ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών

Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών

Ειδίκευση Λογισμικού και Πληροφοριακών Συστημάτων,

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παράλληλοι Αλγόριθμοι Κατηγοριοποίησης Κειμένου σε Κατανεμημένο Περιβάλλον με χρήση Spark

Parallel Text Classification Algorithms in Distributed Environments using Spark

Πλάντζα Παναγιώτα Δανάη
A.M. Mcse19056

Εισηγητής: **Μάμαλης Βασίλης**

(Κενό φύλλο)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παράλληλοι Αλγόριθμοι Κατηγοριοποίησης Κειμένου σε Κατανεμημένο Περιβάλλον με χρήση Spark

Πλάντζα Παναγιώτα Δανάη
A.M. mcse19056

Εισηγητής:

Μάμαλης Βασίλης

Vasileios Mamalis
Digitally signed by Vasileios Mamalis
Date: 2022.07.22 02:54:47 +03'00'

Εξεταστική Επιτροπή:

Πάντζιου Γραμματή
Καντζάβελου Ιωάννα

Grammati Pantziou
Digitally signed by Grammati Pantziou
Date: 2022.07.22 11:21:32 +03'00'

Ioanna Kantzavelou
Digitally signed by Ioanna Kantzavelou
Date: 2022.07.22 10:21:49 +03'00'

Ημερομηνία εξέτασης 21/07/2022

(Κενό φύλλο)

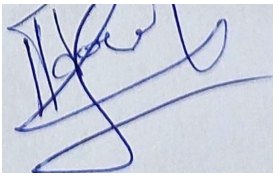
ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Η κάτωθι υπογεγραμμένη Πλάντζα Παναγιώτα Δανάη του Κωνσταντίνου, με αριθμό μητρώου mcse19056, φοιτήτρια του Προγράμματος Μεταπτυχιακών Σπουδών Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών, ειδίκευση Λογισμικού και Πληροφοριακών Συστημάτων του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών.της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής 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/Η Δηλών/ούσα



(Κενό φύλλο)

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία ασχολείται με την παράλλη κατηγοριοποίησης κειμένου σε κατανεμημένο περιβάλλον. Μελετήθηκε η εξαγωγή χαρακτηριστικών από κείμενο και η κατηγοριοποίηση του κειμένου βάση αυτών των χαρακτηριστικων. Η μελέτη επικεντρώθηκε στο σειριακός αλγόριθμος κατηγοριοποιησης SVM, και στους τρόπους παραλληλοποίησης του. Υλοποιήθηκαν 2 εκδόσεις παράλληλου SVM για κατηγοριοποίηση μεταξύ δύο και τριών κλάσεων. Η υλοποίηση των αλγορίθμων εγινε σε περιβάλλον spark. Συλλέχθηκαν Tweets σε μια βάση Cassandra μέσω της διεπαφής του Twitter για Spark. Στη συνέχεια εκπαιδεύτηκαν μοντέλα με τις διαφορετικές εκδόσεις του SVM βάση αυτών των δεδομένων και τέλος συγκρίθηκε η απόδοση των μοντέλων σε χρόνο και ακρίβεια.

ABSTRACT

This thesis deals with the parallel classification of text in a distributed environment. It studies the extraction of features from text and the use of these features to classify the text. The study focuses on the serial SVM classification algorithm, and researches techniques to parallelize it. Two versions of parallel SVM were implemented for text classification between two and three classes. The algorithms were implemented with the use of Spark engine. To analyze the performance of implemented algorithms tweets were collected on a cassandra database via the Spark Twitter Streaming API. With this dataset we trained a model for each case with different versions of SVM. Finally the performance of the models was compared in time and accuracy on a diffrent dataset.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Παράλληλη Επεξεργασία
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: παράλληλος αλγόριθμος, ταξινόμηση, κατηγοριοποίηση, SVM, SMO, Tweeter, Cassada

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

1.1 Περιγραφή του αντικειμένου της διπλωματικής εργασίας

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη του SVM αλγόριθμος για κατηγοριοποίηση κειμενου και η παραλληλοποίηση του. Θα παρουσιαστει η υλοποίηση μιας ολοκληρωμένης εγκατάστασης από την συλλογή των δεδομένων εκπαίδευσης και ελέγχου μέχρι την εκπαίδευση του μοντέλου, τη κατηγοριοποίηση των δεδομένων ελέγχου και τέλος την αξιολόγηση του κάθε μοντέλου. Τα αποτελέσματα από διαφορετικές τακτικές παραλληλοποίησης θα συγκριθούν μεταξυ τους. Επίσης θα υλοποιηθούν και διαφορετικές εκδόσεις του αλγορίθμου για την υποστήριξη της κατηγοριοποίησης μεταξύ 3 και άνω κλάσεων πέραν της κλασικής δυαδικής κατηγοριοποίησης που υποστηρίζεται από τον βασικό SVM αλγόριθμο.

1.2 Θεωρητικό Υπόβαθρο

Όσο περνάνε τα χρόνια ο όγκος των διαθέσιμων δεδομένων μεγαλώνει, έτσι όλο και περισσότερα συστήματα και εφαρμογές χρησιμοποιούν κατανεμημένες λύσεις για την επεξεργασία τους. Έχει δοθεί λοιπόν μεγάλη έμφαση στην έρευνα και ανάπτυξη του επιστημονικού πεδίου που ασχολείται με τον κατανεμημένο υπολογισμό τεράστιου όγκου δεδομένων και με την κατανεμημένη αποθήκευση τους . Κύριοι

στόχοι την έρευνας να βρεθούν λύσεις που προσφέρουν γρήγορη, αποτελεσματική και ασφαλή επεξεργασία και αποθήκευση.

1.2.1 Μεγάλα Δεδομένα

Με τον όρο Big Data αναφερόμαστε σε πολύ μεγάλο όγκο δεδομένων , τα οποία συλλέγονται χωρίς να ακολουθούν κάποια καθορισμένη δομή πράγμα που κάνει την ανάλυση τους δύσκολη . Τα δεδομένα αυτά συλλέγονται με μεγάλους ρυθμούς αλλα δεν αποτελούν κάποια πηγή γνώσης αφού είναι ακαθόριστης μορφής και περιέχουν πολύ θόρυβο. Με την έννοια θόρυβο εννοούμε ότι τα δεδομένα δεν είναι απολύτως αληθινά ή σωστά. Λόγο των ιδιαίτερων αυτών χαρακτηριστικών των big data έχει υπάρξει η ανάγκη για μεγάλη επεξεργαστική δύναμη για την ανάλυση του και τεράστιου αποθηκευτικού χώρος για την συντήρηση τους. Σε αυτό το σημείο έρχεται να βοηθήσει η κατανεμημένη υπολογιστική και τα κατανεμημένα συστήματα αρχείων.

1.2.2 Κατανεμημένα υπολογιστικά συστήματα και Κατανεμημένα συστήματα αποθήκευσης

Ένα σύστημα αρχείων ονομάζουμε κατανεμημένο όταν είναι διαμοιρασμένο σε πολλούς κόμβους σε αντίθεση με ενα κεντροποιημένο συστημα που όλα τα αρχεία βρίσκονται αποθηκευμένα σε ένα φυσικό μηχάνημα. Η κατανομή των δεδομένων σε κόμβους συνδεδεμένους μεταξεί τους γίνεται με τρόπο τέτοιο τρόπο ώστε ο τελικός χρήστης να μην επηρεαζεται ,έτσι για αυτόν η πρόσβαση στα αρχεία γίνεται κάτω από ένα φαινομενικά ενιαίο σύστημα. Το προφανή πλεονέκτημα τέτοιων συστημάτων είναι ότι μπορούν να υποστηρίξουν πολύ μεγαλύτερη ποσότητα δεδομένων, αλλα και

ακόμα και όταν θα φτάσουν το όριο τους μπορούμε εύκολα να τα κλιμακωσουμε οριζόντια προσθετοντας κι άλλους κόμβους.

Πλεονεκτήματα

- Υψηλή διαθεσιμότητα

  Διαθεσιμότητα αρχείων ακόμα και αν κάποιος κόμβος πέσει.
- Πολλαπλοί χρήστες
- Χρήση δεδομένων από απόσταση
- Δινατότητα οριζόντιας κλιμάκωσης

  Μπορεί να υποστηρίξει μεγαλύτερο όγκο δεδομένων με προσθήκη κόμβων

Μειονεκτήματα

- Χρειάζεται προσοχή στην ασφάλεια
- Μπορεί να υπάρχουν καθυστερήσεις λόγω δικτύου
- Πιο περίπλοκο

Παραδείγματα

- NFS – Network File System.
- CIFS – Common Internet File System. Απόγονος του SMB.
- SMB – Server Message Block
- Hadoop.
- NetWare

1.3 Χρησιμοποιούμενα Εργαλεία

1.3.1 Κατανεμημένος προγραμματισμός σε Hadoop

Το Hadoop είναι μια συλλογή από εργαλεία ανοικτού κώδικα που παρέχουν υπηρεσίες που βοηθούν τη διαχείριση ανάλυση και αποθήκευση μεγάλου όγκου δεδομένων σε κατανεμημένο περιβάλλον.
Αποτελείται από τα παρακάτω εργαλεία

- HDFS: Hadoop Distributed File System
- YARN: Yet Another Resource Negotiator
- MapReduce: Programming based Data Processing
- Spark: In-Memory data processing
- PIG, HIVE: Query based processing of data services
- HBase: NoSQL Database
- Mahout, Spark MLLib: Machine Learning libraries
- Solar, Lucene: Searching and Indexing
- Zookeeper: Managing cluster
- Oozie: Job Scheduling

1.3.2 Το μοντέλο προγραμματισμού MapReduce

Το MapReduce είναι ένα μοντέλο προγραμματισμού χρησιμοποιείται για την παράλληλη επεξεργασία μεγάλου συνόλου δεδομένων. Το πρόγραμμα MapReduce λειτουργεί σε δύο φάσεις, συγκεκριμένα, Map και Reduce. Η Map χωρίζει τα δεδομένα τα οποία επεξεργάζονται παράλληλα, ενώ η Reduce απομειώνει τα αποτελέσματα της πρώτης φάσης.
Το Hadoop οικοσύστημα περιέχει το Hadoop MapReduce που χρησιμοποιείται για τη συγγραφή και εκτέλεση τέτοιων προγραμμάτων.

1.3.3 Παράλληλος προγραμματισμός με Spark

Το Spark είναι ένα σύνολο εργαλείων προγραμματισμού για τη σύνταξη και εκτέλεση παραλληλοποιήσιμου κώδικα για την επεξεργασία μεγάλο όγκο δεδομένων σε κατανεμημένο περιβάλλον. Έχει σχεδιαστεί για να εκτελεί τόσο μαζική επεξεργασία (παρόμοια με το MapReduce) όσο και επεξεργασία δεδομένων πουν καταφθάνουν σε ροή. Επίσης εκτελεί διαδραστικά ερωτήματα και χρησιμοποιείται για μηχανική εκμάθηση. Το spark χωρίζει τα δεδομένα προς επεξεργασία σε σε μικρότερες ομάδες που θα τις επεξεργαστούν ξεχωριστα και παράλληλα οι διαθέσιμοι κόμβοι. Τείνει να θεωρείται από τα πιο γρήγορα εργαλεία για αυτή τη δουλειά αφού χρησιμοποιεί RAM και προσωρινή αποθήκευση στη μνήμη για να βελτιώσει περαιτέρω την εκτέλεση των ερωτημάτων. Προσφέρει διεπαφή προγραμματισμού σε Java, Scala, Python. Το Spark μπορεί είτε να λειτουργεί μόνο του είτε σε διαχειριστή συμπλέγματος (cluster manager). Επιπλέον, το Spark έρχεται με αρκετές βιβλιοθήκες που υλοποιούν παράλληλη μηχανική εκμάθηση, επεξεργασία γραφημάτων, ερώτηση δεδομένων τύπου SQL και επεξεργασία ροής ημερομηνιών Υπάρχει μια εσφαλμένη αντίληψη ότι το Apache Spark είναι μια εναλλακτική λύση στο Hadoop. τα δύο μεγάλα πλαίσια δεν αλληλοαποκλείονται, αλλά μπορούν να συνδυαστούν και να δουλέψουν μαζί.

1.3.5 Spark σε συστάδα επεξεργασιτκών κόμβων

Το spark μπορεί να τρέξει τις αυτόνομες εργασίες που του ανατίθενται σε διαφορετικούς επεξεργαστικούς κόμβους μιας συστάδας. Το κεντρικό πρόγραμμα Spark Context συντονίζει αυτές τις εργασίες. Για την κατανομή των πόρων το Spark διαθέτει 4 κύριους διαχειριστές συμπλέγματος ανοιχτού κώδικα: Mesos, Hadoop YARN, Standalone και Kubernetes. Στην απλούστερη περίπτωση του Standalone το spark μπορεί να εκτελέσετε τοπικά σε πολλαπλά νήματα που θα υποστηρίξουμε με αντιστοιχου αριθμού πύρινες στο τοπικό μηχάνημα. Σε αυτήν την περίπτωση δεν χρειάζονται ειδικές ρυθμίσεις απλά δίνεται ο αριθμος των νημάτων στην εφαρμογή σαν

παράμετρος. Αυτή η περίπτωση χρειάζεται μόνο το Spark εγκατεστημένο στον κόμβο επίσης πρέπει να σηκωθούν ο master και οι workers με το χέρι η κάποιο αυτόματο script. Με τον ίδιο μη αυτόματο τρόπο με χρήση του Standalone mode μπορούμε να σηκώσουμε master και workers με το χέρι σε διαφορετικούς κόμβους αρκεί να γίνουν γνωστές οι διευθύνσεις τους μέσα από τα αρχεία ρυθμίσεων. Αυτος ο τρόπος χρησιμοποιήθηκε και στην παρούσα διπλωματική.

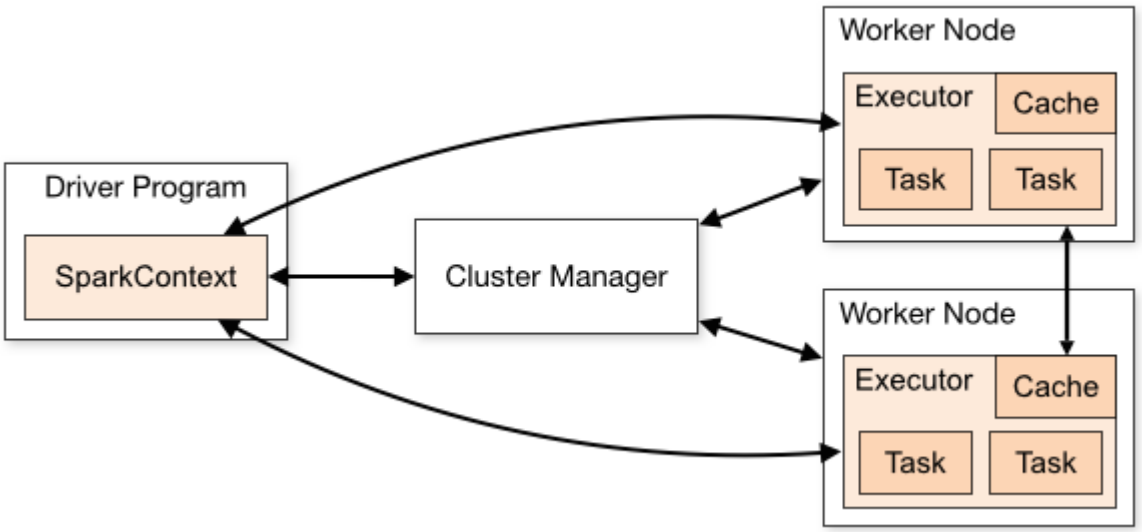


Image source

1.3.6 Χρήση κατανεμημένων δεδομένων στο Spark

Spark και μπορεί να επεξεργαστεί δεδομένα από HDFS, HBase, Cassandra, Hive και οποιοδήποτε Hadoop InputForm.

- Το HDFS είναι ένα από τα κύρια συστατικά του Apache Hadoop είναι ένα κατανεμημένο σύστημα αρχείων που χειρίζεται την αποθήκευση μεγάλων συνόλων δεδομένων σε πολλαπλά φυσικά μηχανήματα και σκληρούς δίσκους.
- Το Apache Cassandra είναι ένα κατανεμημένο σύστημα βάσης δεδομένων ανοιχτού κώδικα που έχει σχεδιαστεί για την αποθήκευση και τη διαχείριση μεγάλων ποσοτήτων δεδομένων σε κοινού τύπου διακομιστές.

Στη παρούσα πτυχιακή χρησιμοποιήθηκε η βάση δεδομένων Cassandra για την ανάγνωση δεδομένων στο Spark..

1.3.7 Μηχανική μάθηση με τη βιβλιοθήκη MLlib του Spark

Το MLlib είναι η βιβλιοθήκη μηχανικής μάθησης του Apache Spark. Αποτελείται από κοινούς αλγόριθμους μάθησης και βοηθητικά προγράμματα, συμπεριλαμβανομένης της ταξινόμησης, της παλινδρόμησης, της ομαδοποίησης, του φιλτραρίσματος, της μείωσης των διαστάσεων και άλλα.

2. ΚΑΤΗΓΟΡΙΟΠΟΙΗΣΗ ΚΕΙΜΕΝΟΥ

2.1 Εισαγωγή – Υπάρχουσες Τεχνικές

Η ταξινόμηση είναι η διαδικασία αναγνώρισης, κατανόησης και ομαδοποίησης ιδεών και αντικειμένων σε προκαθορισμένες κατηγορίες. Οι αλγόριθμοι ταξινόμησης χρησιμοποιούν προ-κατηγοριοποιημένα σύνολα δεδομένα εκπαίδευσης για να προβλέψουν την πιθανότητα τα επόμενα δεδομένα να ταιριάζουν σε μία από τις προκαθορισμένες κατηγορίες. Εν ολίγοις, η ταξινόμηση είναι μια μορφή "αναγνώρισης μοτίβου", με αλγόριθμους ταξινόμησης που εφαρμόζονται στα δεδομένα εκπαίδευσης για να βρουν το ίδιο μοτίβο (παρόμοιες λέξεις ή συναισθήματα, ακολουθίες αριθμών κ.λπ.) σε μελλοντικά σύνολα δεδομένων.

7 Τύποι αλγορίθμων ταξινόμησης

- Logistic Regression.
- Naïve Bayes.
- Stochastic Gradient Descent.
- K-Nearest Neighbours.
- Decision Tree.
- Random Forest.
- Support Vector Machine.

2.2 Εξαγωγή Χαρακτηριστικών (feature extraction)

Η εφαρμογή αλγορίθμων ταξινόμηση σε κείμενο είναι ένα από τους συνηθέστερους τρόπους χρήσης τους. Για την εφαρμογή τέτοιων αλγορίθμων χρειάζεται πρώτα το κείμενο να μετατραπεί σε ένα σύνολο από χαρακτηριστικά μέ αριθμητικού ή λογικού τύπου τιμές. Η εξαγωγή αυτών των χαρακτηριστικών είναι ίσως ένα από τα πιο σημαντικά σταδια. Η Προετοιμασία του κειμένου συνήθως περιλαμβάνει αρχικά

- την αφαίρεση τον σημείων στίξης και των συνδετικών λέξεων που δεν προσφέρουν αξία στην διαδικασία της ταξινόμησης

- Το κείμενο στη συνέχεια πρέπει να μετατραπεί σε ένα σύνολο ανεξάρτητων στοιχείων (bag of words) . Ο απλούστερος τροπος για αυτη τη μετατροπή είναι να χωρίσουμε το κειμενο σε λέξεις.

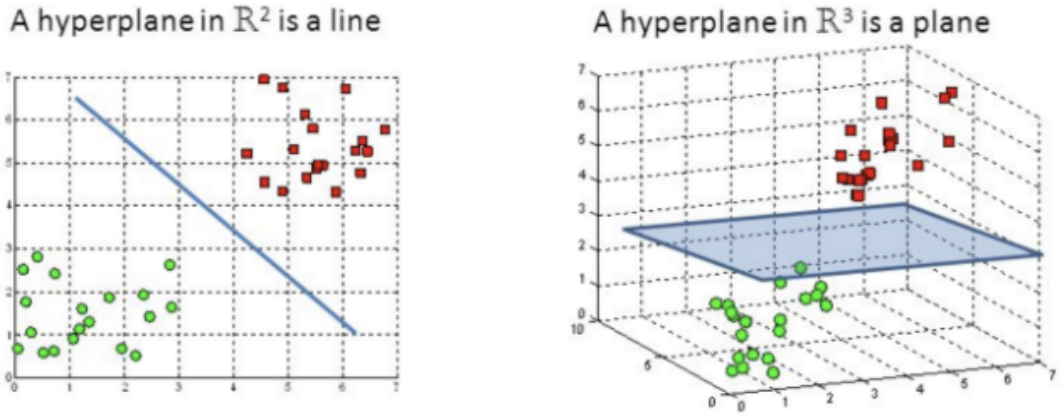
- Με την παραπάνω πρακτική ίσως χαθούν μοτίβα που δημιουργούνται απο την εμφανιση 2 η περισσότερων λέξεων σε συγκεκριμένη σειρά. Σε αυτή τη περιπτωση μπορει να χρησιμοποιηθει μια λιγο διαφορετική τακτικη . Το κείμενο μπορεί να χωριστή αντι για μια λέξη σε κάθε ζευγάρι λέξεων που εμφανίζονται μαζί. Ομοίως μπορούν να βρεθούν όλες οι Ν-αδες, λέξεων που εμφανίζονται μαζι (N-gram).

- Σε αυτό το σημείο εχει δημιουργηθει ενας πίνακας από στοιχεία που αντιπροσωπεύει το αρχικό κείμενο . Από αυτό το σύνολο λέξεων μπορεί πια να εξαχθούν χαρακτηριστικα χρήσιμα για τη ταξινόμηση .

- Το TF-IDF (συχνότητα όρου-αντίστροφη συχνότητα εγγράφων) είναι ένα στατιστικό μέτρο που αξιολογεί πόσο σχετική είναι μια λέξη με ένα έγγραφο σε μια συλλογή εγγράφων. Αυτό γίνεται πολλαπλασιάζοντας δύο μετρήσεις: πόσες φορές εμφανίζεται μια λέξη σε ένα έγγραφο και η αντίστροφη συχνότητα της λέξης σε ένα σύνολο εγγράφων.

2.3 Η μέθοδος SVM (Support Vector Machine)

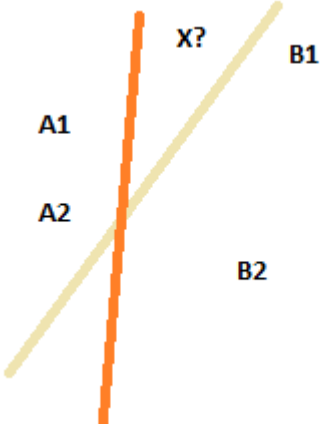
Support Vector Machine, με συντομογραφία SVM είναι ένα υπό επίβλεψη μοντέλο μηχανικής μάθησης για προβλήματα ταξινόμησης δύο ομάδων. Αφού δώσουμε στον SVM ένα σύνολο από ήδη κατηγοριοποιημένα δεδομένα παράγει ένα μοντέλο που με την χρήση του μπορεί να κατηγοριοποιήσει νέα άγνωστα στοιχεία. Είναι αποτελεσματικός σε χώρους υψηλών διαστάσεων, δηλαδή στη κατηγοριοποίηση δεδομένων που χαρακτηρίζονται από πολλά χαρακτηριστικα. Είναι αποτελεσματικός ακόμα και σε περιπτώσεις όπου ο αριθμός των χαρακτηριστικών είναι μεγαλύτερος από τον αριθμό των δειγμάτων.
Ο στόχος του αλγόριθμου είναι να βρει ένα υπερπλάνο σε χώρο Ν-διαστάσεων (Ν-ο αριθμός των χαρακτηριστικών) που ταξινομεί ευδιάκριτα τα σημεία δεδομένων.
Για να διαχωριστούν οι δύο κατηγορίες δεδομένων, υπάρχουν πολλά πιθανά υπερπλάνα που θα μπορούσαν να επιλεγούν. Στόχος μας είναι να βρούμε ένα επίπεδο που να έχει το μέγιστο περιθώριο (margin) , δηλαδή τη μέγιστη απόσταση μεταξύ των σημείων δεδομένων και των δύο κλάσεων.
Εάν ο αριθμός των διαστάσεων δηλαδή των χαρακτηριστικών είναι 2, τότε το υπερπλάνο είναι απλώς μια γραμμή. Εάν ο αριθμός των χαρακτηριστικών εισόδου είναι 3, τότε το υπερπλάνο γίνεται δισδιάστατο επίπεδο και ούτω καθεξής.



Hyperplanes in 2D and 3D feature space

Img source

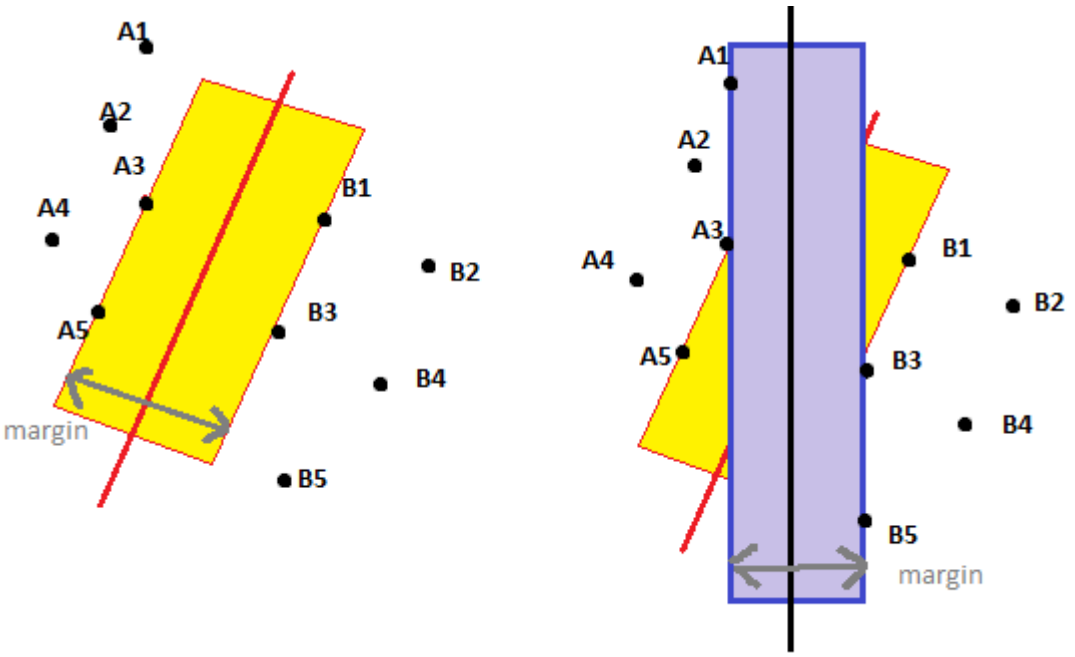
Δηλαδή καταλήγουμε να μιλάμε για ένα πρόβλημα βελτιστοποίησης τη μεγιστοποίηση της απόστασης περιθωρίου. Βελτιστοποίηση όμως ενώ τηρουνται κάποιοι περιορισμοί, τα σημεία της μιας ομάδας να διαχωρίζονται από τα σημεία της άλλης. Όσο πιο ευδιάκριτα χωρισμένα είναι τα σημεία τόσο που εύκολα και με μεγαλύτερη εμπιστοσύνη θα κατατάξουμε τα μελλοντικά σημείο που θα μας δοθούν για ταξινόμηση. Ένα νέο άγνωστο σημείο ταξινομείται σε μια κλάση ανάλογα από την πλευρά του υπερπλάνου που βρίσκεται.



Παράδειγμα

Τα αρχικά δεδομένα μας ανηκουν σε 2 κατηγορίες - κλάσεις Α και Β καί κάθε δεδομένο έχει 2 χαρακτηριστικά (features) οπότε μπορούμε να σχεδιάσουμε τα δεδομένα στο δισδιάστατο επίπεδο με τον άξονα χ να απεικονίζει τις τιμές του

χαρακτηριστικου 1 και τον άξονα Y να απεικονίζει τη τιμή του χαρακτηριστικού 2. Το ζητούμενο αποτέλεσμα του SVM είναι αυτή η γραμμή που διαχωρίζει καλήτερα τα δεδομένα Α από τα δεδομένα Β. Στο σχήμα βλέπουμε 2 πιθανές γραμμές διαχωρισμού. Το νέο αγνωστο δεδομένο Χ στη περίπτωση της κίτρινης γραμμής θα χαρακτηριστει με την κλάση Α ενώ στη περίπτωση διαχωρισμού με την πορτοκαλί γραμμή θα χαρακτηριστεί με την κλάση Β. Για την επιλογή της γραμμής διαχωρισμού πρέπει να βρεθει αρχικά το μεγαλύτερο κενό μεταξύ των δεδομένων των 2 κλάσεων. Το κενό αυτό ονομάζεται margin. Η γραμμή διαχωρισμού είναι η γραμμή στη μέση αυτού του κενού. Διαφορετικές γραμμές διαχωρισμού υπονοούν και διαφορετικό μέγεθος margin.



Το ζητούμενο του SVM είναι όχι απλά να βρει τη γραμμή διαχωρισμού αλλα να βρει τη γραμμή διαχωρισμού που βελτιστοποιεί αυτό το κενό.

Οι δύο πλευρές του margin οριοθετούνται από τα καποια στοιχεία της ομάδας Α απο την μία πλευρά και από κάποια στοιχεία της ομάδας Β από την άλλη πλευρά, αυτά τα στοιχεία ονομάζονται support vectors. Τα support vectors είναι και τα σημαντικότερα στοιχεια η επιλογή τους αλλάζει και την επιλεγμένη επιφάνεια διαχωρισμού.

Τα παραπάνω περιγράφονται με την μαθηματική σχέση

$$f(x) = w^T x + b.$$

img source

Η συνάρτηση της γραμμής διαχωρισμού f. Για όποιο σημείο χ η συνάρτηση είναι αρνητική αυτό ανήκει στην πρώτη κλάση (-1) ενώ θετική αυτό ανήκει στην 2η κλάση (1).

Η συνάρτηση αυτή μπορεί να γραφτει και με τον παρακατω τροπο αυτή θα χρησιμοποιηθεί και στην πορεία στους αλγορίθμους που αναπτύχθηκαν στην παρούσα πτυχιακή

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

Kernel

img source

- Έχουμε τον πίνακα με τα δειγματα **X** καθε θεση του **X** είναι ενας πίνακας με τιμές για κάθε χαρακτηριστικό
- Εχουμε έναν πίνακα **Y** μια θέση για κάθε δείγμα που περιέχει την κλάση κατηγοριοποισης. **-1** για την πρωτη κλάση ή **1** για την δευτερη.
- Ενα πίνακα με τους πολλαπλασιαστές **α** έναν για κάθε 1 θέση του **X**.
- b το bias

Για να βρούμε το fχ[1] τη πρώτη θέση που πίνακα F

- Επαναληπτικά Για κάθε δείγμα X
- πολλαπλασιάζουμε την κλάση (1,-1) του δείγματος
- με τον πολλαπλασιαστή στην αντίστοιχη θέση του δείγματος στον πίνακα α με τους πολλαπλασιαστές
- Και με τον KERNEL (των χαρακτηριστικών του X1 με τα χαρακτηριστικά του X της εκάστοτε επανάληψης)
- Επαναλαμβάνονται τα παραπάνω βηματα για όλα τα X

- Τα αποτελέσματα του επαναληπτικού πολλαπλασιασμού στο τέλος προστίθενται και έχουμε τη πρώτη θέση του πίνακα F

Αντίστοιχα στη συνέχεια θα βρεθεί η θέση F[2], F[3] κλπ

Για το βήμα εύρεσης του Kernel στην πιο απλή περίπτωση του γραμμικού διαχωρισμού των στοιχείων κάνουμε εσωτερικό γινόμενο μεταξυ των δυο Χ.

Δηλαδή KernelFunction(χ1,χ2)=> InnerProduct(χ1,χ2)

Αυτό μεταφράζεται σε εσωτερικό γινόμενο του πίνακα με τα χαρακτηριστικά του χ1 και του πίνακα με τα χαρακτηριστικά του χ2.

Έχοντας δει τα παραπάνω ειναι εμφανές πως μπορούμε να κατηγοριοποιήσουμε ένα νέο δείγμα άγνωστης κλάσης αν έχουμε τον πίνακα με τα χαρακτηριστικά και είναι γνωστά σε εμάς οι πολλαπλασιαστές **α** από την σχέση F(x).

ΑΝ f(x) < 0 τότε y = −1 (κατηγοριοποιήται στην κλάση 1)

ΑΝ f(x) ≥ 0 τότε y = 1 (κατηγοριοποιήται στην κλάση 2)

Τα δείγματα Χ που το αντίστιχο α τους ειναι 0 δεν προσθέτουν κάτι στην F(x) οπότε δεν είναι και χρήσιμα στον ορισμό της συνάρτησης F. Η συνάρτηση F οριοθετείται από τα δείγματα με **α** διάφορο του μηδενός. Αυτα τα δείγματα ειναι και support vectors. Οι αλγόριθμοι που υλοποιήθηκαν στην παρούσα πτυχιακή ασχολούνται με την εύρεση τον πολλαπλασιαστών α (σειριακά και παράλληλα) για τα δεδομένα εκπαίδευσης.

3. ΣΧΕΔΙΑΣΜΟΣ ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗΣ SVM ΑΛΓΟΡΙΘΜΩΝ

Στη παρούσα πτυχιακή δόθηκε ιδιαίτερη σημασία στην εκπαίδευση του μοντέλου με διάφορες εκδόσεις του αλγορίθμου SVM. Ο SVM είναι ένας αλγόριθμος βελτιστοποίησης και η επίλυση του απαιτεί χρήση τετραγωνικού προγραμματισμού. Σε πολύ μεγάλα set δεδομένων και σε δεδομένα με πολλά χαρακτηριστικά η χρήση μια τέτοιας λύσης γίνεται απαγορευτική και σε χρόνο και σε μνήμη. Γι αυτό μελετήθηκε

μια άλλη προσέγγιση για τη λύση SVM προβλημάτων. Ο αλγόριθμος διαδοχικής ελάχιστης βελτιστοποίησης .

3.1 Σειριακός αλγόριθμος Διαδοχικής Ελάχιστης Βελτιστοποίησης

Ο αλγόριθμος διαδοχικής ελάχιστης βελτιστοποίησης (SMO - Sequential Minimal Optimization Algorithm) προτάθηκε από τον John C. Platt το 1998 έχει αποδειχθεί ότι είναι μια αποτελεσματική μέθοδος για την εκπαίδευση Support Vector Machines μοντέλων (SVM). Το SMO διαφέρει από τους περισσότερους αλγόριθμους SVM στο ότι δεν απαιτεί επίλυση με χρήση τετραγωνικού προγραμματισμού. Είναι από τη φύση του σειριακός αλγόριθμος

Όπως αναλύθηκε σε προηγούμενο κεφάλαιο ο στόχος είναι να βρεθεί η παρακάτω σχέση από τα δείγματα εκπαίδευσης Χ, ώστε να μπορεί μετα να λυθεί για οποιοδήποτε άγνωστο χ.

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

Kernel

img source

Η λύση αποτελεί την πολυδιάστατη επιφάνεια που διαχωρίζει τις δύο κλάσεις με τα γνωστα δείγματα. Για ένα άγνωστης κλάσης δείγμα επιστέφει -1 ή 1 ανάλογα με τη κλάση που θα κατηγοριοποιηθεί.

Για να δημιουργήσουμε το μοντέλο που θα μας κατηγοριοποιεί τα άγνωστα δείγματα ψαχνουμε λοιπόν τις άγνωστες τιμές του πίνακα α (πολλαπλασιαστές Lagrange) και τον άγνωστο β (bias). Ξεκινάει ο αλγόριθμος δημιουργώντας έναν πίνακα **α** με θέσεις όσες και ο αριθμός των δειγμάτων εκπαίδευσης **χ** και αρχικοποιούνται όλες οι θέσεις στο 0. Στη συνέχεια σειριακά επιλέγει ένα ζευγάρι από 2 πολλαπλασιαστές **α**, δηλαδή δύο θέσεις αυτού του πίνακα για βελτιστοποίηση. Προσπαθούμε να αλλάξουμε τις τιμές των **α** σε τιμές που βελτιώνουν την **F(x)** δλδ δημιουργούν μια **F(x)** που χωρίζει τα δείγματα εκπαίδευσης με μικρότερο ποσοστό λαθους. Η διαδικασία αυτή συνεχίζεται επαναληπτικά για όλα τα πιθανά ζευγάρια

μέχρι κανένα **α** να μην αλλάξει προς το καλύτερο .Έτσι βελτιστοποιείται το μικρότερο δυνατό υπο-πρόβλημα σε κάθε βήμα ενώ ικανοποιούνται οι περιορισμοί για το επιλεγμένο ζεύγος.

```
var totalCoeffsChanged = false
while (!totalCoeffsChanged ) {

   totalCoeffsChanged = false
   for(j<-xPoints.indices){
     val localCoeffsChanged = examineJ(j)
     totalCoeffsChanged = (totalCoeffsChanged || localCoeffsChanged)
   }
   totalIter += 1

}
```

Εσωτερικά σε κάθε επανάληψη επιλέγεται επιλέγεται με τη σειρά κάθε **α[j]** και ελέγχεται αν η εκάστοτε τιμή του χρειάζεται βελτίωση. Αν όχι η επανάληψη επιστρέφει ότι δεν χρειάστηκε κάποια αλλαγή. Αν όμως επιδέχεται βελτίωση αυτο το **α[j]** γίνεται ζευγάρι με όλα τα υπόλοιπα **α[i]**. Για κάθε ζεύγος ελέγχεται αν η εκάστοτε τιμή του **α[i]** χρειάζεται βελτίωση . Αν τα δύο επίπεδα επαναλήψεων επιστρέψουν χωρίς καμια βελτίωση ο αλγόριθμος τερματίζει και έχουμε τον τελικό πίνακα **α**. Στο τέλος της διαδικασίας κάθε δεδομένο από το αρχικό δείγμα δεδομένων (data set) που δεν έχει μηδενιστεί ο αντίστοιχος πολλαπλασιαστής του στον πίνακα **α** θεωρείται και support vector και οριοθετεί το margin δηλαδή το επίπεδο που διαχωρίζει τις 2 ομάδες κλάσεων.

Τα δείγματα **X** που το αντιστιχο **α** τους ειναι 0 δεν προσθέτουν κάτι στην **F(x)** αφου μηδενίζουν τον όρο που θα προστεθεί, οπότε δεν είναι και χρήσιμα στον ορισμό της συνάρτησης **F**. Η συνάρτησης **F** οριοθετείται από τα δείγματα με πολλαπλασιαστή **α** διάφορο του μηδενός. Αυτα τα δείγματα ειναι και support vectors.

Για την απόφαση του αν ένας πολλαπλασιαστής **α** ειναι βέλτιστος χρησιμοποιούνται η παρακάτω σχέσεις όπως ορίστηκε από τους Karush–Kuhn–Tucker

```
var errorj = f(j) - label(j)
val kktCondition1= label(j) * errorj < -tolerance && coefficients(j) < regularisationParamC
```

```
val kktCondition2= label(j) * errorj > +tolerance && coefficients(j) > 0
```

Υπολογίζεται το πόσο λάθος κατηγοριοποιούνται τα γνωστά σημεία με την ως τώρα συνάρτηση **F(X),** αυτή η τιμή είναι και το **error-j**. Βάση αυτού του **error-j** υπολογίζονται οι λογικές μεταβλητές **kktCondition1** και **kktCondition2** αν είναι και οι δύο ψευδής δεν χρειάζεται βελτιστοποίηση ο επιλεγμένος πολλαπλασιαστής **α**. Σε αντίθετη περίπτωση επιχειρείται η βελτιστοποίηση του ζευγους.

Το νέο **α[j]** υπολογίζεται από την σχέση

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta}$$

Με παρονομαστή $\eta = 2\langle x^{(i)}, x^{(j)}\rangle - \langle x^{(i)}, x^{(i)}\rangle - \langle x^{(j)}, x^{(j)}\rangle.$

Τα **Ej** ειναι το error που μολις υπολογίστηκε πόσο λάθος δηλαδή κατηγοριοποιήθηκε το **X[j]** Με την χρήση του αρχικού **α[j]**.

Σε κώδικα αυτό μεταφράζεται

```
computeDenominatorA = 2 * kernelCell(i, j) - kernelCell(i, i) - kernelCell(j, j)

coefJ_new -= ((label(j) * (errori - errorj)) / denominatorA) //12 optimal αj
coefI_new -= label(i) * label(j) * coefDiffJ
```

Η υπάρχουσα απλουστευμένη υλοποίηση δεν ασχολείται με την περίπτωση που ο παρονομαστης ειναι 0. Στον κώδικα που προτάθηκε από το Platt το 1998 έχει γίνει ειδικός χειρισμός ωστε να μην εξαιρούνται οι συγκεκριμένες περιπτώσεις.

```
val denominatorA = computeDenominatorA(i, j)
if (denominatorA >= 0) {
 return false
}
```

Αφού βρεθεί η νέα τιμή του **α[j]** και του **α[i]** αυτή κόβεται για να περιοριστεί στο πεδίο ορισμού **[L,H]**. Αν είναι μεγαλύτερο του μέγιστου **H** (high) γίνεται ‘ίσο με το **H** αν είναι μικρότερο του ελάχιστου **L** (low) γίνεται ίσο με το **L** .

Για την επιλογή του H και L χρησιμοποιήθηκαν οι παρακάτω συναρτήσεις.

```
def calcLowBound(i: Int, j: Int): Double = {
 if (label(i) == label(j)) scala.math.max(0, coefficients(i) + coefficients (j) -
regularisationParamC)
 else scala.math.max(0, coefficients(j) - coefficients(i))
}

def calcHighBound(i: Int, j: Int): Double = {
 if (label(i) == label(j)) scala.math.min(regularisationParamC, coefficients(i) + coefficients(j))
 else scala.math.min(regularisationParamC, regularisationParamC + coefficients(j) -
coefficients(i))
}
```

Αν τελικά το νέο **α[i]** είναι αρκετά διαφορετικό από το παλιό θεωρείται πως υπάρχει αλλαγή. Αρκετά διαφορετικό εννοούμε να διαφέρει περισσότερο από κάποιο όριο ακρίβειας. Το ίδιο όριο ακρίβειας χρησιμοποιείται για όλες τις μεταβλητές για να θεωρηθεί ότι διαφέρουν από το μηδέν.

```
protected def tolZero(d:Double): Double =  if (abs(d) < tolerance)  0.0 else d
```

Σε κάθε επανάληψη που καταλήγει σε αλλαγή του πίνακα τους πολλαπλασιαστές **α** προσαρμόζεται ανάλογα και η μεταβλητή του **bias** (b)

```
val b1: Double = bias - errori - label(i) * coefDiffI * kernelCell(i, i) - label(j) * coefDiffJ *
kernelCell(i, j)
val b2: Double = bias - errorj - label(i) * coefDiffI * kernelCell(i, j) - label(j) * coefDiffJ *
kernelCell(j, j)


var newBias: Double = bias //19
if (coefI_new > 0 && coefI_new < regularisationParamC) newBias = b1
else if (coefJ_new > 0 && coefJ_new < regularisationParamC) newBias = b2
else newBias = (b1 + b2) / 2
this.bias = newBias;
```

Εδω να σημειωθεί οτι σε κάθε εξωτερική επανάληψη όσο το **error-j** μένει ίδιο επαναχρησιμοποιείται σε όλες τις εσωτερικές επαναλήψεις για να μην χρειαστεί επανυπολογισμός ο οποίος είναι πολύ κοστοβόρος αφού χρειάζεται ο υπολογισμός της **F(x)**.

Caching kernels

Μια ακόμα βελτιστοποίηση που έγινε στους υπολογισμού είναι η επαναχρησιμοποίηση των ήδη υπολογισμένων Kernels. Kernel στην βασική περίπτωση είναι ο υπολογισμός του εσωτερικού γινομένου μεταξύ κάποιου **χ[i]**και **χ[j],** δηλαδή μεταξύ του πίνακα που περιέχει τις τιμές για τα χαρακτηριστικά του στοιχείου **xi** και τον πίνακα που περιέχει τις τιμές για τα χαρακτηριστικά του **xj**.

```
protected def kernelCell(xi: Int, kernelCellIndex: Int): Double = {
 if(localKernels(xi)(kernelCellIndex).isDefined){
   return localKernels(xi)(kernelCellIndex).get
 }
 val array1 = xPoints(xi).features.toArray
 val array2 = xPoints(kernelCellIndex).features.toArray
 val k = KernelCalculator.kernel(array1, array2)

 localKernels(xi)(kernelCellIndex) = Some(k)
 k
}
```

Επίσης εκμεταλλευόμαστε την αντιμεταθετική ιδιότητα των Kernel όπου **K[i,j]** ίσο με **K[j,i]** και μειώνουμε τον χρόνο υπολογισμού της **F(x)**.

Με το τέλος του αλγορίθμου όταν πια είναι γνωστή η **F(x)** για την κατηγοριοποίηση ενός άγνωστης κλάσης στοιχείου **x_unknown** αρκεί να βρεθούν τα χαρακτηριστικά του και να υπολογιστεί η **F(x_Unknown)**. Αν αυτή είναι θετική επιλέγεται η κλάση 1 αν είναι αρνητική επιλέγεται η κλάση 2.

```
protected def margin(features: Array[Double]) :Double=
 svCoefficients.indices.map(i=> svCoefficients(i)* svXPoints(i).label  *
KernelCalculator.kernel(svXPoints(i).features.toArray, features)).sum

def predict( features: Array[Double]): ClassPairPrediction =  {
    val d = margin(features) + getBias

    val klass = if (d >= 0) classPair.classLabel1 else classPair.classLabel2
    new ClassPairPrediction(classPair, klass, d)

}
```

3.2 Παραλληλοποίηση του αλγόριθμου SMO

Ο αλγόριθμος διαδοχικής ελάχιστης βελτιστοποίησης (SMO) είναι και αυτός αρκετά χρονοβόρος όταν χειρίζεται πολύ μεγάλες ομάδες δεδομένων εκπαίδευσης. Μια λύση που μελετήθηκε για την βελτίωση του ήταν η παραλληλοποιηση του . Ο SMO είναι ένας σειριακός αλγόριθμος και κάθε του επανάληψη εξαρτάται από τα αποτελέσματα τις προηγουμενης. Όλες οι προσεγγίσεις παραλληλοποίησης περιλαμβάνουν το διαχωρισμό των δεδομένων σε μικρότερες ομάδες επεξεργασίας και εκτέλεσης του SVM-SMO για τη δημιουργία μοντέλων για αυτά τα επιμέρους μικρότερα προβλήματα. Τέλος συνδυάζουν τα των αποτελέσματα σε ένα συνολικό μοντέλο. Το ζητούμενο μετά την παραλληλοποίηση εκτός της μείωσης του χρόνου εκπαίδευσης ήταν να διατηρείται το απαιτούμενο επίπεδο ακρίβειας ταξινόμησης,

3.2.1 Προσέγγιση Παραλληλοποίησης 1 - Simple

3.2.1.1 Περιγραφή πρώτης προσέγγισης

Τα δεδομένα χωρίζονται σε μικρότερες N ομάδες. Σε κάθε ομάδα τρέχει αλγόριθμος SVM-SMO. Οπως αναφέραμε το πρόβλημα είναι η εύρεση αυτού του υπερ-επιπέδου που μεγιστοποιεί την απόσταση των επιπέδων A1 και A2 , είναι πρόβλημα βελτιστοποίησης και για την λύση του πρέπει να βρεθούν οι πολλαπλασιαστες Lagrange .

Η λύση του SMO μας δίνει αυτό το σετ από πολλαπλασιαστες Lagrange για τους οποίους ικανοποιούνται οι συνθήκες KKT που βελτιστοποιείται ο διαχωρισμός των στοιχείων. Οπότε πραγματοποιήθηκε αρχικά ο διαχωρισμός του μεγάλου συνόλου δεδομένων σε μικρότερα . Σε κάθε ένα από αυτά με χρήση του αλγόριθμου SVM-SMO που αναλήθηκε στο προηγούμενο κεφάλαιο βρέθηκαν αυτοί οι πολλαπλασιαστές που βελτιστοποιουν το μικρότερο πρόβλημα . Τέλος μελετήθηκε ο συνδυασμός αυτών τον πολλαπλασιαστών κατα πόσο βελτιστοποιούν το αρχικό συνολικό προβλημα χωρις να ξανά εκπαιδευτεί δεύτερη φορά το μοντέλο. source

Καθώς η ακρίβεια της λύσης ενός SVM προβλήματος διαχωρισμού εξαρτάται από τη συνολική εικόνα των δεδομένων η εκπαιδευση του σε ξεχωριστές ομάδες φυσικά και υποφέρει αρκετα απο μείωση της ακρίβειας . Αργότερα θα μελετήσουμε διεξοδικά σε πειραματικά κατα πόσο επηρεάζει αυτός ο διαχωρισμός

3.2.1.2 Υλοποίηση σε κώδικα - Κλάση SVMSMOParallelExecutor

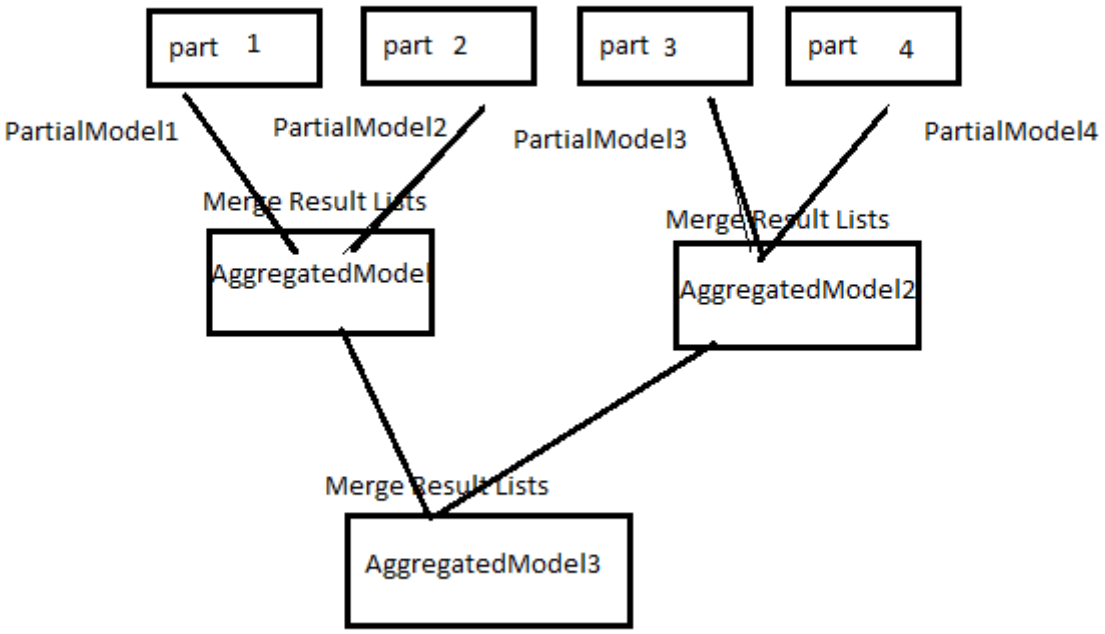
Αρχικά χωρίζεται το σύνολο δεδομένων (dataset) σε N μέρη (partions) κάθε μέρος χαρακτιριζεται απο ένα δείκτη **partIndex**. Για τα δεδομένα κάθε μέρους (partition) τρέχουμε τον επιλεγμένο SVM αλγόριθμος με τη χρηση καποιας υλοποίησης της κλάσης AbstractSVMrunner. Αυτό θα επιστρέψει ένα PartialModel δηλαδή ένα μοντέλο που θα περιέχει τα support vectors και το bias για τα δεδομένα του partition. Δηλαδή έχουμε ην F(X) που ορίζει το υπερεπίπεδο που διαχωρίζει τις 2 κλάσεις για αυτα τα δεδομένα. Το Spark ενώνει τα αποτελέσματα των παράλληλων υπολογισμών σε μορφή δέντρου καλώντας τις μεθόδους add(..) και merge(..) των μοντέλων

```
val trainingDatasetPX: RDD[PointX] = transformDatasetRow(trainingDataset)
val rep =  trainingDatasetPX.repartition(partitions)

val modelAggregator: PartialModel= rep.mapPartitionsWithIndex((partIndex, iter) => {

  val linearSVC: AbstractSVMRunner = newRunner.apply(partIndex)
  val array: Array[PointX] = iter.toArray
  val model = linearSVC.train(pair, array)
  model.setPartitionIndex(partIndex)
  model.setPair(pair)
  Array(model).iterator

}).treeAggregate(new PartialModel())(
  seqOp = (c, v) => c.add(v),
  combOp = (c1, c2) => c1.merge(c2)
)
val aggregatedModel: AggregatedModel = modelAggregator.reconstructPartition()
```



;

```
def merge(aggregatedResult: PartialModel): PartialModel = {
 classPair = aggregatedResult.classPair
 this.svCoefficientsPart ++= aggregatedResult.svCoefficientsPart
 this.svXPointsPart ++= aggregatedResult.svXPointsPart
 this.biasPart ++= aggregatedResult.biasPart
 this.biasSum += aggregatedResult.biasSum
 this
}
```

Κάθε “partial” μοντέλο δηλαδή το μοντέλο του υποσυνόλου δεδομένων, περιέχει μια λίστα με τα support vectors και το index του partition που προήλθαν. Αρχικά καλείται η μέθοδος add(..) και το Partial Model προστίθεται σε ένα άδειο Aggregate Model βάζοντας δίνοντας τη λίστα του σε στο aggregate model. Το aggregated model κρατάει ένα εσωτερικό μαπ με τις λίστες από support vectors που του δόθηκαν με κευ το παρτ index από το οποίο προήλθαν. Στη συνέχεια τα aggregated model γίνονται merge μεταξύ τους κάνοντας merge τα map που περιέχουν. Τελικά επιστρέφεται ένα

aggregated model που περιέχει τις λίστες από όλα τα partitions. Στο τελικό μοντέλο καλούμε την reconstractPartition() η οποία ενώνει τις λίστες σε μια ένιαια.

```
def reconstructPartition(): AggregatedModel = {
 svCoefficients = this.svCoefficientsPart.toSeq.sortBy(_._1).flatMap(_._2).toArray
 svXPoints = this.svXPointsPart.toSeq.sortBy(_._1).flatMap(_._2).toArray
 this.bias = this.biasSum / svXPoints.length
 new AggregatedModel(svCoefficients, svXPoints, this.bias)
}
```

Ουσιαστικά το τελικό μοντέλο περιέχει τα support vectors κάθε ξεχωριστού dataset. Κάθε suppurt vector ;exei epilege;i ap; υποσύνολο των δεδομένων και όχι από την συνολική εικόνα τους πράγμα που θα βλάψει την ακρίβεια κατηγοριοποίησης αργότερα. Όσο περισσότερα partition χρησιμοποιούνται τόσο λιγότερο αντιπροσωπευτικά είναι τα τελικά support vectors

3.2.3 Προσέγγιση Παραλληλοποίησης 2 - Cascade

3.2.3.1 Περιγραφή δεύτερης προσέγγησης - Cascade

Οι αλγόριθμοι SVM βασίζονται στα διανύσματα υποστήριξης (support vectors) για να επιτύχουν την ταξινόμηση. Τα Διανύσματα Υποστήριξης είναι εκείνα που βρίσκονται πιό κοντά στο υπερπλάνο που χωρίζει τα υπόλοιπα διανύσματα με το μέγιστο δυνατόν περιθώριο. Σε αυτη τη προσεγγηση τα δεδομένα χωρίζονται σε μικρότερες N ομάδες. Σε κάθε ομάδα με χρήση SVM βρίσκονται τα support vectors. Στη συνέχεια δημιουργούνται N/2 καινούργιες ομάδες από τα εναπομείναντα σημεία κάθε αρχικής ομάδας αφου εξαλείφθηκαν τα μη support vectors. Αυτη η διαδικασία συνεχίζεται επαναληπτικά logN φορές. Σε κάθε επανάληψη οι α αριθμός των ομάδων έχει υποδιπλασιαστεί. Στην τελευταία επανάληψη θα έχει απομείνει μόνο μια ομάδα

της οποίας τα διανύσματα υποστήριξης (support vector) θα αποτελέσουν και την τελική λύση του αρχικού προβλήματος.

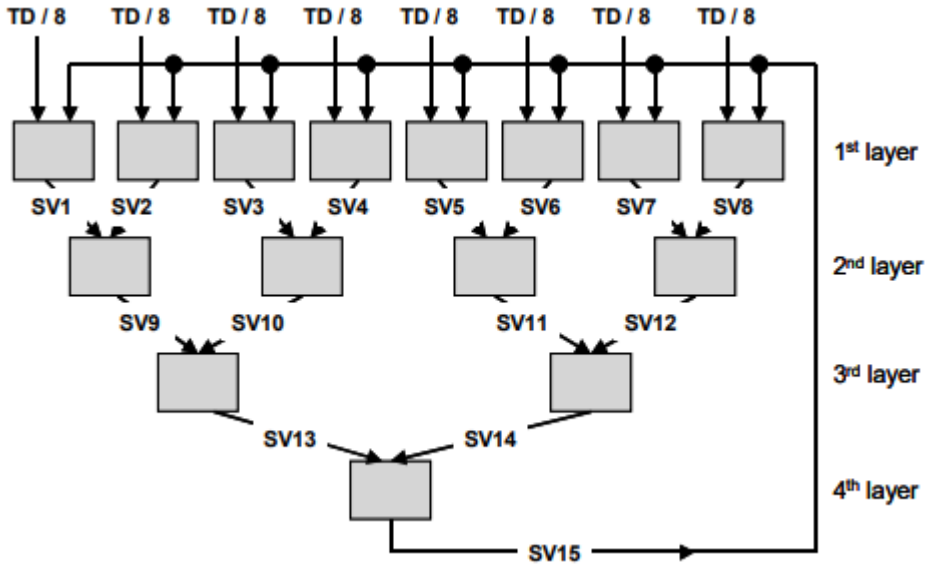
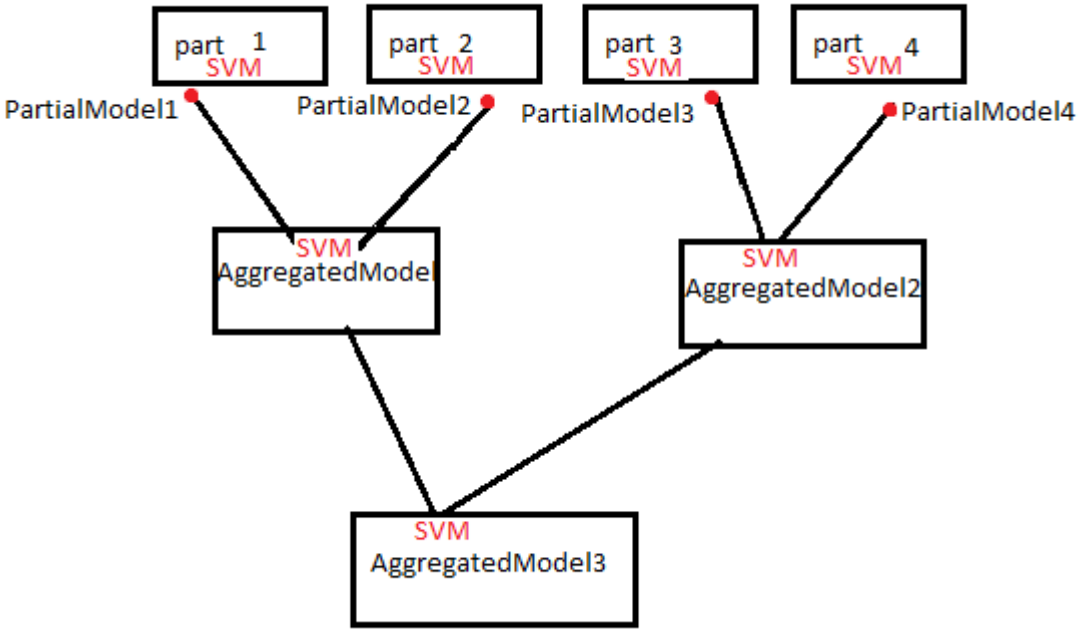


Image source

3.2.3.1 Υλοποίηση σε κώδικα - Κλάση SVMSMOParallelExecutorCascade

Αρχικα χωρίζεται το dataset σε N μέρη (partions) καθε μέρος χαρακτιριζεται απο ένα δείκτη partIndex. Για τα δεδομένα κάθε partition τρέχουμε τον επιλεγμένο SVM αλγόριθμος με τη χρηση καποιας υλοποίησης της κλάσης AbstractSVMrunner. Αυτό θα επιστρέψει ένα PartialModel δλδ ένα μοντέλο που θα περιέχει τα support vectors και το bias για τα στοιχεία του partition. Δηλαδή έχουμε την συνάρτηση F(X) που ορίζει το υπερεπίπεδο που διαχωρίζει τις δύο κλάσεις για αυτά τα δεδομένα. Το Spark ενώνει τα αποτελέσματα των παράλληλων υπολογισμών σε μορφή δέντρου καλώντας τις μεθόδους add(..) και merge(..) των μοντέλων. Σε αντίθεση με την προηγούμενη παράλληλη υλοποίηση κατα την ένωση 2 μοντέλων αφού ενωθούν τα επιμέρους support vectors ξανα τρέχουμε SVM μεταξύ αυτων αυτών των δειγμάτων (support vectors).




```
val trainingDatasetPX: RDD[PointX] = transformDatasetRow(trainingDataset)
val rep =  trainingDatasetPX.repartition(partitions)

val modelAggregator: PartialModelCascade= rep.mapPartitionsWithIndex((partIndex, iter) => {

    val linearSVC: AbstractSVMRunner = newRunner.apply(partIndex)
    val array: Array[PointX] = iter.toArray
    val model: AbstractSVMModel = linearSVC.train(pair, array)
    model.setPartitionIndex(partIndex)
    model.setPair(pair)

    Array(model).iterator

}).treeAggregate(new PartialModelCascade())(

     seqOp = (c, v) => c.add(v, newRunner.apply(c.partIndex)),
     combOp = (c1, c2) => c1.merge(c2, newRunner.apply(c1.partIndex))
)

val aggregatedModel: AggregatedModel = modelAggregator.reconstructPartition()
```

Αφού έχουν τρέξει παράλληλα SVM για κάθε partition επιστρέφεται ένα Partial Model το οποιο περιέχει τα support vector του partition. Αυτα τα μοντέλα περιέχουν τις μεθόδους add(..) και merge(..).

```
def merge(aggregatedResult: PartialModelCascade, linearSVC: AbstractSVMRunner):PartialModelCascade={
 classPair = aggregatedResult.classPair
 partIndex2 = aggregatedResult.partIndex
 if(partIndex==partIndex2)
```

```
    return this

  svCoefficientsPart ++= aggregatedResult.svCoefficientsPart
  svXPointsPart ++= aggregatedResult.svXPointsPart
  biasPart ++= aggregatedResult.biasPart

  if(svXPointsPart.size<=1)/*then cascade 2 partitions*/{
    partIndex = partIndex2
  }
  else/*then cascade 2 partitions*/{
    val svXPointsNew = this.svXPointsPart.toSeq.sortBy(_._1).flatMap(_._2).toArray

    val newCascadedModel: AbstractSVMModel = linearSVC.train(classPair, svXPointsNew)
    // returns new SVMModel(svCoeffs, svXPoints, bias)
    svCoefficientsPart = Map()
    svXPointsPart = Map()
    svCoefficientsPart =
           svCoefficientsPart + (aggregatedResult.partIndex -> newCascadedModel.svCoefficients)
    svXPointsPart = svXPointsPart + (aggregatedResult.partIndex -> newCascadedModel.svXPoints)
  }

  this
}
```

Αυτες οι μέθοδοι θα κληθούν για την ένωση του μοντέλου με κάποιο άλλο. Κατα την ένωση των δύο μοντέλων ενώνονται τα support vectors σε έναν πίνακα αυτός ο πίνακα είναι ένα νέο σύνολο δειγμάτων πάνω στο οποίο θα γίνει τρέξει ο αλγόριθμος SVM . Το αποτέλεσμα αυτής της ένωσης θα είναι τα νέα support vector που θα έχουν λάβει υπόψη την εικόνα των δειγμάτων από τα δύο partitions. Στη συνέχεια αυτό το μοντέλο θα ενωθεί με το αποτέλεσμα ενός άλλου μοντέλου στο ίδιο ύψος του δέντρου. Εκεί κατά τη ένωση θα ξαναγίνει SVM και θα καταλήξουμε με ένα μοντέλο που θα έχει support vector που λαμβάνουν υπόψη την συνολική εικόνα των σημαντικότερων σημείων από κάθε partition. Οταν ποια φτάσουμε στη ρίζα του δέντρου και ενωθούν τα δύο τελικά μοντέλα επιστρέφεται ενα τελικό μοντέλο που τα support vectors είναι τα πιο χαρακτηριστικά σημεία που έχουν ληφθεί από όλα τα partition συνδυαστικά. Είναι εμφανές πως αυτή η τακτική λαμβάνει υπόψη συνδυαστικά όλα τα partitions sthn τελικη λύση αλλα υποφέρει αρκετα απο αποψη ταχύτητας. Η βελτίωση της ταχύτητας σε σχέση με τον αρχικό αλγοριθμο στηρίζεται στο οτι σε κάθε βήμα το δέντρο έχει όλο και λιγότερα δείγματα για να κάνει SVM. Αν όμως το αποτέλεσμα κάθε αλγορίθμου δεν μειώνει αρκετα τα δείγματα και τα support vectors είναι μεγάλο ποσοστό των αρχικών δειγμάτων τότε αντί να βελτιωθεί η ταχύτητα καταλήγουμε να επαναλαμβάνουμε SVMs

με παρόμοιο αριθμό δειγμάτων. Παρακάτω στις δοκιμές θα φανεί όσο μεγαλώνει ο αριθμός των features μεγαλώνει και ο αριθμός των διαστάσεων στις οποίες πρέπει να διαχωριστούν τα δείγματα , στην προκειμένη περίπτωση τα δείγματα ήταν λιγότερα από τις διαστάσεις που έπρεπε να υποστηριχθούν γιαυτο και χρειαζόταν μεγάλος αριθμός από αυτά για να υποστηριχθει ο διαχωρισμός. Ως αποτέλεσμα είχαν αυτή η προσέγγιση να μην συμπεριφέρεται καλά σε θέμα χρόνου.

3.3 Μη γραμμικά διαχωρίσιμα δεδομένα

Όπου δεν είναι δυνατός ο γραμμικός διαχωρισμός, τα δεδομένα χαρτογραφούνται σε χώρο μεγαλύτερης διάστασης, με χρήση των Kernel functions. Με άλλα λόγια, τα Kernel functions μετατρέπουν το μη διαχωρίσιμο πρόβλημα σε διαχωρίσιμα προβλήματα προσθέτοντας και άλλη διάσταση σε αυτό, βρίσκοντας μια διάσταση στην οποια τα δεδομένα μπορούν να διαχωριστούν με κάποιο υπερπλάνο.
Για παράδειγμα
Έχουμε τα παρακάτω μη διαχωρίσιμα δεδομένα στις 2 διαστάσεις **x,y**
Δεν μπορούν να διαχωριστούν από μια γραμμή (2d hyperplane)
Αν φανταστούμε τα ίδια σημεία στον τρισδιάστατο χώρο αφού τους προσθέσουμε μια τρίτη διάσταση **z** η κατάσταση μπορεί να μετατρέπει σε ευκολα διαχωρίσιμα δεδομένα από ένα υπερπλάνο τριών διαστάσεων.

Συναρτήσεις πυρήνες - Kernel functions

- **Linear Kernel** : Η συνάρτηση γραμμικού πυρήνα είναι το εσωτερικό γινόμενο μεταξύ δύο δεδομένων δηλαδή το εσωτερικό γινόμενο μεταξύ δύο διανυσμάτων που ειναι το αθροισμα το άθροισμα του πολλαπλασιασμού κάθε ζεύγους τιμών εισόδου.

```
K(x, xi) = sum(x * xi)
```

```
object LinearKernelCalculator extends KernelCalculatorTrait  {

   override def kernel(features1: Array[Double], features2: Array[Double]): Double
                = dotProduct(features1, features2)

   private def dotProduct(x: Array[Double], k: Array[Double]): Double
                = x.indices.map(i => x(i) * k(i)).sum


}
```

- **Polynomial Kernel** Ένας πολυωνυμικός πυρήνας είναι μια πιο γενικευμένη μορφή του γραμμικού πυρήνα. Ο πολυωνυμικός πυρήνας μπορεί να διακρίνει τον καμπύλο ή τον μη γραμμικό χώρο εισόδου.

  `K(x,xi) = 1 + sum(x * xi)^d`

```
object PolynomialKernelCalculator extends KernelCalculatorTrait{

   override def kernel(features1: Array[Double], features2: Array[Double]): Double
                = polynomialKernel(features1, features2)


   private def dotProduct(x: Array[Double], k: Array[Double]): Double
                = x.indices.map(i => x(i) * k(i)).sum

   private def polynomialKernel(ki: Array[Double], xi: Array[Double]): Double
                = scala.math.pow(dotProduct(ki, xi), 2.0)

}
```

- **Gaussian Kernel**

```
object GaussianKernelCalculator extends KernelCalculatorTrait  {
   private val gm = 1.0

   override  def kernel(features1: Array[Double], features2: Array[Double]): Double
                        =gaussianKernel(features1, features2)

   private def dotProduct(x: Array[Double], k: Array[Double]): Double
                        = x.indices.map(i => x(i) * k(i)).sum


   private def gaussianKernel(x1: Array[Double], x2: Array[Double]): Double
                       =  scala.math.exp(-gm * norm(minusVector(x1, x2)))

   private def norm(x: Array[Double]): Double
```

```
                        = scala.math.sqrt(dotProduct(x, x))

    private def minusVector(x2: Array[Double], x1: Array[Double]): Array[Double]
                        = x2.indices.map(i => x2(i) - x1(i)).toArray

}
```

3.4 Πολυκλασική ταξινόμηση

Αφού υλοποιήθηκε ο SVM-SMO αλγόριθμος για την κατηγοριοποίηση δεδομένων δύο κλάσεων έγινε επέκταση του για την κατηγοριοποίηση δεδομένων που ανήκουν σε περισσότερες από δύο κλάσεις. Για να επιτευχθεί ο ο διαχωρισμό ς N κλάσεων με N>2 υπάρχουν δύο στρατηγικές και οι δύο σαν βάση χρησιμοποιούν τον αλγόριθμο SVM για δυαδικό διαχωρισμό.

3.4.1 Στρατηγική One vs Rest

Δημιουργούνται μοντέλα για τον διαχωρισμό κάθε μια κλάση έναντι όλων των υπολοίπων μαζί . Χρησιμοποιώντας τον απλό SVM αλγόριθμο δημιουργούνται N μοντέλα όσα και οι κλάσεις. Κάθε μοντέλο μπορεί να αποφασίσει αν ένα δείγμα ανήκει σε μία κλάση ή ανήκει στις υπόλοιπες . Δηλαδή θεωρούμε κάθε φορά τις υπόλοιπες κλάσεις σαν μια μεγάλη υπερκλάση. Για την εκπαίδευση κάθε μοντέλου θεωρούμε πως υπάρχουν μόνο 2 κλάσεις μία από τις αρχικές και μια που συμπεριλαμβάνει όλες τις υπόλοιπες οπότε για να εκπαιδεύσουμε ένα τέτοιο μοντέλο χρησιμοποιούμε όλα τα αρχικά δεδομένα εκπαίδευσης αλλάζοντας την ετικέτα τους ώστε να αποτυπώνεται ποιά ανήκουν κάθε φορά στην εικονική υπερκλάση.

Πιο συγκεκριμένα θα εκτελεστεί 3 φορές ο SVM για τη δημιουργία των παρακάτω μοντέλων

1) SVM-1 δίνει το μοντέλο A-Rest:Β Από τα δεδομένα εκπαίδευση [A] ως κλάση A και τα [B ,C] μαζι ώς κλάση REST
2) SVM-2 δίνει το μοντέλο B-Rest: Από τα δεδομένα εκπαίδευση [B] ως κλάση B και τα [A ,C] μαζι ώς κλάση REST
3) SVM-3 δίνει το μοντέλο C-Rest: Από τα δεδομένα εκπαίδευση [C] ως κλάση C και τα [A ,B] μαζι ώς κλάση REST

Στη συνέχεια κατηγοριοποιούμε το άγνωστο δείγμα με χρήση όλων των μοντέλων
μοντέλο A-Rest - Το δείγμα βρίσκεται από την πλευρά του REST
μοντέλο B-Rest- Το δείγμα βρίσκεται από την πλευρά του B
μοντέλο C-Rest -Το δείγμα βρίσκεται από την πλευρά του REST

Καταλήγουμε λοιπόν στο συμπέρασμα πως το δείγμα ανήκει στη κλάση B. Αν το δείγμα κατηγοριοποιηθεί σε 2 κλάσεις που δεν είναι REST επιλέγεται αυτή που έχει το μεγαλύτερο margin.

Υλοποίηση κώδικα

Η υλοποίηση του παραπάνω σεναρίου εκπαίδευσης γίνεται από την κλάδη **OneVsRestSVMExecutor**. Ο κλάση executor δέχεται το αρχικό σύνολο δεδομένων και μια λίστα από ClassPair(class1, class2) αντικείμενα. Κάθε ClassPair αντιπροσωπεύει ένα ζευγάρι κλάσεων(class1 και class2) για το οποίο πρέπει να εκπαιδευτεί και ένα ξεχωριστό μοντέλο. Για τη στρατηγική One vs Rest το ClassPair αποτελείται από μια κλάση από της αρχικής και μια δεύτερη που ειναι σταθερά η ψευδοκλάση REST, δηλαδή όλες οι υπόλοιπες κλάσεις. Για παράδειγμα σε σύνολο δεδομένων που χαρακτηρίζονται από της κλάσεις A, B, C τα ClassPairs που πρέπει να εκπαιδευτούν είναι τα A-Rest, B-Rest, C-Rest. Ο κώδικας μετασχηματίζει για κάθε εκπαίδευση το αρχικό σύνολο δεδομένων δίνοντας την κλάση 1 στο πρώτο συνθετικό του ClassPair και την κλάση -1 σε όλα τα υπόλοιπα δείγματα.

```
val featureExtractor = newFeatureExtractor.apply()
val dfFeatures: DataFrame = featureExtractor.fitIDFModelAndExtractFeatures(trainDf)
val testDfFeatures: DataFrame = featureExtractor.extractFeature(testDf)

val modelsPerPair: Map[ClassPair, AbstractSVMModel] = pairs.map(pair => {


   val dfFiltered: DataFrame = stringLblToInt(dfFeatures, pair)

   val model: AbstractSVMModel = train(pair, dfFiltered)
   model.setPair(pair)
   model.setFeaturesExtractor(featureExtractor)
   (pair, model)

}).toMap
```

Παρατηρούμε πως και τα 3 μοντέλα θα εκπαιδευτούν πάνω στό ίδιο σύνολο δειγμάτων. Άρα και τα 3 μοντέλα έχουν το ίδιο σύνολο λέξεων για αυτό μπορούμε να εξάγουμε τα χαρακτηριστικά (features) συνολικά πριν ξεκινήσουν οι ξεχωριστές διαδικασίες εκπαίδευσης. Στο τέλος της μεθοδου train του executor έχουμε ενα Map με ένα μοντέλο ανά ClassPair. Για την κατηγοριοποίηση ενος άγνωστου δείγματος δίνουμε αυτό το Map στην κλάση **SVMPredictionOneVsRest**. Εκεί υπάρχει η μέθοδος predictRow(..) που θα δεχτεί το άγνωστο δείγμα και θα το κατηγοριοποιήσει χρησιμοποιώντα το Map και τα μοντέλα που δόθηκαν. Η μέθοδος predictRow κατηγοριοποιεί το δείγμα με κάθε μοντέλο ξεχωριστά και μετα επιλέγει κλάση νικητή.

```
def predictRow(body: String, lbl: String, features: Array[Double]): String = {

   val predictions: Iterable[ClassPairPrediction] = modelPerClassPair.values
       .map(m =>  m.predict(features))

   val nonRestWinners: Iterable[ClassPairPrediction] = predictions
   .filterNot(p=> p.resultClass.className.equalsIgnoreCase("rest"))

   if(nonRestWinners.nonEmpty) {

       val maxMargin = nonRestWinners.maxBy(_.getMargin)
       maxMargin.resultClass.className

   }else {

       val minMargin = predictions
                 .filter(p => p.resultClass.className.equalsIgnoreCase("rest"))
                 .minBy(_.getMargin)

       minMargin.resultClass.className
   }
}
```

3.4.2 Στρατηγική One vs One

Δημιουργούνται μοντέλα για τον διαχωρισμό κάθε μια κλάσης έναντι κάθε μιας άλλης κλάσης ξεχωριστά. Χρησιμοποιώντας τον απλό SVM αλγόριθμο εκπαιδεύονται N^2 -N μοντέλα. Για την εκπαιδευση κάθε μοντέλου θεωρούμε πως υπάρχουν μόνο δύο από τις αρχικές κλάσεις και διαχωρίζουμε τα δεδομένα εκπαίδευσης που ανήκουν σε αυτές. ΜΕ μόνο αυτα τα δεδομένα λοιπον γίνεται η εκπαίδευση του. Το μοντέλο που θα παραχθεί αποφασίζει σε ποια από αυτές τις δύο είναι πιο πιθανό να ανήκει το δείγμα. Για να ταξινομηθεί πληρως ενα δειγμα πρέπει πρώτα να ταξινομηθεί μεταξύ όλων των ζευγαριών.
πχ αν θελουμε να καταταχθεί το δείγμα μας μεταξυ των κλάσεων ABC
Δημιουργούνται τα μοντέλα
AB- Από τα δεδομένα εκπαίδευση labeled A ή B
BC- Απο τα δεδομένα εκπαίδευση labeled B ή C
AC - Από τα δεδομένα εκπαίδευση labeled A ή C

Στη συνέχεια

Κατηγοριοποιούμε το δείγμα με χρήση όλων των μοντέλων
AB- Το δείγμα βρίσκεται από την πλευρά του A
BC- Το δείγμα βρίσκεται από την πλευρά του C
AC -Το δείγμα βρίσκεται από την πλευρά του A

Καταλήγουμε λοιπόν στο συμπέρασμα πως το δείγμα ανήκει στη κλάση A. Αν υπάρξει ισοπαλία στις κλάσεις αποτελέσματα επιλέγεται αυτη που επιλέχθηκε με μεγαλύτερο margin.
Υλοποίηση κώδικα
Η υλοποίηση του παραπάνω σεναρίου εκπαίδευσης γίνεται απο τον OneVsOneSVMExecutor. Ο executor δέχεται το αρχικό σύνολο δεδομένων

Και μια λίστα από τη ClassPair(class1, class2) αντικείμενα. Κάθε class pair αντιπροσωπεύει ένα ζευγάρι κλάσεων(class1 kai class2) για το οποίο πρέπει να εκπαιδευτεί και ένα ξεχωριστό μοντέλο. Για τη στρατηγική One vs One τα Cllasspairs είναι όλοι οι δυνατοί συνδιασμοί κλάσεων. Για παράδειγμα σε σύνολο δεδομένων που χαρακτηρίζονται από της κλάσεις A, B, C τα ClassPairs που πρέπει να εκπαιδευτούν είναι τα A-B, B-C, A-C.

```
val modelsPerPair: Map[ClassPair, AbstractSVMModel] = pairs.map(pair => {


   val dfFiltered: DataFrame = filterByPair(trainDf, pair)
   val featureExtractor = newFeatureExtractor.apply()
   val dfFeatures: DataFrame = featureExtractor.fitIDFModelAndExtractFeatures(dfFiltered)

   val model: AbstractSVMModel = train(pair, dfFeatures)
   model.setPair(pair)
   model.setFeaturesExtractor(featureExtractor)
   (pair, model)

}).toMap
```

Η υλοποίηση του παραπάνω σεναρίου κατηγοριοποίησης άγνωστου δείγματος γίνεται από τον SVMPredictionOneVsOne

```
def predictRow(body: String, lbl: String, row: Row): String = {

       val predictionsPerClass: Map[ClassLabel,ClassPairPredictionWinner] =

       modelPerClassPair.values.map(m => {

           val features = m.featuresExtractor.extractFeature(row)
           m.predict(features)

       })
      .groupBy((l: ClassPairPrediction) => l.resultClass)

      .mapValues(new ClassPairPredictionWinner(_))

      val winner: (ClassLabel, ClassPairPredictionWinner) =

      predictionsPerClass.max(new Ordering[Tuple2[ClassLabel, ClassPairPredictionWinner]](){

                  override def compare(
                              x: (ClassLabel, ClassPairPredictionWinner),
                              y: (ClassLabel, ClassPairPredictionWinner)): Int = {

                      val countWinner: Int = Ordering[Int].compare(
                              x._2.countTimesWon,
                              y._2.countTimesWon)

                      if(countWinner!=0)
                          return countWinner

                      val marginWinner: Int = Ordering[Double].compare(
                              x._2.maxPrediction.getMargin,
                      y._2.maxPrediction.getMargin)
                      x._2.comment=s"RESULT based on margin"

                      if(marginWinner!=0)
```

```
                        return marginWinner

                    x._2.comment=s"UNDEFINED RESULT margin and count is the same"

                    marginWinner
                }
            })

   val successfulClassification =
        if (winner._1.className.equalsIgnoreCase(lbl))
            s"CORRECT ${winner._1.className}!!! "
        else
            s"WRONG ${winner._1.className}/correct is ${lbl}!!! "

   printResult(successfulClassification)
   return winner._1.className
}
```

Η συνολική υλοποίηση της διαδικασία εκπαιδευσεις και με τις 2 προσεγγισεις One vs One και One vs Rest θα περιγραφει σε παρακάτω κεφάλαιο.

3.5 Εξαγωγή χαρακτηριστικών από κείμενο

3.5.1 Προετοιμασία κειμένου

Αφαίρεση ενδιάμεσων λέξεων (stopwords)

Τα tweets που διαβάστηκαν από την εφαρμογή αρχικά βρισκόταν σε μη κατάλληλη μορφή για την κατανάλωση τους από τον SVM αλγόριθμο Σε πρώτη φάση αφαιρέθηκαν όλα τα σημεία στίξης όλοι οι χαρακτήρες εικονίδια όπως και τα hashtags. Στη συνέχεια αφαιρέθηκαν όλες οι συνδετικές λέξεις όπως άρθρα και άλλες λέξεις που δεν πρόσφεραν βοήθεια στο χαρακτηρισμό ενός tweet στην εκάστοτε κατηγορία.

Διαχωρισμός σε λέξεις (Tokenizing)

Στη συνέχεια το κείμενο μετατράπηκε σε ακολουθία λέξεων και κρατήθηκαν σε αυτή τη λίστα μόνο οι μοναδικές λέξεις. Πλέον κάθε σημείο προς κατηγοριοποίηση αντιπροσωπεύεται από μία λίστα μοναδικών λέξεων (bag of words).
Σε αυτό το σημείο δεν έχει σημασία η σειρά των λέξεων απλά η ύπαρξη τους.

3.5.2 Υπολογισμός βαρών με TF /IDF

Αρχικά μετράται η συχνότητα εμφάνισης κάθε λέξης στην λίστα λέξεων κάθε tweet. Μερικές όμως λέξεις είναι πιο σημαντικές από άλλες για την κατηγοριοποίηση κάθε κειμένου για αυτό θα αποδοθεί ένα βάρος σε κάθε λέξη. Για την απόδοση βάρους σε κάθε λέξη θα βρεθεί η συχνότητα εμφάνισης κάθε λέξης στο σύνολο των tweet που εξετάζονται και αντίστροφα οι συχνότερες λέξεις θα έχουν μικρότερο αναλογικά βάρος με τις πιο σπάνιες. Οπότε ο αντίστροφος της συχνότητας θα χρησιμοποιηθεί σαν βάρος, Ο πολλαπλασιασμός του αριθμού - βάρους κάθε λέξης στο σύνολο των tweets με τη συχνότητα της στο εκάστοτε tweet αποτελεί ένα αριθμητικό χαρακτηριστικό του. Τελικά σε κάθε λέξη ενός tweet αντιστοιχεί ένα αριθμητικό χαρακτηριστικο(βάρος). Οι λέξεις του συνολικού λεξικού που δεν εμφανίζονται στο tweet θα έχουν βαρος μηδέν. Το σύνολο αυτών των Ν βαρών όλων τον λέξεων του λεξικού για το συγκεκριμένο tweet αποτελεί το σημείο που θα κατηγοριοποιηθεί από τον SVM, το οποίο θα είναι ένα σημείο στο Ν-διάστατο χώρο. Ως λεξικό εννοούμε όλες τις λέξεις που εμφανίζονται έστω μια φορά σε κάποιο tweet από το σύνολο δεδομένων μας. Μπορουμε στη συνέχεια να περιορίσουμε τον αριθμό διαστάσεων κρατώντας τις Ν πιο σπανιες λέξεις από το λεξικό. Επιλέχθηκαν αυτές που εμφανίζονται πιο σπάνιες αφού θεωρούνται και πιο σημαντικές. Για την υλοποίηση του αλγορίθμου TF/IDF έχει χρησιμοποιηθεί η παράλληλη υλοποίηση του Mllib .

Ο IDF εξαρτάται από το συνολικό λεξικό που δημιουργήθηκε από τα δείγματα εκπαίδευσης. Αν αλλάξουν τα δείγματα αλλάζει το λεξικό και η συνολική εμφάνιση μιας λέξης άρα αλλάζει και το βάρος της. Γι αυτόν τον λόγο ένα μοντέλο IDF εκπαιδεύεται με το σύνολο τον δειγμάτων εκπαίδευσης . Με αυτό το μοντέλο εξάγουμε τα χαρακτηριστικά (features) σε κάθε ένα δείγμα ξεχωριστα . Κατα την διαδικασία ελέγχου που θα δοθούν τα άγνωστα δείγματα θα χρησιμοποιήσουμε αυτό το ήδη

εκπαιδευμένο μοντέλο για να εξάγουμε τα χαρακτηριστικά του άγνωστης κατηγορίας δείγματος. Δηλαδη θα χρησιμοποιηθεί το λεξιλόγιο και τα βάρη των λέξεων των δειγμάτων εκπαίδευσης για να βρεθούν τα χαρακτηριστικά από τα δείγματα ελέγχου. Αυτή η λεπτομέρεια είναι σημαντική ιδιαίτερα στην περίπτωση της πολυκλασικής one vs one εκπαίδευσης αφού κάθε διαχωρισμός του αρχικού δείγματος ανά ζευγάρι κλάσεων δίνει και ένα διαφορετικό λεξικό με διαφορετικά βάρη στις λέξεις.

3.5.3 Υπολογισμός N-gram

Σε μια δεύτερη παραλλαγή πέραν τον χαρακτηριστικών που παρήχθησαν από την εμφάνιση των μεμονωμένων λέξεων σε κάθε κείμενο χρησιμοποιήθηκαν επιπλέων χαρακτηριστικά που παρήχθησαν από τη συχνότητα εμφάνισης δύο ή τριών συνεχόμενων λέξεων. Αρχικά κάθε tweet μετατράπηκε σε λίστα από λέξεις. Στη συνέχεια μετατράπηκε σε λίστα από ζευγάρια 2 -3 συνεχόμενων λέξεων΄. Με τη ίδια διαδικασία που ακολουθήθηκε πριν μετρήθηκε η συχνότητα εμφάνισης κάθε ζευγαριού στο σύνολο των δειγμάτων tweet και ο αντίστροφος του πολλαπλασιάστηκε με τη συχνότητα εμφάνισης του ζευγαριού στο συγκεκριμένο tweet. Ετσι προστέθηκαν επιπλέων χαρακτηριστικά σε κάθε κειμενο που έδιναν περισσότερη σημασια στη σειρά εμφάνισης των λέξεων. Αργότερα θα γίνει μελέτη στα αποτελέσματα που έδωσε κάθε προσέγγιση.

4. ΠΕΡΙΓΡΑΦΗ ΒΑΣΙΚΩΝ ΥΛΟΠΟΙΗΜΕΝΩΝ ΣΕΝΑΡΙΩΝ

Παρακάτω θα περιγραφούν η συνολική διαδικασίας εκπαίδευση και ελέγχου για τα τέσσερα διαφορετικά σενάρια που υλοποιήθηκαν.

4.1 Η κλάση SVMSMOExecutors

Κατα την υλοποίηση δημιουργήθηκαν τέσσερις κλάσεις που υλοποιούν την κλάση AbstractSVMExecutor. Κάθε τέτοια executor κλάση τρέχει ένα ολοκληρωμένο σενάριο εκπαίδευσης SVM μοντελου και αξιολόγησης του.

- onevsone.SVMSMOParallelExecutorSimple
- onevsone.SVMSMOParallelExecutorCascade
- onevsrest.SVMSMOParallelExecutorSimple
- onevsrest.SVMSMOParallelExecutorrCascade

Ο ίδιος ο executor δέχεται από την εφαρμογή κατα τη δημιουργία του κλάσεις με τις στρατηγικές που θα ακολουθήσει στις διάφορες φάσεις της συνολικής διαδικασίας.

1) Δέχεται από την εφαρμογή πρώτον μία υλοποίηση της κλάσης FeatureExtractor η οποία δίνει στον executor τη στρατηγική την οποία θα χρησιμοποιήσει για να μετατρέψει κάθε γραμμή από τον πίνακα δεδομένων από απλό κείμενο σε αριθμητικά χαρακτηριστικά (features).
2) Επίσης δέχεται εξωτερικά μια υλοποίηση της κλάση SVMRunner. Η κλάση SVMRunner εκτελεί έναν SVM αλγόριθμος για μια ομάδα δεδομένων και επιστρέφει ένα μοντέλο.
3) Και τέλος δέχεται και τον αριθμό τον partitions που θα χωριστούν τα δεδομένα στη περίπτωση της παραλληλοποίησης.

```
val executor = new SVMSMOParallelExecutor(
                            /*Feature Extractor*/
                            () => new MLlibSimpleFeatureExtractor(3000, 2),
                            /*SVM Algorithm  Runner*/
                            _ => new SimplifiedPrattSVMSMORunner(),
                            /*Number of Partitions*/
                            partitions = 1)
```

Αφού δημιουργηθεί ένας Executor με αυτά τα πεδία μπορεί να τρέξει με τη χρήση της μεθόδου execute.

```
        tableName = "tvmovies_covid_sports"
        testTableName = "tvmovies_covid_sports_test"



        )

executor.execute(tableName, testTableName, pairs)
```

Κάθε execute δέχεται το όνομα του πίνακα με τα δεδομένα εκπαίδευσης και τον πίνακα με τα δεδομένα ελέγχου. Επίσης δέχεται μία δομή που περιέχει ένα σύνολο από κλάσεις, τις κλάσεις τις οποίες θα εκπαιδευτεί το μοντέλο να διακρίνει. Αν ο πίνακα δεδομένων περιέχει και δεδομένα που δεν ανήκουν σε αυτές αυτα θα εξαιρεθούν. Οι κλάσεις δίνονται σε μορφή πιθανόν ζευγαριών για να υποστηριχθεί ο πολυκλασικός ταξινομητής. Μετα την κλήση της execute ακολουθούν 3 φάσεις .

Φάση 1 Διαβασμα δεδομένων εισόδου / Εξαγωγή χαρακτηριστικών

1) Διαβάζονται τα δεδομένα από τον input πίνακα σε μορφή Dataframe και φιλτράρονται με το σύνολο τον κλάσεων που δόθηκαν σαν είσοδος. Το Dataframe αποτελείται από γραμμές rows που κάθε μια αντιστοιχεί σε ένα tweet.
2) Στη συνέχεια χρησιμοποιείται ο επιλεγμένος FeatureExtractor για να μετασχηματιστεί το Dataframe και και τελικά να προστεθεί σε κάθε γραμμή μια κολώνα που περιέχει vector με τα χαρακτηριστικά.

Φάση 2 Εκπαίδευση μοντέλου

3) Κατα τη φάση αυτή θα εκπαιδευτούν τα μοντέλα τα οποία είναι πάνω από έναν στη περίπτωση του πολυκλασικού ταξινομητή. Για κάθε διακριτό ζευγάρι

κλάσεων εκπαιδεύουμε ένα ξεχωριστό μοντέλο. Οι διαφορετικές στρατηγικές που ακολουθήθηκαν θα αναλυθούν παρακάτω στην ανάλυση της υλοποίησης καθε συγκεκριμένης κλάσης executor.

4) Η εκπαίδευση των μοντέλων για κάθε διαφορετικό ζευγάρι κλάσεων γίνεται διακριτά και ανεξάρτητα από τα υπόλοιπα ζευγάρια. Τρέχουμε δηλαδή μια ολοκληρωμένη φάση εκπαίδευσης (train) για όλα τα δεδομένα που ανήκουν στο συγκεκριμένο ζευγάρι και δημιουργούμε ένα ολοκληρωμένο μοντέλο που μπορεί να διακρίνει δεδομένα αυτών των δύο κλάσεων. Αργότερα στην φάση της πρόβλεψης (predict) θα χρειαστούν συνδυαστικά ολα τα εκπαιδευμένα μοντέλα για να βγει αποτέλεσμα για μια γραμμή. Αν δοθεί σαν είσοδος μόνο ένα ζευγάρι κλάσεων ο αλγόριθμος αμέσω εκφυλίζεται σε δυαδικό SVM και αφου υπάρχει μόνο ένα ζευγάρι θα έχουμε και αποτέλεσμα μόνο ένα μοντελο. Οπότε οι πολυκλασικοί (multiclass) executors υποστηρίζουν και δυαδική(binary) ταξινόμηση. Τέλος να ειπωθεί πως κάθε ζευγάρι κλάσεων καλέι ξεχωριστά και σειριακά ena SVM runnel. Η ίδια η εκπαίδευση όμως δεν είναι σειριακή οπως αναλύσαμε νωρίτερα χωρίζουμε τα δεδομενα σε ομάδες (partition). Αρ ο κάθε ξεχωριστός SVM μπορεί να τρέξει σειριακά ή παράλληλα αλλα η εκπαίδευση των διαφορετικών μοντέλων γίνεται σειριακά η παραλληλία είναι εσωτερική.
5) Σε κάθε κύκλο εκπαίδευσης ζευγαριών κλάσεων απομονώνουμε από το αρχικό σετ δεδομένων αυτά που ανήκουν στις εκάστοτε 2 επιλεγμένες κλάσεις. Εδώ υπάρχει άλλη μια διαφοροποίηση μεταξύ των executors καθώς μερικοί βρίσκουν σε αυτό το σημείο τα χαρακτηριστικά (features) κάθε γραμμής και όχι νωρίτερα οπως ειπώθηκε στο βήμα 2. Η διαφοροποίηση θα αναλυθει στον one vs rest και one vs one executor.
6) Το αρχικού dataframe που αφορά τις κλάσεις του τρέτρέχοντος ζευγαριού χωρίζεται σε κομμάτια ανάλογα με το επίπεδο παραλληλοποίησης που έχει επιλεγεί.

7) Καθε κομμάτι εκπαιδεύεται παράλληλα με τη χρήση του επιλεγμένου SVMRunner και παράγει ένα partial μοντελο.
8) Εδώ υπάρχει μια διαφοροποίηση μεταξύ των executor ανάλογα πως θα ενωθούν ξανα τα παραλληλοποιημένα κομμάτια (simple aggregate vs cascade)
9) Τελικά σε όλες όμως τις υλοποιήσεις θα καταλήξουμε με ένα τελικό συνολικό μοντέλο AggregatedModel για κάθε ζευγάρι κλάσεων. Στη περίπτωση που το ζευγάρι κλάσεων είναι μόνο ένα θα εχουμε την απλή δυαδίκη περίπτωση ταξινόμησης (classification).

Φάση 3 Ταξινόμηση πρόβλεψη δεδομένων ελέγχου

10) Διαβάζονται τα δεδομένα ελέγχου σε Dataframe
11) Για κάθε μια γραμμή Dataframe με χρήση του AggregatedModel βρίσκεται η κλάση που προβλέπεται από το μοντέλο

```
val pre = new SVMPredictionOneVsOne(modelPerClassPair)
pre.predictWholeDataframe(testDataset)
```

12) Μετασχηματίζεται το dataframe και προστίθεται η κολώνα prediction

Η εύρεση της κλάση prediction στη περίπτωση που έχουμε μόνο δύο ζευγάρια ειναι απλή είναι αυτη που θα δημιουργήσει το ένα μοναδικό μοντέλο. Στην περίπτωση όμως του πολυκλασικού ταξινομητή γίνεται με 2 στρατηγικές αφού υπάρχουν τόσα αποτελέσματα όσα και μοντέλα. Η επεξήγηση θα γινει σε αργότερα με την ανάλυση των one vs one και one vs rest executors που θα εξηγηθούν οι λεπτομέρειες της κάθε στρατηγικής.

Φάση 4 Υπολογισμός ακρίβειας πρόβλεψης

Το μετασχηματισμένο dataframe δινεται στην κλάση του MLlib Multiclass Metrics η οποια συγκρίνοντας την πραγματικά κλάση κάθε γραμμής με την υπολογισμένη στην κολόνα Prediction θα μετρήσει την τελική ακρίβεια πρόβλεψης που πέτυχε το μοντέλο.

4.2 Η κλάση onevsone.SVMSMOParallelExecutorSimple

Ο συγκεκριμένος executor συνδυάζει τη στρατηγική one vs one για να πετύχει κατηγοριοποίση μεταξυ πολλαπλων κλάσεων και παραλληλοποίηση σε κάθε ξεχωριστή εκπαίδευση μοντέλου χρησιμοποιώντας των απλό τρόπο ένωσης των μοντέλων που παράγονται παράλληλα.

Στην περίπτωση που έχουμε δεδομένα που ανήκουν σε τρεις κλάσεις Α , Β και Γ πρέπει να εκπαιδευτούν ξεχωριστά μοντέλα για κάθε πιθανό ζευγάρι κλάσεων

-μοντέλο Α-Β

-μοντελο Α- Γ

-μοντέλο Β-Γ

Δίνεται λοιπόν στην εντολή εκπαυδευσης μια λίστα με αυτά τα ζευγάρια.
Η κλάση executor σε τρείς σειριακούς κύκλου θα τρέξει SVM για να παράξει αυτά τα μοντέλα. Ο SVM για κάθε μοντέλο ξεχωριστά θα τρέξει σειριακά ή παράλληλα. Σε αυτό το σημείο να σημειωθεί πως θα μπορούσε η παραλληλοποίηση να μεταφερθεί και να γίνει παράλληλη εκπαίδευση των ξεχωριστών μοντέλων με χρήση σειριακού SVM. Η συγκεκριμένη τακτική θα έχει καλύτερα αποτελέσματα γιατί θα απολάβαναι τη σίγουρα καλύτερη απόδοση σε ακρίβεια του σειριακού SVM χωρίς τα μειονεκτήματα της παραλληλοποίησης του. Η εκπαίδευση των ξεχωριστών μοντέλων για τα διαφορετικα ζευγάρια είναι τελειως αυτόνομες εργασίες χωρίς εξαρτηση μεταξυ τους ενώ η παραλληλοποίηση του ίδιου του SVM εσωτερικά σε ένα ζευγάρι ρίχνει συνήθως την ακρίβεια των αποτελεσμάτων της ταξινόμησης. Το θέμα της συγκεκριμένης πτυχιακής όμως εστίασε στην παραλληλοποίηση του ίδιου του SVM αλγορίθμου.

Αφού γίνει η αρχική ανάγνωση των δεδομένων εισόδου και αφού ξεκινήσει η εκπαίδευση κάθε μοντέλου για κάθε ζευγάρι ακολουθεί η μετατροπή του κειμένου σε χαρακτηριστικά. Στην συγκεκριμένη στρατηγική η μετατροπή του κειμένου κάθε εγγραφής σε χαρακτηριστικά διαφέρει ανάλογα με το σε ποιό συνολο ζευγαριών αναφερόμαστε.Χρησιμοποιούμε τη στρατηγική TF/IDF για την εξαγωγη

χαρακτηριστικών η οποία βρίσκει την συχνότητα μιας λέξεις στο κείμενο και μετά την αντίστροφη συχνότητα που εξαρτάται απο το ποσο συχνα εμφανίζεται η λέξη στο σύνολο τον κειμένων οπότε είναι εμφανές ότι εξαρτόμαστε κάθε φορά από το σύνολο των tweets. Το συνολο των tweets εισόδου όμως διαφέρει ανάλογα των διαφορετικών ζευγαριών κλάσεων που επεξεργαζόμαστε κάθε φορά. Τα χαρακτηριστικά μας ξανα υπολογίζονται σε κάθε κύκλο για την εκπαίδευση κάθε διαφορετικού ζευγος κλάσεων.

Παράδειγμα

Έστω ότι προσπαθούμε να βρούμε τα χαρακτηριστικά του tweet1

Κατα την εκπαίδευση του μοντέλου ΑΒ

Αν η λέξη “παράδειγμα” εμφανίζεται σε ένα tweet μια φορά

- και εμφανίζεται στο σύνολο των tweet που έχουν κλάση Α ή Β 10 φορές
- ενώ εμφανίζεται στο σύνολο των tweet που έχουν κλάση Β ή Γ 20 φορές

έχει εμφανώς μεγαλύτερη βαρύτητα στην πρώτη περίπτωση γιατί είναι πιο σπάνια. Οπότε σε αυτήν τη στρατηγική ο feature extractor καλείται σε κάθε υποσύνολο δεδομένων σε κάθε κύκλο εκπαίδευσης ζευγαριού κλάσεων.

Τα υπόλοιπα βήματα συνεχίζουν οπως αναφέρθηκαν προτήτερα.

Το μοντέλο κάθε ζευγαριού ξεκινάει παράλληλα σε p partitions. Εδώ χρησιμοποιείται η απλή παραλληλοποίηση. Χωρίζεται το dataframe σε p μέρη και τρέχει έναν SVM για κάθε μέρος παράλληλα με τα υπόλοιπα. Κάθε partition θα παράξει ένα PartialSVMModel. Αυτό περιέχει τα support vectors από το υποσύνολο δεδομένων του. Για p partition έχουμε τελικά p ομάδες από support vectors. Στην απλή παραλληλοποίηση ενώνουμε τα επιμερους συνολα με support vectors σε ενα ενιαιο συνολο. Αυτό είναι και το τελικό aggregate model που θα κάνει το διαχωρισμό μεταξυ του ζευγους κλάσεων.

Αφου τελειώσει ο κύκλος εκπαίδευσης ξεκινάει η πρόβλεψη του συνόλου δεδομένων ελέγχου.Επιλέγεται σειριακή καθε ενα δεδομένο ελέγχο, βρίσκονται τα χαρακτηριστικά του σημείου για κάθε ζεύγος κλάσεων και ταξινομείται σε μια κλάση από το κάθε ζεύγος με το αντίστοιχο μοντέλο. Για να βρούμε τα χαρακτηριστικά

χρησιμοποιείται ο feature extractor που όπως αναλύσαμε προτήτερα χρησιμοποιεί την συχνότητα εμφάνισης κάθε λέξης στο σύνολο των αρχικών δεδομένων που έγινε η εκπαίδευση.

Καταλήγουμε για το εκάστοτε άγνωστο σημείο με τόσες προβλέψεις όσες και τα ζευγάρια

- Από το μοντέλο ΑΒ βρίσκουμε αν το σημειο θα άνηκε στην κλάση Α ή Β έστω το αποτέλεσμα Α
- Απο το μοντέλο ΑΓ βρίσκουμε αν το σημειο θα άνηκε στην κλάση Α ή Γ έστω το αποτέλεσμα Α
- Από το μοντέλο ΓΒ βρισκουμε αν το σημειο θα άνηκε στην κλάση Γ ή Β έστω το αποτέλεσμα Γ

Η κλάση που επιλέχθηκε περισσότερες φορές είναι και η τελική πρόβλεψη.

Σε λίγες περιπτώσεις δεν υπάρχει ξεκάθαρος νικητής. Καθώς μπορεί και στις τρείς περιπτώσεις να ανήκει σε διαφορετική κλάση. Σε αυτή την περίπτωση επιλέγεται η κλάση που νίκησε με μεγαλύτερη απόσταση από τη γραμμή διαχωρισμού (μεγαλύτερο margin).

4.3 Η κλάση onevsone.SVMSMOParallelExecutorCascade

Η συγκεκριμένη παραλλαγή είναι ίδια με την προηγούμενη όσον αφορά το πολυκλασικό μέρος διαφέρει μόνο στην παραλληλοποίηση του SVM αλγορίθμου εσωτερικά σε κάθε ζευγάρι. Σε αυτή την περίπτωση κατα την εκπαίδευση ενός συνόλου δεδομένων χωρίζουμε αυτό το σύνολο σε p partitions. Αυτά εκπαιδεύονται παράλληλα με χρήση του επιλεγμένου SVM Runner. Κάθε ένα partition επιστρέφει ενα PartialSVMModel που περιέχει ποιά από τα δεδομένα εισόδου ειναι τα SVM support vectors του υποσυνόλου. Στη συνεχεια σε μορφή δέντρου τα αποτελέσματα συγχωνεύονται. Σε κάθε συγχώνευση δύο φύλλων του δέντρου δημιουργειται ενα

μικρότερο dataframe που αποτελειται μονο απο τα support vectors του κάθε partial μοντέλου. Σε αυτό το νέο dataframe τρέχει ξανά SVM για να δημιουργηθεί ένα νέο μοντέλο με λιγότερα support vectors. Σε εκθετικό αριθμό επαναληψεων θα καταλήξουμε με ένα τελικό Aggregated model με τα τελικά support vectors.
Στη συνέχεια θα γίνει η πρόβλεψη κάθε γραμμής όπως και στην στην προηγούμενη περίπτωση.

4.4 Η κλάση onevsrest.SVMSMOParallelExecutorSimple

Αυτή η παραλλαγή συνδυάζει τη στρατηγική one vs rest για να πετύχει κατηγοριοποίση μεταξύ πολλαπλών κλάσεων και παραλληλοποίηση σε κάθε ξεχωριστή εκπαίδευση μοντέλου χρησιμοποιώντας τον απλό τρόπο ένωσης των μοντέλων που παράγονται παράλληλα. Στην περίπτωση που έχουμε δεδομένα που ανήκουν σε τρείς κλάσεις Α , Β και Γ πρέπει να εκπαιδευτούν ξεχωριστά μοντέλα για κάθε πιθανή κλάση (ενώ προηγουμένως εξετάζαμε όλου τους διαφορετικούς συνδυασμούς ζευγαριών τώρα έχουμε μια εκπαίδευση για κάθε κλάση)
Κάθε ξεχωριστή κλάση εκπαιδεύεται έναντι των δεδομένων όλων των υπόλοιπων κλάσεων μαζί τα οποία ανήκουν σε μια εικονική κλάση REST.
Οπότε έχουμε για το παράδειγμα μας την εκπαίδευση των παρακάτω μοντελων

- μοντέλο Α-REST(Β,Γ)
- μοντελο Β- REST(Α,Γ)
- μοντέλο Γ-REST(Α,Β)

Σε όλους τους συνδυασμούς εκπαίδευσης θα χρησιμοποιηθεί όλο το αρχικό σύνολο των δεδομένων οπότε δεν χρειάζεται ξεχωριστή εξαγωγή χαρακτηριστικών σε κάθε ζεύγος. Το βάρος των λέξεων στο λεξικό σε όλες τις περιπτώσεις είναι κοινό. Οπότε σε αυτή την παραλλαγή το αρχικό dataframe θα μετασχηματιστεί συνολικά και θα προστεθεί η κολόνα features πριν ξεκινήσει η επιμέρους εκπαίδευση καθε μοντέλου.

Στη συνέχεια θα χρησιμοποιηθει η παράλληλη εκπαίδευση κάθε μοντέλου με τον απλό τρόπο όπως εξηγήθηκε προηγουμένως στο παράδειγμα 1.

Κατα την φάση πρόβλεψης έχουμε τρία μοντέλα

Εξετάζουμε κάθε σημείο από τα δεδομένα ελέγχου με κάθε μοντέλο ξεχωριστά

-μοντέλο A-REST προβλέπει ότι το σημείο στην κλάση REST

-μοντελο B- REST προβλέπει ότι το σημείο ανήκει στην κλάση REST

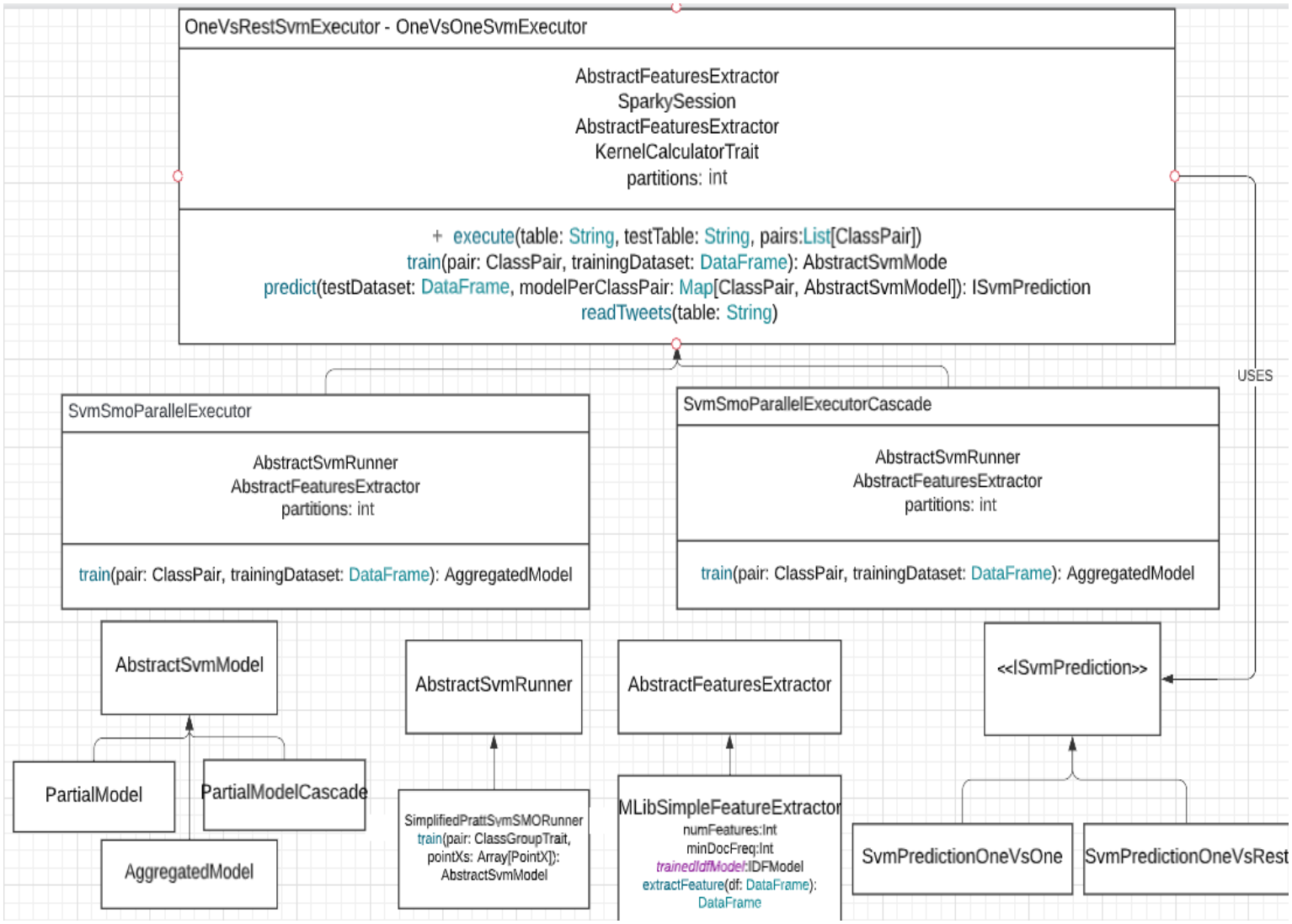
-μοντέλο Γ-REST προβλέπει ότι το σημείο ανήκει στην κλάση Γ

Αρα τελική πρόβλεψη είναι αυτή που δεν έχει επιλέξει τη κλάση REST

Αν υπάρξει αποτέλεσμα με δύο κλάσεις νικητές επιλέγεται αυτη που νίκησε με μεγαλύτερη απόσταση από τη γραμμή διαχωρισμού (μεγαλύτερο margin).

4.5 Η κλάση onevsrest.SVMSMOParallelExecutorCascade

Τέλος αυτή η υλοποίηση συνδέει την στρατηγική one vs rest και τη τύπου cascade παραλληλοποίηση του SVM αλγορίθμου. Η υλοποίηση είναι ίδια με αυτή που περιγράφηκε στο κεφάλαιο 4.4 για την κλάση onevsrest.SVMSMOParallelExecutorSimple με μοναδική διαφορά οτι εσωτερικά σε κάθε SVM χρησιμοποιείται η cascade παραλληλοποίηση όπως περιγράφηκε σο κεφάλαιο 4.3 όπως για την κλάση onevsone.SVMSMOParallelExecutorCascade.


OneVsRestSvmExecutor - OneVsOneSvmExecutor
AbstractFeaturesExtractor
SparkySession
AbstractFeaturesExtractor
KernelCalculatorTrait
partitions: int
+ execute(table: String, testTable: String, pairs:List[ClassPair])
train(pair: ClassPair, trainingDataset: DataFrame): AbstractSvmMode
predict(testDataset: DataFrame, modelPerClassPair: Map[ClassPair, AbstractSvmModel]): ISvmPrediction
readTweets(table: String)
USES
SvmSmoParallelExecutor
AbstractSvmRunner
AbstractFeaturesExtractor
partitions: int
train(pair: ClassPair, trainingDataset: DataFrame): AggregatedModel
SvmSmoParallelExecutorCascade
AbstractSvmRunner
AbstractFeaturesExtractor
partitions: int
train(pair: ClassPair, trainingDataset: DataFrame): AggregatedModel
AbstractSvmModel
PartialModel
PartialModelCascade
AggregatedModel
AbstractSvmRunner
SimplifiedPrattSvmSMORunner
train(pair: ClassGroupTrait, pointXs: Array[PointX]): AbstractSvmModel
AbstractFeaturesExtractor
MLibSimpleFeatureExtractor
numFeatures:Int
minDocFreq:Int
trainedIdfModel:IDFModel
extractFeature(df: DataFrame): DataFrame
<<ISvmPrediction>>
SvmPredictionOneVsOne
SvmPredictionOneVsRest

5. ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ

5.1 Φάση 1 Serial SVM vs Parallel SVM

5.1.1 Serial vs SimpleParallel SVM - 3000 Features - 525 Data

Παρακάτω θα αξιολογηθεί η εκπαιδευση του SVM σε ένα σύνολο 525 δεδομένων 2 κλάσεων, 291 δεδομένα από την πρώτη κλάση και 230 απ΄τη δεύτερη. Κατα τη φάση 1 θα εξαχθούν 3000 features για κάθε tweet με χρήση του MLlibSimpleFeatureExtractor. Αυτή η μέθοδος για το συγκεκριμένο train dataset μας δίνει μοντέλο SVM με 3000 διαστάσεις. Η φάση 1 θα εστιάσει στη σύγκριση της εκπαίδευσης του μοντέλου σειριακά και παράλληλα σε 2, 3 και 4 partitions.
Κατα την η σειριακή εξέταση θα χρησιμοποιηθεί ο executors

- SVMSMOSerialExecutor

Κατα την η παράλληλη εξέταση θα χρησιμοποιηθεί ο executor

- SVMSMOParallelExecutor

Serial

```
SVMSMOSerialExecutor
Training time:  63 secs
Support Vectors: 328/525
accuracy: 0.8352941176470589
Predict TEST- ended: 1.297 secs -
p0---->> BROKEN LOOP AT itr 6 - SVpoints: 328/ 525- Time 62805 millis
```

2 partitions

```
SVMSMOParallelExecutor
- Training time:   14secs
Support Vectors: 407/525
```

```
accuracy 0.8588235294117647
Predict TEST- ended: 1.518 secs -
p0---->> BROKEN LOOP AT itr 6 - SVpoints: 196/261 - Time 12993 millis
p1---->> BROKEN LOOP AT itr 7 - SVpoints: 211/264  - Time 14271 millis
```

3 partitions

```
SVMSMOParallelExecutor
p0---->> BROKEN LOOP AT itr 6 - SVpoints: 150/172 - Time 4839 millis -
p2---->> BROKEN LOOP AT itr 6 - SVpoints: 157/176 - Time 5298 millis -
p1---->> BROKEN LOOP AT itr 6 - SVpoints: 150/177 - Time 5298 millis -
------ Results ------
Training time:  5secs
Support Vectors: 457/525
Test Duration 108  millis
accuracy 0.8941176470588236
Predict TEST- ended: 1.604 secs -
```

4 partitions

```
SVMSMOParallelExecutor
p3---->> BROKEN LOOP AT itr 5 - SVpoints: 115/130 - Time  2604 millis
p1---->> BROKEN LOOP AT itr 5 - SVpoints: 120/130 - Time 2663 millis
p2---->> BROKEN LOOP AT itr 6 : SVpoints: 120/131 - Time 2880 millis
p0---->> BROKEN LOOP AT itr 6 - SVpoints: 120/134 - Time 3036 millis
------ Results ------
Training time:           3secs
Support Vectors: 475/525
accuracy 0.8941176470588236
Predict TEST- ended: 1.607 secs -
```

	Serial	**Parallel 2 part**	**Parallel 3 part**	**Parallel 4 part**
Time	63 sec	14 sec	5 sec	3 sec
accuracy	0.83	0.85	0.89	0.89
Support Vectors	328/525	407/525	457/525	475/525
Support Vectors Per partition	1-328/525	1-196/261 2-211/264	1-150/172 2-157/176 3-150/177	1-115/130 2-120/130 3-120/131 4-120/134

Παρατηρήσεις

Ο χρόνος μειώνεται καθώς αυξάνεται η παραλληλοποίηση ενώ η ακρίβεια παραμένει στα ίδια επίπεδα.

5.1.2 Serial vs Parallel Cascade SVM - 3000 features - 525 Data

Παρακάτω θα αξιολογηθεί η εκπαιδευση του SVM σε ένα σύνολο 525 ακριβώς όπως και στη φάση 1 με μόνη διαφορά ότι κατα την η παράλληλη εξέταση θα χρησιμοποιηθεί ο executor SVMSMOParallelExecutorCascade.

2 partitions

```
p0-->> LOOP iter6- Data: 262 SVpoints: 210/262 - Time 12490 millis
p1-->> LOOP iter6- Data: 263 SVpoints: 199/263 - Time 12647 millis
ADD--SVM between parts (0,1)- LOOP iter11- SVpoints: 317/409  - Time 51323 millis
Training time: 64 secs
Support Vectors: 317/525
accuracy 0.8470588235294118
Predict TEST- ended: 1.234 secs -
```

3 partitions

```
p2-->> iter5- SVpoints: 156/173 - Time 5179 millis -
p0-->> iter5- SVpoints: 158/176 - Time 5344 millis
p1-->> iter7- SVpoints: 152/176 - Time 5777 millis
ADD--SVM between parts (2,0)- LOOP iter5- SVpoints: 250/314 - Time 17201 millis
ADD--SVM between parts (2,1)- LOOP iter6- SVpoints: 317/402 - Time 35887 millis
Training time: 58 secs
Support Vectors: 317/525
accuracy 0.8470588235294118
Predict TEST- ended: 1.371 secs -
```

4 partitions

```
p2-->> LOOP iter5- SVpoints: 119/133 - Time 2917 millis
p0-->> LOOP iter5 -SVpoints: 125/132 - Time 2934 millis
p3-->> LOOP iter7- SVpoints: 117/129 - Time  3081 millis
p1-->> LOOP iter7- SVpoints: 120/131 - Time 3139 millis
ADD--SVM between parts (2,0)- 244 SVpoints: 203/244 - Time 11144 millis
ADD--SVM between parts (2,3)- LOOP iter6  SVpoints: 255/320 - Time 20504 millis
ADD--SVM between parts (2,1)- LOOP iter7- SVpoints: 311/375 - Time 34501 millis
Training time:   69 secs
```

Support Vectors: 311/525

accuracy 0.8470588235294118

Predict TEST- ended: 1.247 secs -

	Serial	**Cascade 2part**	**Cascade 3part**	**Cascade 4part**
Time	63 sec	64 sec	58 sec	69 sec
accuracy	0.83	0.84	0.84	0.84
Support Vectors	328/525	317/525	317/525	311/525
Support Vectors Per partition	1-328/525	1-210/262 2-199/263 MERGE-317/409	1-156/173 2-158/176 3-152/176 MERGE-250/314 MERGE-317/402	1-119/133 2-125/132 3-117/129 4-120/131 MERGE-203/244 MERGE-255/320 MERGE-311/375

Παρατηρήσεις

Στην συγκεκριμένη περίπτωση ο αριθμός των σημείων (525) δεν είναι κατα πολύ μεγαλύτερος από τον αριθμό των διαστάσεων(3000). Για να υποστηριχθει ο διαχωρισμός σε κάθε διάσταση είναι πολύ πιθανόν να χρειαστεί πολύ μεγάλος αριθμός support vectors. Το εκπαιδευμένο μοντέλο μπορεί να καταλήξει να χρησιμοποιήσει ακόμα και σχεδόν όλα τα δεδομένα σαν support vectors. Ο cascade SVM στηρίζεται στην μείωση των δεδομένων σε κάθε γύρο για επιτευχθεί η μείωση του χρόνου κατά την παραλληλοποίηση. Όπως φαίνεται στα αποτελέσματα για παράδειγμα του cascade SVM με 2 παράλληλα μέρη τα 525 δεδομένα χωρίστηκαν σε 2 μέρη και έτρεξαν 2 παράλληλοι SVM για 210 και 199 δεδομένα .Αυτη η πρώτη φάση κράτησε μονο 17 δευτερόλεπτα έδωσε σαν αποτέλεσμα 210 support vectors για το

πρώτο μέρος και 199 για το δεύτερο. Αυτά τα support vectors θα ενωθούν στη συνέχεια και θα ξανα γίνει SVM σε αυτά. Το πρόβλημα εδώ είναι ότι όταν ενωθούν ξανα καταλήγουμε σχεδόν στο ίδιο μέγεθος προβλήματος που ξεκινήσαμε. Η Συγκεκριμένη μέθοδος θα ήταν αποτελεσματική αν είχαμε λιγότερα χαρακτηριστικά - διαστάσεις οποτε κάθε 2 παράλληλοι SVM μείωναν εκθετικά το αρχικό πρόβλημα. Στις δύο παρακάτω φάσεις θα εξεταστεί αναλυτικότερα η περίπτωση

-900 χαρακτηριστικα με 525 δεδομένα και

-900 χαρακτηριστικα με 3885 δεδομένα

κρατώντας τις υπόλοιπες μεταβλητές ίδιες ώστε να να βρεθεί το σημείο που αρχίζει και γίνεται αποδοτικός η αλγόριθμος τύπου cascade.

	Serial	Cascade 2part	Cascade 3part	Cascade 4part
Time	63 sec	64 sec	58 sec	69 sec
accuracy	0.83	0.84	0.84	0.84
Support Vectors	328/525	317/525	317/525	311/525
Support Vectors Per partition	1-328/525	1-210/262 2-199/263 MERGE-317/409	1-156/173 2-158/176 3-152/176 MERGE-250/314 MERGE-317/402	1-119/133 2-125/132 3-117/129 4-120/131 MERGE-203/244 MERGE-255/320 MERGE-311/375

5.2 Φάση 2 Simple Parallel SVM vs Parallel Cascade SVM

5.2.3 Serial SVM vs Parallel Cascade SVM - 900 Features - 525 Data

Όπως φάνηκε στην προηγούμενο πίνακα ο αλγόριθμος παραλληλοποίησης cascade δεν βελτίωσε αρκετά τον χρόνο στις 3000 διατάσεις ανά σημείο. Σε αυτή τη φάση θα συγκρίνουμε τις ίδιες περιπτώσεις αλλα με τις μισές διαστάσεις ώστε να βελτιωθεί ο χρόνος αφού θα μειωθούν σα support σημεία.

Serial

```
p0---->>  LOOP AT itr 8 - 525 SVpoints: 354/525 - Time 64711 millis
Training time: 65 secs
Support Vectors: 354/525
accuracy: 0.8352941176470589
Predict TEST- ended: 0.965 secs -
```

Parallel 2 partitions - Cascade-

```
p0---->> LOOP AT itr 6 - SVpoints: 203/261 - Time 8532 millis
p1---->> LOOP AT itr 6 - SVpoints: 211/264 - Time 10559 millis
ADD----  LOOP AT itr 8 - SVM between partitions (0,1)- SVpoints: 329/414 - Time 38741 millis
Training time:  49 secs
AggregatedCascadeModel null - Support Vectors: 329/526
accuracy: 0.8705882352941177
Predict TEST- ended: 0.974 secs -
```

Serial -1500 Features

```
p1---->> LOOP AT itr 7- SVpoints: 211/261  - Time 8494 millis
p0---->> LOOP AT itr 7- SVpoints: 201/264  - Time 8917 millis
Training time:  9 secs
Support Vectors: 412/525
accuracy: 0.8352941176470589
Predict TEST- ended: 1.081 secs -
```

525 Data 1500 Features			
	Seria	**Cascade 2 part**	**Parallel 2 Part**
Time	65 sec	49sec	9sec
accuracy	0.83	0.87	0.83
Support Vectors	328/525	317/525	317/525

Support Vectors Per partition	1-328/525 -65	1-203/261 - 8s 2-211/264 - 10s MERGE-329/414 - 38s	1-211/261 - 8s 2-201/264 - 9s

Παρατηρήσεις

Παρατηρούμε ότι πράγματι με την μείωση των διαστάσεων ο παράλληλος cascade αλγόριθμος απέδωσε καλήτερα σε θέμα χρόνου με την αυξηση των partitions. Ενω διατήρησε αρκετή καλη ακρίβεια στα αποτελέσματα. Ο χρόνος μειώθηκε από 49 έναντι του σειριακού που έκανε 65 δευτερόλεπτα.Στις 9000 διαστάσεις έκανε ίδιο χρόνο με τον σειριακό Αυτη η μείωση έγκειται στα λιγότερα χαρακτηριστικά-διαστάσεις αρα και υπολογισμό πολύ μικρότερων εσωτερικών γινομένων. Η cascade περίπτωση λόγο του επιπλέων βήματος ένωσης των αποτελεσμάτων με έναν επιπλέων γύρο SVM προσθέτει 40 περισσότερα δευτερόλεπτα σε σχέση με την απλή παράλληλη περίπτωση. Ο παράλληλος cascade αλγόριθμος τελικά παρα το επιπλέον βήμα αποδίδει ακόμα καλύτερα από τον σειριακό χωρις μείωση της ακρίβειας. Παρα τις λιγοτερες διαστάσεις αυτές είναι ακόμα πολλές για τον αριθμό των δεδομένων οπότε τα 2 πρώτα παράλληλα βήματα δεν μείωσαν αρκετα τον αριθμό των support vectors ώστε το τελευταίο βήμα να είναι αρκετα πιο αποδοτικό.

5.2.4 Serial SVM vs Parallel Cascade SVM - 900 Features - 3885 Data

Σε αυτήν την φάση θα τρέξουμε τον cascade αλγόριθμο σε πολύ περισσότερα δεδομένα από τις διαστάσεις για να αποδειχθεί πως σε μια τέτοια περίπτωση κάθε βήμα του αλγορίθμου μειώνει αρκετά το αρχικό μέγεθος του προβλήματος ώστε να γίνει πιο αποδοτικός σε χρόνο απο τον σειριακό. Θα δοκιμαστει ο αλγόριθμος cascade σε 3885 δεδομενα ώστε να είναι αρκετά περισσότερα από τις 900 διαστάσεις.

Serial -900 Features -900 Features-3885 Data

```
p0---->> PARTITION SVM run on partIndex 0- LOOP AT itr 16:  SVpoints: 853/3885 - Time 14448733millis
Training time:   14449secs
Support Vectors: 853/3885
accuracy: 0.7647058823529411
```

Cascade Parallel 2 partitions - 900 Features - 3885 Data

```
p0---->> PARTITION SVM run on partIndex 0- LOOP AT itr 8: SVpoints: 585/1944  - Time 1416527 millis
p1---->> PARTITION SVM run on partIndex 1- LOOP AT itr 11: SVpoints: 597/1941  - Time 1930029 millis

ADD----LOOP AT itr 13 SVM between partitions (0,1)- SVpoints: 746/1182 - Time 976secs
Training time: 2907secs
Support Vectors: 746/3885
accuracy: 0.7411764705882353
Predict TEST- ended: 0.62 secs
```

Simple Parallel - 900 Features - 3885 Data

```
p0---->> PARTITION SVM run on partIndex 0- LOOP AT itr 8: SVpoints: 585/1944 - Time 1415105 millis
p1---->> PARTITION SVM run on partIndex 1- LOOP AT itr 11: SVpoints: 597/1941 - Time 1972454 millis
Training time:  1972secs
Support Vectors: 1182/3885
accuracy: 0.8117647058823529
```

3885 Data 900 Features			
	Seria	**Cascade 2 part**	**Parallel 2 Part**
Time	14449 sec	2907 sec	1972 sec
accuracy	0.76	0.73	0.80
Support Vectors	210/525	746/3885	746/3885
Support Vectors Per partition	1-853/38857	1-585/1944 - 1416 sec 2-597/1941 - 1930 sec MERGE-746/1182 - 976 sec	1-585/1944 -1415 sec 2-597/1941 -1972 sec

Παρατηρήσεις

Παρατηρούμε ότι πράγματι με την μείωση των διαστάσεων ο παράλληλος cascade αλγόριθμος σε κάθε κύκλο SVM μείωσε τα support vectors σε 1182 από τα αρχικά 3885 δεδομένα εισόδου. Αυτό και πάλι δεν ήταν αρκετό για να ξεπεράσει τον απλό παράλληλο αλγόριθμο.

5.3 Φάση 3 Πολυκλασική Ταξινόμηση (Multiclass)

5.3.1 One vs Rest - One vs One - 1 partition

Παρακάτω θα αξιολογηθεί η σειριακή πολυκλασική ταξινόμηση με SVM σε ένα σύνολο 525 δεδομένων τριών κλάσεων, Θα εξαχθούν 60000 features για κάθε tweet με χρήση του MLlibSimpleFeatureExtractor. Και θα συγκριθούν οι δύο στρατηγικές

- One vs Rest
- One vs One

1 Partition Serial - 525 Data 60000 Features		
	One vs Rest	**One vs one**
Time	346+363+350=**1059 sec**	152+150+172 = **474 sec**
accuracy	0.81	0.823

Παρατηρήσεις

Οι δυο αλγόριθμοι για σειριακή ταξινόμηση τριών κλάσεων απέδωσαν σε επίπεδο ακρίβειας σχεδόν το ίδιο με την στρατηγική One vs One να υπερτερεί ελάχιστα. Σε επίπεδο ταχύτητας η στρατηγική One vs One απέδωσε πολύ καλύτερα από την One vs Rest.

5.3.1 One vs Rest - One vs One - 3 partition

Παρακάτω θα αξιολογηθεί η παράλληλη πολυκλασική ταξινόμηση με SVM σε ένα σύνολο 525 δεδομένων τριών κλάσεων, Θα εξαχθούν 60000 features για κάθε tweet με χρήση του MLlib SimpleFeatureExtractor. Και θα συγκριθούν οι δύο στρατηγικές

- One vs Rest
- One vs One

Κατα τον παράλληλο SVM αλγόριθμο θα χρησιμοποιηθούν τρία partitions.

3 Partition Serial -525 Data 60000 Features		
	One vs Rest	**One vs one**
Time	42+45+48=**95 sec**	18+22+21 = 61 sec
accuracy	0.82	0.83

Παρατηρήσεις

Οι δυο αλγόριθμοι για παράλληλη ταξινόμηση τριών κλάσεων απέδωσαν σε επίπεδο ακρίβειας σχεδόν το ίδιο με την στρατηγική One vs One να υπερτερεί ελάχιστα. Σε επίπεδο ταχύτητας η στρατηγική One vs One απέδωσε πολύ καλύτερα από την One vs Rest.

5.4 Φάση 4 - Γραμμικός και Μη γραμμικός διαχωρισμός

Κρατώντας σταθερα 2 κλασεις στις δοκιμές

- Linear Kernel - dot product
- PolynomialKernel
- Gaussian Kernel

Η συγκεκριμένη φάση δεν θα σχολιαστει αφού δεν απέδωσε αποτελέσματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] John C. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines”, Microsoft Research Technical Report MSR-TR-98-14 April 21, 1998

[2] Zanni,Serafini,Zanghirati, “Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems”, 2006

[3] ScienceDirect, “ADC and DAC for biomedical application”

[4] Keerthi, Shevade Bhattacharyya ,Murthy, “Improvements to Platt's SMO Algorithm for SVM Classifier Design ”, National University of Singapore

[5] CS229 Simplified SMO Algorithm 1 CS 229, Autumn 2009 The Simplified SMO Algorithm

[6] Kernel Support Vector Machines for Classification and Regression in C#- APRIL 27, 2010 / CESARSOUZA

[7] An Idiot’s guide to Support vector machines (SVMs) R. Berwick

[8] A MapReduce based Parallel SVM for Large Scale Spam Filtering - Godwin Caruana1 , Maozhen Li1,3 and Man Qi2

[9] MapReduce based RDF Assisted Distributed SVM for High Throughput Spam Filtering

[10] An Ontology Enhanced Parallel SVM for Scalable Spam Filter Training - Godwin Caruana1 , Maozhen Li, and Yang Liu