου περιθωρίου**, maximum margin classifier), που αποδίδει καλύτερα σε σχέση με τα αντίστοιχα μικρών περιθωρίων, αφού τα τελευταία είναι εύκολη λεία στην υπερπροσαρμογή του μοντέλου. Σε περίπτωση που ένας μικρός αριθμός σφαλμάτων εκπαίδευσης είναι ανεκτός για το μοντέλο υπό σχεδίαση, μπορούν να υποστηριχθούν και κλάσεις που δεν είναι γραμμικά διαχωρίσιμες. Με αυτά τα χαρακτηριστικά ορίζεται ένας **κατηγοριοποιητής απαλού περιθωρίου** (soft margin classifier).

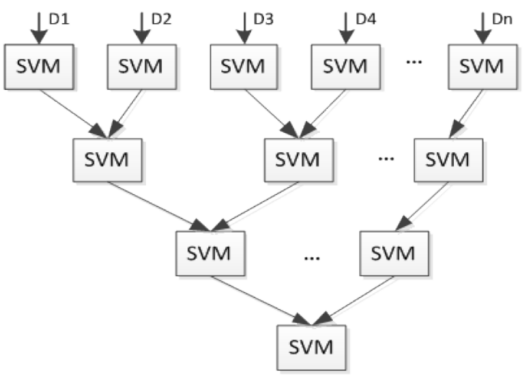
Εάν από την άλλη τα δεδομένα του μοντέλου της μηχανής διανυσμάτων υποστήριξης έχουν μη γραμμικά όρια απόφασης στον χώρο των χαρακτηριστικών, επιστρατεύονται οι **μη γραμμικές μηχανές διανυσμάτων υποστήριξης**. Με αυτές τα δεδομένα μετασχηματίζονται σε έναν νέο χώρο συντεταγμένων υψηλότερης διάστασης όπου εκεί είναι δυνατή η χρήση ενός γραμμικού ορίου απόφασης για την ταξινόμηση των στιγμιότυπων. Ύστερα από αυτή τη μετατροπή τα δεδομένα μπορούν να οπτικοποιηθούν στην πρότερη μορφή τους δημιουργώντας έτσι ένα μη γραμμικό όριο απόφασης.



Σχήμα 1.3 - Μετασχηματισμός Δεδομένων από 2 σε 3 Διαστάσεις για Πλήρη Διαχωρισμό

Από άποψη επίδοσης οι μηχανές διανυσμάτων υποστήριξης μπορούν να προσαρμόζουν τις παραμέτρους του μοντέλου αλλάζοντας το περιθώριο του διαχωριστικού και να είναι λιγότερο ευαίσθητες στο θόρυβο και τα διπλότυπα του συνόλου εκπαίδευσης επιλέγοντας μόνο μια μερίδα στιγμιοτύπων. Όμως ο χρόνος εκπαίδευσης ενός τέτοιου μοντέλου ενδέχεται να είναι μεγάλος λόγω της ιδιαίτερα πολύπλοκης διαδικασίας υπολογισμών για την επιλογή του βέλτιστου υπερεπιπέδου. Γενικότερα οι μηχανές διανυσμάτων υποστήριξης περιέχουν εν γένει το ζήτημα του **τετραγωνικού προγραμματισμού** (quadratic programming) για να διαχωριστούν τα διανύσματα υποστήριξης από τα λοιπά δεδομένα εκπαίδευσης, οπότε υπάρχει η ανάγκη βελτιστοποίησης μιας δευτεροβάθμιας συνάρτησης που προκύπτει βάσει της μεθόδου με γραμμικούς περιορισμούς. Λόγω της πολυσύνθετης φύσης του τετραγωνικού προγραμματισμού, διατίθενται μεν τεχνικές για μια πιο κλιμακώσιμη και ταχεία κατασκευή μοντέλου με πιο περίπλοκες βιβλιοθήκες, μα μια πιο προσγειωμένη προσέγγιση θα ήταν αυτή του **παραλληλοποίησης του αλγορίθμου.**

Η διαδικασία παράλληλης κατασκευής ενός μοντέλου SVM περιέχει την διαίρεση του συνόλου εκπαίδευσης σε υποενότητες, με κάθε μία από αυτές να εκπαιδεύεται σε ένα ξεχωριστό μοντέλο SVM, έως ότου να συγκεντρωθούν τα διανύσματα υποστήριξης κάθε τέτοιου μοντέλου και να δοθούν ως είσοδος στον επόμενο κύκλο αυτής της διαδικασίας. Συμπεραίνουμε λοιπόν ότι ο υπολογισμός του καλύτερου διαχωριστικού για την εφαρμογή που σχεδιάζεται το εν λόγω μοντέλο θα προέρχεται μέσα από μια αλληλουχία επαναλήψεων και φιλτράρισμα των αναγκαίων στιγμιοτύπων για αυτή την διεργασία, όπως φαίνεται σχηματικά και παρακάτω.



Σχήμα 1.4 - Ροή Υποενοτήτων Δεδομένων Εκπαίδευσης σε Παράλληλη Υλοποίηση SVM

1.1.3 Ανάλυση Συναισθημάτων

Με τον όρο ανάλυση συναισθημάτων (sentiment analysis) ή εξόρυξη γνώμης (opinion mining) αναφερόμαστε κυρίως στην μελέτη της προσωπικής άποψης και των συναισθημάτων ενός συνόλου οντοτήτων (έγγραφα, εικόνες, ή κάποιο άλλο είδος πολυμέσου) απέναντι σε κάποια άλλη οντότητα (όπως π.χ. ένα συγκεκριμένο πρόσωπο, κάποιο προϊόν, ή ένα γενικό θέμα προς συζήτηση). Παρόλο που οι δύο όροι χρησιμοποιούνται εδώ σαν συνώνυμα, έχουν τεθεί στο παρελθόν διαχωρισμοί έτσι ώστε με την εξόρυξη γνώμης να ορίζεται η εξαγωγή και ανάλυση της γνώμης ενός συνόλου ατόμων περί μιας οντότητας, ενώ με την ανάλυση συναισθημάτων να περιγράφεται μια διεργασία που εμπεριέχει επεξεργασία φυσικής γλώσσας και ανάκτηση πληροφορίας πάνω σε ένα σύνολο εγγράφων για τον προσδιορισμό της συναισθηματικής πολικότητας μιας οντότητας.

Η ανάλυση συναισθημάτων θεωρείται γενικότερα ως ένα πρόβλημα κατηγοριοποίησης με επίλυση μέσω μεθόδων της μηχανικής μάθησης. Μέσω αυτής ένας χρήστης αποσκοπεί στην εξαγωγή και αναγνώριση της συναισθηματικής πληροφορίας που βρίσκεται σε ένα στοιχείο εισόδου (εδώ για παράδειγμα ένα αρχείο κειμένου), ούτως ώστε να προσδιοριστεί η κλάση συναισθηματικής πολικότητας του που μας αφορά για αυτή την μελέτη.

Φυσικά με την ανθρώπινη επικοινωνία να έχει βρει τις προϋποθέσεις στο διαδίκτυο για την ανταλλαγή απόψεων, κοινοποίηση αντιδράσεων, ή ανάγνωση προσωπικών σχολίων για προϊόντα και υπηρεσίες (κυρίως) μέσω κειμένου, γίνεται αντιληπτή η δυνατότητα και ανάγκη για την μελέτη αυτού του πολύτιμου όγκου πληροφορίας. Με τις κατάλληλες συνθήκες για την εφαρμογή μιας τέτοιας ανάλυσης να διαμορφώνονται σε πραγματικό χρόνο, βρίσκει πρόσφορο έδαφος σε χώρους σαν το marketing (π.χ. για την μελέτη των προτιμήσεων των καταναλωτών) μέσω των κοινωνικών μέσων δικτύωσης (για μια αντιπροσωπευτική σφυγμομέτρηση των εντυπώσεων σε ό,τι αφορά κάποια υπηρεσία ή προϊόν). Για παράδειγμα, το Twitter ως μια δημόσια microblogging πλατφόρμα που περιέχει σαν πυρήνα επικοινωνίας μικρά μηνύματα κειμένου έχει γίνει ουκ ολίγες φορές πηγή εξόρυξης γνώμης. Αυτό οφείλεται εν μέρει και στην ευκολία "ομαδοποίησης" αναρτήσεων ανά θεματική ενότητα με χρήση των ετικετών (hashtags) ώστε να μπορούν να χρησιμοποιηθούν σχετικά εύκολα ως δεδομένα εκπαίδευσης σε κάποιο μοντέλο μηχανικής μάθησης για κατηγοριοποίηση ενός συνόλου μηνυμάτων.

Παρόλα αυτά, πρέπει πάντα να λαμβάνεται υπόψη ότι τα δεδομένα που προέρχονται από μια τέτοια πλατφόρμα είναι συχνά επιρρεπή σε θόρυβο λόγω της δυνατότητας συμμετοχής οποιουδήποτε χρήστη με οποιαδήποτε σκοπιμότητα. Γενικότερα η ανάλυση κειμένου οποιουδήποτε είδους εξαρτάται σε μεγάλο βαθμό από την ποιότητα της πληροφορίας που καλείται να επεξεργαστεί. Ειδικές περιπτώσεις γραπτού λόγου όπως ο σαρκασμός ή τα συμφραζόμενα στα οποία βασίζεται ένα σώμα κειμένου (που αναφέρθηκαν και παραπάνω) επηρεάζουν σε μεγάλο βαθμό την αποτελεσματικότητα του μοντέλου. Για αυτό είναι ιδιαίτερα σημαντικό σε υλοποιήσεις που βασίζονται σε μηχανική μάθηση (όπως αυτές που αναπτύχθηκαν για την παρούσα εργασία) τα δεδομένα εισόδου να επιλέγονται και αξιολογούνται από ανθρώπους που μπορούν να διακρίνουν τέτοιες κρίσιμες περιπτώσεις βάσει των προσωπικών τους γνώσεων πάνω στον τομέα ή στο δείγμα ατόμων που εργάζεται το μοντέλο.

1.2 MapReduce

Όπως επισημάνθηκε στην αρχή, στην εποχή των big data προκύπτει όλο και πιο συχνά η ανάγκη ανάλυσης και διαχείρισης σημαντικού όγκου δεδομένων όσο το δυνατόν πιο γρήγορα. Μια προσέγγιση για την αντιμετώπιση αυτού του μεγέθους συλλογών δεδομένων θα μπορούσε να είναι η εφαρμογή αυτών σε έναν υπερυπολογιστή (supercomputer) με τα κατάλληλα υλικά χαρακτηριστικά

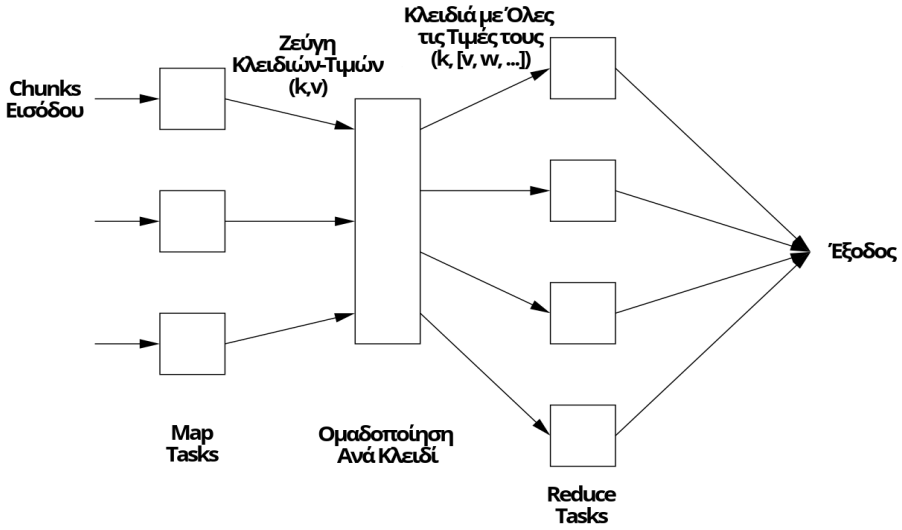
ώστε να υποστηρίξει τον μεγάλο φόρτων αυτών των υπολογισμών παραλληλοποιημένα σε κλίμακα ενός ή μερικών κόμβων τέτοιων μηχανημάτων. Αυτή η προσέγγιση, βέβαια, προϋποθέτει στον πυρήνα της μια οικονομική και υλικοτεχνική ευχέρεια που την καθιστά αποτρεπτική για εφαρμογές που δεν χρειάζονται απαραίτητα τέτοια ισχύ ή οργανισμούς που δεν μπορούν να υποστηρίξουν ένα τέτοιο εγχείρημα.

Σε αυτές τις συνθήκες προτιμάται η ανάπτυξη ενός συστήματος βασισμένο στον παραλληλισμό διεργασιών στο επίπεδο των **υπολογιστικών συστοιχιών** (computing clusters) που αποτελούνται από κοινούς υπολογιστές με συμβατικά υλικά χαρακτηριστικά που βρίσκονται διαθέσιμα στο εμπόριο, και συνδέονται μεταξύ τους μέσω διακοπτών δικτύου (switches) ή καλωδίων Ethernet. Αυτή η πρακτική είναι γνωστή ως **scale-out**, και μπορεί να υποστηρίξει δυναμικά μικρό η μεγάλο όγκο δεδομένων προς επεξεργασία, συνδέοντας είτε μικρό είτε μεγάλο αριθμό μηχανημάτων στην τοπολογία της συστοιχίας που θα χρησιμοποιηθεί.

Βέβαια με την διασύνδεση πολλαπλών υπολογιστών μεταξύ τους προκύπτει το ζήτημα αποθήκευσης και αναφοράς των δεδομένων από και για κάθε ανεξάρτητο μηχάνημα. Αυτή την ανάγκη σχεδιάστηκαν να εξυπηρετούν τα **κατανεμημένα συστήματα αρχείων** (distributed file systems) για την πλήρη αξιοποίηση της παραλληλοποίησης των υπολογισμών αλλά και την αξιοπιστία ενός συστήματος που ορίζεται από ανεξάρτητους υπολογιστικούς κόμβους που ενδέχεται να καταρρεύσουν ανά πάσα στιγμή (έτσι ώστε οι υπολογισμοί να συνεχίσουν και μετά από την κατάρρευση ενός ή περισσότερων υπολογιστών). Για αυτό τα συγκεκριμένα συστήματα αρχείων φροντίζουν κάθε αρχείο ενός κόμβου να αποθηκεύεται εφεδρικά και σε μερικούς άλλους κόμβους, ώστε να μπορεί να προσπελαστεί ανά πάσα στιγμή όταν ο πρώτος κόμβος και τα αρχεία του δεν είναι διαθέσιμα.

Έχοντας όμως υπόψη ότι αναφερόμαστε πάντα στον τομέα του cluster computing, θα πρέπει να αναμένουμε αρχεία (εισόδου, εξόδου ή ενδιάμεσα των υπολογισμών) της τάξης των gigabyte και άνω, τα οποία ως επί το πλείστον δεν θα ενημερώνονται συχνά παρά μόνο ίσως για την προσάρτηση νέων δεδομένων στα ήδη υπάρχοντα. Για αυτό το λόγο τα αρχεία σε ένα τέτοιου είδους σύστημα διαιρούνται σε **chunks**, δηλαδή κομμάτια δεδομένων (συχνά μεγέθους 64 megabytes, όμως αυτό μπορεί να οριστεί κι από τον χρήστη) που αντιγράφονται αρκετές φορές σε αρκετούς υπολογιστές του δικτύου. Η εύρεση κάθε chunk για ένα έγγραφο από έναν κόμβο γίνεται μέσω μιας οντότητας που αναφέρεται ως **master node** ή **name node**, που και αυτή με τη σειρά της αντιγράφεται αρκετές φορές όπως τα αρχεία ευρετηρίου σε έναν ειδικό κατάλογο που μπορεί να προσπελάσει κάθε υπολογιστική μονάδα για να συλλέξει τα κομμάτια ενός ή περισσότερων εγγράφων.

Δίνοντας περαιτέρω προσοχή στα περί κατανεμημένων συστημάτων αρχείων, γίνεται εμφανής μεν η πρόληψη κατάρρευσης ενός υπολογιστή στη συστοιχία στο επίπεδο των εγγράφων, όμως σημαντική είναι και η μέριμνα διαίρεσης των υπολογισμών σε **διεργασίες** για την επανεκκίνηση των υπολογισμών σε περίπτωση κατάρρευσης χωρίς να επηρεάζονται τα υπόλοιπα στιγμιότυπα διεργασιών. Αυτό γίνεται δυνατό μέσω του υπολογιστικού μοντέλου του MapReduce, που αρχικά αναπτύχθηκε εσωτερικά για λογαριασμό της Google με σκοπό την διαχείριση υπολογισμών πάνω σε μεγάλους όγκους πληροφορίας με χαρακτηριστική ανοχή σε σφάλματα υλικού. Η κύρια δομή αυτού του τρόπου υπολογισμού περιέχει την σύνταξη δύο συναρτήσεων, **Map** και **Reduce** (των οποίων τα στιγμιότυπα διεργασιών καλούνται mappers και reducers αντίστοιχα), με το ίδιο το σύστημα να διαχειρίζεται όλα τα υπόλοιπα (δηλαδή τον συγχρονισμό των διεργασιών, την εξισορρόπηση φόρτου, την προσαρμογή σε περίπτωση σφάλματος, και την παράλληλη εκτέλεση των στιγμιοτύπων).



Σχήμα 1.5 - Σχηματική Απεικόνιση Υπολογισμού σε MapReduce

Βήμα Υπολογισμού	Περιγραφή
Map	Ένας αριθμός Map tasks δέχονται ως είσοδο έναν αριθμό chunks από το κατανεμημένο σύστημα αρχείων, τα οποία μετατρέπουν σε μια ακολουθία ζευγαριών κλειδιού-τιμής με τον τρόπο που ορίζει ο χρήστης στην αντίστοιχη συνάρτηση.
Group By Key	Τα ζεύγη του προηγούμενου βήματος συλλέγονται και ταξινομούνται ανά κλειδί από μια διεργασία ελέγχου (master controller) πριν μοιραστούν σε έναν αριθμό Reduce tasks (με όσα ζεύγη έχουν ίδιο κλειδί να προκύπτουν πάντα στο ίδιο Reduce task).
Reduce	Κάθε Reduce task επεξεργάζεται κάθε κλειδί ξεχωριστά συγκεντρώνοντας όλες τις τιμές του (πάλι, με τον τρόπο που ορίζει ο χρήστης στην αντίστοιχη συνάρτηση) και με τα αποτελέσματα που προκύπτουν να αποθηκεύονται στο κατανεμημένο σύστημα αρχείων.

Πίνακας 1.4 - Βήματα Υπολογισμού σε MapReduce

Κοιτώντας πιο ειδικά τις μεθόδους Map, κάθε διεργασία αυτού του τύπου αναγνωρίζει τα chunks που λαμβάνει ως μια σειρά στοιχείων οποιασδήποτε μορφής. Επιλέγεται έτσι από κάθε mapper ένα στοιχείο σαν όρισμα και παράγεται ένας αριθμός ζευγαριών κλειδιού-τιμής (σε όποιο τύπο δεδομένων έχει επιλέξει ο χρήστης, χωρίς να υπάρχει περιορισμός μοναδικότητας για κάθε κλειδί).

Η ομαδοποίηση ανά κλειδί που λαμβάνει χώρα σε ύστερο χρόνο γίνεται αυτοματοποιημένα από το σύστημα και ανεξάρτητα από τις επιλογές και προτιμήσεις του χρήστη στις υλοποιήσεις των μεθόδων Map και Reduce. Συγκεκριμένα εδώ η διεργασία **master controller** συγκεντρώνει σε μια λίστα όλες τις τιμές από τα ζεύγη που έχουν την ίδια τιμή κλειδιού και στέλνει αυτές τις δομές στα Reduce tasks (γνωρίζοντας από πριν τον αριθμό των tasks, συχνά ορισμένο από τον χρήστη). Η λίστα των τιμών ενός συγκεκριμένου κλειδιού αποστέλλονται πάντα στο ίδιο Reduce task.

Στα των μεθόδων Reduce, αυτές λαμβάνουν ως είσοδο ένα κλειδί με μια λίστα των τιμών του, όπως περιγράφηκε στη προηγούμενη φάση. Βάσει των προδιαγραφών που όρισε ο χρήστης στην

υλοποίηση της αντίστοιχης μεθόδου, εκτελούνται οι reducers πάνω σε διαφορετικά chunks και τα αποτελέσματα έχουν την μορφή ενός συνόλου ζευγαριών κλειδιού-τιμής που μπορεί να διαφέρουν (χωρίς αυτό να είναι απόλυτο) σε μορφή από τα αντίστοιχα των Map tasks.

Αξίζει να παρατηρηθεί από τα παραπάνω πως φαίνεται ότι προβλέπεται η ικανότητα μιας μεθόδου Reduce να είναι ορισμένη με τέτοιο τρόπο ώστε να σχετίζεται άμεσα με τα αποτελέσματα της μεθόδου Map (π.χ. σε ένα παράδειγμα καταμέτρησης του πλήθους των εμφανίσεων κάθε λέξης σε ένα σύνολο εγγράφων). Σε αυτές τις συνθήκες μπορεί να εφαρμοστεί συμπληρωματικά η έννοια του **Combiner** που μεταφέρει την λειτουργία των reducers στο τελικό στάδιο των mappers, δημιουργώντας ένα τοπικό ενδιάμεσο βήμα για κάθε Map task. Σε αυτό το στάδιο συναθροίζονται μεταξύ τους τα ζεύγη με ίδιο κλειδί προτού συναθροιστούν ξανά και ταξινομηθούν για να αναλάβουν τα αποτελέσματα αυτά οι reducers.

Σε αυτό το προγραμματιστικό μοντέλο οι διεργασίες που εκτελούνται παράλληλα σε διαφορετικά μηχανήματα ονομάζονται **εργάτες** (workers) και ελέγχονται από την επιβλέπουσα οντότητα του master controller. Επί της ουσίας ο master controller έχει όλες τις αρμοδιότητες που κρίνονται αφηρημένες από την πλευρά του χρήστη για την βελτίωση της παραγωγικότητας σε ένα κατά τα άλλα περίπλοκο μοντέλο διαχείρισης ενός παράλληλου και κατανεμημένου υπολογισμού. Ενδεικτικά δημιουργεί όσα Map και Reduce tasks όρισε ο χρήστης, τα εκχωρεί σε εργάτες και παρακολουθεί την κατάσταση κάθε ενός από αυτούς. Το τελευταίο μέρος έχει ιδιαίτερη σημασία στην περίπτωση που ένας υπολογιστικός κόμβος (ακόμα και αυτός που στεγάζει τον ίδιο τον master controller) καταρρεύσει. Με εξαίρεση το ενδεχόμενο κατάρρευσης του εξοπλισμού που περιέχει τη κύρια διεργασία ελέγχου εργατών (που συνεπάγεται φυσικά την γενική κατάρρευση της εφαρμογής), είναι αναγκαία η πρόνοια για σφάλματα υλικού ή λογισμικού ούτως ώστε να μην επηρεάζεται η λειτουργία του υπολογισμού. Αυτό αντιμετωπίζεται με την ενημέρωση κατάστασης του master controller από τους κόμβους ανά τακτά χρονικά διαστήματα με τέτοιο τρόπο ώστε να εντοπιστεί η αστοχία και να διαταχθεί η επανεκτέλεση όλων των tasks σε κάποιο κόμβο (σε περίπτωση αστοχίας σε ένα Map task) ή να ενημερωθεί και κοινοποιηθεί η κατάσταση του κόμβου στα υπόλοιπα μέλη της διάταξης (σε περίπτωση αστοχίας σε ένα Reduce task).

Το μοντέλο του MapReduce βρίσκει συχνή χρήση σε αλγορίθμους με πολλαπλασιασμούς ή άλλες πιο πολύπλοκες πράξεις πάνω σε πίνακες ή διανύσματα, υπολογισμούς που αφορούν το πεδίο της σχεσιακής άλγεβρας, και άλλες διαδικασίες που δύνανται να παραλληλοποιηθούν κατά τους υπολογισμούς και να συναθροιστούν στο τέλος τα αποτελέσματα τους. Η εύκολη εφαρμογή και επέκταση του MapReduce βέβαια δεν είναι χαρακτηριστικά που επιλύουν τα ζητήματα κόστους επικοινωνίας που υπάρχουν σε αυτό μα και εν γένει σε μοντέλα παράλληλου υπολογισμού. Για έναν ενδεικτικό υπολογισμό σε MapReduce, το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης αφιερώνεται στην επικοινωνία και τον συγχρονισμό των υπολογισμών από βήμα σε βήμα. Αυτό βέβαια δεν σημαίνει ότι ένας αλγόριθμος βελτιστοποιείται απαραίτητα με όσο λιγότερη επικοινωνία γίνεται στις συνιστώσες που απαρτίζουν την παράλληλη υλοποίηση του, αφού αλγόριθμοι με διαφορετικό σκοπό και εφαρμογή έχουν και διαφορετικές ανάγκες επικοινωνίας.

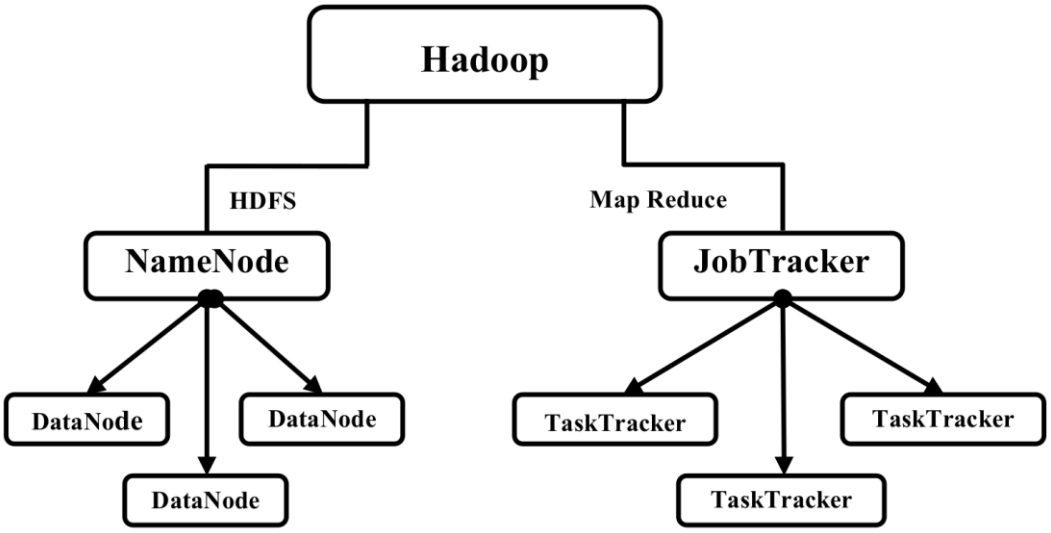
1.2.1 Apache Hadoop

Με προφανή τα οφέλη και τις δυνατότητες επέκτασης του MapReduce, αναπτύχθηκαν αρκετές βιβλιοθήκες για την υποστήριξη του μοντέλου σε γλώσσες όπως Python, Java, Scala, R και άλλες. Η πιο γνωστή υλοποίηση ανοιχτού λογισμικού για το MapReduce είναι το Apache Hadoop, μια σουίτα

που πέρα από τον βασικό κορμό της διαθέτει και έναν αριθμό εργαλείων (που ονομάζεται και **οικοσύστημα**) σε μορφή επεκτάσεων για ειδικές εφαρμογές πάνω σε επεξεργασία και διαχείριση δεδομένων με παράλληλο και κατανεμημένο τρόπο.

Το Apache Hadoop είναι επί της ουσίας η open source εναλλακτική για την κατανεμημένη επεξεργασία και αποθήκευση δεδομένων στην κλίμακα των big data στο μοντέλο του MapReduce χρησιμοποιωντας υλικοτεχνική υποδομή μικρού κόστους. Το framework αυτό χαρακτηρίζεται βεβαίως από μια κλιμακωτή αρχιτεκτονική, ούτως ώστε να λειτουργεί από έναν κόμβο έως έναν μεγάλο αριθμό κόμβων. Αποθηκεύει τα δεδομένα που προκύπτουν ή δίνονται ως είσοδος κατανεμημένα στον εξοπλισμό μιας συστοιχίας και διαχειρίζεται τις αστοχίες που ανιχνεύονται κατά την εκτέλεση μιας εφαρμογής για την παροχή υπηρεσιών ακόμα και σε οριακές περιπτώσεις που παρουσιάζονται σφάλματα σε πολλούς κόμβους.

Οι κόμβοι σε μια συστοιχία που χρησιμοποιεί το Hadoop διακρίνονται σε δύο τύπους, τον **master node** και τον **worker node** (αναφερόμενοι στα αντίστοιχα στοιχεία που εξετάστηκαν γενικότερα παραπάνω). Ο master node οφείλει να γνωρίζει την τοποθεσία των chunks για κάθε αρχείο στους worker nodes, και για αυτό εκτελεί χρέη διαχειριστή του συστήματος αρχείων (εδώ γνωστό ως HDFS) του cluster. Ένας worker node από την άλλη είναι αρμόδιος για την εύρεση και ανάκτηση των δεδομένων εισόδου βασιζόμενος στις πληροφορίες που δίνονται από τον master node, την επεξεργασία όπως ορίστηκε από τον χρήστη, και την περιοδική αναφορά στον master node σε ό,τι αφορά τα νέα δεδομένα που προκύπτουν και αποθηκεύει τοπικά.



Σχήμα 1.6 - Είδη Κόμβων στο Apache Hadoop

Σε μια πιο γενική εικόνα, στο Hadoop δεν γίνεται τίποτα παραπάνω από την φόρτωση των δεδομένων, την επεξεργασία αυτών κατανεμημένα στους worker nodes βάσει της υλοποίησης στην μέθοδο Map, και την συλλογή των ενδιάμεσων δεδομένων από τους worker nodes βάσει της υλοποίησης στην μέθοδο Reduce. Αυτές οι δύο μέθοδοι (καθώς και μια κύρια συνάρτηση για τον συντονισμό και την εξειδικευμένη προσαρμογή των μεθόδων) είναι και εδώ οι μοναδικές που καλείται ο χρήστης να αναπτύξει για την εκτέλεση μιας διεργασίας πάνω στο framework του Hadoop, ακολουθώντας κατά γράμμα την λογική του MapReduce.

Το Hadoop μπορεί να χαρακτηριστεί στον πυρήνα του μέσα από τις τέσσερις κύριες συνιστώσες που τον αποτελούν και μελετούνται στις επόμενες τέσσερις παραγράφους, ενώ μία ακόμη παράγραφος αφιερώνεται στην ενδεικτική αναφορά μερικών υλοποιήσεων του οικοσυστήματος του Hadoop.

1.2.1.1 Hadoop Common

Αυτή η συνιστώσα (παλαιότερα γνωστή με το όνομα Hadoop Core) περιέχει την συλλογή των εργαλείων και βιβλιοθηκών που απαιτούνται για την υποστήριξη των υπολοίπων συνιστωσών (HDFS, MapReduce, YARN) που θα περιγραφούν στην συνέχεια, καθώς και τα αρχεία που χρειάζονται για την εκκίνηση του Hadoop σε ένα σύστημα.

1.2.1.2 Hadoop Distributed File System (HDFS)

Το HDFS είναι το κατανεμημένο σύστημα αρχείων που υλοποιήθηκε για την διασύνδεση των δεδομένων που βρίσκονται σε ένα υπολογιστικό cluster που λειτουργεί με Hadoop. Λόγω της υποστήριξης εξοπλισμού χαμηλού κόστους χαρακτηρίζεται από υψηλή ανοχή σφαλμάτων, χωρίς αυτό να επηρεάζει απαραίτητα τον ρυθμό μετάδοσης πληροφορίας μεταξύ κόμβων για την εξυπηρέτηση ανάκτησης και επεξεργασίας μεγάλων συλλογών δεδομένων. Επιπροσθέτως υποστηρίζει την αποθήκευση των δεδομένων σε μορφή αρχείων κειμένου για μια πιο φιλική και ευανάγνωστη αξιολόγηση των αποτελεσμάτων μιας εργασίας από ανθρώπους. Είναι επίσης υπεύθυνο για την διαίρεση των μεγάλων αρχείων σε chunks και την εφεδρική αποθήκευση τους σε διαφορετικούς κόμβους σε περίπτωση κατάρρευσης ενός ή περισσότερων κόμβων. Όπως προαναφέρθηκε, για λογαριασμό του HDFS υφίστανται δύο τύποι κόμβων, οι master nodes για τη διαχείριση του συστήματος αρχείων και worker nodes για την αποθήκευση των δεδομένων.

1.2.1.3 Hadoop MapReduce

Η συνιστώσα του MapReduce στο Hadoop είναι υπεύθυνη για ό,τι αφορά τον κατανεμημένο υπολογισμό μιας εφαρμογής. Φροντίζει να διαβάζει τα δεδομένα από το HDFS (αν και όχι απαραίτητα, αφού μπορεί να προσπελάσει και τοπικά αρχεία ή βάσεις δεδομένων) και να διαχωρίζει τον υπολογιστικό φόρτο που προκύπτει σε ένα σύνολο κόμβων, εκφράζοντας τον σε μορφή ζευγαριών κλειδιού-τιμής. Αντίστοιχα με την συνιστώσα του HDFS, το MapReduce εκφράζεται από μεριάς του μέσα από δύο τύπους κόμβων. Ο πρώτος είναι ο **job tracker** που είναι υπεύθυνος για την ανάθεση διεργασιών σε ένα σύνολο που αποτελείται από στιγμιότυπα του δεύτερου τύπου κόμβου, τον **task tracker**. Ο job tracker είναι ένας και μοναδικός στην όλη διάταξη (που σημαίνει ότι σε περίπτωση κατάρρευσης, όλοι οι υπολογισμοί θα πάψουν) και βρίσκεται στον master node παρακολουθώντας τις διεργασίες που εκτελούν οι task trackers μέσω τακτικών αναφορών που αποστέλλονται από τους τελευταίους για την διασφάλιση ότι ένας κόμβος είναι σε λειτουργία και επεξεργάζεται δεδομένα εκείνη τη στιγμή. Ένας task tracker λαμβάνει διεργασίες απο τον job tracker και τις εκτελεί μέσα από στιγμιότυπα υλοποιήσεων των μεθόδων Map και Reduce, στέλνοντας περιοδικές αναφορές επιβεβαίωσης ότι δεν έχει καταρρεύσει.

1.2.1.4 Hadoop YARN (Yet Another Resource Negotiator)

Το YARN είναι υπεύθυνο για τον χρονοπρογραμματισμό και τα αιτήματα πόρων που προκύπτουν σε κατανεμημένα συστήματα. Ουσιαστικά πρόκειται για μια κεντρική οντότητα

διαχείρισης πόρων που διευθύνει πως η κάθε εφαρμογή κάνει χρήση των πόρων ενός συστήματος Hadoop, μέσω αναφορών που δίνονται από κάθε κόμβο ούτως ώστε να προωθείται μια διεργασία σε κάποιον εν λειτουργία κόμβο σε περίπτωση κατάρρευσης. Κατά μία έννοια το YARN μπορεί να χαρακτηριστεί σαν ένα είδος λειτουργικού συστήματος για ολόκληρη τη συστοιχία που αναφέρεται, προσφέροντας σημαντικά εργαλεία για να αξιοποιήσει κάθε εφαρμογή τις δυνατότητες που δίνονται από την παράλληλη δομή επεξεργασίας και αποθήκευσης πόρων του Hadoop. Για αυτή την τόσο ευαίσθητη και σημαντική πτυχή είναι υψίστης σημασίας το YARN να χαρακτηρίζεται από κλιμακωσιμότητα, αξιοπιστία, διαθεσιμότητα και βέλτιστη χρήση της συστοιχίας όπου στεγάζεται το εκάστοτε σύστημα.

1.2.1.5 Hadoop Ecosystem

Πέρα από τις συνιστώσες στον πυρήνα του Hadoop, αυτή η πλατφόρμα χαρακτηρίζεται όπως αναφέρθηκε προηγουμένως και από την επεκτασιμότητα του μέσω ενός συνόλου εργαλείων που είτε είναι σχεδιασμένα πάνω σε αυτό, είτε σχετίζονται σε μεγάλο βαθμό μαζί του. Τα εργαλεία αυτά αποτελούν το οικοσύστημα του Hadoop και εστιάζουν στην επεξεργασία συνόλων δεδομένων με εξειδικευμένους τρόπους. Σε αυτή την ενότητα αναφέρονται ενδεικτικά μερικές από αυτές τις επεκτάσεις.

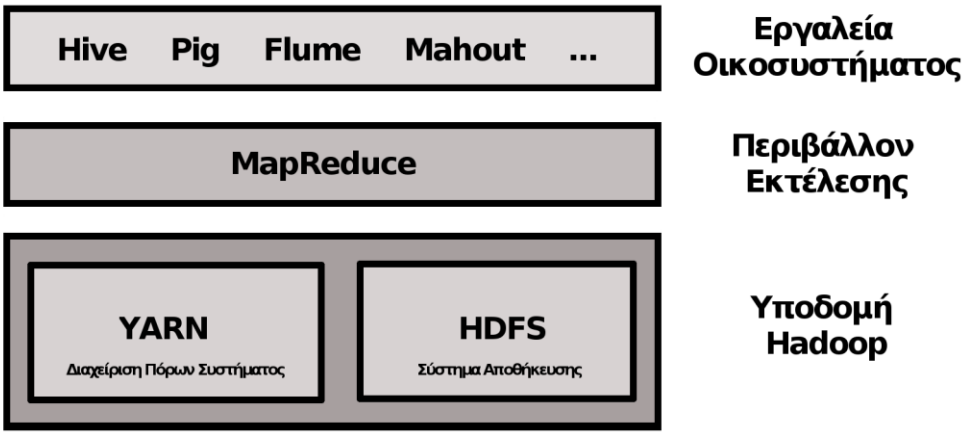
Όπου χρειάζεται περαιτέρω ανάλυση, αναζήτηση, ή σύνοψη των αποτελεσμάτων μιας εφαρμογής πάνω σε μια αποθήκη δεδομένων (data warehouse), μπορεί να επιστρατευτεί το εργαλείο του **Apache Hive**. Πρόκειται για ένα σύστημα διαχείρισης αποθηκών δεδομένων που δίνει τη δυνατότητα να εφαρμοστεί λογική πινάκων στα αποθηκευμένα δεδομένα μέσω εντολών σε γλώσσα HiveQL που τρέχουν στο MapReduce. Με αυτό το εργαλείο γίνεται προσπάθεια παρουσίασης ενός συνόλου δεδομένων σε κατανεμημένο περιβάλλον με όρους βάσεων δεδομένων, ώστε να μετατραπούν τα αδόμητα δεδομένα σε μια πιο καλά δομημένη μορφή. Μπορεί να χρησιμοποιηθεί σαν μια παροδική λύση αν ο χρήστης ενός συστήματος Hadoop δεν είναι τόσο εξοικειωμένος με το μοντέλο του MapReduce όσο με τις εύκολες εκδοχές των γλωσσών τύπου SQL. Τα παραπάνω φυσικά δεν αντικαθιστούν την ανάγκη εξειδικευμένης γνώσης για την δημιουργία πολύπλοκων εφαρμογών για ειδικούς σκοπούς.

Έχοντας γνώση της αναγκαίας συνθήκης για αλληλουχία εκτέλεσης διαφόρων μεθόδων Map και Reduce με συγκεκριμένο τρόπο, παρατηρείται ότι αυτό το μοτίβο μπορεί να παρουσιάσει αρκετά σχεδιαστικά και όχι μόνο ζητήματα ώστε μια εφαρμογή να συμμορφωθεί σε αυτό. Αυτό καλείται να αντιμετωπίσει το **Apache Pig** που μέσω ενός ειδικού συντακτικού με όνομα Pig Latin υποστηρίζει ευέλικτες δομές δεδομένων και άλλα προγραμματιστικά οφέλη για να περιγράψει τους μετασχηματισμούς και τις εργασίες στα δεδομένα εισόδου με τέτοιο τρόπο που να αποκρύπτονται οι "αλυσίδες" MapReduce εργασιών που απαιτούνται εσωτερικά. Με αυτό τον τρόπο ο χρήστης βρίσκεται σε ένα πιο φιλικό επίπεδο αφαίρεσης για το σχεδιαστικό κομμάτι της εφαρμογής, με υλοποιήσεις που περιγράφονται μόλις σε μερικές γραμμές με Pig Latin. Παρά τα οφέλη του, η χρήση του Pig δεν εγγυάται πάντα καλύτερη επίδοση σε σχέση με τα αντίστοιχα προγράμματα σε κλασικό MapReduce για πιο εξειδικευμένους αλγορίθμους.

Όταν τα δεδομένα προς επεξεργασία δεν βρίσκονται ήδη αποθηκευμένα τοπικά ή στο HDFS, αλλά βρίσκονται σαν ροές δεδομένων σε κάποιον απομακρυσμένο εξυπηρετητή, μια λύση δίνει το εργαλείο του **Apache Flume**, μέσω του οποίου το Hadoop μπορεί να συγκεντρώσει και αποθηκεύσει αυτές τις ροές στο HDFS. Το Flume αναμένει κάποια ορισμένα (συνήθως από τον χρήστη) γεγονότα

(events) που λαμβάνουν χώρα από την πηγή των δεδομένων ώστε αυτά να αντιγραφούν στο σύστημα αρχείων του Hadoop κι από εκεί και πέρα λαμβάνει χώρα η επεξεργασία τους σε μια MapReduce εργασία. Βρίσκει χρήση κυρίως σε εφαρμογές που απαιτούν την ανάλυση δεδομένων που προκύπτουν σε πραγματικό χρόνο.

Δεν είναι λίγες οι φορές τώρα που οι αλγόριθμοι που καλούνται οι χρήστες να εφαρμόσουν σε ένα μοντέλο MapReduce φαίνονται πολυσύνθετοι και με υψηλό βαθμό ρίσκου για ενδεχόμενα λάθη υλοποίησης. Τέτοιες περιπτώσεις προκύπτουν φυσικά συχνά και σε αλγορίθμους που αφορούν κατηγοριοποίηση, συσταδοποίηση, ή φιλτράρισμα δεδομένων, με το μεγάλο μέγεθος δεδομένων εισόδου να μεγαλώνει την ανάγκη για την εφαρμογή παράλληλων υπολογισμών όπου είναι δυνατόν. Αυτή η ανάγκη υποστήριξης πολύπλοκων εφαρμογών που αφορούν κυρίως μηχανική μάθηση (ή γενικότερα εξόρυξη δεδομένων) στο πλαίσιο του MapReduce χωρίς να χρειάζεται η πλήρης υλοποίηση τους από τον χρήστη οδήγησε στην δημιουργία του **Apache Mahout**. Σχεδιάστηκε για να προσφέρει την δυνατότητα χρήσης της βιβλιοθήκης του μέσω εντολών στην γραμμή εντολών του συστήματος, απλοποιώντας περαιτέρω την διαδικασία δημιουργίας ενός μοντέλου μηχανικής μάθησης σε μια αλληλουχία μερικών απλών βημάτων. Παρόλα αυτά, διεκόπη η υποστήριξη του μοντέλου MapReduce από το Mahout για να επικεντρωθεί η πλατφόρμα στην επεξεργασία συγκεκριμένων τύπων υπολογισμών. Έτσι σήμερα το Mahout έχει στραφεί σε πιο σύγχρονες και δημοφιλείς λύσεις για επεξεργασία στο επίπεδο της κύριας μνήμης (έναντι του MapReduce μοντέλου που βασίζεται πάντα στην ανάκτηση, επεξεργασία, και αποθήκευση στον δίσκο), αντίστοιχα με τον τρόπο εκτέλεσης που χαρακτηρίζει την επέκταση του Apache Spark που θα μελετήσουμε εκτενέστερα στη συνέχεια μιας και εστιάζει περισσότερο σε εφαρμογές που αφορούν το συγκεκριμένο πλαίσιο της παρούσας εργασίας.



Σχήμα 1.7 - Δομή Συνιστωσών Apache Hadoop

1.2.2 Apache Spark

Η προσήλωση του Hadoop στον χώρο αποθήκευσης του HDFS προξενεί εύλογες απορίες σχετικά με την εν γένει καθυστέρηση που προκύπτει όταν τα (ενδιάμεσα και τελικά) αποτελέσματα μιας εφαρμογής αποθηκεύονται και ανακτώνται από τους δίσκους των κόμβων της συστοιχίας. Η πιο απλή προσέγγιση για την παράκαμψη αυτού του ζητήματος είναι η ανάκτηση και επεξεργασία των δεδομένων στα διάφορα στάδια μιας εφαρμογής στην κύρια μνήμη για την πιο γρήγορη αποθήκευση, ανάγνωση, και εγγραφή της πληροφορίας. Αυτή τη φιλοσοφία σχεδιάστηκε για να ακολουθεί και εφαρμόζει εξ αρχής και το Apache Spark, μια πλατφόρμα για γρήγορους και γενικού τύπου υπολογισμούς μέσω μιας συστοιχίας ικανής να διατηρεί μεγάλα σύνολα δεδομένων στη μνήμη χωρίς να χρησιμοποιεί απαραίτητα το MapReduce σαν προγραμματιστικό μοντέλο. Αυτό βέβαια δεν το

αποκόπτει σε καμία περίπτωση από τα χαρακτηριστικά του Hadoop που είδαμε και προηγουμένως, μιας και το Spark έχει την ευχέρεια να υποστηρίξει δομές αποθήκευσης σαν το HDFS και να τρέξει στον μηχανισμό του YARN. Ευέλικτα σχεδιασμένο για συνεργασία με οντότητες του Hadoop και με την παροχή λειτουργικών διευκολύνσεων προς τον χρήστη μέσω των πολύτιμων εσωτερικών του βιβλιοθηκών, γίνεται ένα πρώτης τάξης εργαλείο για την ανάλυση και εξόρυξη δεδομένων στο επίπεδο των big data.

Εσωτερικά το Spark αποτελείται από μια σειρά συνιστωσών που αποτελούν τον βασικό πυρήνα του, με την μικρή διαφοροποίηση εδώ ότι όλες οι συνιστώσες είναι σχεδιασμένες με τέτοιο τρόπο που μπορούν να θεωρηθούν υπό συγκεκριμένες συνθήκες μία αδιαίρετη οντότητα. Το τελευταίο είναι άμεσο αποτέλεσμα του σκεπτικού περί της άμεσης υιοθέτησης ενδεχόμενων βελτιώσεων που έγιναν στους μηχανισμούς ενός κομματιού της πλατφόρμας από τα υπόλοιπα κομμάτια. Έτσι σε αντίθεση με το Hadoop που τα κομμάτια του λειτουργούν ξεχωριστά μεταξύ τους, το Spark ενθαρρύνει την άμεση συνεργασία μεταξύ των μελών του με αυτό να έχει άμεσο αποτέλεσμα στον χρόνο επεξεργασίας και την συντήρηση της υποδομής, τα οποία και περιγράφονται ένας προς ένα στην συνέχεια.

1.2.2.1 Spark Core

Το Spark Core είναι η μονάδα που περιέχει τα βασικά στοιχεία του Spark (όπως τα εργαλεία για την διαχείριση της μνήμης και τον χρονοπρογραμματισμό των διεργασιών) και τους ορισμούς της ιδιαίτερης δομής δεδομένων που προσφέρει η πλατφόρμα με το όνομα Ανθεκτικά Κατανεμημένα Σύνολα Δεδομένων (Resilient Distributed Datasets) ή **RDD**. Τα RDD είναι μια αμετάβλητη συλλογή δεδομένων που βρίσκονται αποθηκευμένα κατανεμημένα σε έναν αριθμό κόμβων. Με την συλλογή να διαχωρίζεται στους κόμβους, το RDD διαμελίζεται στα υποσύνολα του που ονομάζονται **partitions**, τα οποία μπορεί να επεξεργαστεί κάθε κόμβος ξεχωριστά. Με την δημιουργία ενός RDD προσφέρονται δύο λειτουργίες επάνω του. Η πρώτη λειτουργία είναι ο **μετασχηματισμός** (transformation) μέσω του οποίου δημιουργείται ένα καινούριο RDD όπως το ορίζει ο χρήστης, ενώ η δεύτερη είναι η **ενέργεια** (action) όπου γίνονται υπολογισμοί πάνω σε ένα RDD και προκύπτουν αποτελέσματα τα οποία επιστρέφονται στον χρήστη ή αποθηκεύονται σε ένα σύστημα αρχείων όπως το HDFS.

Για εργασίες που αφορούν big data, τα RDD μπορούν να δηλωθούν οποτεδήποτε σε μια διαδικασία, όμως υπολογίζονται με την πρώτη φορά που χρησιμοποιούνται σε μια ενέργεια (και επανυπολογίζονται με κάθε επόμενη ενέργεια πάνω σε αυτά). Αυτό είναι συνειδητή σχεδιαστική επιλογή για την αντιμετώπιση της φόρτωσης υπέρογκων δεδομένων στην μνήμη. Αν παρόλα αυτά μια εφαρμογή ωφελείται από την προσωρινή αποθήκευση ενός RDD έναντι του επανυπολογισμού του κάθε φορά, το Spark δύναται να συγκρατήσει τα δεδομένα που αποτελούν το RDD με κάθε υπολογιστή να αποθηκεύει τα partitions του. Σε περίπτωση κατάρρευσης, το Spark μπορεί είτε να χρησιμοποιήσει τα αντίγραφα εκείνου του partition που βρίσκονται σε άλλους κόμβους, είτε να υπολογίσει ξανά αυτά τα partitions επί τόπου.

1.2.2.2 Spark SQL

Όταν χρειάζεται επεξεργασία σε (ημι)δομημένα δεδομένα, το Spark μπορεί να υποστηρίξει ερωτήματα (queries) πάνω σε αυτά με τρόπο αντίστοιχο του Apache Hive με τη βοήθεια της συνιστώσας του Spark SQL. Μέσω αυτής οι χρήστες μπορούν να ανακτήσουν δεδομένα από πηγές με δομημένη μορφή (όπως δεδομένα αποθηκευμένα σε τύπο JSON ή Hive) και να ενσωματωθούν τα ερωτήματα πάνω σε αυτά στην κύρια υλοποίηση.

Για να γίνει κάτι τέτοιο δυνατό, το Spark SQL χρησιμοποιεί έναν ειδικό τύπο του RDD που αποτελείται από εγγραφές, γνωστός ως ***DataFrame***, που αποθηκεύει τα δεδομένα βέλτιστα βάσει της δομής τους. Το DataFrame κατά μία έννοια θυμίζει έναν πίνακα σε μια βάση δεδομένων, με τις εγγραφές του να αναπαριστώνται σαν ένα σύνολο στοιχείων με στήλες. Πέρα όμως από αυτή την ιδιαιτερότητα, σε ένα DataFrame μπορούν να εφαρμοστούν μετασχηματισμοί και ενέργειες όπως και σε ένα απλό RDD. Βεβαίως είναι δυνατή και η μετατροπή ενός RDD σε DataFrame, εφαρμόζοντας το σε μια δομή πίνακα με διακριτές στήλες για κάθε εγγραφή. Σχετικά με την προσωρινή αποθήκευση των δεδομένων στο πλαίσιο του Spark SQL, ακολουθείται το ίδιο μοτίβο των RDD έτσι ώστε οι εγγραφές να διατηρηθούν στην μνήμη κατά στήλη (για λόγους οικονομίας χώρου) ως το τέλος της εκτέλεσης της εφαρμογής στο Spark. Η ευελιξία της προσωρινής αποθήκευσης στο Spark SQL έχει ως αποτέλεσμα την έως και εκατό φορές πιο γρήγορη εκτέλεση ενός ερωτήματος από την αντίστοιχη εκτέλεση στο Hive, όπως έχει παρατηρηθεί σε σχετικές μελέτες.

1.2.2.3 Spark Streaming

Αντίστοιχα με κάποια εργαλεία όπως το Apache Flume στο Hadoop για την επεξεργασία ροών δεδομένων από εξωτερικές πηγές, η πλατφόρμα του Spark προτείνει εσωτερικά για αυτή την περίπτωση την χρήση του Spark Streaming, το οποίο προσφέρει δυνατότητες πάνω στα δεδομένα των ροών αρκετά παρόμοιες με αυτές της χρήσης RDD. Πιο συγκεκριμένα χρησιμοποιεί έναν τύπο δεδομένων διακριτής ροής (discretized stream) ή **Dstream** που αποτελείται από έναν αριθμό στιγμιοτύπων RDD που ανακτήθηκαν σε ένα καθορισμένο χρονικό διάστημα και δέχεται κυρίως δύο λειτουργίες πάνω στα δεδομένα του (μαζί με τις λειτουργίες ενός RDD), τον **μετασχηματισμό** (transformation) για δημιουργία ενός εκ νέου DStream, και τις **εργασίες εξόδου** (output operations) για την παρουσίαση των δεδομένων ή την εγγραφή τους σε ένα σύστημα εκτός της δικαιοδοσίας της συστοιχίας που εφαρμόζεται το Spark.

Γενικότερα το Spark Streaming δέχεται τα δεδομένα από μία πηγή και τα διαιρεί σε μικρά υποσύνολα περιοδικά βάσει ενός χρονοδιαγράμματος. Κατα τη διάρκεια ενός κβάντου χρόνου δημιουργείται μία θέση για ένα υποσύνολο δεδομένων (τύπου RDD) που στη συνέχεια διαμορφώνεται με την πρόσληψη πληροφορίας από την ροή. Η ακολουθία αυτών των υποσυνόλων είναι (όπως αναφέρθηκε) ένα DStream για το οποίο ισχύει ό,τι ίσχυε και για ένα RDD όσον αφορά τον υπολογισμό των περιεχομένων του με την πρώτη χρήση αυτών στο πρόγραμμα. Χαρακτηριστική ιδιαιτερότητα του DStream είναι η χρήση περιοδικών **checkpoints** για την διατήρηση δεδομένων που μπορούν να ανακτηθούν αργότερα με ασφάλεια σε περίπτωση κατάρρευσης.

1.2.2.4 GraphX

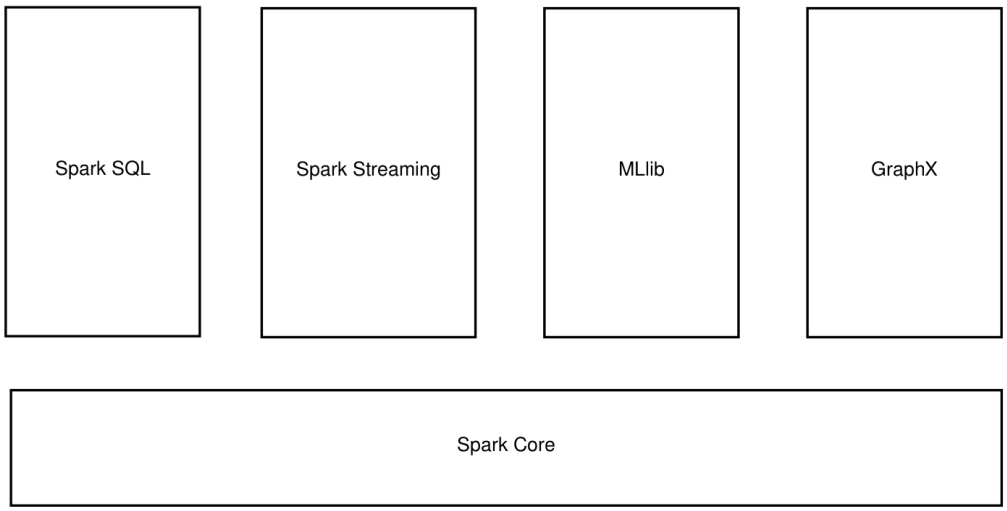
Η συνιστώσα του GraphX είναι υπεύθυνη για τον χειρισμό και τον παράλληλο υπολογισμό πάνω σε γράφους, επεκτείνοντας και αυτή με τη σειρά της τον τύπο του RDD. Με το GraphX μπορούν να δημιουργηθούν κατευθυνόμενοι γράφοι με τιμές και ιδιότητες σε κάθε ακμή και κόμβο καθώς και χρήσιμες ειδικές λειτουργίες για την διαχείριση τους. Δεν θα επεκταθεί περαιτέρω η μελέτη σε αυτό το κομμάτι της πλατφόρμας μιας και οι εφαρμογές του δεν είναι ιδιαίτερα σχετικές για το αντικείμενο της εργασίας.

1.2.2.5 MLlib (Machine Learning Library)

Το πιο σημαντικό και χρήσιμο κομμάτι του Spark για την παρούσα εργασία είναι φυσικά αυτό που σχετίζεται με την μηχανική μάθηση, προσφέροντας αλγόριθμους και υποστηρικτικές υποδομές για την παράλληλη εκτέλεση εφαρμογών για υψηλή ποιότητα εξόρυξης γνώσης και ταχύτητα εκτέλεσης. Έχοντας από προεπιλογή τα οφέλη της πλατφόρμας του Spark για την αποθήκευση δεδομένων στην κύρια μνήμη έναντι του δίσκου, δεν προκαλεί έκπληξη ότι μια εφαρμογή σε MLlib εμφανίζεται σε ενδεικτικές περιπτώσεις έως και εννιά φορές γρηγορότερη από την αντίστοιχη εφαρμογή σε Mahout.

Στο MLlib οι συλλογές μπορούν να έχουν την μορφή ενός DataFrame ή ενός RDD που δίνονται ως είσοδος σε έναν από τους παράλληλους αλγόριθμους που υποστηρίζει το MLlib για εκπαίδευση του μοντέλου και την αξιολόγηση του πάνω στα δεδομένα ελέγχου. Οι αλγόριθμοι καλύπτουν ένα εύρος εργασιών που ανήκουν ή βρίσκουν χρήση γύρω από την μηχανική μάθηση από την **εξαγωγή χαρακτηριστικών** μιας συλλογής μέσω της μεθόδου TFIDF ως την **εκτίμηση των αποτελεσμάτων** μέσω στατιστικών μεθόδων όπως η μήτρα σύγχυσης, ή από τις (γραμμικές ή λογιστικές) **παλινδρομήσεις** και **κατηγοριοποιήσεις** (π.χ. Naïve Bayes, SVM, κ.α.) της επιβλεπόμενης μάθησης ως την **συσταδοποίηση** (όπως ο K-means αλγόριθμος) και το **φιλτράρισμα δεδομένων** (βλ. την μέθοδο του PCA).

Φυσικά και εδώ η λειτουργία των αλγορίθμων του MLlib εξαρτάται από την ποιότητα των εισόδων τους, για αυτό και πρέπει να λαμβάνεται ειδική μέριμνα για την προετοιμασία και εκκαθάριση των δεδομένων που θα χρησιμοποιηθούν σε αλγορίθμους μηχανικής μάθησης. Χρειάζεται επίσης να αναφερθεί πως έχει παρατηρηθεί ότι ενώ για σχετικά μικρές συλλογές δεδομένων εισόδου οι αλγόριθμοι μηχανικής μάθησης στο Spark εύκολα ξεπερνούν το Hadoop σε επίδοση, σε εισόδους μεγαλύτερης κλίμακας που η κύρια μνήμη του συστήματος πιθανώς δεν επαρκεί για την επεξεργασία τους το Hadoop μπορεί σε συγκεκριμένες περιπτώσεις να φανεί πιο αποτελεσματικό. Για αυτό τον λόγο έχει αρκετή σημασία η ρύθμιση των παραμέτρων (όπως τον αριθμό των επαναλήψεων για τα βήματα μιας συσταδοποίησης ή τις σταθερές για την ρύθμιση των μοντέλων κατηγοριοποίησης) που προσφέρονται για κάθε αλγόριθμο καθώς και η βελτιστοποίηση της διαχείρισης της μνήμης καθ' όλη τη διάρκεια της εκτέλεσης βάσει των απαιτήσεων της εφαρμογής.



Σχήμα 1.8 - Δομή Συνιστωσών Apache Spark

2. Ανάπτυξη Υλοποιήσεων Ανάλυσης Συναισθημάτων Κειμένου

Μελετώντας όσα αναφέρθηκαν επιγραμματικά ή πιο αναλυτικά στη προηγούμενη ενότητα δόθηκε σαν πρώτο βήμα μία έμφαση σε επίπεδο δομής και περιγραφής ούτως ώστε να γίνει αρκετά ξεκάθαρη η εσωτερική "διαίρεση" του θέματος σε δύο βασικά τμήματα. Το ένα τμήμα αφορά γενικότερα την εξόρυξη δεδομένων για την εξαγωγή χρήσιμης πληροφορίας από μία αποθήκη δεδομένων κειμένου, ενώ το άλλο περιέχει τον παράλληλο υπολογισμό (με έμφαση στο προγραμματιστικό μοντέλο του MapReduce και παρεμφερή αυτού) συνόλων δεδομένων σε εξαιρετικά μεγάλες κλίμακες που οι κοινές σειριακές υλοποιήσεις δεν μπορούν να τις διαχειριστούν εύκολα ή αποδοτικά.

Αυτή η διαίρεση του πεδίου στο οποίο κυμαίνεται η εργασία κρίνεται συγχρόνως τεχνητή για λόγους ευκολίας στην ανάλυση και αναγκαία για τα ξεχωριστά χαρακτηριστικά που εκφράζουν κάθε τμήμα ξεχωριστά. Βεβαίως μια τέτοια επίπλαστη διχοτόμηση των θεμελίων δεν αποσκοπεί διόλου στην αποκλειστική ανάλυση κάθε τμήματος ξεχωριστά, αφού εύκολα προκύπτουν οι συνδέσεις μεταξύ των τμημάτων για την συνεργασία τους στην ανάπτυξη μιας εφαρμογής που στεγάζεται σε υπαρκτό εξοπλισμό και εργάζεται σε πραγματικά δεδομένα. Παρόλα αυτά εύλογα προκύπτει το ζήτημα πάνω στα κριτήρια που θα μελετηθεί κάθε εφαρμογή ανάλυσης συναισθημάτων κειμένου, καθορίζοντας ανάλογα τις μετρικές για να αξιολογηθεί συνολικά η αποτελεσματικότητα της. Το πιο προφανές μέτρο που μπορεί να χρησιμοποιηθεί για την επίδοση μιας υλοποίησης κατηγοριοποίησης δειγμάτων είναι φυσικά η ορθότητα πρόβλεψης του μοντέλου, δηλαδή το ποσοστό σωστών προβλέψεων κάθε υλοποίησης πάνω στα δεδομένα ελέγχου, συγκρίνοντας την ετικέτα κάθε στιγμιοτύπου προς την πρόβλεψη στην οποία κατέληξε κάθε εφαρμογή. Ένα τέτοιο μέτρο θεωρείται σχετικό για ό,τι αφορά την εξέταση της επίδοσης μιας υλοποίησης κατηγοριοποίησης, όμως πρέπει να ληφθεί υπόψη και η μελέτη της επίδοσης της παράλληλης εκτέλεσης σε κατανεμημένο περιβάλλον. Για να αναπτυχθούν εύρωστες εφαρμογές από άποψη παραλληλοποίησης υπολογισμών, χρειάζεται να ζυγιστούν σωστά οι ανάγκες που καλούνται να εξυπηρετήσουν, διατηρώντας την ισορροπία μεταξύ της καλύτερης ενδεχόμενης επίδοσης κατηγοριοποίησης και της βέλτιστης επίδοσης εκτέλεσης εν παραλλήλω σε μια υπολογιστική συστοιχία.

Για να γίνει όσο πιο ξεκάθαρη γίνεται αυτή η παραδοχή πριν αρχίσει η παρουσίαση των υλοποιήσεων που αναπτύχθηκαν για την παρούσα μελέτη, κρίνεται αναγκαίο η ανάλυση τους να είναι βασισμένη σε δύο επίπεδα, την **όσο το δυνατόν μεγαλύτερη ακρίβεια κατηγοριοποίησης στιγμιοτύπων ελέγχου** και της **όσο το δυνατόν πιο αποδοτική λειτουργία των αλγορίθμων με παράλληλο τρόπο**.

Για αυτό είναι υψίστης σημασίας σε αυτό το σημείο να καθοριστεί η βάση στην οποία **θα διαχωριστεί κάθε υλοποίηση σε δύο στιγμιότυπα**, μια **απλή έκδοση** και μια **τροποποιημένη έκδοση** της εκάστοτε μεθόδου κατηγοριοποίησης, με την τροποποιημένη έκδοση να επικεντρώνεται στην προσπάθεια απόκτησης καλύτερης επίδοσης στην εξόρυξη κειμένου και στην παράλληλη εκτέλεση. Αυτό αποσκοπεί σε μια ξεκάθαρη σύγκριση των αποτελεσμάτων που θα εκφράζουν τις ποιοτικές διαφορές μεταξύ των στιγμιοτύπων κάτω από κοινές συνθήκες (με ίδια πλατφόρμα και ίδια δεδομένα εισόδου μεταξύ των απλών και τροποποιημένων εκδόσεων). Η εύρεση αυτής της βάσης πρέπει να επικεντρωθεί στην εφαρμογή της πάνω στα χαρακτηριστικά (εδώ συγκεκριμένα τις λέξεις κάθε στιγμιοτύπου κειμένου) τα οποία θα επεξεργάζεται το μοντέλο κάθε υλοποίησης, ώστε είτε να διαφοροποιεί τις υλοποιήσεις μεταξύ τους μέσω επιλογής χαρακτηριστικών, είτε να αναπαριστά τα χαρακτηριστικά στην μορφή που ορίζει η πλατφόρμα που χρησιμοποιείται. Το πρώτο σκέλος αφορά περισσότερο την επιλογή των πιο σχετικών από τα χαρακτηριστικά για την μείωση του θορύβου και

κατ' επέκταση αύξηση της αποτελεσματικότητας ταξινόμησης που μπορεί να διερευνηθεί στο Hadoop, ενώ το δεύτερο αγγίζει πιο τεχνικά θέματα επί της μετατροπής ακατέργαστων δεδομένων σε διανύσματα χαρακτηριστικών για χρήση σε τεχνολογίες όπως το Spark. Αυτές οι δύο πλατφόρμες είναι και αυτές που θα χρησιμοποιηθούν για την ανάπτυξη των εφαρμογών στο πλαίσιο της πειραματικής μελέτης για αυτή την εργασία.

Με τα περιβάλλοντα εκτέλεσης να έχουν οριστεί, το βάρος τώρα πέφτει εξ ολοκλήρου στο κριτήριο που θα διαφοροποιεί τις υλοποιήσεις μεταξύ τους ως προς την αποτελεσματικότητα σε κατηγοριοποίηση και παράλληλη εκτέλεση. Όπως αναφέρθηκε και στην υποενότητα περί εξόρυξης κειμένου, ένα βήμα που μπορεί να εφαρμοστεί στην εξόρυξη κειμένου είναι η επιλογή χαρακτηριστικών, όπου φιλτράρονται τα πιο σχετικά από αυτά στην συλλογή εκπαίδευσης για να δημιουργηθεί το μοντέλο ανεπηρέαστο από χαρακτηριστικά που μπορούν να δράσουν ως θόρυβος. Επικεντρώνοντας λοιπόν (για χάρη μιας πιο στοχευμένης ανάλυσης) στην ταξινόμηση κειμένου που εργάζεται σε επίπεδο εγγράφων, εύκολα συμπεραίνει κανείς ότι μια πρόσθετη βελτίωση κατά την προεπεξεργασία για το μοντέλο θα μπορούσε να είναι η εύρεση των πιο σημαντικών λέξεων που περιέχει κάθε έγγραφο. Μια στρατηγική πάνω σε αυτό θα ήταν να θεωρηθούν ως πιο σημαντικοί οι όροι που εμφανίζονται συχνότερα, όμως αυτή η προσέγγιση μοιάζει αρκετά αφελής μιας και είναι αναμενόμενο σε μια τέτοια περίπτωση οι υπερβολικά κοινές λέξεις (όπως άρθρα σαν τα *ο*/*η*/*το*, σύνδεσμοι όπως το *και*, ή λέξεις που χρησιμοποιούνται αρκετά συχνά σε μία γλώσσα που ονομάζονται και **stop words**) να θεωρηθούν πιο σημαντικές έναντι πιο σπάνιων όρων. Αυτό θα είχε ως αποτέλεσμα την χρήση όρων που δεν περιέχουν απαραίτητα σημαντική (και πιο συγκεκριμένα εδώ συναισθηματική) πληροφορία. Για αυτό αρκετά συχνά επιλέγονται οι πιο σπάνιοι από τους όρους σε ένα σώμα κειμένου, χωρίς αυτό να είναι ούτε εδώ ο κανόνας αφού μόνο η σπανιότητα μιας λέξης δεν μπορεί να χρησιμοποιηθεί σαν κριτήριο. Ένας σπάνιος όρος μπορεί να είναι είτε μία λέξη που δεν συνηθίζεται να χρησιμοποιείται από το ευρύ κοινό σαν εναλλακτική ενός όρου ενταγμένου στην καθομιλουμένη (όπως το *αλεξιβρόχιο* έναντι της *ομπρέλας* ή το *αμφίψωμο* έναντι του *τοστ*), είτε μια τεχνική ορολογία που χρησιμοποιείται ειδικότερα σε έναν τομέα ή θεματική ενότητα (όπως το *όφσαϊντ* ή η *αρμογή*). Η κύρια διαφορά μεταξύ των δύο αυτών μορφών σπάνιων όρων είναι η συγκέντρωση που παρουσιάζουν σε έναν αριθμό εγγράφων μιας συλλογής, με την δεύτερη μορφή όπως είναι αντιληπτό να εμφανίζεται σε περισσότερα έγγραφα από ό,τι η πρώτη, επειδή είναι αρκετά πιθανότερο να εμφανιστεί ένας τέτοιος όρος ξανά μετά την πρώτη του χρήση σε συγκεκριμένες συλλογές κειμένου.

Από τα παραπάνω προκύπτει η ανάγκη για χρήση ενός μέτρου που θα υποδεικνύει την ποσότητα συγκέντρωσης εμφάνισης όρων σε μια συλλογή κατά όρο, ώστε να γίνει στη συνέχεια η επιλογή των πιο σχετικών από αυτούς. Το μέτρο που μπορεί να υποστηρίξει αυτή την ανάγκη ονομάζεται **TFIDF** (Term Frequency - Inverse Document Frequency) και είναι μια αρκετά δημοφιλής στατιστική τεχνική που χρησιμοποιείται στην επιλογή χαρακτηριστικών για την ευρετηρίαση των όρων βάσει της σημαντικότητας που έχουν στο έγγραφο σε συνάρτηση με τα υπόλοιπα έγγραφα στην συλλογή. Βασίζεται στον υπολογισμό μιας τιμής που λειτουργεί σαν **βάρος** για κάθε όρο ανά έγγραφο, συνδυάζοντας την αντίστροφη αναλογία της συχνότητας του όρου στη συλλογή των εγγράφων με το ποσοστό των εγγράφων που εμφανίζεται ο όρος αυτός στην συλλογή.

Ο υπολογισμός αυτού του βάρους για έναν όρο στο TFIDF αποτελείται από δύο μέρη, με το πρώτο να είναι το **TF** (Term Frequency) που εκφράζει την συχνότητα ενός όρου μέσα σε ένα έγγραφο, σύμφωνα με την παρακάτω σχέση.

$$TF_{t,d} = \frac{n_{t,d}}{\sum_i n_{i,d}}$$

Σχέση 2.1 - Σχέση Υπολογισμού Συχνότητας Όρου σε Έγγραφο

Όπου:

- Ως $n_{t,d}$ ορίζεται ο αριθμός των εμφανίσεων του όρου *t* σε ένα έγγραφο *d*.
- Ως $\Sigma_i n_{i,d}$ ορίζεται ο συνολικός αριθμός των όρων που υπάρχουν σε ένα έγγραφο *d* (χρησιμοποιείται εδώ σαν μέθοδος κανονικοποίησης λόγω της διαφοράς μήκους κειμένου από έγγραφο σε έγγραφο, το οποίο συνεπάγεται την εύλογη πιθανότητα ένας όρος να εμφανίζεται περισσότερες φορές σε ένα έγγραφο μεγαλύτερου μήκους).

Το δεύτερο μέρος του υπολογισμού περιέχει το **IDF** (Inverse Document Frequency) που αφορά την αντίστροφη συχνότητα των εγγράφων για τον λογαριασμό ενός όρου. Αυτό εν ολίγοις εκφράζει το πόσο σχετικός ή όχι είναι ένας όρος σταθμίζοντας τον αναλόγως σε ολόκληρη τη συλλογή των εγγράφων. Ο λόγος ύπαρξης του IDF στο TFIDF έγκειται στο ότι ο υπολογισμός των TF βαρών υπονοεί ότι όλοι οι όροι είναι το ίδιο σημαντικοί, οπότε με το IDF διαχωρίζονται οι υπερβολικά κοινοί όροι από τους πιο σπάνιους που φέρουν ενδεχομένως χρήσιμη πληροφορία, μειώνοντας το βάρος στους πρώτους και αυξάνοντας το στους δεύτερους, μέσω της παρακάτω σχέσης.

$$IDF_t = \log(\frac{n}{df_t})$$

Σχέση 2.2 - Σχέση Υπολογισμού Αντίστροφης Συχνότητας Εγγράφων για Όρο *t*

Όπου:

- Ως *n* ορίζεται ο συνολικός αριθμός των εγγράφων στην συλλογή.
- Ως df_i ορίζεται η συχνότητα εγγράφου για τον όρο *t* (δηλαδή υποδηλώνει τον αριθμό των εγγράφων που περιέχουν τον όρο *t*).

Συνδυάζοντας τις δύο παραπάνω σχέσεις με έναν απλό πολλαπλασιασμό, εξάγεται το TFIDF βάρος ενός όρου που αναφέρεται σε ένα έγγραφο όπως φαίνεται πιο αναλυτικά και στη συνέχεια.

$$TFIDF_{t,d} = TF_{t,d} \cdot IDF_t = \frac{n_{t,d}}{\sum_i n_{i,d}} \cdot \log(\frac{n}{df_t})$$

Σχέση 2.3 - Σχέση Υπολογισμού Βάρους TFIDF για Όρο σε Έγγραφο

Εξυπακούεται ότι το μέτρο του TFIDF εξαρτάται σε μεγάλο βαθμό από την ίδια την συλλογή στην οποία χρησιμοποιείται. Αυτό σημαίνει ότι η αποτελεσματικότητα για την εύρεση των πιο σχετικών χαρακτηριστικών είναι ανάλογη της ποιότητας των συνόλων εισόδου. Μια τέτοια συνθήκη θα μπορούσε να θεωρηθεί αποτρεπτική σε περιπτώσεις που τα δεδομένα εισόδου δεν βρίσκονται εντελώς υπό τον ποιοτικό έλεγχο του χρήστη, όμως η εσωτερική προσαρμογή των βαρών του TFIDF βάσει των ίδιων των δεδομένων κάνει ιδιαίτερα ελκυστική τη χρήση τους. Ένας ακόμη λόγος υπέρ της χρήσης του μέτρου του TFIDF είναι η ευχέρεια αναπαράστασης των χαρακτηριστικών στα έγγραφα ως **διανύσματα χαρακτηριστικών**, σε μια μορφή που μπορεί να αξιοποιήσει μια υλοποίηση σαν το Spark (όπως αναφέρθηκε παραπάνω και θα φανεί αργότερα κατά την περιγραφή των υλοποιήσεων).

Μένοντας στο τελευταίο σκέλος, έχει ενδιαφέρον να σχολιαστεί η χρήση του TFIDF έναντι άλλων επιλογών που μπορούσαν να πάρουν την θέση του μέσα στους σκοπούς της εργασίας. Σε ό,τι αφορά την επεξεργασία των δεδομένων εισόδου σε μια προσπάθεια βελτίωσης της επίδοσης στην κατηγοριοποίηση, δόθηκε προηγουμένως έμφαση στο κριτήριο του πόσο σχετικός ή όχι είναι ένας όρος σε ένα έγγραφο συναρτήσει ολόκληρης της συλλογής εγγράφων. Αναγνωρίστηκε ότι υπερβολικά κοινές λέξεις δεν μπορούν να φέρουν ίσο ποσοστό πληροφορίας με τις πιο σπάνιες. Για αυτό μια πιο απλή πρακτική επί της επιλογής του πιο ουσιώδη όρου στο φιλτράρισμα των χαρακτηριστικών θα μπορούσε να θεωρηθεί η **χρήση μιας λίστας stop words λέξεων**, όπου κάθε όρος σε αυτή θα αγνοείται ενώ οι υπόλοιποι κρατούνται για την δημιουργία του μοντέλου. Η απλότητα όμως μιας τέτοιας λίστας κρύβει στο εννοιολογικό του πλαίσιο την παραδοχή ότι τα δεδομένα εισόδου περιέχουν είτε κείμενο σε μία συγκεκριμένη γλώσσα (όπου μια λίστα με αγγλικά stop words θα είχε εφαρμογή μόνο πάνω σε λέξεις και κείμενα γραμμένα στην αγγλική γλώσσα), είτε κείμενο τέτοιου περιεχομένου ώστε η λίστα να λειτουργεί επαρκώς (μιας και ένα κείμενο ιατρικού περιεχομένου έχει διαφορετικό λεξιλόγιο από μία ανάρτηση ενός μέσου χρήστη στο διαδίκτυο). Τέτοια ζητήματα μπορούν φυσικά να αντιμετωπιστούν με την παραμετροποίηση της λίστας πάνω στο ίδιο το περιεχόμενο των δεδομένων εισόδου, όμως στο πιο πρακτικό επίπεδο υλοποίησης και χρήσης αυτής σε ένα κατανεμημένο σύστημα, δημιουργούνται ερωτηματικά για την αποτελεσματικότητα της. Η λίστα βεβαίως μπορεί να αποθηκευτεί σε ένα κατανεμημένο σύστημα αρχείων όπως το HDFS, όμως η εξέταση κάθε όρου για το αν μπορεί να παραληφθεί ή όχι στην δημιουργία του μοντέλου μεταφράζεται στην σάρωση της λίστας ξανά και ξανά. Δηλαδή το πρόγραμμα θα υποχρεούται να ανακτά στοιχεία από τον δίσκο τόσες φορές όσες είναι και οι λέξεις στην είσοδο του, διαβάζοντας την λίστα πολλές φορές και μάλιστα παράλληλα μέσα από τα διαφορετικά στιγμιότυπα που τρέχουν σε διαφορετικά μηχανήματα, το οποίο συνεπάγεται μεγάλες καθυστερήσεις στο χρόνο εκτέλεσης. Αν από την άλλη επιλεγεί η φόρτωση της λίστας στην κύρια μνήμη για να αποφευχθούν οι επαναλαμβανόμενες αναγνώσεις στο δίσκο, τίθεται στο τραπέζι το θέμα του μήκους μιας τέτοιας λίστας με σκοπό να απομείνουν αρκετοί πόροι που θα εξασφαλίζουν την εύρυθμη λειτουργία κάθε κόμβου στη συστοιχία. Τέτοιες επιπρόσθετες προσπάθειες για την διαχείριση μιας λίστας που εγγυάται μόνο απλότητα στο σχεδιασμό αλλά βαριές επιπλοκές στο λειτουργικό κομμάτι μιας υλοποίησης, κάνουν την χρήση την να μην μοιάζει τόσο φρόνιμη.

Στα της αναπαράστασης των χαρακτηριστικών σε μορφή διανύσματος, μια τεχνική που αξίζει να αναφερθεί είναι αυτή των **n-grams** όπου ένας αριθμός λέξεων (ορισμένος από τον χρήστη) που βρίσκονται σε αλληλουχία σε ένα κείμενο θεωρείται ως ένα χαρακτηριστικό. Με αυτό τον τρόπο μία εφαρμογή που λειτουργεί με 1-gram αναπαριστά κάθε έγγραφο με χαρακτηριστικά τις λέξεις που το αποτελούν, ενώ μια εφαρμογή που λειτουργεί με 2-grams σχηματίζει τα χαρακτηριστικά ως ζεύγη λέξεων που βρίσκονται από την αρχή του κειμένου, και ούτω καθεξής. Ακολουθεί σε γενικές γραμμές το μοντέλο του **bag of words** όπου δεν συγκρατείται η ακριβής σειρά όλου του κειμένου σε ένα έγγραφο. Σε αυτη την αναπαράσταση τα βάρη στα n-grams που εμφανίζονται πιο σπάνια είναι προφανώς μεγαλύτερα από τα υπόλοιπα που περιέχουν πιο κοινές λέξεις, επηρεάζοντας ανάλογα την ταξινόμηση των στιγμιοτύπων σε κλάσεις. Αρκετά συχνά προτιμούνται οι υλοποιήσεις των 2-grams και 3-grams για καλύτερη ανίχνευση πληροφορίας μέσα στο κείμενο έναντι μεγαλύτερου βαθμού n-grams, μιας και τα τελευταία θα έχουν σαν αποτέλεσμα την δημιουργία μεγάλου αριθμού χαρακτηριστικών για την αναπαράσταση κάθε εγγράφου. Παρόλα αυτά κάτι τέτοιο δεν είναι καθόλου απίθανο να συμβεί και στις πιο συμβατικές εκδοχές αυτής της τεχνικής, αφού ενδέχεται να συντηρούνται διαστάσεις για χαρακτηριστικά τα οποία εμφανίζονται σπάνια. Επιπροσθέτως η επιλογή της εκδοχής που θα χρησιμοποιηθεί μπορεί να είναι στην ευχέρεια του χρήστη ώστε να καταλήξει εμπειρικά σε μία από αυτές, όμως επηρεάζει ευθέως τον τρόπο με τον οποίο θα εξαχθεί οποιαδήποτε χρήσιμη ή μη πληροφορία από μια συλλογή εγγράφων αφού η διαίρεση ενός εγγράφου σε αυθαίρετα κομμάτια ίσου μήκους δεν εγγυάται απαραίτητα την επιθυμητή σταχυολόγηση σε όρους που

εμφανίζονται αρκετές φορές με διαφορετικές περικλείουσες λέξεις. Λόγω αυτής της εξάρτησης από την σειρά με την οποία εμφανίζονται οι όροι σε ένα έγγραφο και παρά την ευκολία που προσφέρει στον σχεδιασμό και την εφαρμογή του στην εξαγωγή χαρακτηριστικών, τα n-grams εγείρουν ερωτήματα για το αν μπορούν να υποστηρίξουν μια υλοποίηση γενικού σκοπού πάνω σε δεδομένα που ενδεχομένως διαφέρουν σε δομή ή ύφος μεταξύ τους.

Κλείνοντας την περιγραφή της λογικής που χαρακτηρίζει τις ακόλουθες υλοποιήσεις, ιδιαίτερα σημαντικό είναι να υπάρχει μια προεργασία πάνω στα συστατικά που αποτελούν ένα σώμα κειμένου χωρίς να φέρουν κάποια πληροφορία ώστε να θεωρηθούν χαρακτηριστικά προς μελέτη. Όροι όπως σημεία στίξης, υπερσύνδεσμοι, και αριθμοί μπορούν να ανιχνευθούν και να παραληφθούν κατά την μετατροπή των εγγράφων σε διανύσματα για να μην επιβαρυνθεί το μοντέλο από μεγάλο μέγεθος διαστάσεων προς διαχείριση με χαρακτηριστικά που κατά πάσα πιθανότητα θα οξύνουν τον θόρυβο και θα επηρεάσουν αρνητικά την τα αποτελέσματα ενός μοντέλου κατηγοριοποίησης.

2.1 Υλοποιήσεις στο Hadoop

Σαν μια συνέχεια της υποενότητας για το Hadoop στο πρώτο μέρος της εργασίας, εδώ η περιγραφή μιας εφαρμογής αγγίζει περισσότερες τεχνικές λεπτομέρειες και διαδικαστικές λειτουργίες για την ανάπτυξη των ακόλουθων προγραμμάτων στην εν λόγω πλατφόρμα.

Ο τρόπος διάρθρωσης μιας εφαρμογής η οποία βασίζεται στο περιβάλλον του Hadoop επιτρέπει την χρήση **καθολικών μετρητών** (global counters) στο εσωτερικό του προγράμματος. Ένας τέτοιος μηχανισμός είναι ιδιαίτερα χρήσιμος αλλά και κρίσιμος από πλευράς ορισμού και λειτουργίας μιας παράλληλης εκτέλεσης ενός αλγορίθμου, προσφέροντας επίσης τη δυνατότητα υπολογισμού της επίδοσης του στην ταξινόμηση των δειγμάτων. Το τελευταίο γίνεται εφικτό υπολογίζοντας τα στοιχεία που αποτελούν την μήτρα σύγχυσης και αναφέρονται επιγραμματικά παρακάτω για την βασική περίπτωση της δυαδικής κατηγοριοποίησης (θεωρώντας δύο κατηγορίες, θετική και αρνητική πολικότητα εγγράφου) η οποία και θα μελετηθεί από εδώ και πέρα.

Μετρητής	Περιγραφή
True Positive	Ο αριθμός των στιγμιοτύπων ελέγχου που ταξινομήθηκε ορθά στην θετική κατηγορία.
False Positive	Ο αριθμός των στιγμιοτύπων ελέγχου που ταξινομήθηκε λανθασμένα στην θετική κατηγορία.
True Negative	Ο αριθμός των στιγμιοτύπων ελέγχου που ταξινομήθηκε ορθά στην αρνητική κατηγορία.
False Negative	Ο αριθμός των στιγμιοτύπων ελέγχου που ταξινομήθηκε λανθασμένα στην αρνητική κατηγορία.

Πίνακας 2.1 - Καθολικοί Μετρητές για την Μήτρα Σύγχυσης Δυαδικής Κατηγοριοποίησης

Για την ανάπτυξη μιας εφαρμογής κατηγοριοποίησης κειμένου (με απώτερο σκοπό εδώ φυσικά την ανάλυση συναισθημάτων) στην πλατφόρμα του Hadoop επιλέχθηκε να υλοποιηθεί ο αλγόριθμος του Naïve Bayes, όπως αυτός μελετήθηκε και στην αντίστοιχη υποενότητα του πρώτου μέρους, λόγω της ιδιαίτερα απλής διαδικασίας για την δημιουργία ενός πιθανοτικού μοντέλου ταξινόμησης.

2.1.1 Naïve Bayes

Δίνοντας βάση σε όσα αναφέρθηκαν και στην πρώτη ενότητα της εισαγωγής, προκύπτει το συμπέρασμα ότι μια εφαρμογή κατηγοριοποίησης υπάγεται στο γενικότερο πεδίο της μηχανικής μάθησης, με κατανοητή την ανάγκη δημιουργίας δύο λειτουργιών εκπαίδευσης και ελέγχου για το μοντέλο της. Οι λειτουργίες αυτές είναι διακριτές όσον αφορά τα δεδομένα που λαμβάνουν στην είσοδο όμως περιέχουν την κατά κανόνα αλληλουχία μεταξύ τους, με το μοντέλο που δημιουργείται στην φάση της εκπαίδευσης να εφαρμόζεται στη συνέχεια πάνω στα δεδομένα ελέγχου της δεύτερης φάσης.

Με μια επιπλέον ανάγνωση στα βασικά στοιχεία του τρόπου λειτουργίας του Hadoop γίνεται ξεκάθαρη η ανάγκη τήρησης του οδηγού πάνω στον οποίο βασίζεται για εκφραστεί η επεξεργασία που περιέχει η μέθοδος του Naïve Bayes με όρους μεθόδων Map και Reduce. Σύμφωνα με αυτή τη μέθοδο, πέρα από τις πιθανότητες ενός χαρακτηριστικού για κάθε κατηγορία, χρησιμοποιούνται οι πιθανότητες εκ των προτέρων και οι πιθανότητες υπό συνθήκη για κάθε χαρακτηριστικό. Οι πιθανότητες αυτές, που αποτελούν τον πυρήνα του μοντέλου του Naïve Bayes, χρησιμοποιούνται φυσικά για τον υπολογισμό της τελικής υπό συνθήκη πιθανότητας κάθε κατηγορίας ώστε εν τέλει να επιλεχθεί εκείνη με την μεγαλύτερη τιμή για την πρόβλεψη της ετικέτας ενός δείγματος στην συλλογή ελέγχου. Για την διαχείριση και τον υπολογισμό αυτών των πιθανοτήτων σε μια εφαρμογή που αποτελείται αυστηρά από δύο τύπους συναρτήσεων και εκτείνεται σε δύο φάσεις, ενθαρρύνεται η χρήση επιπρόσθετων καθολικών μετρητών που θα είναι προσβάσιμοι για ανάγνωση και επεξεργασία σε όλο το εύρος της υλοποίησης. Οι μετρητές που χρειάζονται στον αλγόριθμο εδώ είναι οι ακόλουθοι.

Μετρητής	Περιγραφή
Πλήθος Δειγμάτων	Ο αριθμός όλων των στιγμιοτύπων που περιέχει το σύνολο εκπαίδευσης.
Πλήθος Δειγμάτων Κατηγορίας *y*	Ο αριθμός των στιγμιοτύπων στο σύνολο εκπαίδευσης που ανήκουν στην κατηγορία *y*.
Πλήθος Λέξεων Κατηγορίας *y*	Ο αριθμός των όρων που υπάρχουν σε στιγμιότυπα τα οποία ανήκουν στην κατηγορία *y*.
Πλήθος Χαρακτηριστικών	Ο αριθμός των χαρακτηριστικών από τα οποία αποτελείται το μοντέλο ταξινόμησης.

Πίνακας 2.2 - Καθολικοί Μετρητές για την Υλοποίηση του Naïve Bayes στο Hadoop

2.1.1.1 Εκπαίδευση Μοντέλου

Θέτοντας ως παραδοχή ότι η συλλογή εκπαίδευσης βρίσκεται σε μορφή ενός ή περισσότερων αρχείων κειμένου τα οποία η εφαρμογή θα σαρώνει παράλληλα γραμμή προς γραμμή, μένει να βρεθεί η δομή κλειδιού-τιμής που θα δοθεί σαν αποτέλεσμα με το πέρας της φάσης της εκπαίδευσης. Με τους μετρητές που έχουν οριστεί να αναλαμβάνουν την καταμέτρηση γενικών παραγόντων που χρησιμοποιούνται μετέπειτα για τον υπολογισμό πιθανοτήτων, το βάρος πέφτει στην επεξεργασία των χαρακτηριστικών που θα εξαχθούν για να αποτιμηθούν οι υπό συνθήκη πιθανότητες $P(x_i|y)$ κάθε ενός από αυτά και για κάθε κατηγορία. Η υπό συνθήκη πιθανότητα $P(x_i|y)$ εκφράζεται με την παρακάτω σχέση (εφαρμόζοντας εδώ και την εξομάλυνση Laplace).

$$P(x_i|y) = \frac{\sum x_i, y + 1}{\sum x, y \cdot |V|}$$

Σχέση 2.4 - Πιθανότητα Υπό Συνθήκη ένα Χαρακτηριστικό x_i να Βρίσκεται σε Έγγραφο Κλάσης y

Όπου:

- Ως $\Sigma x_i,y$ εκφράζεται το πλήθος των εμφανίσεων του χαρακτηριστικού x_i στα έγγραφα κλάσης y του συνόλου εκπαίδευσης.
- Ως $\Sigma x,y$ εκφράζεται το πλήθος των λέξεων σε στιγμιότυπα κλάσης y.
- Ως $|V|$ εκφράζεται το συνολικό πλήθος των χαρακτηριστικών στο σύνολο εκπαίδευσης.

Οι δύο τελευταίοι συντελεστές της σχέσης υπολογίζονται από τους αντίστοιχους καθολικούς μετρητές που έχουν οριστεί εξ αρχής. Αυτό πρακτικά σημαίνει ότι στη φάση της εκπαίδευσης το πρόγραμμα μετράει για κάθε χαρακτηριστικό το πλήθος των εμφανίσεων του σε έγγραφα κάθε κλάσης ξεχωριστά, ή αλλιώς ότι τα ζευγάρια κλειδιού-τιμής που επιστρέφονται θα έχουν ως κλειδί ένα χαρακτηριστικό και σαν τιμή τον αριθμό εμφανίσεων του σε κάθε κατηγορίας.

Η συνάρτηση Map εδώ υποδέχεται τα δεδομένα και σαρώνει τα αρχεία εισόδου γραμμή προς γραμμή (όπου κάθε γραμμή θεωρείται ως ένα έγγραφο). Κατά την ανάγνωση κάθε εγγράφου κρατείται η ετικέτα κατηγορίας του για την συνολική καταμέτρηση του πλήθους των εγγράφων που ανήκουν σε κάθε κατηγορία. Έπειτα το κείμενο φιλτράρεται ώστε να μετατραπούν όλοι οι χαρακτήρες σε πεζούς και να μην συμπεριλαμβάνονται όροι όπως υπερσύνδεσμοι, αριθμοί, σημεία στίξης, και άλλα στοιχεία. Στη συνέχεια διασπάται το κείμενο σε ένα διάνυσμα λέξεων (του οποίου το μήκος προστίθεται στον αντίστοιχο μετρητή), με κάθε χαρακτηριστικό να γίνεται κλειδί σε ένα ζεύγος κλειδιού-τιμής που φέρει ως τιμή την ετικέτα κατηγορίας στην οποία αναφέρεται.

```
Map(σύνολο_εκπαίδευσης)
{
    ΔΕΙΓΜΑΤΑ++
    ετικέτα_κατηγορίας = γραμμή[1]
    κείμενο = γραμμή[2].φιλτράρισμα_όρων()

    αν(ετικέτα_κατηγορίας == “Θετικό”)
    {
        ΘΕΤΙΚΑ_ΔΕΙΓΜΑΤΑ++
        ΘΕΤΙΚΕΣ_ΛΕΞΕΙΣ += κείμενο.διαχωρισμός_όρων(“ “).μήκος
    }
    αλλιώς
    {
        ΑΡΝΗΤΙΚΑ_ΔΕΙΓΜΑΤΑ++
        ΑΡΝΗΤΙΚΕΣ_ΛΕΞΕΙΣ += κείμενο.διαχωρισμός_όρων(“ “).μήκος
    }

    για(κάθε λέξη στο κείμενο.διαχωρισμός_όρων(“ “))
        γράψε(λέξη, ετικέτα κατηγορίας)
}
```

Αλγόριθμος 2.1 - Συνάρτηση Map για την Εκπαίδευση του Naïve Bayes μοντέλου

Στη συνάρτηση Reduce για την φάση της εκπαίδευσης, τα ζευγάρια κλειδιού-τιμής που δημιουργήθηκαν από την Map συνάρτηση συναθροίζονται εσωτερικά από το Hadoop κατά κλειδί (δηλαδή κάθε λέξη φέρει σαν τιμές όλα τα στιγμιότυπα ετικετών κατηγορίας). Κάθε reducer αναλαμβάνει κατά τα γνωστά ένα κλειδί για να επεξεργαστεί τις τιμές του, με τον καθολικό μετρητή για το πλήθος των χαρακτηριστικών στο μοντέλο να προσαυξάνεται ανάλογα. Βάσει του αποτελέσματος στο οποίο στοχεύει η εκπαίδευση, σε αυτό το βήμα αρκεί να καταμετρηθούν οι εμφανίσεις κάθε ετικέτας στο σύνολο των τιμών. Το πλήθος εμφανίσεων κάθε ετικέτας εκφράζει το πλήθος εμφάνισης αυτού του χαρακτηριστικού σε έγγραφα της κατηγορίας με την εκάστοτε ετικέτα.

```
Reduce(κλειδί, τιμές[])
{
      ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ++

      θετικές_εμφανίσεις = 0
      αρνητικές_εμφανίσεις = 0

      για(κάθε τιμή στο τιμές[])
      {
            αν(τιμή == "Θετικό")
                  θετικές_εμφανίσεις++
            αλλιώς
                  αρνητικές_εμφανίσεις++
      }

      γράψε(κλειδί, (θετικές_εμφανίσεις, αρνητικές_εμφανίσεις))
}
```

Αλγόριθμος 2.2 - Συνάρτηση Reduce για την Εκπαίδευση του Naïve Bayes μοντέλου

2.1.1.2 Έλεγχος Μοντέλου

Εφόσον στον προηγούμενο κύκλο εργασίας υπολογίστηκαν αποσπασματικά όλοι οι παράγοντες που απαρτίζουν τις πιθανότητες που αποτελούν το μοντέλο, στον επόμενο κύκλο εργασίας απομένει ο υπολογισμός των πιθανοτήτων ταξινόμησης εγγράφων για κάθε κλάση έτσι ώστε να επιλεγεί για κάθε έγγραφο η ετικέτα που εμφανίζει την μέγιστη από αυτές.

Πριν την σάρωση της συλλογής ελέγχου από τους mappers ελέγχου, είναι απαραίτητη η ανάγνωση των αποτελεσμάτων εκπαίδευσης του προηγούμενου κύκλου και ο υπολογισμός των συνιστωσών που αποτελούν την σχέση από την οποία προκύπτουν οι πιθανότητες ταξινόμησης για κάθε κατηγορία. Το Hadoop προσφέρει την δυνατότητα για την κλήση μιας συνάρτησης **Setup** πριν την εκτέλεση της συνάρτησης Map από κάθε mapper, η οποία χρησιμοποιείται εδώ ώστε να μην επιβαρυνθεί το πρόγραμμα από μια ενδεχομένως κοστοβόρα σειριακή διαδικασία υπολογισμών των πιθανοτήτων. Σε αυτή την συνάρτηση προετοιμασίας υπολογίζονται οι εκ των προτέρων πιθανότητες κάθε κατηγορίας, σαρώνονται τα αποτελέσματα του προηγούμενου κύκλου εργασίας σε δύο δομές λεξικού για τα χαρακτηριστικά των δύο κλάσεων (με κλειδί το ίδιο το χαρακτηριστικό και τιμή το πλήθος εμφανίσεων σε έγγραφα της αντίστοιχης κλάσης), και μέσω αυτών υπολογίζονται οι υπό συνθήκη πιθανότητες για κάθε χαρακτηριστικό που βρίσκεται σε έγγραφα κάθε κατηγορίας για να αποθηκευθούν σε δύο νέες δομές λεξικού (με κλειδί το χαρακτηριστικό και τιμή την υπό συνθήκη πιθανότητα).

```
Setup()
{
	πιθανότητα_θετικής_κλάσης = ΘΕΤΙΚΑ_ΔΕΙΓΜΑΤΑ / ΔΕΙΓΜΑΤΑ

	πιθανότητα_αρνητικής_κλάσης = ΑΡΝΗΤΙΚΑ_ΔΕΙΓΜΑΤΑ / ΔΕΙΓΜΑΤΑ

	για(κάθε αρχείο στο αποτελέσματα_φάσης_εκπαίδευσης)
	{
		για(κάθε γραμμή στο αρχείο)
		{
			θετικά_χαρακτηριστικά[].πρόσθεσε(γραμμή[0], γραμμή[1])
			αρνητικά_χαρακτηριστικά[].πρόσθεσε(γραμμή[0], γραμμή[2])
		}
	}

	για(κάθε στοιχείο στις λίστες θετικών και αρνητικών χαρακτηριστικών)
	{
		πιθανότητες_θετ_χαρακτηριστικών[].πρόσθεσε(στοιχείο.κλειδί, στοιχείο.τιμή + 1 /
				(ΘΕΤΙΚΕΣ_ΛΕΞΕΙΣ  + ΔΕΙΓΜΑΤΑ))

		πιθανότητες_αρν_χαρακτηριστικών[].πρόσθεσε(στοιχείο.κλειδί, στοιχείο.τιμή + 1 /
				(ΑΡΝΗΤΙΚΕΣ_ΛΕΞΕΙΣ  + ΔΕΙΓΜΑΤΑ))
	}
}
```

Αλγόριθμος 2.3 - Συνάρτηση Setup για τον Έλεγχο του Naïve Bayes μοντέλου

Με την εκτέλεση της συνάρτησης Setup το πρόγραμμα έχει υπολογίσει τις δύο συνιστώσες που χρειάζονται να πολλαπλασιαστούν για τον υπολογισμό της πιθανότητας ταξινόμησης ενός εγγράφου για κάθε κλάση. Πάνω σε αυτά τα θεμέλια προχωρά η υλοποίηση μέσω μιας συνάρτησης Map όπου αντίστοιχα με την Map της εκπαίδευσης δέχεται σαν είσοδο αρχεία κειμένου και τα σαρώνει γραμμής προς γραμμή για να απομονώσει τα στοιχεία κάθε εγγράφου. Εξάγει το αναγνωριστικό του εγγράφου, το κείμενο του, καθώς και την ετικέτα κατηγορίας. Στη συνέχεια φιλτράρει το κείμενο κάθε εγγράφου από όρους χωρίς συναισθηματική πληροφορία και διασπά το φιλτραρισμένο κείμενο που προέκυψε σε ένα διάνυσμα λέξεων. Ύστερα, βάσει της σχέσης της πιθανότητας ταξινόμησης, υπολογίζεται το γινόμενο όλων των πιθανοτήτων των χαρακτηριστικών (από το λεξιλόγιο του μοντέλου που παρουσιάζονται στο έγγραφο) για κάθε κλάση, και η ποσότητα που προκύπτει πολλαπλασιάζεται με την εκ των προτέρων πιθανότητα κάθε κλάσης. Οι δύο πιθανότητες συγκρίνονται μεταξύ τους και η μέγιστη αυτών καταδεικνύει την πρόβλεψη ετικέτας κατηγορίας του μοντέλου για το συγκεκριμένο έγγραφο. Εν τέλει συγκρίνεται η ετικέτα κατηγορίας του εγγράφου με την πρόβλεψη του μοντέλου για την εξέταση της αποτελεσματικότητας του μοντέλου (βασιζόμενη στην στατιστική καταμέτρηση των καθολικών μετρητών για το πλήθος των εγγράφων που κατηγοριοποιήθηκαν ορθά ή μη) και εγγράφονται στην έξοδο τα ζευγάρια κλειδιού-τιμής που φέρουν ως κλειδί το αναγνωριστικό του εγγράφου μαζί με το κείμενο του και ως τιμή την ετικέτα κατηγορίας που προέβλεψε ο Naïve Bayes αλγόριθμος.

```
Map(σύνολο_ελέγχου)
{
	αναγνωριστικό_εγγράφου = γραμμή[0]
	ετικέτα_κατηγορίας = γραμμή[1]
	κείμενο = γραμμή[2].φιλτράρισμα_όρων()
```

```
    θετική_πιθανότητα = 1
    αρνητική_πιθανότητα = 1

    όροι[] = κείμενο.διαχωρισμός_όρων(“ “)

    για(κάθε λέξη στη λίστα των όρων)
    {
        για(κάθε στοιχείο από τις λίστες θετικών και αρνητικών χαρακτηριστικών)
        {
                θετική_πιθανότητα *= θετικά_χαρακτηριστικά.ανάκτησε(λέξη)
                αρνητική_πιθανότητα *= αρνητικά_χαρακτηριστικά.ανάκτησε(λέξη)
        }
    }

    θετική_πιθανότητα *= πιθανότητα_θετικής_κλάσης
    αρνητική_πιθανότητα *= πιθανότητα_αρνητικής_κλάσης

    αν(θετική_πιθανότητα > αρνητική_πιθανότητα)
    {
        αν(ετικέτα_κατηγορίας == “Θετικό”)
            TRUE_POSITIVE++
        αλλιώς
             FALSE_POSITIVE++

        γράψε((αναγνωριστικό_εγγράφου, κείμενο), “Θετικό”)
    }
    αλλιώς
    {
        αν(ετικέτα_κατηγορίας == “Αρνητικό”)
            TRUE_NEGATIVE++
        αλλιώς
            FALSE_POSITIVE++

        γράψε((αναγνωριστικό_εγγράφου, κείμενο), “Αρνητικό”)
    }
}
```

Αλγόριθμος 2.4 - Συνάρτηση Map για τον Έλεγχο του Naïve Bayes μοντέλου

Εφόσον έχουν προκύψει τα αποτελέσματα και οι μετρικές που χρειάζονται σε αυτό το σημείο, δεν υπάρχει κάποια περαιτέρω λειτουργία που χρειάζεται να αναπτυχθεί για την συνάθροιση των ζευγαριών κλειδιών-τιμών της συνάρτησης Map του ελέγχου κατά κλειδί, επομένως δεν υπάρχει λόγος να υλοποιηθεί για αυτή τη φάση κάποια συνάρτηση Reduce.

Κύκλος Εργασίας	Φάση Κύκλου Εργασίας	Δομή Ζευγαριών Κλειδιού-Τιμής	Περιγραφή Φάσης
Εκπαίδευση Μοντέλου	Map	**Είσοδος** **Κλειδί:** θέση δείκτη αρχείου **Τιμή:** γραμμή αρχείου **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** ετικέτα κλάσης	● Υποδοχή συνόλου εκπαίδευσης. ● Φιλτράρισμα κειμένου εγγράφου εκπαίδευσης. ● Διάσπαση κειμένου σε λίστα λέξεων. ● Συσχέτιση κάθε λέξης με την ετικέτα κλάσης του εγγράφου στο οποίο εμφανίστηκε.
	Reduce	**Είσοδος** **Κλειδί:** λέξη **Τιμή:** ετικέτα κλάσης **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** (θετικές εμφανίσεις, αρνητικές εμφανίσεις)	● Υπολογισμός πλήθους εμφανίσεων κάθε λέξης σε έγγραφα κάθε κλάσης. ● Συσχέτιση κάθε λέξης με τις εμφανίσεις της σε κάθε κλάση. ● Προετοιμασία συντελεστών για την κατασκευή του μοντέλου.
Έλεγχος Μοντέλου	Map	**Είσοδος** **Κλειδί:** θέση δείκτη αρχείου **Τιμή:** γραμμή αρχείου **'Εξοδος** **Κλειδί:** (έγγραφο, κείμενο) **Τιμή:** ετικέτα κλάσης	● Εφαρμογή συντελεστών για την ρύθμιση του μοντέλου. ● Υποδοχή συνόλου ελέγχου. ● Φιλτράρισμα κειμένου εγγράφου ελέγχου. ● Διάσπαση κειμένου σε μια λίστα λέξεων. ● Υπολογισμός πιθανοτήτων για το ενδεχόμενο ένα έγγραφο να ανήκει σε κάθε κλάση. ● Ορισμός ετικέτας κλάσης με την μεγαλύτερη πιθανότητα για τον λογαριασμό κάθε εγγράφου.
	Reduce	Δεν ορίζεται	Δεν ορίζεται

Πίνακας 2.3 - Διάγραμμα Κύκλων Εργασίας Υλοποίησης Naïve Bayes στο Hadoop

2.1.2 Τροποποιημένη Έκδοση Naïve Bayes

Η προηγούμενη υλοποίηση του Naïve Bayes κυμαίνεται στον βασικό τρόπο λειτουργίας του αλγορίθμου, αυτή όμως η προσέγγιση φέρεται σε όλα τα χαρακτηριστικά με τον ίδιο ακριβώς τρόπο, με έναν υπερβολικά κοινό όρο να θεωρείται ισάξιος ενός πιο σπάνιου και ενδεχομένως πιο σχετικού όρου. Είναι σαφές ότι υπάρχει έδαφος για μία προσπάθεια βελτίωσης του μοντέλου επιλέγοντας στοχευμένα τα χαρακτηριστικά τα οποία θα αποτελέσουν το λεξιλόγιο του, για μια πιο στοχευμένη εφαρμογή του πάνω στα δεδομένα εισόδου.

Μελετώντας τις μεθόδους επιλογής χαρακτηριστικών, φαίνεται πως υπάρχει η δυνατότητα συνδυασμού του μοντέλου από το Naïve Bayes με το μέτρο του TFIDF που εφαρμόζεται στην κλίμακα των χαρακτηριστικών ανά έγγραφο. Μια τέτοια προσπάθεια βελτίωσης περιλαμβάνει την επιλογή των πιο σημαντικών χαρακτηριστικών σε κλίμακα εγγράφων με παράλληλο τρόπο, και την μετέπειτα κατασκευή και εφαρμογή του μοντέλου πάνω στο σύνολο ελέγχου που διέπεται από την ίδια λογική που ακολουθείται στην απλή έκδοση του Naïve Bayes.

Αντίστοιχα με την προηγούμενη υλοποίηση αξιοποιείται κι εδώ ο μηχανισμός των καθολικών μετρητών για την αποδοτική επικοινωνία κάθε mapper ή reducer, με κάποιες ποσότητες να έχουν κομβικό χαρακτήρα στην επεξεργασία που επιτελείται σε μια σειρά κύκλων εργασιών που συνδέονται αλυσιδωτά μεταξύ τους. Οι μετρητές αυτοί παρουσιάζονται στον πίνακα που ακολουθεί.

Μετρητής	Περιγραφή
Πλήθος Δειγμάτων	Ο αριθμός όλων των στιγμιοτύπων που περιέχει το σύνολο εκπαίδευσης, πριν την διαδικασία επιλογής δειγμάτων.
Πλήθος Χαρακτηριστικών	Ο αριθμός των χαρακτηριστικών από τα οποία αποτελείται το μοντέλο ταξινόμησης.
Πλήθος Νέων Δειγμάτων	Ο αριθμός των στιγμιοτύπων του συνόλου εκπαίδευσης που απέμειναν από την επιλογή των πιο σχετικών όρων και την ανασυγκρότηση των στιγμιοτύπων.
Πλήθος Δειγμάτων Κατηγορίας *y*	Ο αριθμός των στιγμιοτύπων στο νέο σύνολο εκπαίδευσης που ανήκουν στην κατηγορία *y*.
Πλήθος Λέξεων Κατηγορίας *y*	Ο αριθμός των όρων που υπάρχουν σε στιγμιότυπα τα οποία ανήκουν στην κατηγορία *y*.

Πίνακας 2.4 - Καθολικοί Μετρητές για την Υλοποίηση του Τροποποιημένου Naïve Bayes στο Hadoop

2.1.2.1 Καταμέτρηση Όρων

Το πρώτο βήμα για την κατασκευή μιας λειτουργίας που υπολογίζει βαθμολογίες TFIDF είναι η στοιχειώδης και απλή λειτουργία της καταμέτρησης εμφανίσεων κάθε λέξης σε κάθε έγγραφο. Μέσω αυτού του κύκλου εργασίας καταμετρώνται οι εμφανίσεις των λέξεων και σαρώνεται κάθε έγγραφο ξεχωριστά συγκρατώντας το αναγνωριστικό του και το κείμενο που περιέχει. Αντίστοιχα με τον πρώτο κύκλο εργασίας στην προηγούμενη υλοποίηση, κατά την υποδοχή κάθε εγγράφου στην συνάρτηση Map οι όροι φιλτράρονται σε μια προσπάθεια προεπεξεργασίας και εξάγονται οι όροι που θα στοιχειοθετηθούν μαζί με το αναγνωριστικό εγγράφου (έτσι ώστε να είναι γνωστό σε ποιο έγγραφο γίνεται η καταμέτρηση των εμφανίσεων του όρου) σαν ένα σύνθετο κλειδί στο ζεύγος κλειδιού-τιμής που θα σχηματιστεί. Εφόσον οι mappers έχουν σαν σκοπό τον ορισμό των ζευγαριών που θα δωθούν στους reducers για τον υπολογισμό της καταμέτρησης, η αρχικοποιημένη τιμή κάθε ζευγαριού

κλειδιού-τιμής θα είναι η μονάδα, μιας και το πρόγραμμα γνωρίζει πως ο όρος θα βρίσκεται τουλάχιστον μία φορά σε αυτό το έγγραφο.

Ιδιαίτερη σημασία εδώ έχει η αποθήκευση της ετικέτας κατηγορίας κάθε εγγράφου. Μία επιλογή αποθήκευσης αυτής της πληροφορίας μπορεί να είναι η εγγραφή των αναγνωριστικών των εγγράφων που ανήκουν σε μία από τις κλάσεις σε ένα αρχείο που θα σαρώνεται όταν στην κατασκευή του μοντέλου πρέπει να αποσαφηνίζεται σε ποια κατηγορία ανήκει κάθε έγγραφο. Όμως η ανάγκη σάρωσης του αρχείου για λογαριασμό κάθε εγγράφου ξεχωριστά θα φέρει μοιραία καθυστερήσεις σε μεγάλα σύνολα εισόδου ή συλλογές με άνιση κατανομή δειγμάτων ανά κλάση. Επίσης η ύπαρξη ενός τέτοιου αρχείου που εκτελεί χρέη αναφοράς για ολόκληρο το εύρος της υλοποίησης μπορεί να παρουσιάσει προβλήματα κατά την ταυτόχρονη εγγραφή διαφορετικών mappers, ή ακόμα χειρότερα να χρειαστεί συγχρονισμός πολλών στιγμιοτύπων αυτού του αρχείου για λογαριασμό κάθε κόμβου. Για να αποφευχθεί μια τέτοια τετριμμένη προσέγγιση, προτιμήθηκε να **συγκρατείται η ετικέτα κατηγορίας** μίας από τις δύο κλάσεις (εδώ επιλέγεται η θετική) **συμπεριλαμβάνοντας έναν συγκεκριμένο χαρακτήρα** (εδώ επιλέγεται φυσικά ο χαρακτήρας "+") **στο τέλος κάθε αναγνωριστικού εγγράφου** στο σύνθετο κλειδί των ζευγαριών κλειδιού-τιμής που δημιουργούνται στους mappers. Με αυτό τον τρόπο, όταν χρειαστεί στην υλοποίηση να γνωστοποιηθεί σε ποια κλάση ανήκει κάθε έγγραφο, αρκεί να ελεγχθεί αν το αναγνωριστικό κάθε εγγράφου φέρει ή όχι στο τέλος τον χαρακτήρα που ορίστηκε εξ αρχής.

```
Map(σύνολο_εκπαίδευσης)
{
      ΔΕΙΓΜΑΤΑ++
      αναγνωριστικό_εγγράφου = γραμμή[0]
      ετικέτα_κατηγορίας = γραμμή[1]

      αν(ετικέτα_κατηγορίας == "Θετικό")
          αναγνωριστικό_εγγράφου.πρόσθεσε("+")

      κείμενο = γραμμή[2].φιλτράρισμα_όρων()

       για(κάθε λέξη στο κείμενο.διαχωρισμός_όρων(" "))
            γράψε((λέξη, αναγνωριστικό_εγγράφου), 1)
}
```

Αλγόριθμος 2.5 - Συνάρτηση Map για την Καταμέτρηση Όρων

Στο σκέλος της συνάρτησης Reduce προκύπτουν για κάθε χαρακτηριστικό ανά έγγραφο το πλήθος των εμφανίσεων τους σε μορφή ζευγαριού κλειδιού-τιμής.

```
Reduce(κλειδί, τιμές[])
{
      άθροισμα_εμφανίσεων = 0

      για(κάθε τιμή στο τιμές[])
            άθροισμα_εμφανίσεων += τιμή

      γράψε(κλειδί, άθροισμα_εμφανίσεων)
}
```

Αλγόριθμος 2.7 - Συνάρτηση Reduce για την Καταμέτρηση Όρων

2.1.2.2 Συχνότητα Όρων

Το κύριο ζητούμενο εδώ είναι το μήκος του κειμένου κάθε εγγράφου σε λέξεις, έτσι ώστε στο τέλος αυτού του κύκλου να επιστρέφονται ζευγάρια κλειδιού-τιμής με σύνθετο κλειδί μία λέξη με το έγγραφο της και σύνθετη τιμή τον αριθμό εμφανίσεων της στο έγγραφο με το μήκος αυτού.

Η συνάρτηση Map εναλλάσει τα στοιχεία, φέροντας το αναγνωριστικό κάθε εγγράφου στο κλειδί κάθε ζεύγους και ορίζοντας ως σύνθετη τιμή την λέξη με το πλήθος εμφάνισης της στο εν λόγω έγγραφο. Έτσι τα ζευγάρια κλειδιού-τιμής που θα παραλάβουν οι reducers θα ορίζονται για κάθε έγγραφο ξεχωριστά για να υπολογιστεί με αυτό τον τρόπο το μήκος τους σε λέξεις.

```
Map(κλειδί, τιμή)
{
	σύνθετο_κλειδί[] = κλειδί.διάσπαση_σύνθετου_κλειδιού()
	λέξη = σύνθετο_κλειδί[0]
	αναγνωριστικό_εγγράφου = σύνθετο_κλειδί[1]

	γράψε(αναγνωριστικό_εγγράφου, (λέξη, τιμή))
}
```

Αλγόριθμος 2.8 - Συνάρτηση Map για την Συχνότητα Όρων

Το σκέλος της Reduce συνάρτησης περιορίζεται αρχικά στην καταμέτρηση του πλήθους των τιμών που υποδηλώνουν το μήκος όρων κάθε εγγράφου. Έπειτα εναλλάσσονται ξανά οι ποσότητες, φέροντας ως σύνθετο κλειδί την λέξη με το έγγραφο της και σχηματίζοντας ως σύνθετη τιμή το πλήθος εμφανίσεων αυτού του όρου στο έγγραφο μαζί με το μήκος του εγγράφου. Η εναλλαγή των ποσοτήτων σε κάθε ζευγάρι κλειδιού-τιμής γίνεται με χρήση λιστών για τις λέξεις και το πλήθος εμφανίσεων τους, που σαρώνονται για να δωθούν στα ζευγάρια που θα προκύψουν.

```
Reduce(κλειδί, τιμές[])
{
	αναγνωριστικό_εγγράφου = κλειδί
	μήκος_εγγράφου = 0

	για(κάθε τιμή στο τιμές[])
	{
		σύνθετη_τιμή[] = τιμή.διάσπαση_σύνθετης_τιμής()

		λίστα_λέξεων.πρόσθεσε(σύνθετη_τιμή[0])
		λίστα_πλήθους_εμφανίσεων.πρόσθεσε(σύνθετη_τιμή[1])

		μήκος_εγγράφου++
	}

	για(κάθε λέξη, πλήθος_εμφανίσεων στις λίστες λέξεων, πληθών εμφανίσεων)
		γράψε((λέξη, αναγνωριστικό_εγγράφου), (πλήθος_εμφανίσεων, μήκος_εγγράφου))
}
```

Αλγόριθμος 2.9 - Συνάρτηση Reduce για την Συχνότητα Όρων

2.1.2.3 TFIDF Όρων

Με τους συντελεστές για τον υπολογισμό της συχνότητας όρων να έχουν οριστεί στον προηγούμενο κύκλο εργασίας, απομένει η αριθμητική πράξη της διαίρεσης για την ανάκτηση της τιμής της συχνότητας, καθώς και ο υπολογισμός της αντίστροφης συχνότητας εγγράφων για τα χαρακτηριστικά που παρουσιάζονται σε αυτά.

Στα της Map συνάρτησης, χρειάζεται μετασχηματισμός επί των ζευγαριών κλειδιού-τιμής με σκοπό να βρεθεί κάθε χαρακτηριστικό στο κλειδί και τα υπόλοιπα στοιχεία να τοποθετηθούν στην τιμή κάθε ζευγαριού. Με αυτή την εναλλαγή γίνεται προσπάθεια να λάβει χώρα η διαδικασία υπολογισμού για κάθε χαρακτηριστικό σε κάθε έγγραφο στην Reduce συνάρτηση αυτού του βήματος.

```
Map(κλειδί, τιμή)
{
	σύνθετο_κλειδί[] = κλειδί.διάσπαση_σύνθετου_κλειδιού()
	σύνθετη_τιμή[] = κλειδί.διάσπαση_σύνθετης_τιμής()

	λέξη = σύνθετο_κλειδί[0]
	αναγνωριστικό_εγγράφου = σύνθετο_κλειδί[1]
	πλήθος_εμφανίσεων = σύνθετη_τιμή[0]
	μήκος_εγγράφου = σύνθετη_τιμή[1]

	γράψε(λέξη, (αναγνωριστικό_εγγράφου, πλήθος_εμφανίσεων, μήκος_εγγράφου))
}
```

Αλγόριθμος 2.10 - Συνάρτηση Map για τον Υπολογισμό των TFIDF Βαρών

Στην πλευρά της συνάρτησης Reduce, απομένει ο υπολογισμός του μέτρου αντίστροφης συχνότητας εγγράφων για να βρεθεί το TFIDF βάρος, που υπολογίζεται ως ο λογάριθμος του πηλίκου του πλήθους δειγμάτων στην συλλογή προς την συχνότητα εγγράφου ενός χαρακτηριστικό (δηλαδή τον αριθμό των δειγμάτων που περιέχουν το χαρακτηριστικό). Με τον αριθμό των δειγμάτων που περιέχει το σύνολο εκπαίδευσης να είναι ήδη γνωστό μέσω του αντίστοιχου καθολικού μετρητή, απομένει η διευθέτηση της συνιστώσας της συχνότητας εγγράφου για κάθε χαρακτηριστικό. Έχοντας ορίσει από την Map συνάρτηση κάθε χαρακτηριστικό ως το κλειδί των ζευγαριών κλειδιού-τιμής, εννοείται εδώ ότι η συχνότητα εγγράφου κάθε χαρακτηριστικού ισούται με τον αριθμό των ζευγαριών κλειδιού-τιμής που έχουν συναθροιστεί κατά κλειδί σε αυτή τη φάση αυτού του κύκλου εργασίας. Με την διάσπαση της σύνθετης τιμής κάθε ζευγαριού που δέχεται ένας reducer σε αναγνωριστικό του εγγράφου, πλήθος εμφανίσεων χαρακτηριστικού, και μήκος εγγράφου, υπολογίζεται το TFIDF βάρος κάθε χαρακτηριστικού.

```
Reduce(κλειδί, τιμές[])
{
	λέξη = κλειδί

	για(κάθε τιμή στο τιμές[])
	{
		σύνθετη_τιμή[] = τιμή.διάσπαση_σύνθετης_τιμής()
		αναγνωριστικό_εγγράφου = σύνθετη_τιμή[0]
		πλήθος_εμφανίσεων = σύνθετη_τιμή[1]
		μήκος_εγγράφου = σύνθετη_τιμή[2]
```

```
            tf = πλήθος_εμφανίσεων / μήκος_εγγράφου
            idf = log(ΔΕΙΓΜΑΤΑ / τιμές[].μήκος)
            tfidf = tf * idf

            γράψε((λέξη, αναγνωριστικό_εγγράφου), tfidf)
        }
}
```

Αλγόριθμος 2.11 - Συνάρτηση Reduce για τον Υπολογισμό των TFIDF Βαρών

2.1.2.4 Επιλογή Όρων

Έχοντας προσδιορίσει το βάρος κάθε χαρακτηριστικού ανά έγγραφο στο μέτρο του TFIDF, μπορούν να κινηθούν οι διαδικασίες για το "κοσκίνισμα" τους με τέτοιο τρόπο ώστε να μην ληφθούν υπόψη αυτά που εμφανίζονται λιγότερο σχετικά στο σύνολο εκπαίδευσης. Ο τρόπος με τον οποίο μπορεί να επιτευχθεί κάτι τέτοιο δίνεται από την ταξινόμηση των ζευγαριών κλειδιού-τιμής που προέρχονται από τον προηγούμενο κύκλο εργασίας ως προς τις TFIDF βαθμολογίες και τον ορισμό ενός αριθμού (ή ποσοστού) χαρακτηριστικών που θα διατηρηθούν στην επόμενη φάση, με τα υπόλοιπα να μην λαμβάνονται υπόψη. Με την περάτωση αυτού του κύκλου εργασίας, κάθε χαρακτηριστικό ανά έγγραφο θα έχει ταξινομηθεί κατά TFIDF βάρος σε μία λίστα από όπου θα απομακρυνθούν οι εγγραφές με τα μικρότερα βάρη έως ότου το πλήθος των εγγραφών να ανταποκρίνεται στο ποσοστό σημαντικότερων χαρακτηριστικών που έχει οριστεί.

Ίσως ο σημαντικότερος παράγοντας προς ρύθμιση στην τροποποιημένη υλοποίηση του Naïve Bayes στο Hadoop, το ποσοστό επιλεγμένων χαρακτηριστικών καθορίζει σε μεγάλο βαθμό τα αποτελέσματα του προγράμματος στην επίδοση ταξινόμησης αλλά και τις χρονικές επιδόσεις της εκτέλεσης, που θα είναι μοιραία ανάλογες με το μέγεθος των χαρακτηριστικών το οποίο θα χαρακτηρίζει το λεξιλόγιο που θα διαθέτει το μοντέλο στο βήμα της εκπαίδευσης. Με την βασική υλοποίηση του Naïve Bayes στο Hadoop που αναλύθηκε προηγουμένως να θεωρεί όλα τα χαρακτηριστικά το ίδιο ακριβώς σχετικά, ο παράγοντας αυτός σημαίνει την κύρια διαφοροποίηση μεταξύ αυτών των δύο. Το ποσοστό χαρακτηριστικών που θα επιλεγεί μπορεί να προκύψει μετά από μελέτη πάνω σε θεωρίες της στατιστικής (όπως για παράδειγμα με την αρχή του Pareto, που χαρακτηρίζεται από το αξίωμα του κανόνα 80/20 ο οποίος ισχυρίζεται ότι το 20% των δεδομένων μπορεί να είναι αντιπροσωπευτικό του υπόλοιπου 80%) ή εμπειρικά μετά από δοκιμές διαφορετικών ποσοστών στα ίδια ακριβώς δεδομένα. Έπειτα από δοκιμές έγινε φανερό ότι η καλύτερη ποιότητα κατηγοριοποίησης εμφανίζεται **στην επιλογή του 75% των πιο σχετικών χαρακτηριστικών των δεδομένων εκπαίδευσης**. Ο ορισμός αυτής της ποσότητας επαφίεται στην αντιμετώπιση που θα επιλεγεί προγραμματιστικά ώστε να γίνει η διαλογή των χαρακτηριστικών με αυτό το ποσοστό.

Για λογαριασμό της Map συνάρτησης εδώ, πρέπει να αποφασιστεί εκ των προτέρων ο τρόπος με τον οποίο θα συνταχθούν τα ζευγάρια κλειδιού-τιμής, μιας και σε αυτό το σημείο τα δεδομένα βρίσκονται σε επίπεδο χαρακτηριστικού ανά έγγραφο πριν την εισαγωγή τους στην επόμενη φάση της εκπαίδευσης του μοντέλου. Μια πρώτη προσέγγιση θα ήταν να τεθεί από κάθε mapper σαν κλειδί η βαθμολογία TFIDF και ως τιμή το εκάστοτε χαρακτηριστικό με το αναγνωριστικό εγγράφου του για να ταξινομηθούν τα ζευγάρια εσωτερικά πριν οδηγηθούν στους reducers, όμως κάτι τέτοιο θα είχε ως αποτέλεσμα οι reducers να καλούνται κάθε φορά για ζευγάρια που έχουν ίδιο κλειδί το οποίο φυσικά δεν νοείται στην περίπτωση που εξετάζεται εδώ. Παρακάμπτοντας την υλοποίηση κάποιας Reduce συνάρτησης για να επιλυθεί αυτό το ζήτημα προξενείται η ανάγκη για την ανάπτυξη ενός επιπλέον

κύκλου εργασίας που θα εργαστεί στην επιλογή του 75% των πιο σχετικών χαρακτηριστικών, επιβαρύνοντας το πρόγραμμα από άποψη χρόνου και πολυπλοκότητας σχεδιασμού.

Μία άλλη οπτική για την μεταφορά των χαρακτηριστικών στην συνάρτηση Reduce είναι να συγκεντρωθούν οι πληροφορίες του κάθε χαρακτηριστικού με το αναγνωριστικό εγγράφου του μαζί με την TFIDF βαθμολογία του στο πεδίο της τιμής των ζευγαριών κλειδιού-τιμής που θα δημιουργηθούν, θέτοντας παράλληλα στο πεδίο του κλειδιού μια κενή συμβολική τιμή *NULL* για να συγκεντρωθούν όλα τα ζευγάρια στο ίδιο στιγμιότυπο reducer και να γίνουν οι απαραίτητες ενέργειες μέσω ενός και μόνο κόμβου. Κάτι τέτοιο φυσικά θα αποτελούσε επί της ουσίας ένα σειριακό κομμάτι υπολογισμού σε ένα κατά τα άλλα παράλληλο υπολογισμό. Επίσης δεν υπάρχει κανενός είδους εγγύηση για το αν μπορεί να εξυπηρετήσει ο εξοπλισμός ενός κόμβου το σύνολο όλων των δεδομένων που θα κληθεί να επεξεργαστεί μέσω ενός reducer, μιας και στην συγκεκριμένη μελέτη δίνεται ιδιαίτερη έμφαση στην κλιμακωσιμότητα μιας εφαρμογής, οπότε μια τέτοια προσέγγιση κρίνεται ακατάλληλη και προφανώς επιζήμια για την επίδοση.

Τα παραπάνω ενισχύουν την επιλογή του αναγνωριστικού εγγράφου ως κλειδί στα ζευγάρια κλειδιού-τιμής που πρέπει να δημιουργήσουν οι mappers σε αυτό το βήμα, αφήνοντας κάθε χαρακτηριστικό με την TFIDF βαθμολογία του στην τιμή κάθε ζευγαριού. Με αυτό τον τρόπο κάθε reducer αναφέρεται σε ένα συγκεκριμένο έγγραφο ούτως ώστε με το πέρας αυτού του βήματος να δοθεί στην επόμενη φάση ένας αριθμός ζευγαριών κλειδιού-τιμής που θα είναι όμοια στο σύνολο τους με την συλλογή δεδομένων εκπαίδευσης που δόθηκαν στην αρχή της υλοποίησης.

```
Map(κλειδί, τιμή)
{
      σύνθετο_κλειδί[] = κλειδί.διάσπαση_σύνθετου_κλειδιού()

      λέξη = σύνθετο_κλειδί[0]
      αναγνωριστικό_εγγράφου = σύνθετο_κλειδί[1]
      TFIDF = τιμή

      γράψε(αναγνωριστικό_εγγράφου, (λέξη, TFIDF))
}
```

Αλγόριθμος 2.12 - Συνάρτηση Map για την Επιλογή Χαρακτηριστικών

Περνώντας στην συνάρτηση Reduce, εξυπακούεται ότι εφόσον όλα τα ζευγάρια κλειδιού-τιμής φέρουν ως κλειδί τον αναγνωριστικό κάθε εγγράφου (με ή χωρίς τον χαρακτήρα που υποδηλώνει την ετικέτα κατηγορίας του), η συνάρτηση αυτή θα εκτελεστεί ξεχωριστά για κάθε έγγραφο, ώστε να αξιοποιηθεί ο καθολικός μετρητής του πλήθους των εγγράφων έτσι όπως αυτά διαμορφώθηκαν. Στην συνέχεια συγκεντρώνονται για κάθε έγγραφο τα χαρακτηριστικά του και ταξινομούνται κατά TFIDF έτσι ώστε να επιλεγεί το 75% των πιο σχετικών εξ αυτών. Ύστερα αξιοποιούνται οι καθολικοί μετρητές που αφορούν το πλήθος δειγμάτων και λέξεων κάθε κλάσης ανάλογα με τον χαρακτήρα που φέρει (ή όχι) το αναγνωριστικό εγγράφου στο κλειδί κάθε ζευγαριού. Στην έξοδο κάθε ζευγάρι κλειδιού-τιμής θα φέρει ως κλειδί το αναγνωριστικό εγγράφου και ως τιμή το φιλτραρισμένο κείμενο με τα πιο σημαντικά χαρακτηριστικά. Με αυτό τον τρόπο η επόμενη φάση υποδέχεται σαν είσοδο μια φιλτραρισμένη εκδοχή της αρχικής συλλογής εκπαίδευσης που εφαρμόστηκε ως είσοδος στον πρώτο κύκλο εργασίας αυτής της εφαρμογής.

```
Reduce(κλειδί, τιμές[])
{
	ΝΕΑ_ΔΕΊΓΜΑΤΑ++

	ταξινομημένα_χαρακτηριστικά[] = τιμές[].ταξινόμησε_κατά_TFIDF()
	μήκος_χαρακτηριστικών = ταξινομημένα_χαρακτηριστικά[].μήκος

	όσο(ταξινομημένα_χαρακτηριστικά[].μήκος > μήκος_χαρακτηριστικών * 75 / 100)
		ταξινομημένα_χαρακτηριστικά,αφαίρεσε_πρώτο_στοιχείο()

	αν(κλειδί.τελειώνει_σε("+"))
	{
		ΘΕΤΙΚΑ_ΔΕΙΓΜΑΤΑ++
		ΘΕΤΙΚΕΣ_ΛΕΞΕΙΣ += ταξινομημένα_χαρακτηριστικά[].μήκος()
	}
	αλλιώς
	{
		ΑΡΝΗΤΙΚΑ_ΔΕΙΓΜΑΤΑ++
		ΑΡΝΗΤΙΚΕΣ_ΛΕΞΕΙΣ += ταξινομημένα_χαρακτηριστικά[].μήκος()
	}

	κείμενο_εγγράφου = ""

	για(κάθε χαρακτηριστικό στη λίστα ταξινομημένων χαρακτηριστικών)
		κείμενο_εγγράφου += χαρακτηριστικό

	γράψε(κλειδί, κείμενο_εγγράφου)
}
```

Αλγόριθμος 2.13 - Συνάρτηση Reduce για την Επιλογή Χαρακτηριστικών

2.1.2.5 Εκπαίδευση Μοντέλου

Με τα ζευγάρια κλειδιού-τιμής από το τελευταίο βήμα να αναπαριστούν επαρκώς τα έγγραφα του διαμορφωμένου κατά επιλογή χαρακτηριστικών συνόλου εκπαίδευσης, ο κύκλος εργασίας για την εκπαίδευση του μοντέλου βαίνει στην ίδια λογική λειτουργίας με την απλή υλοποίηση του Naïve Bayes. Με τους καθολικούς μετρητές να λειτουργούν όπως έχει προαναφερθεί για τις ποσότητες των χαρακτηριστικών και εγγράφων κάθε κλάσης, καθώς και για τις γενικές πληροφορίες συνολικού πλήθους χαρακτηριστικών και εγγράφων, έχουν οριστεί οι περισσότεροι παράγοντες για τον υπολογισμό πιθανότητας ένα έγγραφο ελέγχου να ανήκει σε κάθε μία από τις κλάσεις που υποστηρίζει το μοντέλο. Η προτελευταία πληροφορία που απομένει προς υπολογισμό πριν το βήμα του ελέγχου είναι και εδώ το πλήθος εμφανίσεων κάθε χαρακτηριστικού σε έγγραφα κάθε κλάσης, για να προσδιοριστούν οι αντίστοιχες πιθανότητες υπό συνθήκη.

Για το σκέλος της Map συνάρτησης που υποδέχεται ως είσοδο τα διαμορφωμένα έγγραφα, αρκεί η διάσπαση των χαρακτηριστικών τους και παραγωγή ζευγαριών κλειδιού-τιμής με το χαρακτηριστικό να λαμβάνει τη θέση του κλειδιού, ενώ η ετικέτα κατηγορίας του εγγράφου ορίζεται ως η τιμή όλων των ζευγαριών που προέρχεται από την διάσπαση του εκάστοτε εγγράφου στους mappers. Με αυτό τον τρόπο μετατοπίζεται στους reducers ο φόρτος για την καταμέτρηση του πλήθους εμφάνισης κάθε όρου σε έγγραφα θετικής και αρνητικής κλάσης.

```
Map(κλειδί, τιμή)
{
      σύνθετη_τιμή[] = τιμή.διάσπαση_σύνθετης_τιμής()

      κείμενο_εγγράφου = σύνθετη_τιμή[0]
      ετικέτα_κατηγορίας = σύνθετη_τιμή[1]

      χαρακτηριστικά[] = κείμενο_εγγράφου.διαχωρισμός_όρων(" ")

      για(κάθε χαρακτηριστικό στο χαρακτηριστικά[])
            γράψε(χαρακτηριστικό, ετικέτα_κατηγορίας)
}
```

Αλγόριθμος 2.14 - Συνάρτηση Map για την Εκπαίδευση Μοντέλου

Κάθε reducer λαμβάνει για λογαριασμό ενός χαρακτηριστικού έναν αριθμό τιμών με τις ετικέτες κατηγορίας, όπου αρκεί αυτές να καταμετρηθούν με την βοήθεια δύο τοπικών μετρητών για να υπολογιστεί το πλήθος εμφάνισης του εκάστοτε χαρακτηριστικού σε έγγραφα της αντίστοιχης κλάσης. Σε αυτό το σημείο επίσης αυξάνεται και ο καθολικός μετρητής που εκφράζει το συνολικό πλήθος των χαρακτηριστικών και αποτελεί τον τελευταίο παράγοντα που χρειάζεται η εφαρμογή για την εκπαίδευση του μοντέλου πριν την υποδοχή της συλλογής ελέγχου στον επόμενο και τελικό κύκλο εργασίας.

```
Reduce(κλειδί, τιμές[])
{
      ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ++

      θετικές_εμφανίσεις = 0
      αρνητικές_εμφανίσεις = 0

      για(κάθε τιμή στο τιμές[])
      {
            αν(τιμή == "ΘΕΤΙΚΟ")
                  θετικές_εμφανίσεις++
            αλλιώς
                  αρνητικές_εμφανίσεις++
      }

      γράψε(κλειδί, (θετικές_εμφανίσεις, αρνητικές_εμφανίσεις))
}
```

Αλγόριθμος 2.15 - Συνάρτηση Reduce για την Ανασυγκρότηση Εγγράφων

2.1.2.6 Έλεγχος Μοντέλου

Ο κύκλος εργασίας για τον έλεγχο του μοντέλου χρειάζεται και σε αυτή την έκδοση την προετοιμασία των πιθανοτήτων που θα χρησιμοποιηθούν στην διαλογή αυτού του βήματος πριν την υποδοχή του συνόλου ελέγχου. Σαν προετοιμασία λογίζεται και εδώ ο υπολογισμός των εκ των προτέρων πιθανοτήτων *P(y)* για την θετική και την αρνητική κλάση. Με το τέλος αυτής της συνάρτησης Setup η εφαρμογή διαθέτει ό,τι χρειάζεται για να επεξεργαστεί εν τέλει τα δείγματα ελέγχου ώστε να προκύψουν οι προβλέψεις του μοντέλου κατηγοριοποίησης.

```
Setup()
{
	πιθανότητα_θετικής_κλάσης = ΘΕΤΙΚΑ_ΔΕΙΓΜΑΤΑ / ΔΕΙΓΜΑΤΑ

	πιθανότητα_αρνητικής_κλάσης = ΑΡΝΗΤΙΚΑ_ΔΕΙΓΜΑΤΑ / ΔΕΙΓΜΑΤΑ

	για(κάθε αρχείο στο αποτελέσματα_φάσης_εκπαίδευσης)
	{
		για(κάθε γραμμή στο αρχείο)
		{
			θετικά_χαρακτηριστικά[].πρόσθεσε(γραμμή[0], γραμμή[1])
			αρνητικά_χαρακτηριστικά[].πρόσθεσε(γραμμή[0], γραμμή[2])
		}
	}

	για(κάθε στοιχείο στις λίστες θετικών και αρνητικών χαρακτηριστικών)
	{
		πιθανότητες_θετ_χαρακτηριστικών[].πρόσθεσε(στοιχείο.κλειδί, στοιχείο.τιμή + 1 /
				(ΘΕΤΙΚΕΣ_ΛΕΞΕΙΣ  + ΔΕΙΓΜΑΤΑ))

		πιθανότητες_αρν_χαρακτηριστικών[].πρόσθεσε(στοιχείο.κλειδί, στοιχείο.τιμή + 1 /
				(ΑΡΝΗΤΙΚΕΣ_ΛΕΞΕΙΣ  + ΔΕΙΓΜΑΤΑ))
	}
}
```

Αλγόριθμος 2.16 - Συνάρτηση Setup για τον Έλεγχο του Naïve Bayes Μοντέλου

Η εφαρμογή καταλήγει στην Map συνάρτηση που λαμβάνει τη συλλογή ελέγχου και απομονώνει από κάθε έγγραφο το αναγνωριστικό του, το κείμενο του, καθώς και την ετικέτα κατηγορίας του. Φιλτράρει από το κείμενο τους όρους που ενδέχεται να περιέχουν πληροφορία και διαχωρίζει το κείμενο που απομένει σε λέξεις ώστε να αναπαριστάται στην μορφή ενός διανύσματος. Στη συνέχεια, με κάθε λέξη στο διάνυσμα υπολογίζεται το γινόμενο των πιθανοτήτων των χαρακτηριστικών για κάθε κλάση κι ύστερα αυτή η ποσότητα πολλαπλασιάζεται με την αντίστοιχη εκ των προτέρων πιθανότητα. Έπειτα συγκρίνονται οι εκ των προτέρων πιθανότητες μεταξύ τους και η μέγιστη εξ αυτών αποτελεί την πρόβλεψη του μοντέλου, ενώ στο τέλος συγκρίνεται η πρόβλεψη του μοντέλου με την ετικέτα που φέρει το έγγραφο ώστε να αξιοποιηθούν οι καθολικοί μετρητές της μήτρας σύγχυσης και να μελετηθεί το πλήθος των εγγράφων που ταξινομήθηκαν σωστά ή λανθασμένα. Σαν έξοδο οι mappers σχηματίζουν ζευγάρια κλειδιού-τιμής όπου το αναγνωριστικό του εγγράφου με το κείμενο του τοποθετούνται σαν σύνθετο κλειδί και η ετικέτα κατηγορίας που προέβλεψε του μοντέλο τίθεται ως τιμή.

```
Map(σύνολο_ελέγχου)
{
	αναγνωριστικό_εγγράφου = γραμμή[0]
	ετικέτα_κατηγορίας = γραμμή[1]
	κείμενο = γραμμή[2].φιλτράρισμα_όρων()

	θετική_πιθανότητα = 1
	αρνητική_πιθανότητα = 1

	όροι[] = κείμενο.διαχωρισμός_όρων(“ “)
```

```
    για(κάθε λέξη στη λίστα των όρων)
    {
        για(κάθε στοιχείο από τις λίστες θετικών και αρνητικών χαρακτηριστικών)
        {
                θετική_πιθανότητα *= θετικά_χαρακτηριστικά.ανάκτησε(λέξη)
                αρνητική_πιθανότητα *= αρνητικά_χαρακτηριστικά.ανάκτησε(λέξη)
        }
    }

    θετική_πιθανότητα *= πιθανότητα_θετικής_κλάσης
    αρνητική_πιθανότητα *= πιθανότητα_αρνητικής_κλάσης

    αν(θετική_πιθανότητα > αρνητική_πιθανότητα)
    {
        αν(ετικέτα_κατηγορίας == "Θετικό")
            TRUE_POSITIVE++
        αλλιώς
            FALSE_POSITIVE++

        γράψε((αναγνωριστικό_εγγράφου, κείμενο), "Θετικό")
    }
    αλλιώς
    {
        αν(ετικέτα_κατηγορίας == "Αρνητικό")
            TRUE_NEGATIVE++
        αλλιώς
            FALSE_POSITIVE++

        γράψε((αναγνωριστικό_εγγράφου, κείμενο), "Αρνητικό")
    }
}
```

Αλγόριθμος 2.17 - Συνάρτηση Map για τον Έλεγχο του Naïve Bayes Μοντέλου

Και εδώ η ανάπτυξη μιας συνάρτησης Reduce κρίνεται περιττή σε αυτό τον κύκλο εργασίας, μιας και τα αποτελέσματα καθώς και οι μετρικές ταξινόμησης που χρειάζονται για την αποτίμηση της αποτελεσματικότητας της εφαρμογής προέκυψαν εσωτερικά των mappers, με κάθε στιγμιότυπο να εργάζεται για ένα έγγραφο την φορά. Έτσι, δεν απαιτείται καμία συνάθροιση των ζευγαριών κλειδιού-τιμής που προέκυψαν κατά κλειδί και η τροποποιημένη υλοποίηση του αλγόριθμου Naïve Bayes στο Hadoop ολοκληρώνεται αισίως σε αυτό το σημείο.

Κύκλος Εργασίας	Φάση Κύκλου Εργασίας	Δομή Ζευγαριών Κλειδιού-Τιμής	Περιγραφή Φάσης
Καταμέτρηση Όρων	Map	**Είσοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** (εμφανίσεις λέξης, μήκος εγγράφου) **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** (έγγραφο, εμφανίσεις λέξης, μήκος εγγράφου)	• Υποδοχή συνόλου εκπαίδευσης. • Φιλτράρισμα κειμένου εγγράφου εκπαίδευσης. • Διάσπαση κειμένου σε μια λίστα λέξεων. • Συσχέτιση κάθε λέξης με την μονάδα που υποδεικνύει το ελάχιστο πλήθος εμφάνισης της σε ένα έγγραφο. • Μαρκάρισμα όλων των αναγνωριστικών θετικών εγγράφων.
	Reduce	**Είσοδος** **Κλειδί:** λέξη **Τιμή:** (έγγραφο, εμφανίσεις λέξης, μήκος εγγράφου) **'Εξοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** TFIDF βαθμολογία	• Καταμέτρηση του συνολικού πλήθους εμφάνισης κάθε όρου σε ένα έγγραφο. πλήθος εμφάνισης της σε ένα έγγραφο.
Συχνότητα Όρων	Map	**Είσοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** TFIDF βαθμολογία **'Εξοδος** **Κλειδί:** έγγραφο **Τιμή:** (λέξη, TFIDF βαθμολογία)	• Μετάθεση αναγνωριστικού κάθε εγγράφου στο κλειδί κάθε ζευγαριού κλειδιού-τιμής για την συνάθροιση κατά έγγραφο στην συνάρτηση Reduce.
	Reduce	**Είσοδος** **Κλειδί:** έγγραφο **Τιμή:** (λέξη, TFIDF βαθμολογία) **'Εξοδος** **Κλειδί:** έγγραφο **Τιμή:** κείμενο	• Υπολογισμός των παραγόντων για την συχνότητα ενός όρου ανά έγγραφο. πλήθος εμφάνισης της σε ένα έγγραφο. • Μετάθεση κάθε λέξης σε σύνθετο κλειδί με το αναγνωριστικό εγγράφου για κάθε ζευγάρι κλειδιού-τιμής.
TFIDF Όρων	Map	**Είσοδος** **Κλειδί:** έγγραφο **Τιμή:** (κείμενο, ετικέτα κλάσης) **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** ετικέτα κλάσης	• Μετάθεση του αναγνωριστικού εγγράφου στην σύνθετη τιμή των ζευγαριών κλειδιού-τιμής για την συνάθροιση κατά λέξη στην συνάρτηση Reduce.

	Reduce	**Είσοδος** **Κλειδί:** λέξη **Τιμή:** ετικέτα κλάσης **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** (θετικές εμφανίσεις, αρνητικές εμφανίσεις)	• Υπολογισμός των TFIDF βαρών για κάθε όρο ξεχωριστά σε κάθε έγγραφο. • Μετάθεση του αναγνωριστικού εγγράφου στο σύνθετο πια κλειδί ζευγαριών κλειδιού-τιμής για την στοίχιση κατά όρο ανά έγγραφο στον επόμενο κύκλο εργασίας.
Επιλογή Όρων	Map	**Είσοδος** **Κλειδί:** θέση δείκτη αρχείου **Τιμή:** γραμμή αρχείου **'Εξοδος** **Κλειδί:** (έγγραφο, κείμενο) **Τιμή:** ετικέτα κλάσης	• Μετάθεση αναγνωριστικού εγγράφου στο κλειδί και σχηματισμός σύνθετης τιμής χαρακτηριστικού με την TFIDF βαθμολογία του στα ζευγάρια κλειδιού-τιμής ώστε να ανασυγκροτηθεί κάθε έγγραφο πριν την φάση της εκπαίδευσης του μοντέλου.
	Reduce	**Είσοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** (εμφανίσεις λέξης, μήκος εγγράφου) **'Εξοδος** **Κλειδί:** λέξη **Τιμή:** (έγγραφο, εμφανίσεις λέξης, μήκος εγγράφου)	• Επιλογή των κορυφαίων όρων με την μεγαλύτερη βαθμολογία TFIDF στο 75% επί του συνόλου. • Επανασύνταξη του κειμένου κάθε εγγράφου με τους εναπομείναντες όρους. • Προσδιορισμός του πλήθους δειγμάτων και λέξεων για κάθε κλάση. • Τοποθέτηση του κειμένου του εγγράφου στην τιμή των ζευγαριών κλειδιού-τιμής.
Εκπαίδευση Μοντέλου	Map	**Είσοδος** **Κλειδί:** λέξη **Τιμή:** (έγγραφο, εμφανίσεις λέξης, μήκος εγγράφου) **'Εξοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** TFIDF βαθμολογία	• Υποδοχή επιλεγμένου συνόλου εκπαίδευσης. • Διάσπαση κειμένου σε μια λίστα λέξεων. • Συσχέτιση κάθε λέξης με την ετικέτα της κλάσης την οποία φέρει το έγγραφο στο οποίο εμφανίστηκε.

	Reduce	**Είσοδος** **Κλειδί:** (λέξη, έγγραφο) **Τιμή:** TFIDF βαθμολογία **'Εξοδος** **Κλειδί:** έγγραφο **Τιμή:** (λέξη, TFIDF βαθμολογία)	• Υπολογισμός πλήθους εμφανίσεων μιας λέξης σε έγγραφα κάθε κλάσης, για κάθε λέξη. • Συσχέτιση κάθε λέξης με τις εμφανίσεις της σε κάθε κλάση. • Προετοιμασία συντελεστών για την κατασκευή του μοντέλου.
Έλεγχος Μοντέλου	Map	**Είσοδος** **Κλειδί:** έγγραφο **Τιμή:** (λέξη, TFIDF βαθμολογία) **'Εξοδος** **Κλειδί:** έγγραφο **Τιμή:** κείμενο	• Εφαρμογή συντελεστών για την ρύθμιση του μοντέλου. • Υποδοχή συνόλου ελέγχου. • Φιλτράρισμα κειμένου εγγράφου ελέγχου. • Διάσπαση κειμένου σε μια λίστα λέξεων. • Υπολογισμός πιθανοτήτων για το ενδεχόμενο ένα έγγραφο να ανήκει σε μία κλάση, για κάθε κλάση. • Ορισμός ετικέτας κλάσης με την μεγαλύτερη πιθανότητα για τον λογαριασμό κάθε εγγράφου.
	Reduce	Δεν ορίζεται	Δεν ορίζεται

Πίνακας 2.4 - Διάγραμμα Κύκλων Εργασίας Τροποποιημένης Υλοποίησης Naïve Bayes στο Hadoop

2.2 Υλοποιήσεις στο Spark

Με απώτερο σκοπό την ανάλυση μεταξύ των λειτουργιών στις πλατφόρμες των Hadoop και Spark, κρίνεται αναγκαίο να υλοποιηθούν εφαρμογές με τέτοιο τρόπο ώστε αυτές να τεθούν ευθέως σε σύγκριση. Με τις υλοποιήσεις στο Hadoop να κινούνται γύρω από τον αλγόριθμο Naïve Bayes για την κατηγοριοποίηση δειγμάτων βάσει συναισθήματος, είναι ξεκάθαρη η προσέγγιση που πρέπει να ακολουθηθεί (σε πρώτο χρόνο τουλάχιστον) και εδώ για την ανάπτυξη υλοποιήσεων στην πλατφόρμα του Spark.

Στη μεταφορά των υλοποιήσεων από την μία πλατφόρμα στην άλλη χρειάζεται να μελετηθούν οι όροι με τους οποίους λειτουργεί το Spark για να αξιοποιηθούν τα οφέλη που προσφέρει με τον υψηλό βαθμό αφαίρεσης αλλά και την εξαιρετικά μεγάλη ευελιξία σχεδιασμού που το χαρακτηριίζει σε πιο πολύπλοκες εφαρμογές. Όπως έχει αναφερθεί και παραπάνω, τα οφέλη αυτά έχουν να κάνουν με τον τρόπο με τον οποίο αποθηκεύονται τα δεδομένα (στη μνήμη έναντι του δίσκου) και με τους χρήσιμους μηχανισμούς που περιέχουν οι βιβλιοθήκες στο Spark. Με μια ματιά στις βιβλιοθήκες του Spark,

φαίνεται πως ένα μεγάλο μέρος των λειτουργιών που σχεδιάστηκαν για τις εφαρμογές του Hadoop υπάρχουν έτοιμες και υποστηρίζονται ως υλοποιήσεις προς χρήση με τον τρόπο που ορίζεται από τις αντίστοιχες τους κλάσεις.

Η ύπαρξη εξειδικευμένων μηχανισμών μέσα στο Spark για να παρακαμφθεί ο λεπτομερής σχεδιασμός της υλοποίησης ενός αλγορίθμου σαν τον Naïve Bayes, προτρέπει στην διερεύνηση και άλλων αλγορίθμων που χαίρουν υποστήριξης στο εσωτερικό της πλατφόρμας. Ένας αλγόριθμος από αυτούς είναι και εκείνος των Support Vector Machines (που αναφέρθηκε και αναλύθηκε ως προς την λειτουργία του στην πρώτη ενότητα), όπου και επιλέγεται για να υλοποιηθεί στο Spark συμπληρωματικά και ξεχωριστά από την εφαρμογή του Naïve Bayes, με σκοπό την τελική σύγκριση σχετικά με το ποια εφαρμογή και μέθοδος μπορεί να επιτύχει υψηλότερη επίδοση κατηγοριοποίησης και εκτέλεσης σε ένα κατανεμημένο περιβάλλον.

Παρόμοια με τις ήδη υλοποιημένες κλάσεις για την δημιουργία μοντέλων των αλγορίθμων κατηγοριοποίησης που έχουν αναφερθεί ως τώρα, το Spark περιέχει μηχανισμούς για την εξαγωγή και επιλογή χαρακτηριστικών, με έναν από αυτούς να αφορά το μέτρο TFIDF το οποίο χρησιμοποιήθηκε και στην τροποποιημένη έκδοση του Naïve Bayes αλγόριθμου στο Hadoop. Στην προκειμένη περίπτωση βέβαια το Spark δίνει την επιπλέον δυνατότητα να χρησιμοποιηθούν τα βάρη του TFIDF και για την διανυσματική αναπαράσταση των εγγράφων, έτσι ώστε το πρόγραμμα να μην χρειάζεται ειδική μέριμνα για τημ διαχείριση των δεδομένων σε μορφή κειμένου έναντι της προτιμητέας επεξεργασίας πάνω σε δεδομένα αριθμητικής μορφής. Απομένουν από αυτό το σημείο και μετά τα λοιπά διαδικαστικά μέρη για την ανάπτυξη υλοποιήσεων που θα εφαρμόζουν την κατηγοριοποίηση εγγράφων σε κλάσεις που αναπαριστούν συναισθηματική πολικότητα. Βεβαίως εδώ συμπεριλαμβάνεται η προεπεξεργασία των δεδομένων εισόδου (τα οποία εδώ μπορούν να χωριστούν ψευδοτυχαία σε δεδομένα εκπαίδευσης και ελέγχου βάσει ενός ποσοστού) και η εξαγωγή των χαρακτηριστικών (που μπορεί να υλοποιηθεί κάνοντας και πάλι χρήση λειτουργιών που βρίσκονται εντός των βιβλιοθηκών της πλατφόρμας).

Σε ό,τι αφορά την προσπάθεια βελτίωσης των αλγορίθμων Naïve Bayes και Support Vector Machines στο Spark, αξίζει να μελετηθούν οι μέθοδοι που περιέχει αυτό εσωτερικά για να εκτιμηθεί η προτιμότερη προσέγγιση που θα ακολουθηθεί, ούτως ώστε να μην συμπεριληφθούν και δράσουν τα λιγότερο σχετικά χαρακτηριστικά ως θόρυβος στην κατασκευή του μοντέλου. Μετά από σύντομη μελέτη στις υλοποιημένες λειτουργίες που υποστηρίζει το Spark για τον υπολογισμό του IDF μέτρου, βρέθηκε η μέθοδος **setMinDocFreq** που λαμβάνει ως όρισμα τον ελάχιστο αριθμό εγγράφων που πρέπει να βρίσκεται ένας όρος ώστε αυτός να χαρακτηριστεί σχετικός. Η μέθοδος αυτή σαρώνει το ευρετήριο όρων που σχηματίζεται για τις εμφανίσεις κάθε χαρακτηριστικού, και βρίσκοντας κάθε χαρακτηριστικό με πλήθος εμφάνισης μικρότερο από το όρισμα μηδενίζει το μέτρο IDF του ώστε το βάρος του στο TFIDF να μηδενιστεί.

2.2.1 Naïve Bayes

Αρχικά η εξαγωγή των χαρακτηριστικών από τις συλλογές εισόδου μπορεί να γίνει με χρήση συναρτήσεων Map με τέτοιο τρόπο ώστε πρώτα να απομονώνεται κάθε έγγραφο ξεχωριστά κι έπειτα να δημιουργηθούν ζευγάρια κλειδιού-τιμής που συγκρατείται το κείμενο και η κλάση κάθε εγγράφου σε μορφή RDD. Με την παράλληλη προεπεξεργασία των δεδομένων εξασφαλίζεται ο καθαρισμός του κειμένου από άσχετους όρους και η υλοποίηση προχωρά στην μετατροπή των διαμορφωμένων δεδομένων από RDD σε DataFrame για στοιχειοθετηθούν τα ζευγάρια κλειδιού-τιμής σαν εγγραφές σε

έναν πίνακα με στήλες, ώστε να συμμορφωθεί η υλοποίηση στην απαιτούμενη μορφή για τα βήματα του διαχωρισμού των όρων, υπολογισμού των μέτρων TFIDF, και της κατασκευής του μοντέλου, όπου όλα αναλαμβάνονται από τις αντίστοιχες λειτουργίες που προσφέρονται από το Spark.

```
NB(συλλογή_εισόδου)
{
      είσοδος = διάβασε(συλλογή_εισόδου).map(έγγραφο).map((ετικέτα, κείμενο))

      έγγραφα_dataframe = έγγραφα.μετατροπή_σε_DF("ετικέτα", "κείμενο")
      όροι = έγγραφα_dataframe.διαχωρισμός_όρων("όροι")

      έγγραφα_tf = όροι.TF("tf")
      έγγραφα_idf = όροι.IDF("idf")
      έγγραφα_tfidf =  υπολόγισε_TFIDF(έγγραφα_tf, έγγραφα_idf)

      σύνολο_εκπαίδευσης = έγγραφα_tfidf.ποσοστό(75%)
      σύνολο_ελέγχου = έγγραφα_tfidf.ποσοστό(25%)

      μοντέλο = NaïveBayes(σύνολο_εκπαίδευσης, σύνολο_ελέγχου)
}
```

Αλγόριθμος 2.20 - Υλοποίηση Naïve Bayes Μοντέλου στο Spark

2.2.2 Τροποποιημένη Έκδοση Naïve Bayes

Με τα βήματα για την ολοκληρωμένη υλοποίηση της απλής έκδοσης να έχουν οριστεί με ξεκάθαρο και λιτό τρόπο, γίνεται εύκολος ο χειρισμός της σε μια προσπάθεια βελτίωσης. Μοναδική διαφοροποίηση μεταξύ της απλής και της τροποποιημένης έκδοσης είναι η εφαρμογή της μεθόδου που περιγράφηκε προηγουμένως έτσι ώστε να φιλτραριστούν τα χαρακτηριστικά βάσει των εμφανίσεων τους στο σύνολο εκπαίδευσης, με αυτά που εμφανίζονται λιγότερο από έναν ορισμένο αριθμό εμφανίσεων να μην λαμβάνονται υπόψη. Έπειτα από δοκιμές επιλέχθηκε το όριο των **πέντε (5) εγγράφων** όπου και παρουσιάστηκε η βέλτιστη κατηγοριοποίηση και εκτέλεση.

```
Modified_NB(συλλογή_εισόδου)
{
      είσοδος = διάβασε(συλλογή_εισόδου).map(έγγραφο).map((ετικέτα, κείμενο))

      έγγραφα_dataframe = έγγραφα.μετατροπή_σε_DF("ετικέτα", "κείμενο")
      όροι = έγγραφα_dataframe.διαχωρισμός_όρων("όροι")

      έγγραφα_tf = όροι.TF("tf")
      έγγραφα_idf = όροι.IDF("idf").με_ελάχιστη_εμφάνιση_όρου(5)
      έγγραφα_tfidf =  υπολόγισε_TFIDF(έγγραφα_tf, έγγραφα_idf)

      σύνολο_εκπαίδευσης = έγγραφα_tfidf.ποσοστό(75%)
      σύνολο_ελέγχου = έγγραφα_tfidf.ποσοστό(25%)

      μοντέλο = NaïveBayes(σύνολο_εκπαίδευσης, σύνολο_ελέγχου)
}
```

Αλγόριθμος 2.21 - Υλοποίηση Τροποποιημένου Naïve Bayes Μοντέλου στο Spark

2.2.3 Support Vector Machines

Σε μια αντίστοιχη διαδικασία αυτής που ακολουθήθηκε για τις δύο εκδόσεις του αλγόριθμου Naïve Bayes, η υλοποίηση των Support Vector Machines ξεκινά διαβάζοντας τα δεδομένα εισόδου απομονώνοντας το κείμενο και την ετικέτα κατηγορίας από κάθε. Ύστερα η RDD δομή που προέκυψε μετατρέπεται σε DataFrame σύμφωνα με τα απαιτούμενα του Spark για την εφαρμογή των βαρών TFIDF, ώστε τα δεδομένα εκπαίδευσης και ελέγχου να βρίσκονται στην κατάλληλη μορφή για να δοθούν ως ορίσματα στην ειδική λειτουργία του Spark για την κατηγοριοποίηση με χρήση ενός γραμμικού μοντέλου SVM. Το μοντέλο αυτό μπορεί να παραμετροποιηθεί περαιτέρω για ό,τι αφορά τον μέγιστο αριθμό επαναλήψεων που θα εκτελεστούν, το όριο βάσει του οποίο θα ταξινομούνται τα οριακά στιγμιότυπα, την αλλαγή των βαρών απόστασης μεταξύ στιγμιοτύπων, τον προσδιορισμό της παραμέτρου κανονικοποίησης για το μοντέλο, και άλλα. Στην προκειμένη περίπτωση επιλέγεται να γίνει χρήση της πρώτης και τελευταίας μεθόδου παραμετροποίησης, ορίζοντας τον αριθμό των επαναλήψεων εκτέλεσης του μοντέλου τόσο όσο η κατηγοριοποίηση να είναι ικανοποιητική δίχως να εμφανίζονται μεγάλες καθυστερήσεις, καθώς και την παράμετρο κανονικοποίησης του μοντέλου για να δηλωθεί ο βαθμός κατά τον οποίο το μοντέλο θα δεχθεί εσφαλμένες ταξινομήσεις στιγμιοτύπων. Μετά από μελέτη προτιμήθηκε να οριστεί το μέγιστο πλήθος επαναλήψεων εκτέλεσης του αλγορίθμου στις **δέκα φορές** και να οριστεί η παράμετρος κανονικοποίησης στην ποσότητα του **μηδέν κόμμα δύο** (**0.2**) όπου και παρουσιάστηκε η βέλτιστη κατηγοριοποίηση και εκτέλεση.

```
SVM(συλλογή_εισόδου)
{
      είσοδος = διάβασε(συλλογή_εισόδου).map(έγγραφο).map((ετικέτα, κείμενο))

      έγγραφα_dataframe = έγγραφα.μετατροπή_σε_DF(“ετικέτα”, “κείμενο”)
      όροι = έγγραφα_dataframe.διαχωρισμός_όρων(“όροι”)

      έγγραφα_tf = όροι.TF(“tf”)
      έγγραφα_idf = όροι.IDF(“idf”)
      έγγραφα_tfidf =  υπολόγισε_TFIDF(έγγραφα_tf, έγγραφα_idf)

      σύνολο_εκπαίδευσης = έγγραφα_tfidf.ποσοστό(75%)
      σύνολο_ελέγχου = έγγραφα_tfidf.ποσοστό(25%)

      μοντέλο = SVM(σύνολο_εκπαίδευσης, σύνολο_ελέγχου).επαναλήψεις(10)
                              .κανονικοποίηση(0.2)
}
```

Αλγόριθμος 2.22 - Υλοποίηση Support Machine Vectors Μοντέλου στο Spark

2.2.4 Τροποποιημένη Έκδοση Support Vector Machines

Και εδώ η μοναδική διαφορά μεταξύ των εκδόσεων για λογαριασμό του ίδιο αλγόριθμου είναι η εφαρμογή της επιλογής χαρακτηριστικών που παρουσιάζονται ως τα πιο σχετικά, υποδεικνύοντας ως λιγότερο σχετικά αυτά που έχουν τον μικρότερο αριθμό εμφανίσεων σε έγγραφα της συλλογής εισόδου. Για λόγους συνέπειας και απευθείας σύγκρισης των αποτελεσμάτων με τα αντίστοιχα των υλοποιήσεων για τον αλγόριθμο του Naïve Bayes, επιλέχθηκε και εδώ το όριο των **πέντε (5) εγγράφων** για τον εντοπισμό των χαρακτηριστικών με τις λιγότερες εμφανίσεις και την ρύθμιση του μέτρου IDF τους ανάλογα, χωρίς περαιτέρω αλλαγές στα βήματα που εκτελεί η υλοποίηση εσωτερικά.

```
Modified_SVM(συλλογή_εισόδου)
{
      είσοδος = διάβασε(συλλογή_εισόδου).map(έγγραφο).map((ετικέτα, κείμενο))

      έγγραφα_dataframe = έγγραφα.μετατροπή_σε_DF(“ετικέτα”, “κείμενο”)
      όροι = έγγραφα_dataframe.διαχωρισμός_όρων(“όροι”)

      έγγραφα_tf = όροι.TF(“tf”)
      έγγραφα_idf = όροι.IDF(“idf”).με_ελάχιστη_εμφάνιση_όρου(5)
      έγγραφα_tfidf =  υπολόγισε_TFIDF(έγγραφα_tf, έγγραφα_idf)

      σύνολο_εκπαίδευσης = έγγραφα_tfidf.ποσοστό(75%)
      σύνολο_ελέγχου = έγγραφα_tfidf.ποσοστό(25%)

          μοντέλο = SVM(σύνολο_εκπαίδευσης, σύνολο_ελέγχου).επαναλήψεις(10)
                                .κανονικοποίηση(0.2)
}
```

Αλγόριθμος 2.23 - Υλοποίηση Τροποποιημένου Support Machine Vectors Μοντέλου στο Spark

3. Πειραματικά Αποτελέσματα και Αξιολόγηση

Ως πρώτος και πιο θεμελιώδης παράγοντας για να καταστεί δυνατή η αξιολόγηση των μετρήσεων ορίζεται το περιβάλλον στο οποίο θα εξεταστούν οι υλοποιήσεις που έχουν αναπτυχθεί στο πλαίσιο της εργασίας. Είναι αντιληπτό από ό,τι αναφέρθηκε ως τώρα πως χρειάζεται σαν εξοπλισμός μια συστοιχία υπολογιστών που θα αποτελείται από έναν αριθμό κόμβων ο οποίος θα μπορεί να μεταβάλλεται για τους σκοπούς της εξέτασης κάθε υλοποίησης στις μετρήσεις επί της επιτάχυνσης (speedup, όπως θα δούμε παρακάτω) που χρειάζεται την προσθήκη περισσότερων κόμβων στην συστοιχία. Για την πειραματική υλοποίηση και αξιολόγηση των εν λόγω εφαρμογών χρησιμοποιήθηκε εργαστηριακός εξοπλισμός που φιλοξενείται στους χώρους του τμήματος Μηχανικών Πληροφορικής και Υπολογιστών στο Πανεπιστήμιο Δυτικής Αττικής. Η συστοιχία που προέκυψε αποτελείται από **τρεις (3) υπολογιστές** με τα χαρακτηριστικά που παρουσιάζονται στον παρακάτω πίνακα.

Χαρακτηριστικό	Εξοπλισμός Κόμβου
Επεξεργαστής	Intel i3-4160, 3.6GHz
Πυρήνες	2
Νήματα	4
Κύρια Μνήμη	8GB
Αποθηκευτικός Χώρος	250GB

Πίνακας 3.1 - Χαρακτηριστικά Κόμβων Συστοιχίας Πειραματικής Αξιολόγησης

Σε κάθε κόμβο βρίσκονται ρυθμισμένες κατάλληλα οι εγκαταστάσεις των πλατφόρμων των Hadoop και Spark στις τελευταίες ευσταθείς εκδόσεις που ήταν διαθέσιμες κατά την διάρκεια των μετρήσεων (με το **Hadoop** να εγκαθίσταται για την έκδοση **3.3.0** και το **Spark** για την έκδοση **3.0.1**). Το σύνολο των κόμβων στην συστοιχία είναι συνδεδεμένοι μεταξύ τους σε ένα **τοπικό δίκτυο ταχύτητας ενός (1) gigabyte** και φυσικά υποστηρίζεται το κατανεμημένο σύστημα αρχείων HDFS που είναι συμβατό με υλοποιήσεις και και στις δύο πλατφόρμες..

Περνώντας στην είσοδο των υλοποιήσεων που αναπτύχθηκαν έως τώρα, ιδιαίτερο ενδιαφέρον έχει η επιλογή της μορφής και του είδους των δεδομένων που θα δοθούν για την εκπαίδευση και τον έλεγχο ενός μοντέλου κατηγοριοποίησης. Η προφανής μορφή ενός ή περισσότερων αρχείων κειμένου μπορεί να εστιαστεί περισσότερο στην σάρωση εγγράφων σε μορφή γραμμής έτσι ώστε να μπορεί να εφαρμοστεί εύκολα η λειτουργία της συνάρτησης Map (και στις δύο πλατφόρμες) κατά την προεπεξεργασία των συλλογών εισόδου. Μία απεικόνιση εγγράφων που μπορεί να εξυπηρετήσει αυτή την ιδέα αυτή των αρχείων τύπου **CSV** (Comma-Separated Values) όπου κάθε γραμμή θεωρείται ως μία ξεχωριστή εγγραφή που αποτελείται από στήλες που περιέχουν διαφορετικά στοιχεία και χωρίζονται μεταξύ τους με κόμμα. Μέσω αρχείων τέτοιας μορφής θα μπορούσε να παρουσιαστεί σε κάθε γραμμή το κείμενο κάθε στιγμιοτύπου μαζί με την ετικέτα κατηγορίας και το αναγνωριστικό του, δηλαδή τα στοιχεία τα οποία χρησιμοποιήθηκαν στην σχεδίαση των αλγορίθμων για τον χαρακτηρισμό κάθε ενός εγγράφου.

Στην συνέχεια τίθεται το θέμα επιλογής του τύπου δεδομένων εισόδου που θα εφαρμοστεί στις υλοποιήσεις. Κατά την αναζήτηση διπλωματικών και δημοσιευμένων εργασιών που το αντικείμενο τους ταυτίζεται θεματικά με αυτό της παρούσας μελέτης, παρατηρήθηκε το φαινόμενο επιλογής μερικών εξαιρετικά κοινών συνόλων δεδομένων (όπως για παράδειγμα η συλλογή κειμένου 20Newsgroups) ως είσοδο. Αυτές οι συλλογές αφορούν ως επί το πλείστον υποδειγματικές εκτελέσεις (που αφορούν συνήθως την κατηγοριοποίηση ή την συσταδοποίηση δειγμάτων) για την παρουσίαση της εύρυθμης λειτουργίας μιας υλοποίησης, όμως σε μια προσπάθεια επικαιροποίησης των υλοποιήσεων υπό όρους που προσεγγίζουν ρεαλιστικές εφαρμογές, αποφεύχθηκε η επιλογή τέτοιων παραδοσιακών συλλογών για χάρη κάποιου συνόλου που μπορεί να βρίσκεται στην κλίμακα των big data (έστω και θεωρητικά, χωρίς να χρειαστεί ενδεχομένως να εφαρμοστούν μεγέθη εισόδου gigabytes στις υλοποιήσεις που θα εξεταστούν) και αφορά γενικότερα ανάγκες του παρόντος. Προτιμήθηκε έτσι η χρήση της συλλογής κειμένου **Sentiment140** που περιέχει **1.578.627 tweets** από το Twitter, τα οποία παρουσιάζονται ανά γραμμή και περιέχουν τα στοιχεία που αναφέρθηκαν προηγουμένως σε μορφή CSV. Τα στιγμιότυπα που την αποτελούν φέρουν μία από τις δύο ετικέτες κατηγοριών που υποδηλώνουν την θετική ή αρνητική συναισθηματική πολικότητα που εκφράζει κάθε ένα από αυτά, ενώ το μέγεθος της συλλογής σε συνδυασμό με τον τύπο των στιγμιοτύπων αυτών καθαυτών επιτρέπουν την ικανοποιητική μελέτη της λειτουργίας των υλοποιήσεων σε επίπεδο ορθής ταξινόμησης και παράλληλης εκτέλεσης, καθώς και στην γενίκευση χρήσης αυτών των υλοποιήσεων για την εφαρμογή τους πάνω σε νέα σύνολα δεδομένων τέτοιου τύπου.

Επιπλέον χρειάζεται να προσδιοριστεί το ποσοστό επί κάθε συνόλου που θα αφιερωθεί για την εκπαίδευση και τον έλεγχο κάθε μοντέλου. Η επιλογή αυτών των ποσοστών δεν ακολουθεί τόσο μια προκαθορισμένη θεωρία επί του μοντέλου αυτού καθ' αυτού, παρά εξαρτάται από την ίδια την συλλογή δεδομένων εισόδου που αναφέρονται. Γενικότερα σε εφαρμογές μηχανικής μάθησης υπάρχει η ανάγκη εύρεσης μιας ισορροπίας μεταξύ των ποσοτήτων εκπαίδευσης και ελέγχου έτσι ώστε να είναι τα δεδομένα εκπαίδευσης αρκετά και αντιπροσωπευτικά, την ίδια στιγμή που τα δεδομένα ελέγχου δεν πρέπει να είναι ούτε τόσα πολλά ώστε να στερούν σημαντική λεπτομέρεια για την κατασκευή του μοντέλου αλλά ούτε και τόσα λίγα ώστε να δείχνουν στα αποτελέσματα μια λανθασμένη επίδοση υψηλής ή χαμηλής ορθότητας λόγω μικρού αριθμού δειγμάτων προς έλεγχο. Για να προσδιοριστεί η διαίρεση της συλλογής εισόδου σε ποσοστά χρειάζεται να γίνουν δοκιμαστικές εκτελέσεις των υλοποιήσεων για λογαριασμό του ίδιου συνόλου με διαφορετικές ποσότητες εκπαίδευσης και ελέγχου κάθε φορά. Στις δοκιμές που έγιναν στην προκειμένη περίπτωση η καλύτερη επίδοση στα αποτελέσματα παρουσιάστηκε όταν χρησιμοποιήθηκε **το 75% των εγγράφων για την εκπαίδευση** και **το 25% των εγγράφων για τον έλεγχο του μοντέλου.**

Όπως επισημάνθηκε και προηγουμένως, για να κριθεί πλήρης η αξιολόγηση κάθε υλοποίησης χρειάζεται να μελετηθούν οι εκτελέσεις της με τέτοιο τρόπο που να εξετάζονται όλοι οι παράγοντες που αφορούν την επίδοση της σε ό,τι αφορά την κατηγοριοποίηση και την εκτέλεση σε κατανεμημένο περιβάλλον. Λόγω αυτής της ανάγκης οι μετρήσεις που προέκυψαν για τους σκοπούς της παρούσας ενότητας χωρίζονται στη συνέχεια ανά είδος επίδοσης λόγω της διττής φύσης των υλοποιήσεων, πριν αξιολογηθούν τα αποτελέσματα συνολικά στην τελευταία και συμπερασματική ενότητα.

3.1 Επίδοση Κατηγοριοποίησης

Η αξιολόγηση μιας εφαρμογής κατηγοριοποίησης δεδομένων αφορά κυρίως το κατά πόσο ένα συγκεκριμένο μοντέλο ταξινόμησης των δειγμάτων καταφέρνει να κατηγοριοποιεί ορθά τα έγγραφα ελέγχου. Για να εκφραστεί η αποτελεσματικότητα μιας τέτοιας υλοποίησης χρησιμοποιείται όπως προαναφέρθηκε η μήτρα σύγχυσης που παρουσιάζει τα πλήθη των δειγμάτων που ταξινομήθηκαν βάσει

της σύγκρισης της ετικέτας κατηγορίας ενός δείγματος έναντι της πρόβλεψης του μοντέλου. Από αυτές τις ποσότητες μπορούν να εξαχθούν μετρικές που ποσοτικοποιούν την αποτελεσματικότητα. Το πιο εύλογο μέτρο βάσει του οποίου μπορεί να αξιολογηθεί ένα μοντέλο είναι φυσικά η **ορθότητα** (accuracy) του, δηλαδή η ποσότητα των σωστών προβλέψεων του προς το σύνολο των προβλέψεων, χρησιμοποιώντας τους παράγοντες της μήτρας σύγχυσης όπως φαίνεται στην παρακάτω σχέση.

$$A = \frac{TP + TN}{TP + FP + FN + TN}$$

Σχέση 3.1 - Σχέση Υπολογισμού Ορθότητας A (Accuracy) Μοντέλου

Παρόλα αυτά η ορθότητα ενδέχεται να μην είναι το καταλληλότερο μέτρο για να αξιολογηθεί μία υλοποίηση αφού μπορεί να δείχνει ως υψηλά αποδοτικό ένα μοντέλο που ταξινομεί καλά τα δείγματα που ανήκουν στην κατηγορία με τα περισσότερα δείγματα στο σύνολο ελέγχου. Αυτό το φαινόμενο εντάσσεται γενικότερα στο πρόβλημα ανισορροπίας των στιγμιοτύπων, μιας και σε ορισμένες συλλογές εισόδου δεν εγγυάται ότι όλες οι κλάσεις θα έχουν ισόποσα δείγματα μεταξύ τους. Εναλλακτικά μπορούν να χρησιμοποιηθούν δύο άλλες μετρικές, της **ακρίβειας** (precision) και της **ανάκλησης** (recall), που χρησιμοποιούνται και πιο συχνά στην αξιολόγηση τέτοιων συστημάτων. Υπολογίζονται από τις σχέσεις που φαίνονται παρακάτω και επιστρέφουν ένα ποσοστό εγγράφων που έχουν ταξινομηθεί ορθά ως προς τα ανακτημένα και συναφή αντίστοιχα στιγμιότυπα..

$$P = \frac{TP}{TP + FP}$$

Σχέση 3.2 - Σχέση Υπολογισμού Ακρίβειας P (Precision) Μοντέλου για Θετική Κλάση

$$R = \frac{TP}{TP + FN}$$

Σχέση 3.3 - Σχέση Υπολογισμού Ανάκλησης R (Recall) Μοντέλου για Θετική Κλάση

Η χρήση των δύο ανεξάρτητων αυτών μέτρων αξιολόγησης παρακάμπτει μεν μια ενδεχομένως εσφαλμένη καλή επίδοση ενός συστήματος με τη χρήση της ορθότητας, όμως εκφράζουν την ακρίβεια και την ανάκληση αντίστοιχα μόνο για λογαριασμό μιας κλάσης κάθε φορά (με τις παραπάνω σχέσεις, για παράδειγμα, να αναφέρονται στην ακρίβεια και ανάκληση για έγγραφα της θετικής κλάσης). Για μια εκφραστεί επαρκώς μια υλοποίηση στο σύνολο των εγγράφων που δέχεται σαν είσοδο, χρειάζεται να συνδυαστούν η ακρίβεια και η ανάκληση κάθε κλάσης (εδώ συγκεκριμένα της θετικής και αρνητικής) ώστε να βρεθεί η σταθμισμένη τιμή τους. Οι σταθμισμένες τιμές της ακρίβειας και της ανάκλησης υπολογίζονται με την βοήθεια των σχέσεων που παρουσιάζονται στην συνέχεια.

$$P_{weighted} = \frac{(P_{C1} \cdot |C1|) + (P_{C2} \cdot |C2|)}{|C1| + |C2|}$$

Σχέση 3.4 - Σχέση Υπολογισμού Σταθμισμένης Ακρίβειας P (Precision) Μοντέλου

$$R_{weighted} = \frac{(R_{C1} \cdot |C1|) + (R_{C2} \cdot |C2|)}{|C1| + |C2|}$$

Σχέση 3.5 - Σχέση Υπολογισμού Σταθμισμένης Ανάκλησης R (Recall) Μοντέλου

Όπου:

- Ως P_{C1}, P_{C2} εκφράζονται οι ποσότητες ακρίβειας P για τις κλάσεις *C1, C2* αντίστοιχα.
- Ως R_{C1}, R_{C2} εκφράζονται οι ποσότητες ανάκλησης R για τις κλάσεις *C1, C2* αντίστοιχα.
- Ως $|C1|$, $|C2|$ εκφράζονται τα πλήθη των στιγμιοτύπων για τις κλάσεις *C1, C2* αντίστοιχα.

Συνεχίζοντας την μελέτη αυτών των δύο ποσοτήτων από άλλη σκοπιά, καταλήγει κανείς στο συμπέρασμα ότι αφού τα μέτρα αυτά είναι αμοιβαία αντισταθμιστικά μεταξύ τους (ως προς τις ποσότητες "false positive" και "false negative" που παρουσιάζονται για λογαριασμό κάθε κατηγορίας στον παρονομαστή του κλάσματος των σχέσεων τους), τότε μπορεί να παρουσιαστεί για κάποιο συγκεκριμένο σύνολο εισόδου υψηλή ακρίβεια και χαμηλή ανάκληση (ή και αντίστροφα).

Για την κατάλληλη αντιστάθμιση των δύο αυτών μέτρων ώστε να συνδυαστούν σε μία τιμή η οποία θα αφορά την αποτελεσματικότητα ενός μοντέλου, χρησιμοποιείται το **μέτρο F_1** (F_1 measure ή F_1 score) ή αλλιώς ο σταθμισμένος αρμονικός μέσος της ακρίβειας και της ανάκλησης που υπολογίζεται με την παρακάτω σχέση.

$$F_1 = \frac{2PR}{P + R}$$

Σχέση 3.6 - Σχέση Υπολογισμού μέτρου F_1 (F_1 measure) Μοντέλου για Θετική ή Αρνητική Κλάση

Αφού η σχέση του εμπεριέχει τα μέτρα της ακρίβειας και της ανάκλησης, το μέτρο **F_1** προσδιορίζεται και αυτό για κάθε κλάση ξεχωριστά, όμως εάν εφαρμοστούν στην σχέση του οι σταθμισμένες ποσότητες των δύο σχετικών μέτρων, η σταθμισμένη πια τιμή του μέτρου F_1. αξιολογεί την λειτουργία της εφαρμογής συνολικά.

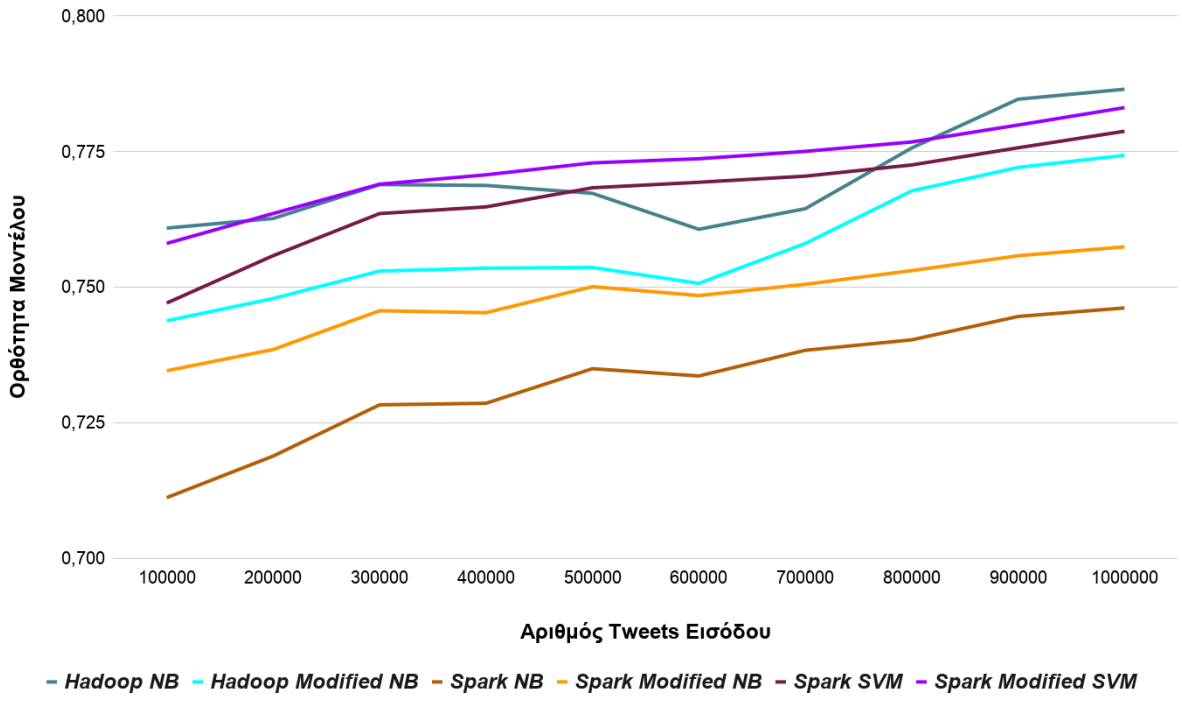
3.1.1 Ορθότητα (Accuracy)

Στο πλαίσιο της μελέτης της ορθότητας κάθε υλοποίησης στα στιγμιότυπα εισόδου δημιουργήθηκαν δέκα σύνολα εκπαίδευσης και ελέγχου για κάθε πλατφόρμα ξεχωριστά που αποτελούνται από εκατό χιλιάδες έως ένα εκατομμύριο στιγμιότυπα, τα οποία αυξάνονται και εδώ κατά εκατό χιλιάδες τη φορά. Κάθε βαθμίδα συνόλου περιέχει τα ίδια ακριβώς στιγμιότυπα, τηρώντας το ίδιο ποσοστό διαχωρισμού των στιγμιοτύπων ως 75% για την εκπαίδευση και 25% για τον έλεγχο κάθε μοντέλου.

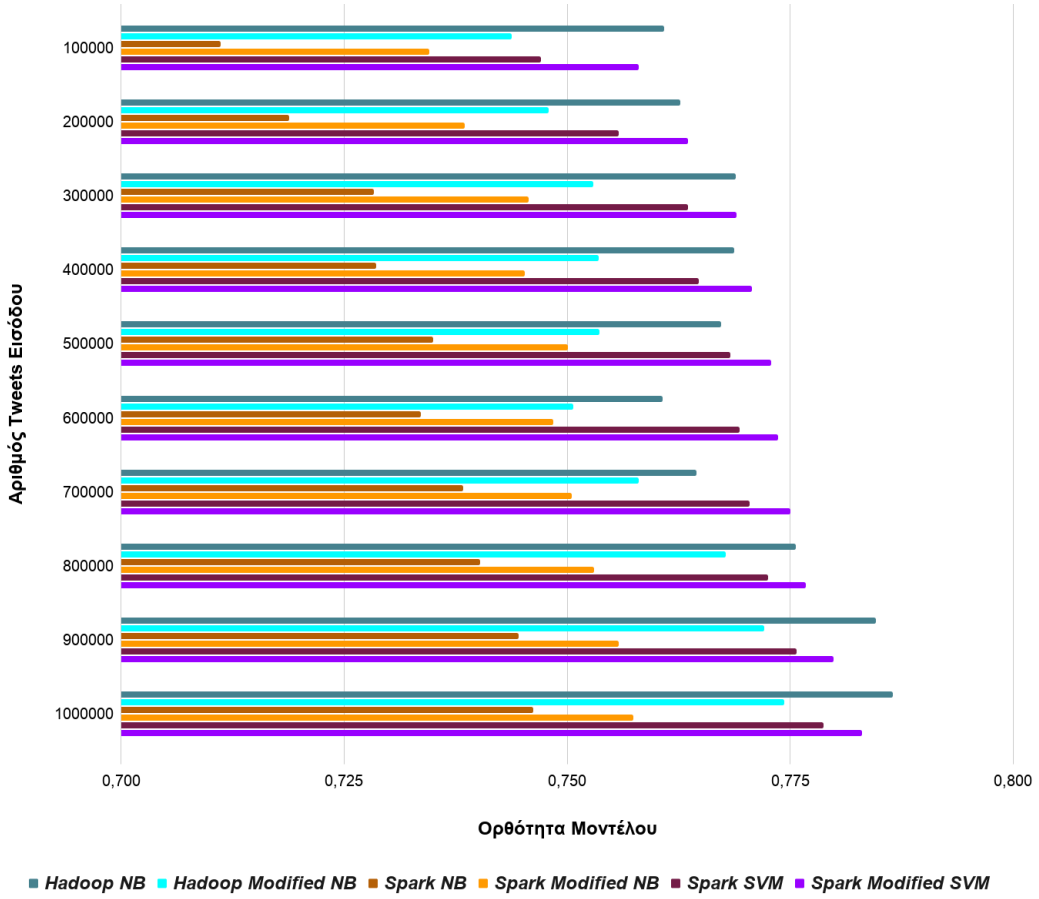
Η ορθότητα που δηλώνεται στον πίνακα παρακάτω εκφράζεται σε σχέση με την μονάδα που εκφράζει την τέλεια ταξινόμηση εγγράφων της εκάστοτε υλοποίησης για το συγκεκριμένο σύνολο εισόδου. Τα αποτελέσματα επιβεβαιώνονται μέσω της πολλαπλής εκτέλεσης κάθε υλοποίησης για τα ίδια σύνολα εισόδου ώστε να μην καταγραφεί κάποια εσφαλμένη μέτρηση (κάτι το οποίο μπορεί να συμβεί σε περίπτωση εφαρμογής λάθος συνόλων μεταξύ των εκτελέσεων, κάποιας τροποποίησης που έγινε εσωτερικά σε μια υλοποίηση μεταξύ των εκτελέσεων, ή κάποιου σφάλματος κατά την εκτέλεση στο πειραματικό περιβάλλον).

	Σύνολα Δειγμάτων Εισόδου (σε Χιλιάδες)									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Hadoop NB	0,761	0,763	0,769	0,769	0,766	0,761	0,763	0,776	0,785	0,785
Hadoop NB_M	0,744	0,748	0,753	0,754	0,754	0,751	0,758	0,768	0,771	0,773
Spark NB	0,711	0,719	0,727	0,726	0,735	0,734	0,737	0,740	0,745	0,745
Spark NB_M	0,735	0,739	0,746	0,744	0,75	0,747	0,751	0,752	0,756	0,756
Spark SVM	0,747	0,756	0,764	0,765	0,767	0,768	0,770	0,773	0,776	0,779
Spark SVM_M	0,758	0,764	0,769	0,771	0,773	0,774	0,775	0,777	0,780	0,783

Πίνακας 3.2 - Ορθότητα Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.1 - Γράφημα Γραμμών Ορθότητας Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.2 - Γράφημα Ράβδων Ορθότητας Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου

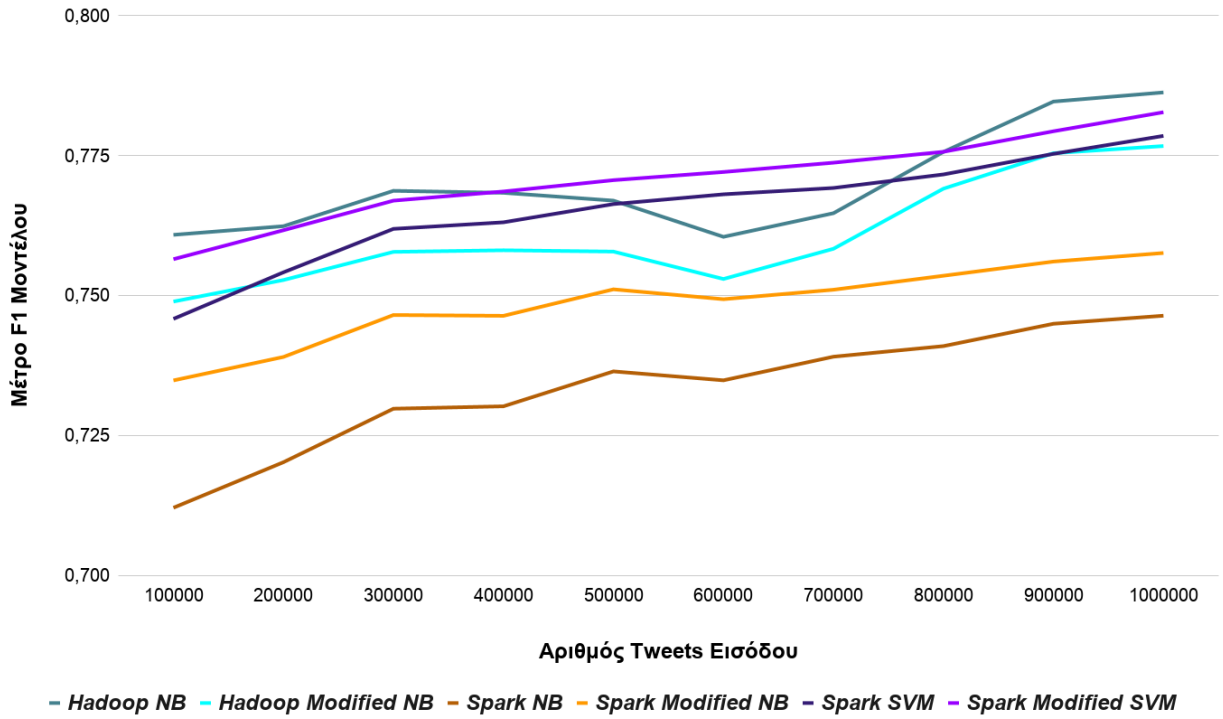
Από τα δεδομένα των μετρήσεων φαίνεται σε πρώτη ανάγνωση ότι στις δοκιμές με τις μεγαλύτερες συλλογές εισόδου κρίνεται πιο αποτελεσματική η απλή έκδοση του Naïve Bayes στο Hadoop, με τις δύο εκδόσεις των SVM εφαρμογών στο Spark και την τροποποιημένη έκδοση του Naïve Bayes στο Hadoop να ακολουθούν. Παρόλα αυτά γίνεται αντιληπτό ότι η απλή έκδοση του Naïve Bayes στο Hadoop παρουσιάζει αρκετή αναποτελεσματικότητα στην περιοχή των μεσαίων συλλογών εισόδου (μια συμπεριφορά που ακολουθεί αντίστοιχα και η αντίστοιχη τροποποιημένη έκδοση), σε σχέση με τις εφαρμογές των SVM μοντέλων που ακολουθούν μια σταθερή ανάπτυξη στην ορθότητα των προβλέψεων τους. Σε ό,τι αφορά τις υλοποιήσεις του Naïve Bayes στο Spark, αυτές κυμαίνονται σε αρκετά χαμηλότερα ποσοστά με την τροποποιημένη έκδοση να πετυχαίνει αρκετά μεγαλύτερη ορθότητα από την απλή έκδοση.

3.1.2 Μέτρο F_1 (F_1 Measure)

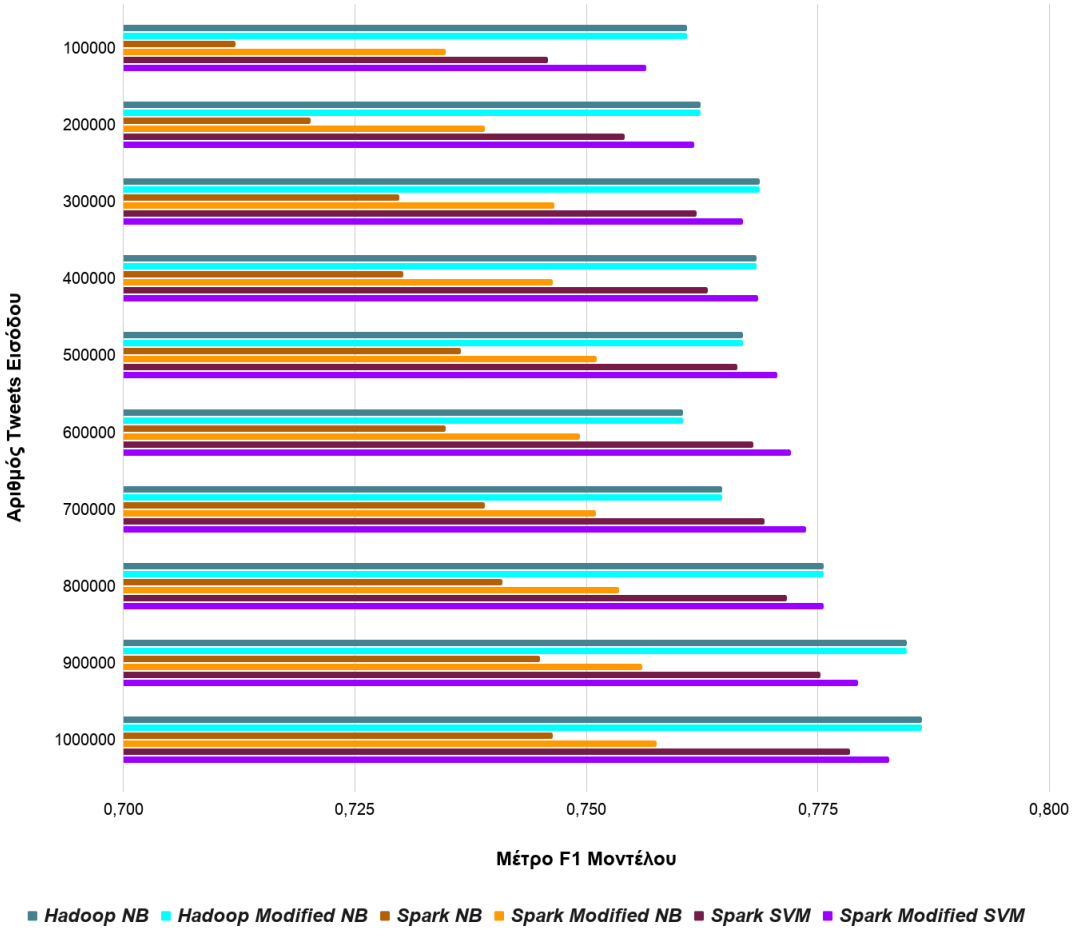
Συμπληρωματικά με την καταμέτρηση της ορθότητας, εφαρμόζονται τα ίδια δέκα (10) σύνολα εκπαίδευσης και ελέγχου για τον υπολογισμό της μέσης τιμής του μέτρου F_1. Με χρήση αυτού του μέτρου αναπαριστάται καλύτερα η επίδραση των λανθασμένα ταξινομημένων εγγράφων σε κάθε κατηγορία παρουσιάζοντας τον αρμονικό μέσο μεταξύ ακρίβειας και ανάκλησης, με το μέτρο F_1 να τείνει προς στο μικρότερο εξ αυτών έτσι ώστε μια υψηλή τιμή του να υποδηλώνει ότι είναι αντίστοιχα υψηλά και τα δύο μέτρα αξιολόγησης που περιέχει. Οι τιμές του μέτρου F_1 κινούνται αντίστοιχα με την ορθότητα σε σχέση με την μονάδα που δείχνει το καλύτερο αποτέλεσμα κατηγοριοποίησης που μπορεί να επιτευχθεί με όλα τα δείγματα να ταξινομούνται σωστά.

	Σύνολα Δειγμάτων Εισόδου (σε Χιλιάδες)									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Hadoop NB	0,761	0,761	0,769	0,767	0,767	0,759	0,765	0,776	0,785	0,785
Hadoop NB_M	0,749	0,753	0,758	0,758	0,758	0,753	0,757	0,769	0,774	0,777
Spark NB	0,711	0,719	0,73	0,729	0,735	0,735	0,738	0,741	0,745	0,745
Spark NB_M	0,735	0,738	0,747	0,745	0,75	0,748	0,75	0,754	0,755	0,758
Spark SVM	0,746	0,753	0,762	0,763	0,765	0,768	0,769	0,773	0,774	0,779
Spark SVM_M	0,755	0,762	0,767	0,769	0,771	0,772	0,775	0,776	0,778	0,783

Πίνακας 3.3 - Μέτρο F_1 Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.3 - Γράφημα Γραμμών Μέτρου F_1 Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.4 - Γράφημα Ράβδων Μέτρου F_1 Κατηγοριοποίησης Δειγμάτων Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου

Σε μια αντίστοιχη μελέτη των υλοποιήσεων για την αποτελεσματικότητα των προβλέψεων κάθε μοντέλου μέσω του μέτρου F_1, επαληθεύεται η αστάθεια που εμφανίζεται για λογαριασμό της απλής έκδοσης του Naïve Bayes στην πλατφόρμα του Hadoop με την τροποποιημένη έκδοση να ακολουθεί πιστά αυτό το μοτίβο. Αντίστοιχα επαληθεύεται και η σταθερή επίδοση των δύο εφαρμογών SVM στο Spark πετυχαίνοντας υψηλές τιμές στο μέτρο F_1, όταν οι υπόλοιπες δύο εφαρμογές για το Naïve Bayes αλγόριθμο στο Spark σημειώνουν αισθητά πιο χαμηλή αποτελεσματικότητα στις προβλέψεις των μοντέλων τους.

Με αυτό τον τρόπο η ανάλυση της επίδοσης της ποιότητας ταξινόμησης των υλοποιήσεων καταλήγει στην διαλογή των τεσσάρων πρώτων σε αποτελεσματικότητα εφαρμογών, αφού οι βασισμένες σε Spark εφαρμογές του Naïve Bayes απέχουν αρκετά τις υψηλότερες τιμές των κριτηρίων ορθότητας και F_1 μέτρου που καταγράφηκαν σε αυτό το σκέλος της αξιολόγησης.

3.2 Επίδοση Παράλληλης Εκτέλεσης

Περνώντας στην αξιολόγηση των υλοποιήσεων σε ό,τι αφορά τον παράλληλο υπολογισμό, χρειάζεται να βρεθούν οι μετρικές που περιγράφουν επαρκώς την επίδοση κάθε μίας από αυτές. Σε αντίθεση με την αξιολόγηση μιας εφαρμογής στο πλαίσιο της ορθής κατηγοριοποίησης, οι μετρήσεις εδώ απασχολούν την μελέτη για την συμπεριφορά των υλοποιήσεων βάσει του περιβάλλοντος στο οποίο εκτελούνται. Ο πιο προφανής παράγοντας για την αξιολόγηση ενός προγράμματος σε αυτούς

τους όρους είναι φυσικά ο **χρόνος εκτέλεσης** ώστε να προτιμηθεί η υλοποίηση που ολοκληρώνεται στον μικρότερο δυνατό χρόνο, όμως κάτι τέτοιο δεν αρκεί για μια ολοκληρωμένη ανάλυση πάνω σε εφαρμογές που εκτελούνται με παραλληλο τρόπο σε περιβάλλον συστοιχίας.

Έχοντας πάντα υπόψη ότι η χρήση των ανεπτυγμένων εφαρμογών γενικεύεται από δεδομένα συνηθισμένων μεγεθών σε δεδομένα της κλίμακας των big data, πρέπει να υπάρχει κάποια μέριμνα για να παρουσιαστεί η επίδοση κάθε υλοποίησης σε διαφορετικά μεγέθη δεδομένων για να ελεγχθεί εάν ο χρόνος εκτέλεσης μεταβάλλεται κλιμακωτά όσο μεγαλώνει ο όγκος των συνόλων εισόδου. Κάπως έτσι, με την χρήση ενός αριθμού συνόλων δεδομένων που διαφέρουν μεταξύ τους σε μέγεθος, μπορεί να ελεγχθεί κάθε μία υλοποίηση ως προς τον χρόνο που απαιτεί για την ολοκλήρωση της εκτέλεσης του ανάλογα με τον φόρτο δεδομένων που καλείται να διεκπεραιώσει, ή με άλλα λόγια να εξεταστεί η **κλιμακωσιμότητα** των υλοποιήσεων. Μια εφαρμογή γενικότερα θεωρείται κλιμακώσιμη όταν ο παραλληλισμός της αυξάνεται (κατά προσέγγιση) γραμμικά με τον όγκο των δεδομένων στην είσοδο, διατηρώντας ίση διαχείριση και επεξεργασία του φόρτου εργασίας για κάθε κόμβο όσο ο όγκος αυτός αυξάνεται.

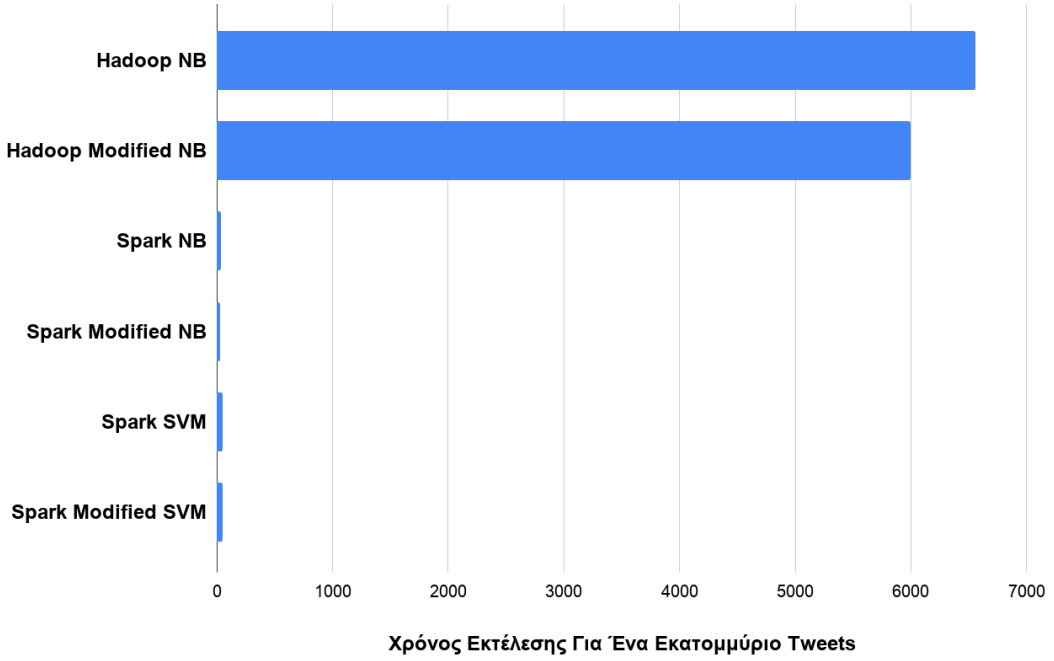
Σε ένα περαιτέρω βήμα χρειάζεται να εξεταστεί και η συμπεριφορά κάθε υλοποίησης σε συνάρτηση με τον αριθμό κόμβων που αποτελούν την συστοιχία σε κάθε δεδομένη στιγμή, για να μελετηθεί η βελτιστοποίηση που παρουσιάζεται στον χρόνο εκτέλεσης όσο προστίθενται περισσότεροι υπολογιστές στην υποδομή. Με άλλα λόγια δηλαδή χρειάζεται να μετρηθεί η **επιτάχυνση** για λογαριασμό κάθε υλοποίησης ώστε να παρουσιαστεί η διαφορά στο χρόνο εκτέλεσης της σε περιβάλλον συστοιχίας (από τον ελάχιστο αριθμό που δικαιολογεί την ύπαρξη μιας συστοιχίας ως το σύνολο των κόμβων που την αποτελούν) έναντι του χρόνου εκτέλεσης που απαιτείται σε μόλις έναν υπολογιστικό κόμβο. Εννοείται εδώ ότι όσο ελαχιστοποιείται ο χρόνος εκτέλεσης με την πρόσθεση κόμβων στην συστοιχία τόσο αυξάνεται η επιτάχυνση.

3.2.1 Χρόνος Εκτέλεσης (Execution Time)

Οι μετρήσεις που αφορούν τον απαιτούμενο χρόνο εκτέλεσης κάθε υλοποίησης αναφέρονται σε μια συλλογή δεδομένων που αποτελείται συνολικά από ένα εκατομμύριο (1.000.000) tweets, όπου το 75% αυτών εφαρμόζεται σαν σύνολο εκπαίδευσης και το υπόλοιπο 25% σαν σύνολο ελέγχου σύμφωνα με τις συμβάσεις που καθορίστηκαν εξ αρχής. Κάθε υλοποίηση εκτελέστηκε τρεις φορές ξεχωριστά για την διασφάλιση της εγκυρότητας των χρόνων ώστε να υπολογιστεί ο μέσος όρος αυτών, ο οποίος και καταγράφεται παρακάτω.

	Hadoop		**Spark**			
	NB	**NB_M**	**NB**	**NB_M**	**SVM**	**SVM_M**
Χρόνος Εκτέλεσης	6561	6000	32	31	48	49

Πίνακας 3.4 - Χρόνος Εκτέλεσης (σε δευτερόλεπτα) Κάθε Υλοποίησης Για Ένα Εκατομμύριο Δείγματα Εισόδου



Γράφημα 3.5 - Γράφημα Ράβδων Χρόνου Εκτέλεσης (σε δευτερόλεπτα) Κάθε Υλοποίησης Για Ένα Εκατομμύριο Δείγματα Εισόδου

Όπως ήταν ήδη αναμενόμενο από όσα έχουν μελετηθεί ως τώρα για τις πλατφόρμες στις οποίες βασίζονται οι υλοποιήσεις, παρατηρείται σημαντική διαφορά στο χρόνο που απαιτείται για την ολοκλήρωση των εφαρμογών στο Hadoop έναντι εκείνων που δρουν στο Spark. Μεταξύ των εφαρμογών στο Hadoop υπάρχει μια διαφορά περίπου εννέα (9) λεπτών, με την απλή έκδοση του Naïve Bayes να κρίνεται η πιο χρονοβόρα πιθανώς λόγω του μεγάλου αριθμού χαρακτηριστικών που καλείται η συγκεκριμένη υλοποίηση να διαχειριστεί. Σε ό,τι αφορά τις υλοποιήσεις στο Spark, οι διαφορές στον χρόνο εκτέλεσης μεταξύ των εκδόσεων κάθε μεθόδου δεν ξεπερνούν το ένα δευτερόλεπτο, ενώ μεταξύ Naïve Bayes και SVM οι εφαρμογές απέχουν μόλις από πέντε έως οκτώ δευτερόλεπτα.

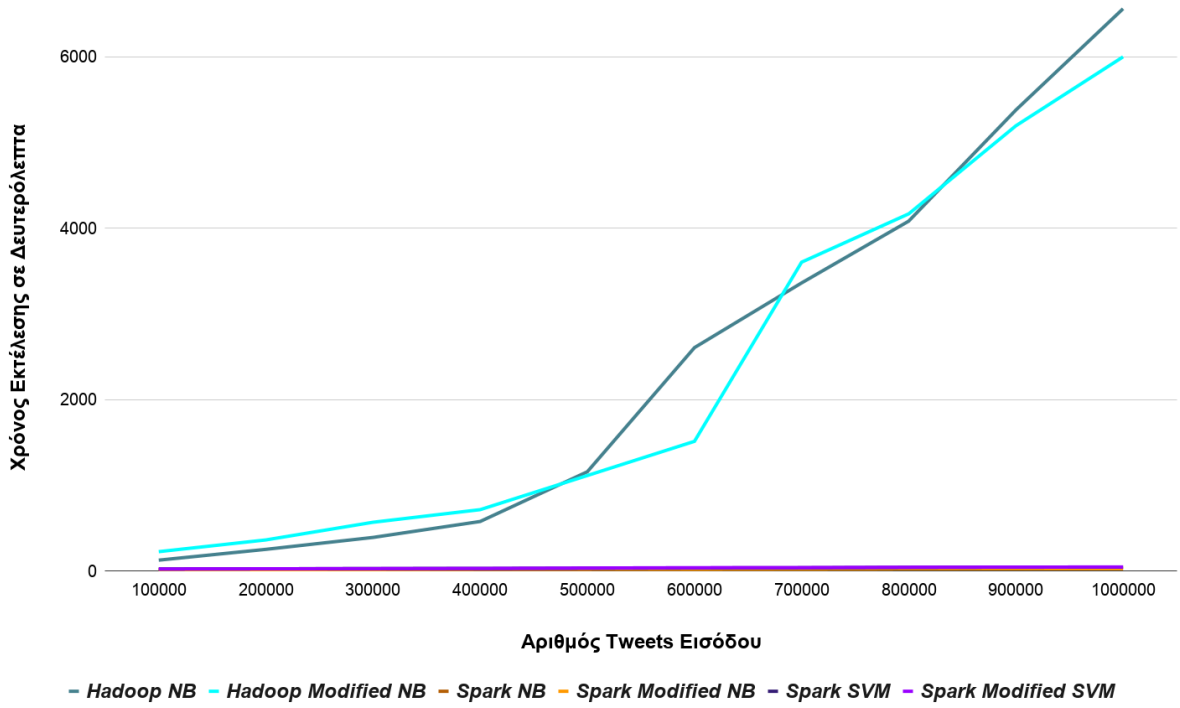
Είναι ξεκάθαρο από τώρα ότι χρησιμοποιώντας ως αποκλειστικό κριτήριο επίδοσης τον χρόνο εκτέλεσης των εφαρμογών, η αξιολόγηση καταλήγει στην προτίμηση οποιασδήποτε υλοποίησης που βασίζεται στην πλατφόρμα του Spark λόγω της τεράστιας εξοικονόμησης χρόνου που σημειώνεται έναντι των καταφανώς χρονοβόρων υλοποιήσεων στο Hadoop. Παρόλα αυτά χρειάζεται να εξεταστεί η συμπεριφορά που μπορεί να έχει κάθε υλοποίηση για διαφορετικές κλίμακες δεδομένων εισόδου ώστε να επιβεβαιωθεί ότι ο χρόνος εκτέλεσης της είναι ικανοποιητικός και συνεπώς ανάλογος με τον όγκο των στιγμιοτύπων που δέχεται στην είσοδο.

3.2.2 Κλιμακωσιμότητα (Scalability)

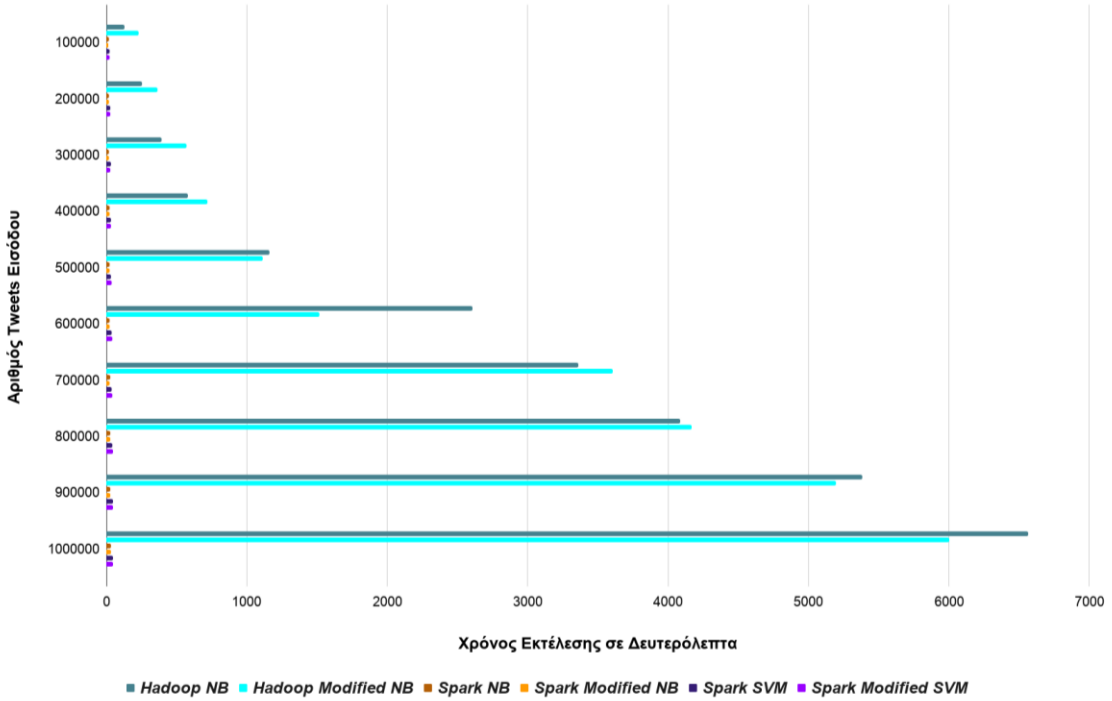
Για την μελέτη της συμπεριφοράς κάθε εφαρμογής στο πλαίσιο της κλιμακωσιμότητας, εξετάστηκε ο χρόνος εκτέλεσης κάθε μίας για λογαριασμό δέκα συλλογών κειμένου (που περιλαμβάνουν από εκατό χιλιάδες έως ένα εκατομμύριο tweets) που χρησιμοποιήθηκαν και προηγουμένως για την λήψη μετρήσεων σχετικά με την ορθότητα και το μέτρο F_1. Για κάθε υλοποίηση και κάθε συλλογή δεδομένων οι εφαρμογές εκτελέστηκαν τρεις φορές ξεχωριστά και υπολογίστηκε ο μέσος όρος των χρόνων που καταγράφηκαν για να μην επηρεαστούν οι τιμές του παρακάτω πίνακα από ενδεχόμενες στιγμιαίες αστοχίες.

	Σύνολα Δειγμάτων Εισόδου (σε Χιλιάδες)									
	100	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Hadoop NB	129	255	394	580	1160	2608	3363	4084	5380	6561
Hadoop NB_M	228	365	571	718	1116	1515	3605	4169	5196	6000
Spark NB	18	19	21	23	24	25	27	28	30	32
Spark NB_M	16	18	20	23	24	25	26	28	30	31
Spark SVM	26	28	31	32	34	38	40	44	47	48
Spark SVM_M	26	28	30	32	37	41	42	46	48	49

Πίνακας 3.5 - Χρόνος Εκτέλεσης (σε δευτερόλεπτα) Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.6 - Γράφημα Γραμμών Χρόνου Εκτέλεσης (σε δευτερόλεπτα) Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου



Γράφημα 3.7 - Γράφημα Ράβδων Χρόνου Εκτέλεσης (σε δευτερόλεπτα) Κάθε Υλοποίησης Ανά Αριθμό Δειγμάτων Εισόδου

Επιβεβαιώνεται εδώ η σημαντική διαφορά στους απαιτούμενους χρόνους μεταξύ των υλοποιήσεων σε διαφορετικές πλατφόρμες. Για λογαριασμό των ίδιων από άποψη όγκου συλλογής εγγράφων, οι υλοποιήσεις στο Hadoop έχουν μια σχετικά σταθερή αλλά και μεγάλη διαφορά μεταξύ διαδοχικών συλλογών, αφού οι χρόνοι εκτέλεσης για τα μικρότερα σετ εισόδων κυμαίνονται σε επιδόσεις λίγων λεπτών ενώ για τα μεγαλύτερα σετ χρειάζεται κάτι παραπάνω από ώρα για την ολοκλήρωση μιας εφαρμογής. Αντιθέτως οι υλοποιήσεις βασισμένες στο Spark δείχνουν μια κλιμακωτή συμπεριφορά που μοιάζει ιδιαίτερα ικανοποιητική αφού για διαδοχικά σετ εισόδου παρατηρούνται εξαιρετικά μικρές διαφορές λίγων δευτερολέπτων με αποτέλεσμα καμία εκτέλεση αυτών των εφαρμογών κατά την λήψη μετρήσεων να χρειάζεται παραπάνω από ένα λεπτό για την ολοκλήρωση της.

Με μια επιπλέον ανάγνωση στα αποτελέσματα γίνεται μεν αντιληπτή μια διαφορά που κυμαίνεται περίπου από δέκα έως είκοσι δευτερόλεπτα υπέρ των υλοποιήσεων για τον Naïve Bayes αλγόριθμο, όμως δεν πρέπει να παραλείπεται ότι οι μετρήσεις που βασίζουν την αξιολόγηση μέχρι τώρα αναφέρονται στο καλύτερο σενάριο κάνοντας χρήση όλων των κόμβων στην συστοιχία που χρησιμοποιήθηκε στο πλαίσιο της εργασίας. Για αυτό τον λόγο είναι χρήσιμη η διερεύνηση της επίδοσης των εφαρμογών στις διάφορες περιπτώσεις που προκύπτουν ανά αριθμό κόμβων συστοιχίας για να επαληθευτεί από κάθε άποψη η αποτελεσματικότητα κάθε μίας από αυτές σε ό,τι αφορά την παράλληλη εκτέλεση τους σε κατανεμημένο περιβάλλον.

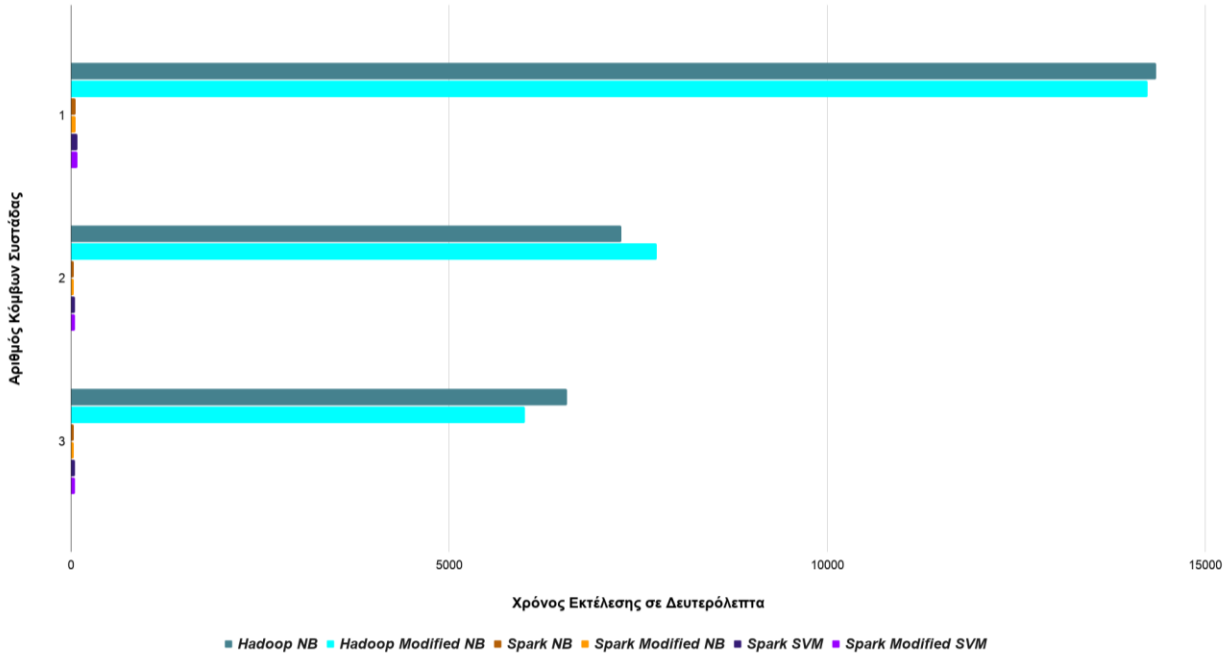
3.2.3 Επιτάχυνση (Speedup)

Για να προσδιοριστεί κατά πόσο επηρεάζεται ο χρόνος ολοκλήρωσης μιας εφαρμογής από τον αριθμό των κόμβων που αποτελούν την συστοιχία στην οποία εκτελείται έγιναν οι κατάλληλες ρυθμίσεις ώστε να ληφθούν μετρήσεις από την περίπτωση ενός και μοναδικού κόμβου έως το συνολικό αριθμό κόμβων στην τοπολογία. Και εδώ κάθε εφαρμογή εκτελέστηκε τρεις φορές ξεχωριστά ούτως

ώστε να εξασφαλιστεί ότι με τον μέσο όρο των μετρήσεων που καταγράφονται αντικατοπτρίζεται ικανοποιητικά η επίδοση των υλοποιήσεων βάσει του αριθμού των κόμβων στους οποίους εκτελούνται.

	Κόμβοι Συστοιχίας		
	1	**2**	**3**
Hadoop NB	14349	7274	6561
Hadoop NB_M	14239	7743	6000
Spark NB	55	33	32
Spark NB_M	55	32	31
Spark SVM	80	49	48
Spark SVM_M	79	50	49

Πίνακας 3.6 - Χρόνος Εκτέλεσης (σε Δευτερόλεπτα) Κάθε Υλοποίησης Για Ένα Εκατομμύριο Δείγματα Εισόδου Ανά Αριθμό Κόμβων Συστοιχίας



Γράφημα 3.8 - Γράφημα Ράβδων Χρόνου Εκτέλεσης (σε Δευτερόλεπτα) Κάθε Υλοποίησης Κάθε Υλοποίησης Για Ένα Εκατομμύριο Δείγματα Εισόδου Ανά Αριθμό Κόμβων Συστοιχίας

Εδώ γίνεται απτή η επίδραση του αριθμού των κόμβων στην συστοιχία ως προς την επίδοση του χρόνου ολοκλήρωσης κάθε εφαρμογής, με εκείνες που βασίζονται στο Hadoop να χρειάζονται τόσο μεγάλους χρόνους για το πέρας της εκτέλεσης τους (σχεδόν δύο ώρες για δύο κόμβους και πάνω από τρεις ώρες για ένα κόμβο) ώστε να αμφισβητείται ευθέως η χρήση τους σε περιβάλλον που δεν μπορεί να προβλεφθεί εξ αρχής ο όγκος των δεδομένων ή ο αριθμός των μηχανημάτων που αποτελούν την συστοιχία. Οι υλοποιήσεις στο Spark από την άλλη δέχονται μεν μια σημαντική επιβάρυνση στον χρόνο

εκτέλεσης με την μείωση των διαθέσιμων πόρων εκτέλεσης, η οποία όμως δεν έχει καμία σχέση με αυτές του Hadoop αφού οι μέγιστες καθυστερήσεις για τις εφαρμογές των δύο μεθόδων ταξινόμησης σε αυτή τη πλατφόρμα μετά βίας υπερβαίνουν το μισό λεπτό της ώρας.

Το συμπέρασμα στο οποίο καταλήγει αυτό το σκέλος της αξιολόγησης των αποτελεσμάτων βρίσκει τις υλοποιήσεις του Hadoop να χαρακτηρίζονται για ακόμη μία φορά από την αρκετά ευρεία διαφορά στους χρόνους εκτέλεσης, παρουσιάζοντας μεγάλα απαιτούμενα διαστήματα για την διεκπεραίωση αντίστοιχης διεργασίας και επεξεργασία ίδιου όγκου δεδομένων με τις υλοποιήσεις του Spark. Οι τελευταίες έδειξαν ικανοποιητική ανοχή σε ό,τι αφορά την κλιμακωτή αύξηση των δεδομένων εισόδου καθώς και στην ρύθμιση του αριθμού διαθέσιμου εξοπλισμού για μεγάλο όγκο δεδομένων, στοιχεία που προφανώς αποδεικνύουν ότι είναι και οι πιο αποδοτικές. Πιο ειδικά, μεταξύ των εφαρμογών που είναι βασισμένες στο Spark, διαφαίνεται ότι αυτές που ορίζονται για την μέθοδο του Naïve Bayes πετυχαίνουν και τους πιο σύντομους χρόνους ολοκλήρωσης ενώ εκείνες των SVM ακολουθούν με μια μικρή (αλλά όχι και τόσο αμελητέα, μιλώντας πάντα για τις επιδόσεις στην πλατφόρμα του Spark) καθυστέρηση.

4. Συμπεράσματα και Επεκτάσεις

Με την ολοκλήρωση της φάσης της αξιολόγησης επί των μετρήσεων γίνεται αισθητή και μετρήσιμη η αποτελεσματικότητα των υλοποιήσεων για κάθε κριτήριο που τέθηκε, όμως στο πλαίσιο της υφιστάμενης μελέτης υπάρχει η ανάγκη μιας συνολικής ανάλυσης για να προσδιοριστεί σε πιο γενικό επίπεδο το σύνολο των αποτελεσμάτων ούτως ώστε να κριθεί η βέλτιστη χρήση τους σε μια ενδεχόμενη εφαρμογή στον πραγματικό κόσμο.

Στην προηγούμενη ενότητα οι μετρήσεις είχαν βεβαίως ως σκοπό να καταδείξουν την συμπεριφορά όλων των εφαρμογών που αναπτύχθηκαν για το εκάστοτε κριτήριο που εξεταζόταν στους δύο ξεχωριστούς τύπους επίδοσης που χρειάστηκε να επικεντρωθεί η εργασία, της κατηγοριοποίησης και της παράλληλης εκτέλεσης σε περιβάλλον συστοιχίας. Ο διαχωρισμός αυτός κατέληξε όπως είναι φυσικό σε δύο διαφορετικές αξιολογήσεις, μία για κάθε τύπο επίδοσης, όπου για το κάθε ζητούμενο συντάχθηκε μια κατάταξη μεταξύ των υλοποιήσεων που υποδήλωνε την αποτελεσματικότητα τους πάνω στα ίδια δεδομένα εισόδου. Για λόγους πληρότητας από τη μεριά της αποτίμησης των μετρήσεων, κρίνεται αναγκαία η σύνθεση αυτών των αποτελεσμάτων με τέτοιο τρόπο ώστε για κάθε υλοποίηση να γνωστοποιηθεί η επίδοση της σε συνάρτηση με το σύνολο των κριτηρίων αξιολογώντας την εφαρμογή τους σε υποθετικά σενάρια χρήσης.

Εάν είναι θεμιτή η αξιοποίηση μιας εκ των εφαρμογών σε ένα σύστημα που χρειάζεται όσο το δυνατό καλύτερη ταξινόμηση εγγράφων, η μελέτη μπορεί να προσανατολιστεί στις εφαρμογές που παρουσίασαν την καλύτερη επίδοση κατηγοριοποίησης, όμως θα πρέπει να ληφθεί υπόψη σε δεύτερο χρόνο και ο παράγοντας της εφαρμογής των αλγορίθμων στο κατανεμημένο περιβάλλον. Πιο συγκεκριμένα θα πρέπει να βρεθεί η κατάλληλη υλοποίηση που παρέχει υψηλή ακρίβεια στις προβλέψεις των εγγράφων, χωρίς να γίνονται ταυτόχρονα εκπτώσεις σε ό,τι αφορά την κλιμακωσιμότητα και την επιτάχυνση της. Οι τελευταίοι αυτοί παράγοντες είναι κρίσιμοι στην επιλογή της βέλτιστης εφαρμογής για αυτό το σενάριο χρήσης, αφού ενδέχεται να μην μπορεί να προβλεφθεί εξ αρχής ο όγκος των δεδομένων ή ο αριθμός των κόμβων που θα είναι διαθέσιμοι στην τοπολογία του συστήματος. Να σημειωθεί πρωτίστως ότι μπορεί μεν μια εφαρμογή να έχει ικανοποιητική επίδοση σε ό,τι αφορά τον χρόνο εκτέλεσης, όμως δεν πρέπει αυτό να επηρεάζει (σε μεγάλο βαθμό, τουλάχιστον) την επίδοση κατηγοριοποίησης. Αντίστοιχα από την άλλη πλευρά πρέπει να υπάρχει μια στοιχειώδη ανοχή σε καθυστερήσεις, αλλά μόνο αν αυτές συνεπάγονται μια αύξηση του ποσοστού ορθότητας των αποτελεσμάτων που δικαιολογεί την προτίμηση αυτής της εφαρμογής έναντι μιας άλλης. Ακολουθώντας αυτό τον συλλογισμό ξεχωρίζουν αρχικά βάσει επίδοσης στο μέτρο F_1 οι κορυφαίες τέσσερις υλοποιήσεις των απλών και τροποποιημένων εκδόσεων του Naïve Bayes στο Hadoop και του SVM στο Spark, λόγω της εξαιρετικά μικρής διαφοράς που παρουσιάζεται μεταξύ τους στα τρία τελευταία και μεγαλύτερα σε όγκο σύνολα εισόδου. Στη συνέχεια μεταφέρεται η μελέτη στα περί της παράλληλης εκτέλεσης, όπου παρατηρείται ξανά η μεγάλη διαφορά στον απαιτούμενο χρόνο ολοκλήρωσης μεταξύ των πλατφορμών που εξετάστηκαν. Γνωρίζοντας δηλαδή ότι οι εφαρμογές που είναι βασισμένες στο Hadoop παρουσιάζουν μεγάλες καθυστερήσεις, είναι προφανές ότι θα προτιμηθούν σε αυτό το σενάριο οι εφαρμογές του SVM στο Spark, με οριακά καλύτερη την τροποποιημένη έκδοση αυτών των δύο.

Για λογαριασμό ενός συστήματος που επικεντρώνεται περισσότερο στην όσο το δυνατό καλύτερη παράλληλη εκτέλεση ενός ικανοποιητικού μοντέλου ταξινόμησης, ακολουθείται η αντίστροφη λογική από το προηγούμενο σενάριο για να κριθεί κάθε υλοποίηση πρωτίστως από την συμπεριφορά της στον χρόνο εκτέλεσης και έπειτα να ληφθεί υπόψη η αποτελεσματικότητα των προβλέψεων κάθε μοντέλου. Σε αυτή την περίπτωση προκύπτει για άλλη μια φορά η μεγάλη διαφορά

στους χρόνους ολοκλήρωσης μεταξύ των πλατφορμών, αφού οι υλοποιήσεις σχεδιασμένες στο Hadoop χρειάζονται πολλαπλάσιο χρόνο για την ολοκλήρωση τους έναντι αυτών στο Spark. Οι τελευταίες με την σειρά τους έχουν πολύ μικρές χρονικές διαφορές μεταξύ τους που δεν ξεπερνούν ποτέ τα είκοσι δευτερόλεπτα, οπότε χρειάζεται περαιτέρω διερεύνηση πάνω στην επίδοση της κατηγοριοποίησης κάθε μιας από τις υλοποιήσεις αυτές. Μελετώντας τις επιδόσεις των εφαρμογών στο Spark για το μέτρο F_1 παρατηρείται σημαντική διαφορά μεταξύ των αλγορίθμων Naïve Bayes και SVM, και μια αρκετά μικρότερη διαφορά μεταξύ κάθε αλγόριθμου σε ό,τι αφορά τις απλές εκδόσεις σε σχέση με τις τροποποιημένες. Ειδικότερα, για κάθε σύνολο δειγμάτων εισόδου οι υλοποιήσεις που εφαρμόζουν την SVM μέθοδο έχουν σταθερά καλύτερη επίδοση στις προβλέψεις των μοντέλων τους από αυτές που εφαρμόζουν τον Naïve Bayes αλγόριθμο. Κρίνοντας τώρα επί των δύο εκδόσεων που κάνουν χρήση του SVM, η τροποποιημένη έκδοση δείχνει να επικρατεί συντηρώντας μια σταθερά καλύτερη επίδοση ταξινόμησης των εγγράφων σε σχέση με την αύξηση του μεγέθους των δεδομένων που καλείται να επεξεργαστεί στην είσοδο.

Παρατηρώντας τα συμπεράσματα των δύο προηγούμενων σεναρίων, προκύπτει με μια πρώτη ανάγνωση ότι μια καλή επίδοση στους χρόνους εκτέλεσης μιας υλοποίησης δεν συνεπάγεται πάντα από καλή επίδοση και στην κατηγοριοποίηση των δειγμάτων προς εξέταση, και το αντίστροφο. Αυτή η διαπίστωση επαληθεύει φυσικά την ανάγκη μιας συνολικής αποτίμησης των συμπεριφορών κάθε εφαρμογής στα κατάλληλα κριτήρια έναντι της ξεχωριστής αξιολόγησης βάσει των ζητούμενων υπό εξέταση (εδώ συγκεκριμένα σε ό,τι αφορά τα ποσοστά σωστών προβλέψεων του μοντέλου και τους χρόνους ολοκλήρωσης των υλοποιήσεων σε διάφορα σενάρια εκτέλεσης), αφού η διττή φύση τέτοιων εφαρμογών που κινούνται ταυτόχρονα στην εξόρυξη κειμένου και τον παράλληλο υπολογισμό δεν μπορεί να έχει μία και μοναδική μετρική ώστε να κριθεί πόσο αποδοτική είναι μια εφαρμογή έναντι μιας άλλης. Οι θεματικές συνιστώσες που αποτελούν τον χώρο στον οποίο δραστηριοποιείται η παρούσα μελέτη προσανατολίζονται κατά μία έννοια αντίθετα μεταξύ τους ως προς τα ζητούμενα για την βελτιστοποίηση των αποτελεσμάτων τους, οπότε είναι σημαντικό να βρεθεί η στοιχειώδης ισορροπία μεταξύ των ζητουμένων, ανάλογα με τους σκοπούς και τα δεδομένα που έχει να επεξεργαστεί το σύστημα στο οποίο θα λειτουργήσει η εκάστοτε εφαρμογή.

Με το αντικείμενο αυτής της εργασίας να είναι η ανάλυση συναισθημάτων σε δεδομένα μορφής κειμένου, το βάρος της συνολικής επίδοσης κάθε υλοποίησης πέφτει στην ίδια την ποιότητα των δεδομένων που εφαρμόζονται στην είσοδο. Η επιλογή μιας συλλογής με tweets για την εκπαίδευση και του έλεγχο του κάθε μοντέλου χαρίζει μια ευκολία σε ό,τι αφορά τον όγκο των δεδομένων εισόδου, μιας και κάθε έγγραφο της συλλογής έχει ένα προκαθορισμένο ανώτατο όριο χαρακτήρων από την ίδια την πλατφόρμα του Twitter που προέρχονται, όμως την ίδια στιγμή θέτονται οι προϋποθέσεις για χαμηλής ποιότητας πληροφορία που θα τροφοδοτήσει κάθε μοντέλο. Δεν μπορεί να εγγυηθεί από κανέναν ότι κάθε έγγραφο σε μια τέτοια συλλογή πληροί ποιοτικά κριτήρια σε ό,τι αφορά τα ορθογραφικά λάθη ή τα σημασιολογικά ασυνάρτητα tweets που έχουν κατά πάσα πιθανότητα σκοπό το spamming. Μπορεί μεν να καταπολεμηθούν αυτά τα ενδεχόμενα με ανάλογες τροποποιήσεις στις υλοποιήσεις, όμως αυτό κατά πάσα πιθανότητα θα λειτουργήσει αρνητικά σε ό,τι έχει σχέση με την παράλληλη υλοποίηση των εφαρμογών για ένα αντίκρυσμα αυξημένης επίδοσης κατηγοριοποίησης που δεν υπάρχουν εγγυήσεις ότι θα εμφανιστεί ή αν θα είναι αρκετά μεγάλη αυτή η αύξηση ώστε να δικαιολογεί τροποποιήσεις τέτοιου μεγέθους. Σαν άγραφος κανόνας, ο θόρυβος στα δεδομένα εισόδου μιας εφαρμογής που ασχολείται με την εξόρυξη κειμένου θα πρέπει να λαμβάνεται υπόψη ως ακόμη ένα κριτήριο για την αξιολόγηση αποτελεσμάτων που μπορεί να εμφανίζονται απογοητευτικά και κατώτερα των περιστάσεων στην επίδοση είτε της ταξινόμησης είτε της παράλληλης εκτέλεσης των αλγορίθμων. Στη συγκεκριμένη περίπτωση οι επιδόσεις των εφαρμογών δείχνουν να είναι ελαφρά άνω

των προσδοκιών οπότε κρίνονται ικανοποιητικά λειτουργικές για τον τύπο εισόδου που καλούνται να διαχειριστούν.

Με το βλέμμα σε ενδεχόμενες επεκτάσεις που προκαλούν ενδιαφέρον και κρίνονται συμβατές με το περιεχόμενο της εργασίας, αρχικά μπορεί να εξεταστεί η περίπτωση όπου χρειάζεται από μία εφαρμογή να επιτυγχάνεται η βέλτιστη επίδοση ορθότητας στην κατηγοριοποίηση των εγγράφων. Εδώ μπορεί να μελετηθεί ένα συμπληρωματικό κομμάτι που επαναπροσδιορίζει τον τρόπο με τον οποίο η τροποποιημένη έκδοση κάθε υλοποίησης επιλέγει τα πιο σχετικά χαρακτηριστικά ώστε να ελαχιστοποιηθεί ο θόρυβος εισόδου. Ένα τέτοιο εγχείρημα μπορεί να αφορά την χρήση ενός άλλου τρόπου αξιολόγησης περί των σημαντικών χαρακτηριστικών, όπως τις πρακτικές που εξετάστηκαν όμως δεν προτιμήθηκαν κατά την ανάπτυξη των υλοποιήσεων για τη συγκεκριμένη μελέτη.

Περνώντας στην άλλη όχθη των περιοχών που δραστηριοποιήθηκε ως αυτό το σημείο η εργασία, θα μπορούσε να μελετηθεί περαιτέρω και με μεγαλύτερη λεπτομέρεια η συμπεριφορά των εφαρμογών σε μια συστοιχία που αποτελείται από περισσότερους κόμβους. Με την αξιοποίηση μιας τέτοιας υποδομής μπορεί να γίνει πιο ξεκάθαρη η επίδοση των υλοποιήσεων σε ό,τι αφορά την κλιμακωσιμότητα και κυρίως την επιτάχυνση, εφόσον η τελευταία όπως είναι αναμενόμενο θα αρχίσει να φθίνει με την αύξηση των διαθέσιμων μηχανημάτων, με τον χρόνο που εξοικονομείται να ελαχιστοποιείται αναλόγως με την προσάρτηση νέων κόμβων στην συστοιχία. Αυτό αξίζει και μπορεί να παρατηρηθεί μιας και μοιραία οι διεργασίες των εφαρμογών θα φτάσουν σε κρίσιμα σημεία που δεν μπορούν να παραλληλοποιηθούν περαιτέρω για λογαριασμό των δεδομένων εισόδου, ενώ θα επιβαρύνονται οι χρόνοι εκτέλεσης λόγω της επικοινωνίας μεταξύ των (αυξημένων) κόμβων για τον συντονισμό των διαδικασιών που έχουν οριστεί. Με τις παραπάνω μετρήσεις που θα μπορούν να ληφθούν σε μια τέτοια, μπορούν να γίνουν αναλόγως και οι απαραίτητες τροποποιήσεις στον σχεδιασμό των υλοποιήσεων στην αναζήτηση καλύτερων συνθηκών που μπορούν να οριστούν για τις εκτελέσεις τους.

Μια άλλη πτυχή της εργασίας που έχει περιθώρια επέκτασης είναι στο ευαίσθητο για τις επιδόσεις θέμα της μορφής και της πηγής των δεδομένων εισόδου. Αν και δεν αναφέρθηκε ρητά στην περιγραφή της συλλογής δειγμάτων που χρησιμοποιήθηκαν για τις πειραματικές μετρήσεις, έως τώρα εξετάστηκαν οι επιδόσεις των υλοποιήσεων βασισμένες στις πιο κατάλληλες συνθήκες, αφού εφαρμόζονται στην είσοδο για την εκπαίδευση και τον έλεγχο των μοντέλων δεδομένα που είναι προετοιμασμένα, στοιχισμένα από πριν, και αποθηκευμένα εξ αρχής στο κατανεμημένο σύστημα αρχείων της συστοιχίας. Ένα ενδιαφέρον συμπληρωματικό κομμάτι πάνω στην εργασία θα μπορούσε να αφορά λοιπόν την εφαρμογή δεδομένων εισόδου (τουλάχιστον) στην φάση του ελέγχου ενός μοντέλου που συγκεντρώνονται σε πραγματικό χρόνο κατά την εκτέλεση μιας εφαρμογής. Μία ανάπτυξη πάνω σε αυτή την ιδέα φυσικά χρειάζεται (πέρα από λίγες μα αναγκαίες τροποποιήσεις στον βασικό κορμό των προγραμμάτων) την συνδρομή σχετικών επεκτάσεων (όπως η πλατφόρμα του Flume για το Hadoop) ή συνιστωσών (όπως η βιβλιοθήκη Spark Streaming του Spark) για να καταστεί κάτι τέτοιο δυνατό, δίνοντας παράλληλα μια ευκαιρία για την μελέτη αυτών των εργαλείων.

Βιβλιογραφία

[1] Khader, M., Awajan, A., & Al-Naymat, G. (2018). Sentiment Analysis Based on MapReduce: A survey. 1-8. 10.1145/3291280.3291795.

[2] Nasir, N., Zafar, K., & Alamgir, Z. (2017). Sentiment Analysis of Social Media Using MapReduce.

[3] Nagdive, A., & Tugnayat, R. (2018). A Review of Hadoop Ecosystem for BigData.

[4] Ghazi, M., & Gangodkar, D. (2015). Hadoop, MapReduce and HDFS: a developer's perspective. Procedia Computer Science. 48. 45-50. 10.1016/j.procs.2015.04.108.

[5] Alsaqqa, S., Al-Naymat, G., & Awajan, A. (2018). A Large-Scale Sentiment Data Classification for Online Reviews Under Apache Spark. Procedia Computer Science. 141. 183-189. 10.1016/j.procs.2018.10.166.

[6] Tan, P., Steinbach, M., Karpatne, A., & Kumar, V. (2018). Introduction to Data Mining (2nd ed.). Pearson.

[7] Sumathy, K. L., & Chidambaram, M. (2013). Text Mining: Concepts, Applications, Tools and Issues An Overview. International Journal of Computer Applications. 80. 29-32. 10.5120/13851-1685.

[8] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press, USA.

[9] Ying, X. (2019). An Overview of Overfitting and its Solutions. Journal of Physics: Conference Series. 1168. 022022. 10.1088/1742-6596/1168/2/022022.

[10] He, Q., Zhuang, F., & Shi, Z. (2010). Parallel Implementation of Classification Algorithms Based on MapReduce. 655-662. 10.1007/978-3-642-16248-0_89.

[11] Sun, Z., & Fox, G. (2012). Study on Parallel SVM Based on MapReduce.

[12] Þrastarson, K. (2018). Design, Implementation and Analysis of a Parallel and Scalable Cascade Support Vector Machine Framework [Master's thesis, University of Iceland]. Skemman.

[13] Bhosale, A., Kulkarni, A., Gadkari, S., & Krothapalli, S. (2015). An Overview of Sentiment Analysis.

[14] Mukherjee, S., & Bhattacharyya, P. (2013). Sentiment Analysis : A Literature Survey.

[15] Rajaraman, A., Leskovec, J., & Ullman, J. (2014). Mining of Massive Datasets (2nd ed.). Cambridge University Press.

[16] Alharbi, A., Alnnamlah, H., & Syed, L. (2018). Classification of Customer Tweets Using Big Data Analytics. 10.1007/978-3-319-78753-4_13.

[17] Kumar, M., Rath, N., Swain, A., & Rath, S. (2015). Feature Selection and Classification of Microarray Data using MapReduce based ANOVA and K-Nearest Neighbor. Procedia Computer Science. 54. 301-310. 10.1016/j.procs.2015.06.035.

[18] Kamdar, A. B., & Rajani, I. K. (2015). Improved Adaptive K Nearest Neighbor algorithm using MapReduce. International Journal of Science, Engineering and Technology Research (IJSETR), 4(6), 2113-2117.

[19] Li, Z. (2014). Naive Bayes algorithm for Twitter sentiment analysis and its implementation in MapReduce [Master's thesis, University of Missouri]. MOspace.

[20] Laada, S. (2014). Suitability of the Spark framework for data classification [Master's thesis, University of Tartu]. DSpace.

[21] Murthy, A. C., Vavilapalli, V. K., Eadline, D., Niemiec, J., & Markham, J. (2014). Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2. Pearson.

[22] Hammond, K., & Varde, A. (2013). Cloud Based Predictive Analytics: Text Classification, Recommender Systems and Decision Support. 607-612. 10.1109/ICDMW.2013.95.

[23] White, T. (2015). Hadoop: The Definitive Guide (4th ed.). O'Reilly.

[24] Jayaraman, R., & Velumani, B. (2018). An Efficient Map Reduce-Based Hybrid NBC-TFIDF Algorithm to mine the Public Sentiment on Diabetes Mellitus - A Big Data approach. Journal of King Saud University - Computer and Information Sciences. 10.1016/j.jksuci.2018.06.011.

[25] Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). Learning Spark: Lightning-Fast Big Data Analysis (1st ed.). O'Reilly.

[26] Kaur, A., Singh, G., & Gupta, R. (2020). Sentimental Analysis Using Various Analytical tools from Hadoop Eco System. Journal of Xidian University. 14. 10.37896/jxu14.5/071.

[27] Moertini, V., Septrianto, M., & Venica, L. (2019). INCREMENTAL PARALLEL CLASSIFIER FOR BIG DATA WITH CASE STUDY: NAÏVE BAYES USING MAPREDUCE PATTERNS. Journal of Theoretical and Applied Information Technology. 97. 3077-3097.

[28] Gilheany, E. (2016). Processing time of TFIDF and Naive Bayes on Spark 2.0, Hadoop 2.6 and Hadoop 2.7: Which Tool Is More Efficient? [Master's thesis, National College of Ireland]. NORMA.

[29] Kanavos, A., Nodarakis, N., Sioutas, S., Tsakalidis, A., Tsolis, D., & Tzimas, G. (2017). Large Scale Implementations for Twitter Sentiment Classification. Algorithms. 10. 33. 10.3390/a10010033.

[30] Amazal, H., Ramdani, M., & Kissi, M. (2018). A Text Classification Approach using Parallel Naive Bayes in Big Data Context. SITA'18: Proceedings of the 12th International Conference on Intelligent Systems: Theories and Applications. 1-6. 10.1145/3289402.3289536.

[31] Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey.

[32] Kandefer, M., & Shapiro, S. (2009). An F-Measure for Context-based Information Retrieval. Commonsense 2009 - Proceedings of the 9th International Symposium on Logical Formalizations of Commonsense Reasoning.

[33] Baltas, A., Kanavos, A., & Tsakalidis, A. (2017). An Apache Spark Implementation for Sentiment Analysis on Twitter Data. 15-25. 10.1007/978-3-319-57045-7_2.

Παράρτημα: Κώδικας Υλοποιήσεων στο Πλαίσιο της Εργασίας

Υλοποίηση Naïve Bayes στο Hadoop

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.Counters;

import java.io.*;
import java.io.IOException;
import java.util.*;
import java.nio.charset.StandardCharsets;


public class NB
{
      public static enum Global_Counters
      {
      TWEETS_SIZE,
      POS_TWEETS_SIZE,
      NEG_TWEETS_SIZE,
      POS_WORDS_SIZE,
      NEG_WORDS_SIZE,
      FEATURES_SIZE,
      TRUE_POSITIVE,
      FALSE_POSITIVE,
      TRUE_NEGATIVE,
      FALSE_NEGATIVE
      }



      /* input:  <byte_offset, line_of_tweet>
      * output: <word, sentiment>
      */
      public static class Map_Training extends Mapper<Object, Text,
Text, Text>
      {
      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {

      context.getCounter(Global_Counters.TWEETS_SIZE).increment(1);

            String line = value.toString();
```

```
            String[] columns = line.split(",");

            // if the columns are more than 4, that means the text of
the post had commas inside,
            // so stitch the last columns together to form the full text
of the tweet
            if(columns.length > 4)
            {
                  for(int i=4; i<columns.length; i++)
                  columns[3] += columns[i];
            }

            String tweet_sentiment = columns[1];
            String tweet_text = columns[3];

            // clean the text of the tweet from links...
            tweet_text =
tweet_text.replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9][a-zA-Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-
]+[a-zA-Z0-9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")
                              .replaceAll("(#|@|&).*?\\w+", "")   //
mentions, hashtags, special characters...
                              .replaceAll("\\d+", "")             //
numbers...
                              .replaceAll("[^a-zA-Z ]", " ")      //
punctuation...
                              .toLowerCase()                      //
turn every character left to lowercase...
                              .trim()                             //
trim the spaces before & after the whole string...
                              .replaceAll("\\s+", " ");           // and
get rid of double spaces

            String sentiment_label = "POSITIVE";

            if(tweet_sentiment.equals("1"))
            {

      context.getCounter(Global_Counters.POS_TWEETS_SIZE).increment(1);

      context.getCounter(Global_Counters.POS_WORDS_SIZE).increment(tweet
_text.split("\\s+").length);
            }
            else
            {

      context.getCounter(Global_Counters.NEG_TWEETS_SIZE).increment(1);

      context.getCounter(Global_Counters.NEG_WORDS_SIZE).increment(tweet
_text.split("\\s+").length);
                  sentiment_label = "NEGATIVE";
            }


            if(tweet_text != null && !tweet_text.trim().isEmpty())
            {
                  String[] tweet_words = tweet_text.split(" ");

                  for(String word : tweet_words)
```

```
                  context.write(new Text(word), new
Text(sentiment_label));
            }
      }
      }

      /* input:  <word, sentiment>
      * output: <word, pos_wordcount@neg_wordcount>
      */
      public static class Reduce_Training extends Reducer<Text, Text,
Text, Text>
      {
      public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
      {

      context.getCounter(Global_Counters.FEATURES_SIZE).increment(1);

            int positive_counter = 0;
            int negative_counter = 0;

            // for each word, count the occurrences in tweets with
positive/negative sentiment
            for(Text value : values)
            {
                  String sentiment = value.toString();
                  if(sentiment.equals("POSITIVE"))
                  positive_counter++;
                  else
                  negative_counter++;
            }

            context.write(key, new Text(String.valueOf(positive_counter)
+ "@" + String.valueOf(negative_counter)));
      }
      }



      /* input: <byte_offset, line_of_tweet>
      * output: <tweet@tweet_text, sentiment>
      */
      public static class Map_Testing extends Mapper<Object, Text, Text,
Text>
      {
      int features_size, tweets_size, pos_tweets_size, neg_tweets_size,
pos_words_size, neg_words_size;
      Double pos_class_probability, neg_class_probability;

      // hashmaps with each word as key and its number of occurrences in
each class as value
      HashMap<String, Integer> pos_words = new HashMap<String,
Integer>();
      HashMap<String, Integer> neg_words = new HashMap<String,
Integer>();

      // hashmaps with each word as key and its probability in each
class as value
      HashMap<String, Double> pos_words_probabilities = new
HashMap<String, Double>();
```

```
	HashMap<String, Double> neg_words_probabilities = new
HashMap<String, Double>();

	// lists holding all probabilities to be multiplied together,
along with the positive/negative class probability
	ArrayList<Double> pos_probabilities_list = new
ArrayList<Double>();
	ArrayList<Double> neg_probabilities_list = new
ArrayList<Double>();

	protected void setup(Context context) throws IOException,
InterruptedException
	{
		// load all counters to be used for the calculation of the
probabilities
		features_size =
Integer.parseInt(context.getConfiguration().get("features_size"));
		tweets_size =
Integer.parseInt(context.getConfiguration().get("tweets_size"));
		pos_tweets_size =
Integer.parseInt(context.getConfiguration().get("pos_tweets_size"));
		neg_tweets_size =
Integer.parseInt(context.getConfiguration().get("neg_tweets_size"));
		pos_words_size =
Integer.parseInt(context.getConfiguration().get("pos_words_size"));
		neg_words_size =
Integer.parseInt(context.getConfiguration().get("neg_words_size"));

		pos_class_probability = ((double) pos_tweets_size) /
tweets_size;
		neg_class_probability = ((double) neg_tweets_size) /
tweets_size;

		// load the model of the last training job and fill two
hashmaps of words with the number of
		// occurrences in positive and negative tweets
		Path training_model = new Path("training");
		FileSystem model_fs =
training_model.getFileSystem(context.getConfiguration());
		FileStatus[] file_status =
model_fs.listStatus(training_model);

		for(FileStatus i : file_status)
		{
			Path current_file_path = i.getPath();

			if(i.isFile())
			{
			BufferedReader br = new BufferedReader(new
InputStreamReader(model_fs.open(current_file_path)));
			String line;

			while((line = br.readLine()) != null)
			{
				String[] columns = line.toString().split("\t");
				String[] pos_and_neg_counts =
columns[1].split("@");

				pos_words.put(columns[0],
Integer.parseInt(pos_and_neg_counts[0]));
```

```
                        neg_words.put(columns[0],
Integer.parseInt(pos_and_neg_counts[1]));
                  }

                  br.close();
                  }
            }

            // calculate all the word probabilities for positive and
negative class (with laplace smoothing)
            for(Map.Entry<String,Integer> entry : pos_words.entrySet())
            {
                  pos_words_probabilities.put(entry.getKey(), ((double)
entry.getValue() + 1) / (pos_words_size + features_size));
                  neg_words_probabilities.put(entry.getKey(), ((double)
neg_words.get(entry.getKey()) + 1) / (neg_words_size + features_size));
            }
      }

      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {
            String line = value.toString();
            String[] columns = line.split(",");

            // if the columns are more than 4, that means the text of
the post had commas inside,
            // so stitch the last columns together to form the post
            if(columns.length > 4)
            {
                  for(int i=4; i<columns.length; i++)
                  columns[3] += columns[i];
            }

            String tweet_id = columns[0];
            String tweet_sentiment = columns[1];
            String tweet_text = columns[3];

            // clean the text of the tweet from links...
            tweet_text =
tweet_text.replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9][a-zA-Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-
]+[a-zA-Z0-9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")
                              .replaceAll("(#|@|&).*?\\w+", "")   //
mentions, hashtags, special characters...
                              .replaceAll("\\d+", "")             //
numbers...
                              .replaceAll("[^a-zA-Z ]", " ")      //
punctuation...
                              .toLowerCase()                      //
turn every character left to lowercase...
                              .trim()                             //
trim the spaces before & after the whole string...
                              .replaceAll("\\s+", " ");           // and
get rid of double spaces

            // initialize the product of positive and negative
probabilities with 1
            Double pos_probability = 1.0;
```

```
            Double neg_probability = 1.0;

            // calculate the product of the probabilities of the words
(+ the class probability) for each class
            if(tweet_text != null && !tweet_text.trim().isEmpty())
            {
                  String[] tweet_words = tweet_text.split(" ");

                  for(String word : tweet_words)
                  {
                  for(Map.Entry<String,Double> entry :
pos_words_probabilities.entrySet())
                  {
                        if(word.equals(entry.getKey()))
                        {
                              pos_probability *=
pos_words_probabilities.get(word);
                              neg_probability *=
neg_words_probabilities.get(word);
                        }
                  }
                  }
            }

            // multiply the product of positive and negative probability
with the class probability of each sentiment
            pos_probability *= pos_class_probability;
            neg_probability *= neg_class_probability;

            // compare and set the max value of the two class
probabilities as the result of the guessed sentiment for every tweet
            if(Double.compare(pos_probability, neg_probability) > 0)
            {
                  if(tweet_sentiment.equals("1"))

      context.getCounter(Global_Counters.TRUE_POSITIVE).increment(1);
                  else

      context.getCounter(Global_Counters.FALSE_POSITIVE).increment(1);

                  context.write(new Text(tweet_id + "@" + tweet_text),
new Text("POSITIVE"));
            }
            else
            {
                  if(tweet_sentiment.equals("0"))

      context.getCounter(Global_Counters.TRUE_NEGATIVE).increment(1);
                  else

      context.getCounter(Global_Counters.FALSE_NEGATIVE).increment(1);

                  context.write(new Text(tweet_id + "@" + tweet_text),
new Text("NEGATIVE"));
            }
      }
      }
```

```
      public static void main(String[] args) throws Exception
      {
      // paths to directories were input, inbetween and final job
outputs are stored
      Path input_dir = new Path(args[0]);
      Path training_dir = new Path("training");
      Path testing_dir = new Path(args[1]);
      Path output_dir = new Path("output");

      Configuration conf = new Configuration();

      FileSystem fs = FileSystem.get(conf);
      if(fs.exists(training_dir))
            fs.delete(training_dir, true);
      if(fs.exists(output_dir))
            fs.delete(output_dir, true);

      long start_time = System.nanoTime();

      Job training_job = Job.getInstance(conf, "Training");
      training_job.setJarByClass(NB.class);
      training_job.setMapperClass(Map_Training.class);
      training_job.setReducerClass(Reduce_Training.class);
      training_job.setNumReduceTasks(3);
      training_job.setMapOutputKeyClass(Text.class);
      training_job.setMapOutputValueClass(Text.class);
      training_job.setOutputKeyClass(Text.class);
      training_job.setOutputValueClass(Text.class);
      TextInputFormat.addInputPath(training_job, input_dir);
      TextInputFormat.setMaxInputSplitSize(training_job,
Long.valueOf(args[2]));
      TextOutputFormat.setOutputPath(training_job, training_dir);
      training_job.waitForCompletion(true);

      int tweets_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.T
WEETS_SIZE).getValue());
      conf.set("tweets_size", String.valueOf(tweets_size));
      int pos_tweets_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.P
OS_TWEETS_SIZE).getValue());
      conf.set("pos_tweets_size", String.valueOf(pos_tweets_size));
      int neg_tweets_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.N
EG_TWEETS_SIZE).getValue());
      conf.set("neg_tweets_size", String.valueOf(neg_tweets_size));
      int pos_words_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.P
OS_WORDS_SIZE).getValue());
      conf.set("pos_words_size", String.valueOf(pos_words_size));
      int neg_words_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.N
EG_WORDS_SIZE).getValue());
      conf.set("neg_words_size", String.valueOf(neg_words_size));
      int features_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.F
EATURES_SIZE).getValue());
      conf.set("features_size", String.valueOf(features_size));

      Job testing_job = Job.getInstance(conf, "Testing");
```

```
      testing_job.setJarByClass(NB.class);
      testing_job.setMapperClass(Map_Testing.class);
      testing_job.setMapOutputKeyClass(Text.class);
      testing_job.setMapOutputValueClass(Text.class);
      testing_job.setOutputKeyClass(Text.class);
      testing_job.setOutputValueClass(Text.class);
      TextInputFormat.addInputPath(testing_job, testing_dir);
      TextInputFormat.setMaxInputSplitSize(testing_job,
Long.valueOf(args[3]));
      TextOutputFormat.setOutputPath(testing_job, output_dir);
      testing_job.waitForCompletion(true);

      System.out.println("EXECUTION DURATION: " + (System.nanoTime() -
start_time) / 1000000000F + " seconds");

      int tp =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.TR
UE_POSITIVE).getValue());
      int fp =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.FA
LSE_POSITIVE).getValue());
      int tn =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.TR
UE_NEGATIVE).getValue());
      int fn =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.FA
LSE_NEGATIVE).getValue());

      System.out.println("\nCONFUSION MATRIX:");
      System.out.printf("%-10s %-10s \n", tp, fp);
      System.out.printf("%-10s %-10s \n\n", fn, tn);

      System.out.printf("%-25s %-10s \n", "ACCURACY: ", ((double) (tp +
tn)) / (tp + tn + fp + fn));
      }
}
```

Υλοποίηση Τροποποιημένου Naïve Bayes στο Hadoop

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
```

```
import org.apache.hadoop.mapreduce.Counters;

import java.io.*;
import java.io.IOException;
import java.util.*;
import java.nio.charset.StandardCharsets;


public class Modified_NB
{
      public static enum Global_Counters
      {
      NUM_OF_TWEETS,
      TWEETS_SIZE,
      POS_TWEETS_SIZE,
      NEG_TWEETS_SIZE,
      POS_WORDS_SIZE,
      NEG_WORDS_SIZE,
      FEATURES_SIZE,
      TRUE_POSITIVE,
      FALSE_POSITIVE,
      TRUE_NEGATIVE,
      FALSE_NEGATIVE
      }



      /* input:  <byte_offset, line_of_tweet>
      * output: <(word@tweet), 1>
      */
      public static class Map_WordCount extends Mapper<Object, Text,
Text, IntWritable>
      {
      private Text word_tweet_key = new Text();
      private final static IntWritable one = new IntWritable(1);

      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {

      context.getCounter(Global_Counters.NUM_OF_TWEETS).increment(1);

            String line = value.toString();
            String[] columns = line.split(",");

            // if the columns are more than 4, that means the text of
the post had commas inside,
            // so stitch the last columns together to form the full text
of the tweet
            if(columns.length > 4)
            {
                  for(int i=4; i<columns.length; i++)
                  columns[3] += columns[i];
            }

            String tweet_id = columns[0];
            String tweet_sentiment = columns[1];
            String tweet_text = columns[3];

            if(tweet_sentiment.equals("1"))
```

```
                tweet_id += '+';

            // clean the text of the tweet from links...
            tweet_text =
tweet_text.replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9][a-zA-Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-
]+[a-zA-Z0-9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")
                              .replaceAll("(#|@|&).*?\\w+", "")   //
mentions, hashtags, special characters...
                              .replaceAll("\\d+", "")             //
numbers...
                              .replaceAll("[^a-zA-Z ]", " ")      //
punctuation...
                              .toLowerCase()                      //
turn every character left to lowercase...
                              .trim()                             //
trim the spaces before & after the whole string...
                              .replaceAll("\\s+", " ");           // and
get rid of double spaces


            if(tweet_text != null && !tweet_text.trim().isEmpty())
            {
                  String[] tweet_words = tweet_text.split(" ");

                  for(int i=0; i<tweet_words.length; i++)
                  {
                  word_tweet_key.set(tweet_words[i] + "@" + tweet_id);

                  context.write(word_tweet_key, one);
                  }
            }
      }
      }

      /* input:  <(word@tweet), 1>
      * output: <(word@tweet), word_count>
      */
      public static class Reduce_WordCount extends Reducer<Text,
IntWritable, Text, IntWritable>
      {
      public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException
      {
            int sum = 0;
            for(IntWritable value : values)
                  sum += value.get();

            context.write(key, new IntWritable(sum));
      }
      }



      /* input:  <(word@tweet), word_count>
      * output: <tweet, (word=word_count)>
      */
      public static class Map_TF extends Mapper<Object, Text, Text,
Text>
```

```
      {
      private Text tweet_key = new Text();
      private Text word_wordcount_value = new Text();

      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {
            String lines[] = value.toString().split("\t");
            String splitted_key[] = lines[0].toString().split("@");

            tweet_key.set(splitted_key[1]);
            word_wordcount_value.set(splitted_key[0] + "=" + lines[1]);

            context.write(tweet_key, word_wordcount_value);
      }
      }

      /* input:  <tweet, (word=word_count)>
      * output: <(word@tweet), (word_count/tweet_length)>
      */
      public static class Reduce_TF extends Reducer<Text, Text, Text,
Text>
      {
      public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
      {
            String tweet = key.toString();

            int total = 0;

            ArrayList<String> word_list = new ArrayList<String>();
            ArrayList<String> count_list = new ArrayList<String>();
            ArrayList<String> word_by_count_list = new
ArrayList<String>();

            for(Text value : values)
            {
                  String[] splitted_key = value.toString().split("=");

                  word_list.add(splitted_key[0]);
                  count_list.add(splitted_key[1]);

                  total += Integer.parseInt(splitted_key[1]);
            }

            // create the count/total in the list
            for(String count : count_list)
                  word_by_count_list.add(count + "/" + total);

            // create the iterator for word_list and word_by_count_list
and write the key-value pair in the context
            Iterator<String> wl = word_list.iterator();
            Iterator<String> wbcl = word_by_count_list.iterator();

            while (wl.hasNext() && wbcl.hasNext())
            {
                  context.write(new Text(wl.next() + "@" + tweet), new
Text(wbcl.next().toString()));
            }
      }
```

```
	}



	/* input:  <(word@tweet), (word_count/tweet_length)>
	* output: <word, (tweet=word_count/tweet_length)>
	*/
	public static class Map_TFIDF extends Mapper<Object, Text, Text,
Text>
	{
	private Text word_key = new Text();
	private Text tweet_wordcount_tweetsize_value = new Text();

	public void map(Object key, Text value, Context context ) throws
IOException, InterruptedException
	{
		String[] columns = value.toString().split("\t");
		String[] splitted_key = columns[0].toString().split("@");

		word_key.set(splitted_key[0]);
		tweet_wordcount_tweetsize_value.set(splitted_key[1] + "=" +
columns[1]);

		context.write(word_key, tweet_wordcount_tweetsize_value);

	}
	}

	/* input:  <word, (tweet=word_count/tweet_length)>
	* output: <(tweet@word), TFIDF>
	*/
	public static class Reduce_TFIDF extends Reducer<Text, Text, Text,
Text>
	{
	private static int num_of_tweets;
	private Double tfidf;

	protected void setup(Context context) throws IOException,
InterruptedException
	{
		Configuration conf = context.getConfiguration();
		num_of_tweets = Integer.parseInt(conf.get("num_of_tweets"));
	}

	public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
	{
		int num_of_tweets_with_this_word = 0;
		ArrayList<String> value_list = new ArrayList<String>();

		for (Text value : values)
		{
			value_list.add(value.toString());
			num_of_tweets_with_this_word++;
		}

		// access the list created above, calculate the TFIDF for
each word and arrange the values in the required format
		for (String value : value_list)
		{
```

```
                  String[] value_arr = value.split("=");
                  String[] divide_data =
value_arr[1].toString().split("/");

                  tfidf =
(Double.parseDouble(divide_data[0])/Double.parseDouble(divide_data[1]))
* Math.log(num_of_tweets/num_of_tweets_with_this_word);

                  context.write(new Text(value_arr[0] + "@" +
key.toString()), new Text(tfidf.toString()));
            }
      }
      }



      /* input:  <(tweet@word), TFIDF>
      * output: <tweet, (word_TFIDF)>
      */
      public static class Map_FeatSel extends Mapper<Object, Text, Text,
Text>
      {
      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {
            String[] columns = value.toString().split("\t");

            String[] splitted_key = columns[0].toString().split("@");

            context.write(new Text(splitted_key[0]), new
Text(splitted_key[1] + "_" + columns[1]));
      }
      }

      /* input:  <tweet, (word_TFIDF)>
      * output: <tweet, text>
      */
      public static class Reduce_FeatSel extends Reducer<Text, Text,
Text, Text>
      {
      private HashMap tweet_words = new HashMap<Text, Double>();

      public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
      {

      context.getCounter(Global_Counters.TWEETS_SIZE).increment(1);

            String tweet_text = "";

            for(Text value : values)
            {
                  String[] columns = value.toString().split("_");
                  tweet_words.put(new Text(columns[0]),
Double.parseDouble(columns[1]));
            }

            // sort the tweet's words by their TFIDF score
            List list = new LinkedList(tweet_words.entrySet());
```

```
            int num_of_words = tweet_words.size();
            tweet_words.clear();

            Collections.sort(list, new Comparator()
                                    {
                                          public int compare(Object o1,
Object o2)
                                          {
                                          return ((Comparable)
((java.util.Map.Entry) (o1)).getValue())

      .compareTo(((java.util.Map.Entry) (o2)).getValue());
                                          }
                                    });

            HashMap sorted_tweet_words = new LinkedHashMap();
            for(Iterator i = list.iterator(); i.hasNext();)
            {
                  java.util.Map.Entry entry = (java.util.Map.Entry)
i.next();
                  sorted_tweet_words.put(entry.getKey(),
entry.getValue());
            }

            // hold the words with the biggest TFIDF score, by trimming
down the ones with the lowest score
            // until the 75% of the words remained in the list
            while((sorted_tweet_words.size() > ((num_of_words * 75) /
100)) && (num_of_words > 1))

      sorted_tweet_words.remove(sorted_tweet_words.keySet().stream().fin
dFirst().get());

            if(key.toString().endsWith("+"))
            {

      context.getCounter(Global_Counters.POS_TWEETS_SIZE).increment(1);

      context.getCounter(Global_Counters.POS_WORDS_SIZE).increment(sorte
d_tweet_words.size());
            }
            else
            {

      context.getCounter(Global_Counters.NEG_TWEETS_SIZE).increment(1);

      context.getCounter(Global_Counters.NEG_WORDS_SIZE).increment(sorte
d_tweet_words.size());
            }

            // put the most relevant words in a string
            Set set = sorted_tweet_words.entrySet();
            Iterator it = set.iterator();
            while(it.hasNext())
            {
                  java.util.Map.Entry me = (java.util.Map.Entry)
it.next();
                  tweet_text += me.getKey().toString() + " ";
            }
```

```
            context.write(key, new Text(tweet_text));
      }
      }



      /* input:  <tweet, text>
      * output: <word, sentiment>
      */
      public static class Map_Training extends Mapper<Object, Text,
Text, Text>
      {
      private Text word_key = new Text();
      private Text sentiment_value = new Text();

      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {
            String[] line = value.toString().split("\t");
            String[] tweet_words = line[1].toString().split(" ");

            if(line[0].endsWith("+"))
                  sentiment_value.set("POSITIVE");
            else
                  sentiment_value.set("NEGATIVE");

            for(String word : tweet_words)
            {
                  word_key.set(word);

                  context.write(word_key, sentiment_value);
            }
      }
      }

      /* input:  <word, sentiment>
      * output: <word, pos_wordcount@neg_wordcount>
      */
      public static class Reduce_Training extends Reducer<Text, Text,
Text, Text>
      {
      public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException
      {

      context.getCounter(Global_Counters.FEATURES_SIZE).increment(1);

            int positive_counter = 0;
            int negative_counter = 0;

            // for each word, count the occurrences in tweets with
positive/negative sentiment
            for(Text value : values)
            {
                  String sentiment = value.toString();

                  if(sentiment.equals("POSITIVE"))
                  positive_counter++;
                  else
                  negative_counter++;
```

```
            }

            context.write(key, new Text(String.valueOf(positive_counter)
+ "@" + String.valueOf(negative_counter)));
      }
      }



      /* input: <byte_offset, line_of_tweet>
      * output: <tweet@tweet_text, sentiment>
      */
      public static class Map_Testing extends Mapper<Object, Text, Text,
Text>
      {
      int features_size, tweets_size, pos_tweets_size, neg_tweets_size,
pos_words_size, neg_words_size;
      Double pos_class_probability, neg_class_probability;

      // hashmaps with each word as key and its number of occurrences in
each class as value
      HashMap<String, Integer> pos_words = new HashMap<String,
Integer>();
      HashMap<String, Integer> neg_words = new HashMap<String,
Integer>();

      // hashmaps with each word as key and its probability in each
class as value
      HashMap<String, Double> pos_words_probabilities = new
HashMap<String, Double>();
      HashMap<String, Double> neg_words_probabilities = new
HashMap<String, Double>();

      // lists holding all probabilities to be multiplied together,
along with the positive/negative class probability
      ArrayList<Double> pos_probabilities_list = new
ArrayList<Double>();
      ArrayList<Double> neg_probabilities_list = new
ArrayList<Double>();

      protected void setup(Context context) throws IOException,
InterruptedException
      {
            // load all counters to be used for the calculation of the
probabilities
            features_size =
Integer.parseInt(context.getConfiguration().get("features_size"));
            tweets_size =
Integer.parseInt(context.getConfiguration().get("tweets_size"));
            pos_tweets_size =
Integer.parseInt(context.getConfiguration().get("pos_tweets_size"));
            neg_tweets_size =
Integer.parseInt(context.getConfiguration().get("neg_tweets_size"));
            pos_words_size =
Integer.parseInt(context.getConfiguration().get("pos_words_size"));
            neg_words_size =
Integer.parseInt(context.getConfiguration().get("neg_words_size"));

            pos_class_probability = ((double) pos_tweets_size) /
tweets_size;
```

```
            neg_class_probability = ((double) neg_tweets_size) /
tweets_size;

            // load the model of the last training job and fill two
hashmaps of words with the number of
            // occurrences in positive and negative tweets
            Path training_model = new Path("training");
            FileSystem model_fs =
training_model.getFileSystem(context.getConfiguration());
            FileStatus[] file_status =
model_fs.listStatus(training_model);

            for(FileStatus i : file_status)
            {
                  Path current_file_path = i.getPath();

                  if(i.isFile())
                  {
                  BufferedReader br = new BufferedReader(new
InputStreamReader(model_fs.open(current_file_path)));
                  String line;

                  while((line = br.readLine()) != null)
                  {
                        String[] columns = line.toString().split("\t");
                        String[] pos_and_neg_counts =
columns[1].split("@");

                        pos_words.put(columns[0],
Integer.parseInt(pos_and_neg_counts[0]));
                        neg_words.put(columns[0],
Integer.parseInt(pos_and_neg_counts[1]));
                  }

                  br.close();
                  }
            }

            // calculate all the word probabilities for positive and
negative class (with laplace smoothing)
            for(Map.Entry<String,Integer> entry : pos_words.entrySet())
            {
                  pos_words_probabilities.put(entry.getKey(), ((double)
entry.getValue() + 1) / (pos_words_size + features_size));
                  neg_words_probabilities.put(entry.getKey(), ((double)
neg_words.get(entry.getKey()) + 1) / (neg_words_size + features_size));
            }
      }

      public void map(Object key, Text value, Context context) throws
IOException, InterruptedException
      {
            String line = value.toString();
            String[] columns = line.split(",");

            // if the columns are more than 4, that means the text of
the post had commas inside,
            // so stitch the last columns together to form the post
            if(columns.length > 4)
            {
```

```
                  for(int i=4; i<columns.length; i++)
                  columns[3] += columns[i];
            }

            String tweet_id = columns[0];
            String tweet_sentiment = columns[1];
            String tweet_text = columns[3];

            // clean the text of the tweet from links..
            tweet_text =
tweet_text.replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9][a-zA-Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-
]+[a-zA-Z0-9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")
                              .replaceAll("(#|@|&).*?\\w+", "")   //
mentions, hashtags, special characters...
                              .replaceAll("\\d+", "")             //
numbers...
                              .replaceAll("[^a-zA-Z ]", " ")      //
punctuation...
                              .toLowerCase()                      //
turn every character left to lowercase...
                              .trim()                             //
trim the spaces before & after the whole string...
                              .replaceAll("\\s+", " ");           // and
get rid of double spaces

            // initialize the product of positive and negative
probabilities with 1
            Double pos_probability = 1.0;
            Double neg_probability = 1.0;

            // calculate the product of the probabilities of the words
(+ the class probability) for each class
            if(tweet_text != null && !tweet_text.trim().isEmpty())
            {
                  String[] tweet_words = tweet_text.split(" ");

                  for(String word : tweet_words)
                  {
                  for(Map.Entry<String,Double> entry :
pos_words_probabilities.entrySet())
                  {
                        if(word.equals(entry.getKey()))
                        {
                              pos_probability *=
pos_words_probabilities.get(word);
                              neg_probability *=
neg_words_probabilities.get(word);
                        }
                  }
                  }
            }

            // multiply the product of positive and negative probability
with the class probability of each sentiment
            pos_probability *= pos_class_probability;
            neg_probability *= neg_class_probability;

            // compare and set the max value of the two class
```

```
probabilities as the result of the guessed sentiment for every tweet
            if(Double.compare(pos_probability, neg_probability) > 0)
            {
                  if(tweet_sentiment.equals("1"))

      context.getCounter(Global_Counters.TRUE_POSITIVE).increment(1);
                  else

      context.getCounter(Global_Counters.FALSE_POSITIVE).increment(1);

                  context.write(new Text(tweet_id + "@" + tweet_text),
new Text("POSITIVE"));
            }
            else
            {
                  if(tweet_sentiment.equals("0"))

      context.getCounter(Global_Counters.TRUE_NEGATIVE).increment(1);
                  else

      context.getCounter(Global_Counters.FALSE_NEGATIVE).increment(1);

                  context.write(new Text(tweet_id + "@" + tweet_text),
new Text("NEGATIVE"));
            }
      }
      }



      public static void main(String[] args) throws Exception
      {
      // paths to directories were input, inbetween and final job
outputs are stored
      Path input_dir = new Path(args[0]);
      Path wordcount_dir = new Path("wordcount");
      Path tf_dir = new Path("tf");
      Path tfidf_dir = new Path("tfidf");
      Path features_dir = new Path("features");
      Path training_dir = new Path("training");
      Path testing_dir = new Path(args[1]);
      Path output_dir = new Path("output");

      Configuration conf = new Configuration();

      FileSystem fs = FileSystem.get(conf);
      if(fs.exists(wordcount_dir))
            fs.delete(wordcount_dir, true);
      if(fs.exists(tf_dir))
            fs.delete(tf_dir, true);
      if(fs.exists(tfidf_dir))
            fs.delete(tfidf_dir, true);
      if(fs.exists(features_dir))
            fs.delete(features_dir, true);
      if(fs.exists(training_dir))
            fs.delete(training_dir, true);
      if(fs.exists(output_dir))
            fs.delete(output_dir, true);

      long start_time = System.nanoTime();
```

```
      Job wordcount_job = Job.getInstance(conf, "Word Count");
      wordcount_job.setJarByClass(Modified_NB.class);
      wordcount_job.setMapperClass(Map_WordCount.class);
      wordcount_job.setCombinerClass(Reduce_WordCount.class);
      wordcount_job.setReducerClass(Reduce_WordCount.class);
      wordcount_job.setNumReduceTasks(3);
      wordcount_job.setMapOutputKeyClass(Text.class);
      wordcount_job.setMapOutputValueClass(IntWritable.class);
      wordcount_job.setOutputKeyClass(Text.class);
      wordcount_job.setOutputValueClass(IntWritable.class);
      TextInputFormat.addInputPath(wordcount_job, input_dir);
      TextInputFormat.setMaxInputSplitSize(wordcount_job,
Long.valueOf(args[2]));
      TextOutputFormat.setOutputPath(wordcount_job, wordcount_dir);
      wordcount_job.waitForCompletion(true);

      // Counting the total number of tweets from the training data in
order to calculate the TFIDF score of each feature
      int num_of_tweets =
Math.toIntExact(wordcount_job.getCounters().findCounter(Global_Counters.
NUM_OF_TWEETS).getValue());
      conf.set("num_of_tweets", String.valueOf(num_of_tweets));

      Job tf_job = Job.getInstance(conf, "TF");
      tf_job.setJarByClass(Modified_NB.class);
      tf_job.setMapperClass(Map_TF.class);
      tf_job.setReducerClass(Reduce_TF.class);
      tf_job.setNumReduceTasks(3);
      tf_job.setMapOutputKeyClass(Text.class);
      tf_job.setMapOutputValueClass(Text.class);
      tf_job.setOutputKeyClass(Text.class);
      tf_job.setOutputValueClass(Text.class);
      FileInputFormat.addInputPath(tf_job, wordcount_dir);
      FileOutputFormat.setOutputPath(tf_job, tf_dir);
      tf_job.waitForCompletion(true);

      Job tfidf_job = Job.getInstance(conf, "TFIDF");
      tfidf_job.setJarByClass(Modified_NB.class);
      tfidf_job.setMapperClass(Map_TFIDF.class);
      tfidf_job.setReducerClass(Reduce_TFIDF.class);
      tfidf_job.setNumReduceTasks(3);
      tfidf_job.setMapOutputKeyClass(Text.class);
      tfidf_job.setMapOutputValueClass(Text.class);
      tfidf_job.setOutputKeyClass(Text.class);
      tfidf_job.setOutputValueClass(Text.class);
      FileInputFormat.addInputPath(tfidf_job, tf_dir);
      FileOutputFormat.setOutputPath(tfidf_job, tfidf_dir);
      tfidf_job.waitForCompletion(true);

      Job feature_selection_job = Job.getInstance(conf, "Feature
Selection");
      feature_selection_job.setJarByClass(Modified_NB.class);
      feature_selection_job.setMapperClass(Map_FeatSel.class);
      feature_selection_job.setReducerClass(Reduce_FeatSel.class);
      feature_selection_job.setNumReduceTasks(3);
      feature_selection_job.setMapOutputKeyClass(Text.class);
      feature_selection_job.setMapOutputValueClass(Text.class);
      feature_selection_job.setOutputKeyClass(Text.class);
      feature_selection_job.setOutputValueClass(Text.class);
```

```
      FileInputFormat.addInputPath(feature_selection_job, tfidf_dir);
      FileOutputFormat.setOutputPath(feature_selection_job,
features_dir);
      feature_selection_job.waitForCompletion(true);

      int tweets_size =
Math.toIntExact(feature_selection_job.getCounters().findCounter(Global_C
ounters.TWEETS_SIZE).getValue());
      conf.set("tweets_size", String.valueOf(tweets_size));
      int pos_tweets_size =
Math.toIntExact(feature_selection_job.getCounters().findCounter(Global_C
ounters.POS_TWEETS_SIZE).getValue());
      conf.set("pos_tweets_size", String.valueOf(pos_tweets_size));
      int neg_tweets_size =
Math.toIntExact(feature_selection_job.getCounters().findCounter(Global_C
ounters.NEG_TWEETS_SIZE).getValue());
      conf.set("neg_tweets_size", String.valueOf(neg_tweets_size));
      int pos_words_size =
Math.toIntExact(feature_selection_job.getCounters().findCounter(Global_C
ounters.POS_WORDS_SIZE).getValue());
      conf.set("pos_words_size", String.valueOf(pos_words_size));
      int neg_words_size =
Math.toIntExact(feature_selection_job.getCounters().findCounter(Global_C
ounters.NEG_WORDS_SIZE).getValue());
      conf.set("neg_words_size", String.valueOf(neg_words_size));

      Job training_job = Job.getInstance(conf, "Training");
      training_job.setJarByClass(Modified_NB.class);
      training_job.setMapperClass(Map_Training.class);
      training_job.setReducerClass(Reduce_Training.class);
      training_job.setNumReduceTasks(3);
      training_job.setMapOutputKeyClass(Text.class);
      training_job.setMapOutputValueClass(Text.class);
      training_job.setOutputKeyClass(Text.class);
      training_job.setOutputValueClass(Text.class);
      FileInputFormat.addInputPath(training_job, features_dir);
      FileOutputFormat.setOutputPath(training_job, training_dir);
      training_job.waitForCompletion(true);

      int features_size =
Math.toIntExact(training_job.getCounters().findCounter(Global_Counters.F
EATURES_SIZE).getValue());
      conf.set("features_size", String.valueOf(features_size));

      Job testing_job = Job.getInstance(conf, "Testing");
      testing_job.setJarByClass(Modified_NB.class);
      testing_job.setMapperClass(Map_Testing.class);
      testing_job.setMapOutputKeyClass(Text.class);
      testing_job.setMapOutputValueClass(Text.class);
      testing_job.setOutputKeyClass(Text.class);
      testing_job.setOutputValueClass(Text.class);
      TextInputFormat.addInputPath(testing_job, testing_dir);
      TextInputFormat.setMaxInputSplitSize(testing_job,
Long.valueOf(args[3]));
      TextOutputFormat.setOutputPath(testing_job, output_dir);
      testing_job.waitForCompletion(true);

      System.out.println("EXECUTION DURATION: " + (System.nanoTime() -
start_time) / 1000000000F + " seconds");
```

```
      int tp =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.TR
UE_POSITIVE).getValue());
      int fp =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.FA
LSE_POSITIVE).getValue());
      int tn =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.TR
UE_NEGATIVE).getValue());
      int fn =
Math.toIntExact(testing_job.getCounters().findCounter(Global_Counters.FA
LSE_NEGATIVE).getValue());

      System.out.println("\nCONFUSION MATRIX:");
      System.out.printf("%-10s %-10s \n", tp, fp);
      System.out.printf("%-10s %-10s \n\n", fn, tn);

      System.out.printf("%-25s %-10s \n", "ACCURACY: ", ((double) (tp +
tn)) / (tp + tn + fp + fn));
      }
}
```

Υλοποίηση Naïve Bayes στο Spark

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark._
import org.apache.spark.SparkContext._

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
import org.apache.spark.ml.classification.NaïveBayes
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

import org.apache.spark.mllib.evaluation.MulticlassMetrics


object NB
{
      // function that splits each line of the .csv file by commas,
while being cautious at the
      // commas that might exist inside the last quoted column of the
line with the tweet text
      def split_csv(line:String) : Array[String] =
      {
      var columns = line.split(",");

      // if the columns are more than 4, that means the text of the post
had commas inside,
      // so stitch the last columns together to form the full text of
the tweet
      if(columns.length > 4)
      {
            for(i <- 4 until columns.length)
```

```
                columns(3) += columns(i);
      }

      return columns
      }

      def main(args: Array[String]): Unit =
      {
      // create a scala spark context for rdd management and a spark
session for dataframe management
      val conf = new SparkConf().setAppName("Naïve Bayes")
      val sc = new SparkContext(conf)

      val start_time = System.nanoTime()

      // read the .csv file with the training data
      val input = sc.textFile("hdfs://mpi6:19000/user/mpi/spark_input_"
+ args(0) + "/tweets.csv", 3)
                        .map(line => split_csv(line))       // split
each line to columns...
                        .map(column =>
                                    {   // map the cleaned up tweet text
as key and sentiment as value
                                        (
                                        column(1).toDouble // set the
sentiment of the tweet as key
                                        ,
                                        column(3) // set the cleaned
up tweet text as value, by cleaning up the text from...

      .replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-9][a-zA-
Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")

      .replaceAll("(#|@|&).*?\\w+", "")   // mentions, hashtags, special
characters...
                                                .replaceAll("\\d+", "")
      // numbers...
                                                .replaceAll("[^a-zA-Z
]", " ")    // punctuation...
                                                .toLowerCase()
      // turn every character left to lowercase...
                                                .trim()
      // trim the spaces before & after the whole string...
                                                .replaceAll("\\s+", " ")
      // and get rid of double spaces
                                        )
                                    }
                            )


      // convert the RDD type sets of training data to DataFrames with
named columns in order to apply the TFIDF measure on them
      import spark.implicits._
      val input_dataframe = input.toDF("label", "tweet")


      // apply TFIDF to the text of training data to have the proper
form of (double, Vectors(double[])) used at the Naïve Bayes classifier
```

```
      val tokenizer = new
Tokenizer().setInputCol("tweet").setOutputCol("words")      // split the
text of each tweet to tokens
      val words_data = tokenizer.transform(input_dataframe)

      val input_hashingTF = new
HashingTF().setInputCol("words")setOutputCol("rawFeatures")   //
calculate TF
      val input_featurized_data = input_hashingTF.transform(words_data)

      val input_idf = new
IDF().setInputCol("rawFeatures").setOutputCol("features")         //
calculate IDF
      val input_idf_model = input_idf.fit(input_featurized_data)

      val input_rescaled_data =
input_idf_model.transform(input_featurized_data)            // calculate
TFIDF
      //input_rescaled_data.select("label", "features").show()

      val Array(training_data, test_data) =
input_rescaled_data.randomSplit(Array(0.75, 0.25), seed = 1234L)

      // create the Naïve Bayes model, train it with the train data and
classify/predict the test data
      val model = new NaïveBayes().fit(training_data)
      val predictions = model.transform(test_data)
      //predictions.show()
      val end_time = System.nanoTime()


      // select each (prediction, true label) set and compute the test
error, convert them to RDD, and use the MulticlassMetrics class
      // to output the confusion matrix and some metrics
      val prediction_and_labels = predictions.select("prediction",
"label").rdd.map(r => (r.getDouble(0), r.getDouble(1)))

      val metrics = new MulticlassMetrics(prediction_and_labels)
      println(metrics.confusionMatrix)
      println("ACCURACY: " + metrics.accuracy)
      println("F1 SCORE: " + metrics.weightedFMeasure)
      println("EXECUTION DURATION: " + (end_time - start_time) /
1000000000F + " seconds")

      sc.stop()
      }
}
```

Υλοποίηση Τροποποιημένου Naïve Bayes στο Spark

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark._
import org.apache.spark.SparkContext._

import org.apache.spark.rdd.RDD
```

```
import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
import org.apache.spark.ml.classification.NaïveBayes
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

import org.apache.spark.mllib.evaluation.MulticlassMetrics


object Modified_NB
{
      // function that splits each line of the .csv file by commas,
while being cautious at the
      // commas that might exist inside the last quoted column of the
line with the tweet text
      def split_csv(line:String) : Array[String] =
      {
      var columns = line.split(",");

      // if the columns are more than 4, that means the text of the post
had commas inside,
      // so stitch the last columns together to form the full text of
the tweet
      if(columns.length > 4)
      {
            for(i <- 4 until columns.length)
                  columns(3) += columns(i);
      }

      return columns
      }

      def main(args: Array[String]): Unit =
      {
      // create a scala spark context for rdd management and a spark
session for dataframe management
      val conf = new SparkConf().setAppName("Modified Naïve Bayes")
      val sc = new SparkContext(conf)

      val start_time = System.nanoTime()

      // read the .csv file with the training data
      val input = sc.textFile("hdfs://mpi6:19000/user/mpi/spark_input_"
+ args(0) + "/tweets.csv", 3)
                        .map(line => split_csv(line))       // split
each line to columns...
                        .map(column =>
                                    {   // map the cleaned up tweet text
as key and sentiment as value
                                          (
                                          column(1).toDouble // set the
sentiment of the tweet as key
                                          ,
                                          column(3) // set the cleaned
up tweet text as value, by cleaning up the text from...

      .replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-9][a-zA-
Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")
```

```
      .replaceAll("(#|@|&).*?\\w+", "")   // mentions, hashtags, special
characters...
                                                .replaceAll("\\d+", "")
      // numbers...
                                                .replaceAll("[^a-zA-Z
]", " ")    // punctuation...
                                                .toLowerCase()
      // turn every character left to lowercase...
                                                .trim()
      // trim the spaces before & after the whole string...
                                                .replaceAll("\\s+", " ")
      // and get rid of double spaces
                                          )
                                    }
                              )


      // convert the RDD type sets of training data to dataframes with
named columns in order to apply the TFIDF measure on them
      import spark.implicits._
      val input_dataframe = input.toDF("label", "tweet")

      // apply TFIDF to the text of training data to have the proper
form of (double, Vectors(double[])) used at the Naïve Bayes classifier
      val tokenizer = new
Tokenizer().setInputCol("tweet").setOutputCol("words")      // split the
text of each tweet to tokens
      val words_data = tokenizer.transform(input_dataframe)

      val input_hashingTF = new
HashingTF().setInputCol("words")setOutputCol("rawFeatures")
      // calculate TF
      val input_featurized_data = input_hashingTF.transform(words_data)

      val input_idf = new
IDF().setMinDocFreq(5).setInputCol("rawFeatures").setOutputCol("features
")  // calculate IDF
      val input_idf_model = input_idf.fit(input_featurized_data)

      val input_rescaled_data =
input_idf_model.transform(input_featurized_data)
      // calculate TFIDF
      //input_rescaled_data.select("label", "features").show()

      val Array(training_data, test_data) =
input_rescaled_data.randomSplit(Array(0.75, 0.25), seed = 1234L)

      // create the Naïve Bayes model, train it with the train data and
classify/predict the test data
      val model = new NaïveBayes().fit(training_data)
      val predictions = model.transform(test_data)
      val end_time = System.nanoTime()

      // select each (prediction, true label) set and compute the test
error, convert them to RDD, and use the MulticlassMetrics class
      // to output the confusion matrix and some metrics
      val prediction_and_labels = predictions.select("prediction",
"label").rdd.map(r => (r.getDouble(0), r.getDouble(1)))
```

```
      val metrics = new MulticlassMetrics(prediction_and_labels)
      println(metrics.confusionMatrix)
      println("ACCURACY: " + metrics.accuracy)
      println("F1 SCORE: " + metrics.weightedFMeasure)
      println("EXECUTION DURATION: " + (end_time - start_time) /
1000000000F + " seconds")

      sc.stop()
      }
}
```

Υλοποίηση Support Vector Machines στο Spark

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark._
import org.apache.spark.SparkContext._

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
import org.apache.spark.ml.classification.LinearSVC

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.evaluation.MulticlassMetrics


object SVM
{
      // function that splits each line of the .csv file by commas,
while being cautious at the
      // commas that might exist inside the last quoted column of the
line with the tweet text
      def split_csv(line:String) : Array[String] =
      {
      var columns = line.split(",");

      // if the columns are more than 4, that means the text of the post
had commas inside,
      // so stitch the last columns together to form the full text of
the tweet
      if(columns.length > 4)
      {
            for(i <- 4 until columns.length)
                  columns(3) += columns(i);
      }

      return columns
      }

      def main(args: Array[String]): Unit =
      {
      // create a scala spark context for rdd management and a spark
session for dataframe management
      val conf = new SparkConf().setAppName("Support Vector Machines")
```

```
      val sc = new SparkContext(conf)

      val start_time = System.nanoTime()

      // read the .csv file with the training data
      val input = sc.textFile("hdfs://mpi6:19000/user/mpi/spark_input_"
+ args(0) + "/tweets.csv", 3)
                        .map(line => split_csv(line))       // split
each line to columns...
                        .map(column =>
                                    {   // map the cleaned up tweet text
as key and sentiment as value
                                          (
                                          column(1).toDouble // set the
sentiment of the tweet as key
                                          ,
                                          column(3) // set the cleaned
up tweet text as value, by cleaning up the text from...

      .replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-9][a-zA-
Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")

      .replaceAll("(#|@|&).*?\\w+", "")   // mentions, hashtags, special
characters...
                                                .replaceAll("\\d+", "")
      // numbers...
                                                .replaceAll("[^a-zA-Z
]", " ")    // punctuation...
                                                .toLowerCase()
      // turn every character left to lowercase...
                                                .trim()
      // trim the spaces before & after the whole string...
                                                .replaceAll("\\s+", " ")
      // and get rid of double spaces
                                          )
                                    }
                              )


      // convert the RDD type sets of training data to dataframes with
named columns in order to apply the TFIDF measure on them
      import spark.implicits._
      val input_dataframe = input.toDF("label", "tweet")


      // apply TFIDF to the text of training data to have the proper
form of (double, Vectors(double[])) used at the Naïve Bayes classifier
      val tokenizer = new
Tokenizer().setInputCol("tweet").setOutputCol("words")      // split the
text of each tweet to tokens
      val words_data = tokenizer.transform(input_dataframe)

      val input_hashingTF = new
HashingTF().setInputCol("words")setOutputCol("rawFeatures")   //
calculate TF
      val input_featurized_data = input_hashingTF.transform(words_data)

      val input_idf = new
```

```
IDF().setInputCol("rawFeatures").setOutputCol("features")         //
calculate IDF
      val input_idf_model = input_idf.fit(input_featurized_data)

      val input_rescaled_data =
input_idf_model.transform(input_featurized_data)            // calculate
TFIDF

      val Array(training_data, test_data) =
input_rescaled_data.randomSplit(Array(0.75, 0.25), seed = 1234L)

      // create the SVM model, train it with the train data and
classify/predict the test data
      val lsvc = new LinearSVC().setMaxIter(10).setRegParam(0.1)
      val lsvc_model = lsvc.fit(training_data)
      val predictions = lsvc_model.transform(test_data)
      val end_time = System.nanoTime()

      // select each (prediction, true label) set and compute the test
error, convert them to RDD, and use the MulticlassMetrics class
      // to output the confusion matrix and some metrics
      val prediction_and_labels = predictions.select("prediction",
"label").rdd.map(r => (r.getDouble(0), r.getDouble(1)))

      val metrics = new MulticlassMetrics(prediction_and_labels)
      println(metrics.confusionMatrix)
      println("ACCURACY: " + metrics.accuracy)
      println("F1 SCORE: " + metrics.weightedFMeasure)
      println("EXECUTION DURATION: " + (end_time - start_time) /
1000000000F + " seconds")

      sc.stop()
      }
}
```

Υλοποίηση Τροποποιημένου Support Vector Machines στο Spark

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark._
import org.apache.spark.SparkContext._

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}
import org.apache.spark.ml.classification.LinearSVC

import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.mllib.evaluation.MulticlassMetrics


object Modified_SVM
{
      // function that splits each line of the .csv file by commas,
while being cautious at the
```

```
      // commas that might exist inside the last quoted column of the
line with the tweet text
      def split_csv(line:String) : Array[String] =
      {
      var columns = line.split(",");

      // if the columns are more than 4, that means the text of the post
had commas inside,
      // so stitch the last columns together to form the full text of
the tweet
      if(columns.length > 4)
      {
            for(i <- 4 until columns.length)
                  columns(3) += columns(i);
      }

      return columns
      }

      def main(args: Array[String]): Unit =
      {
      // create a scala spark context for rdd management and a spark
session for dataframe management
      val conf = new SparkConf().setAppName("Modified Support Vector
Machines")
      val sc = new SparkContext(conf)

      val start_time = System.nanoTime()

      // read the .csv file with the training data
      val input = sc.textFile("hdfs://mpi6:19000/user/mpi/spark_input_"
+ args(0) + "/tweets.csv", 3)
                        .map(line => split_csv(line))       // split
each line to columns...
                        .map(column =>
                                    {   // map the cleaned up tweet text
as key and sentiment as value
                                          (
                                          column(1).toDouble // set the
sentiment of the tweet as key
                                          ,
                                          column(3) // set the cleaned
up tweet text as value, by cleaning up the text from...

      .replaceAll("(?i)(https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-9][a-zA-
Z0-9-]+[a-zA-Z0-9]\\.[^\\s]{2,}|www\\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-
9]\\.[^\\s]{2,}|https?:\\/\\/(?:www\\.|(?!www))[a-zA-Z0-
9]+\\.[^\\s]{2,}|www\\.[a-zA-Z0-9]+\\.[^\\s]{2,})", "")

      .replaceAll("(#|@|&).*?\\w+", "")   // mentions, hashtags, special
characters...
                                                .replaceAll("\\d+", "")
      // numbers...
                                                .replaceAll("[^a-zA-Z
]", " ")    // punctuation...
                                                .toLowerCase()
      // turn every character left to lowercase...
                                                .trim()
      // trim the spaces before & after the whole string...
                                                .replaceAll("\\s+", " ")
```

```
      // and get rid of double spaces
                                          )
                                    }
                              )


      // convert the RDD type sets of training data to dataframes with
named columns in order to apply the TFIDF measure on them
      import spark.implicits._
      val input_dataframe = input.toDF("label", "tweet")


      // apply TFIDF to the text of training data to have the proper
form of (double, Vectors(double[])) used at the Naïve Bayes classifier
      val tokenizer = new
Tokenizer().setInputCol("tweet").setOutputCol("words")      // split the
text of each tweet to tokens
      val words_data = tokenizer.transform(input_dataframe)

      val input_hashingTF = new
HashingTF().setInputCol("words").setOutputCol("rawFeatures")
      // calculate TF
      val input_featurized_data = input_hashingTF.transform(words_data)

      val input_idf = new
IDF().setMinDocFreq(5).setInputCol("rawFeatures").setOutputCol("features
")  // calculate IDF
      val input_idf_model = input_idf.fit(input_featurized_data)

      val input_rescaled_data =
input_idf_model.transform(input_featurized_data)
      // calculate TFIDF

      val Array(training_data, test_data) =
input_rescaled_data.randomSplit(Array(0.75, 0.25), seed = 1234L)

      // create the SVM model, train it with the train data and
classify/predict the test data
      val lsvc = new LinearSVC().setMaxIter(10).setRegParam(0.1)
      val lsvc_model = lsvc.fit(training_data)
      val predictions = lsvc_model.transform(test_data)
      val end_time = System.nanoTime()

      // select each (prediction, true label) set and compute the test
error, convert them to RDD, and use the MulticlassMetrics class
      // to output the confusion matrix and some metrics
      val prediction_and_labels = predictions.select("prediction",
"label").rdd.map(r => (r.getDouble(0), r.getDouble(1)))

      val metrics = new MulticlassMetrics(prediction_and_labels)
      println(metrics.confusionMatrix)
      println("ACCURACY: " + metrics.accuracy)
      println("F1 SCORE: " + metrics.weightedFMeasure)
      println("EXECUTION DURATION: " + (end_time - start_time) /
1000000000F + " seconds")

      sc.stop()
      }
}
```