ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τεχνικές Μηχανικής Μάθησης για την Πρόβλεψη Σφαλμάτων σε Λογισμικό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του
Δρόση Οδυσσέα

Επιβλέπων Καθηγητής : Φωτάκης Δημήτριος,
Αναπληρωτής Καθηγητής ΕΜΠ

ΑΘΗΝΑ, 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΜΠ

Machine Learning Techniques to Predict Software Bugs

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δρόση Οδυσσέα

Επιβλέπων Καθηγητής : Φωτάκης Δημήτριος Αναπληρωτής Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή επιτροπή στις 16/7/2019

(Υπογραφή)	(Υπογραφή)	(Υπογραφή)
……..	……..	…….
Δημήτρης Φωτάκης	Γιώργος Στάμου	Αριστείδης Παγουρτζής
Αν. Καθηγητής	Αν. Καθηγητής	Αν. Καθηγητής
ΕΜΠ	ΕΜΠ	ΕΜΠ

(Υπογραφή) .....

Δρόσης Οδυσσέας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Ευχαριστίες

Στην όλη διαδικασία έρευνας και μελέτης για την εκπόνηση της διπλωματικής μου εργασίας ο επιβλέπων καθηγητής μου, κ. Δημήτριος Φωτάκης, ήταν δίπλα μου και οφείλω να τον ευχαριστήσω ιδιαίτερα για τη βοήθειά του και τις οξυδερκείς επισημάνσεις του. Σημαντικότερη όμως και καταλυτική ήταν η βοήθειά του στις αποφάσεις σχετικά με το μέλλον μου. Φωτίζοντας πτυχές της προσωπικότητάς μου, μου έδειξε το δρόμο για τα επόμενα βήματα.

Πολλές ευχαριστίες επίσης θα ήθελα να εκφράσω προς τα ανθρώπους που στελεχώνουν το τμήμα < R&D NOKIA SOLYTIONS AND NETWORKS HELLAS >για την παραχώρηση των δεδομένων στοιχείων που επεξεργάστηκα και με βάση τα οποία κατέληξα στα συμπεράσματα που παρουσιάζονται στην εργασία μου.

Τέλος ξεχωρίζει πάντα η οφειλή μου στον αδελφό και τους γονείς μου για την υποστήριξή τους , καθ όλο το διάστημα των σπουδών μου με κατανόηση , διακριτικότητα και πολλή αγάπη.

Δρόσης Οδυσσέας
Ιούλιος

Περίληψη

Στις μέρες μας ένα μεγάλο ποσοστό των προγραμματιστών ασχολούνται με την επιδιόρθωση αλγορίθμων και την εύρεση μεθόδων ελαχιστοποίησης των λαθών σε αυτούς. Η ανάγνωση, η κατανόηση και η επιδιόρθωση ενός κώδικα αποτελεί ένα αρκετά δύσκολο πρόβλημα και απαιτεί πολύ μεγάλη υπολογιστική δύναμη αλλά και ανθρώπινο δυναμικό. Ένας πιο αποδοτικός τρόπος επίλυσης του προβλήματος αυτού που παρουσιάζεται και σε όλες τις μεγάλες εταιρίες παγκοσμίου βεληνεκούς θα ήταν μέσω αλγορίθμων να υπολογίζουμε τις πιθανότητες εμφάνισης λαθών λογισμικού είτε γνωρίζοντας τον κώδικα αυτού είτε βασιζόμενοι αποκλειστικά σε δεδομένα του παρελθόντος είτε συνδυάζοντας τα δύο παραπάνω.

Και οι δύο τρόποι υλοποιούνται με αλγορίθμους μηχανικής μάθησης και τις αντίστοιχες τεχνικές. Στην διπλωματική αυτή εργασία θα αφοσιωθούμε στις δυνατότητες πρόβλεψης λαθών λογισμικού χωρίς να γνωρίζουμε τον κώδικα της εφαρμογής ούτε την γλώσσα προγραμματισμού που έχει υλοποιηθεί αλλά μόνο το πλήθος και το είδος των bugs που εμφανίστηκαν από την μέρα που έχουμε αρχίσει την μελέτη. Οι αλγόριθμοι που θα υλοποιηθούν είναι μηχανικής μάθησης και προσαρμόζονται στα δεδομένα που διαθέτουμε χωρίς καμιά πρώιμη γνώση. Επίσης γίνεται αναφορά σε ήδη υπάρχοντες τρόπους επίλυσης του προβλήματος αυτού, σύγκριση αυτών με τις δίκες μου μεθόδους καθώς και διαφορετικούς τρόπος επίλυσης ακολουθώντας την ίδια μέθοδο.

Τέλος, γίνεται σύγκριση των μεθόδων που χρησιμοποιήθηκαν (Poisson regression, Naive Bayes, Decision trees, Neural Networks Relu Activation Function, Logistic Regression) με την χρήση μετρικών που χρησιμοποιούνται για δυαδική ταξινόμηση. Στη συνέχεια αποφαινόμαστε σχετικά με την βέλτιστη προσέγγιση στο πρόβλημα αυτό που έγινε με την χρήση χρονοσειρών με την μέθοδο της Logistic Regression καθώς και τα επιπλέον δεδομένα που θα έπρεπε να διαθέτουμε ώστε να βελτιώσουμε την απόδοση των προσεγγίσεων.

Λέξεις κλειδιά

Αλγόριθμοι μηχανικής μάθησης, Σφάλματα λογισμικού, Λογιστική Παλινδρόμηση, Δυαδική Ταξινόμηση, Μετρικές, Χρονοσειρές, Μαρκοβιανή αλυσίδα

Abstract

Nowadays, a large percentage of developers are engaged in repairing algorithms and finding ways to minimize errors in them. Reading, understanding and correcting a code is a fairly challenging problem and requires a lot of computing power but also human resources. A more efficient way to solve this problem across all major global corporations would be by means of algorithms to calculate the chances of software errors either by knowing the code or by relying solely on past data or by combining the two above.

Both ways are implemented with machine learning algorithms and the corresponding techniques. In this diploma thesis we will devote ourselves to the possibilities of predicting software errors without knowing the application code neither the programming language that has been implemented but only the number and type of bugs that have occurred since the day we started the study. The algorithms to be implemented are mechanical learning and are adapted to the dataset we have without any early knowledge. Also, reference is made to existing ways of solving this problem, comparing these with my trials methods and different ways of solving the same method.

Finally, we compare the methods used (Poisson regression, Naive Bayes, Decision trees, Neural Networks with Relu Activation Function, Logistic Regression) by using metrics designed for binary classification. We then make a decision on the optimal approach to the logistic regression using Logical Regression, as well as the additional data that we should have to improve the performance of the approaches.

Key words

Machine learning algorithms, Software Bugs, Logistic Regression, Binary classification, Metrics, Time series, Markov Chain

Περιεχόμενα

Πίνακας Εικόνων

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1. Ορισμός του προβλήματος

Το ζητούμενο της διπλωματικής αυτής εργασίας είναι η δημιουργία ενός αλγορίθμου για την πρόβλεψη λαθών λογισμικού χωρίς την γνώση του κώδικα της εφαρμογής, αλλά αποκλειστικά βασιζομένου σε ιστορικά δεδομένα του παρελθόντος της εφαρμογής αυτής. Ο στόχος αυτός επιτεύχθηκε με την δημιουργία αλγορίθμου μηχανικής μάθησης και την εκπαίδευσή του με τα ιστορικά δεδομένα. Η είσοδος του αλγορίθμου θα είναι ένας πίνακας Ν x Μ στοιχείων, όπου σε κάθε μια από τις Ν γραμμές του θα είχαμε ένα σφάλμα που εμφανίστηκε και σε κάθε μια από τις Μ στήλες αυτού θα ήταν χαρακτηριστικά αυτού, όπως η ημερομηνία, η εφαρμογή που βρέθηκε και οτιδήποτε άλλο χαρακτηριστικό διαθέτουν τα δεδομένα.

Ο αλγόριθμος μας θα απαντά στην ερώτηση <ποια είναι η πιθανότητα εμφάνισης ενός λάθους αύριο με αυτά τα χαρακτηριστικά> Πιο συγκεκριμένα χωρίζουμε τα δεδομένα μας σε δύο κατηγορίες. Ως ημέρα με σφάλμα και ως ημέρα χωρίς σφάλμα. Στην πρώτη κατηγορία εντάσσεται η ημέρα στην οποία εμφανίστηκε τουλάχιστον ένα σφάλμα με τα χαρακτηριστικά που αναζητούμε.

Ο αλγόριθμος θα προβλέπει, αν η επόμενη μέρα θα θεωρείται ημέρα με σφάλμα ή όχι. Επίσης δίνεται η δυνατότητα στον χρήστη να απομονώνει συγκεκριμένα χαρακτηριστικά των λαθών και να αναζητά αποκλειστικά αυτά. Δηλαδή να ζητάει να προβλέψουμε την πιθανότητα εμφάνισης ενός σφάλματος με συγκεκριμένα χαρακτηριστικά σε κάθε μια από τις Μ στήλες του πίνακα.

1.2. Πρόβλεψη λαθών λογισμικού

Στις μέρες μας το ένα από τα μείζονα θέματα στον τομέα της πληροφορικής είναι η δημιουργία αλγορίθμων μηχανικής μάθησης ώστε να προσαρμόζονται σε κάθε πρόβλημα που προσπαθούμε να επιλύσουμε και εκμεταλλευόμενοι την υπολογιστική δύναμη που διαθέτουν να καταφέρνουν να πετύχουν καλύτερα αποτελέσματα από απλούς αλγορίθμους και να υπερνικήσουν την ανθρώπινη δύναμη.

Από την άλλη πλευρά ένα αρκετά σημαντικό πρόβλημα στον τομέα της πληροφορικής είναι η πρόβλεψη για λάθη λογισμικού σε κώδικες πριν αυτοί εισαχθούν στην αγορά είτε

ακόμα και κατά τη διάρκεια της χρήσης τους. Κάθε εταιρία θα ήθελε με μεγάλη βεβαιότητα να γνωρίζει, αν οι κώδικες που χρησιμοποιούν οι εφαρμογές της είναι ημέρα με σφάλμα ή όχι. Αυτός είναι ο βασικός λόγος που μεγάλο πλήθος προγραμματιστών ασχολείται αποκλειστικά με την επιδιόρθωση αλγορίθμων.

Το ερώτημα που προκύπτει είναι, αν θα καταφέρουμε να πετύχουμε την εκμετάλλευση αλγορίθμων μηχανικής μάθησης ώστε να επιλύσουμε το πρόβλημα της πρόβλεψης λαθών λογισμικού.

Το πρόβλημα της πρόβλεψης λαθών λογισμικού αποτελεί ένα από τα βασικότερα προβλήματα στην ανάπτυξη λογισμικού και την συνεχή επιδιόρθωσή του. Ένας βασικός λόγος που έχει τόσο μεγάλη σημασία είναι η δυνατότητα να προβλέπουμε, αν οι κώδικες που διαθέτουμε είναι χωρίς σφάλματα η όχι, κατά πόσο θα εμφανιστούν λάθη σε αυτούς τις επόμενες μέρες ακόμα και για να βρίσκουμε τα χαρακτηριστικά αυτών που θα εμφανιστούν τις προσεχείς μέρες. Για να μπορέσουμε να πετύχουμε τον σκοπό μας χρειαζόμαστε δεδομένα από το παρελθόν και τα λεγόμενα «static code metrics», που είναι μετρικές οι οποίες μας δίνουν μια αξιολόγηση για την ορθότητα του κώδικα, διαβάζοντας τον αλγόριθμο της εφαρμογής που επιθυμούμε να εξετάσουμε. Επιπλέον θα είναι ιδιαίτερα σημαντικό επίτευγμα για τις εταιρίες, καθώς, αν μπορέσουμε με μεγάλη πιθανότητα να αποφανθούμε πως ένας κώδικας είναι χωρίς σφάλμα, τότε το λογισμικό θα μπορεί να βγει στην αγορά με επιτυχία και να αναβαθμίσει την φήμη της εκάστοτε εταιρίας καθώς δεν θα υπάρχουν προβλήματα από τους χρήστες.

Εκμεταλλευόμενοι το ιστορικό της εμφάνισης των σφαλμάτων ή ένα πλήθος από μετρικές του κώδικα προσπαθούμε για κάθε αλγόριθμο που διαθέτουμε να προσαρμόσουμε τα δεδομένα σε αυτόν ώστε να καταφέρουμε να υπολογίσουμε πιθανότητες εμφάνισης λαθών τις επόμενες μέρες. Βεβαία είναι απαραίτητη προϋπόθεση να υπάρχει ένα μεγάλο πλήθος ιστορικών δεδομένων ώστε να καταφέρει να προσαρμοστεί ο αλγόριθμος μας. Μια τέτοια αξιοσημείωτη προσέγγιση είχε γίνει από την Facebook. Πιο συγκεκριμένα ο αλγόριθμος είχε την δυνατότητα με ελάχιστες τροποποιήσεις να καταφέρει να προσαρμοστεί σε οποιαδήποτε δεδομένα και να βρει τις βέλτιστες παραμέτρους για τον υπολογισμό των ζητούμενων πιθανοτήτων, οι οποίες στη συνεχεία μέσω ενός κατωφλίου στρογγυλοποιούνται σε ακέραιο αριθμό (δυαδικές τιμές, αν θέλουμε την εμφάνιση η μη την επόμενη ημέρα).

Τέλος, είναι απαραίτητο να υπάρχει η σωστή πρόβλεψη λαθών σε κάθε λογισμικό καθώς μπορεί να μειωθεί το κόστος από την έγκυρη πρόβλεψη των λαθών, αφού η διάγνωση ενός λάθους, όταν ακόμα βρισκόμαστε σε διαδικασία ανάπτυξης λογισμικού, είναι λιγότερο ζημιογόνα, διότι μπορεί να διορθωθεί άμεσα και δεν είναι ανάγκη ο υπόλοιπος κώδικας να βασίζεται σε ένα λανθασμένο κομμάτι.

1.3. Μηχανική μάθηση και εξόρυξη δεδομένων

Η επιστήμη της ανάλυσης δεδομένων και της ανακάλυψης μοτίβων σε αυτά, ονομάζεται εξόρυξη δεδομένων (Data Mining) [20]. Είναι μία αυτόματη ή ημιαυτόματη διαδικασία, η οποία εφαρμόζεται σε δεδομένα μεγάλου μεγέθους, με σκοπό την εύρεση συνδυασμών και μοτίβων που ακολουθούν τα δεδομένα αυτά. Αυτή η διαδικασία είναι εξαιρετικά σημαντική, αφού την εύρεση αυτών των μοτίβων ακολουθεί η πρόβλεψη μελλοντικών καταστάσεων και αποτελεσμάτων, βγάζοντας μη προφανή συμπεράσματα από τους κανόνες οι οποίοι προέκυψαν από την ανάλυση των δεδομένων.

Στην περίπτωση που αυτά τα μοτίβα οργανωθούν σε δομές, οι οποίες μπορούν να εξερευνηθούν και να χρησιμοποιηθούν μελλοντικά για την εξαγωγή συμπερασμάτων, τα μοτίβα αυτά ονομάζονται δομικά. Η μηχανική μάθηση ορίζεται ως η συλλογή από τεχνικές με σκοπό την εύρεση και περιγραφή των δομικών μοτίβων, ένα εργαλείο το οποίο βοηθά στην ερμηνεία των δεδομένων και στην πρόβλεψη των πιθανών αποτελεσμάτων.

Η μηχανική μάθηση είναι ένας κλάδος που σχετίζεται με την τεχνητή νοημοσύνη και την στατιστική. Απαραίτητο για την μηχανική μάθηση είναι ένα σύνολο δεδομένων που ονομάζουμε δεδομένα εκπαίδευσης (training data), τα οποία θα χρησιμοποιηθούν για την εκπαίδευση του υπολογιστή ώστε να μπορεί να βγάζει χρήσιμα συμπεράσματα για τα δεδομένα.
Η μηχανική μάθηση μπορεί να χωριστεί σε δύο μεγάλες κατηγορίες:

1.3.1 Επιβλεπόμενη Μάθηση

Επιβλεπόμενη μάθηση είναι ο κλάδος της μηχανικής μάθησης ο οποίος επιχειρεί να εξάγει μία συνάρτηση, που περιγράφει τα δεδομένα που έχουμε στην διάθεσή μας. Κάθε μονάδα δεδομένων αποτελείται από τα χαρακτηριστικά της (features) καθώς και το αποτέλεσμα στο οποίο οδηγεί (label). Ένα παράδειγμα επιβλεπόμενης μάθησης είναι η εκπαίδευση του υπολογιστή να μπορεί να καταλάβει από μία

φωτογραφία, εάν το ζώο, το οποίο εικονίζεται, είναι λύκος ή σκύλος, γνωρίζοντας εκ των προτέρων ποιο είναι το ζώο που περιγράφεται στην φωτογραφία για το κομμάτι της εκπαίδευσης Αφού η μηχανή εκπαιδευτεί, τότε, εάν της δώσουμε μία άγνωστη φωτογραφία, θα είναι σε θέση να βρει ποιο από τα δύο ζώα απεικονίζεται.

1.3.2 Μη Επιβλεπόμενη Μάθηση

Η μη επιβλεπόμενη μάθηση αποτελεί την προσπάθεια της εύρεσης μοτίβων στα δεδομένα, για τα οποία όμως δεν γνωρίζουμε τυχόν κατηγοριοποίηση που ακολουθούν (δηλαδή η ετικέτα είναι άγνωστη). Αντίστοιχο παράδειγμα, θα αποτελούσε, αν είχαμε ένα σύνολο φωτογραφιών από ζώα, χωρίς να γνωρίζουμε τι ζώα απεικονίζονται και εμείς να είχαμε ως σκοπό την εύρεση μοτίβων ώστε να ομαδοποιήσουμε τα ζώα που απεικονίζονται στις φωτογραφίες.

Το πρόβλημά μας, αποτελεί πρόβλημα επιβλεπόμενης μάθησης. Ανάλογα με το είδος της εξόδου, μπορούμε να έχουμε και διαφορετικό είδος επιβλεπόμενης μάθησης. Εάν το αποτέλεσμα αποτελεί μία κατηγορία (για παράδειγμα σκύλος ή λύκος), τότε έχουμε ένα πρόβλημα κατηγοριοποίησης (classification). Αντίθετα, αν προσπαθούμε να προβλέψουμε έναν αριθμό, για παράδειγμα μία τιμή ενός προϊόντος, τότε έχουμε ένα πρόβλημα παλινδρόμησης (regression).

1.4. Οι αλγόριθμοι που χρησιμοποιήθηκαν

Για την διπλωματική εργασία αυτή χρησιμοποιήθηκαν κατά κύριο λόγο αλγόριθμοι μηχανικής μάθησης, καθώς και ένας αλγόριθμος με κατανομή πιθανοτήτων.

Για τον αλγόριθμο κατανομής πιθανοτήτων που χρησιμοποιήσαμε (Poisson Distribution) έχουμε υποθέσει πως τα δεδομένα μας ακολουθούν μια κατανομή πιθανότητας και συγκεκριμένα την Poisson Distribution και το πρόβλημα απλοποιείται στην εύρεση της κατάλληλης παραμέτρου για την κατανομή αυτή. Όμως η υπόθεση πως τα δεδομένα μας προσομοιώνονται από μια από τις γνωστές κατανομές πιθανοτήτων είναι αρκετά λανθασμένη, όπως θα εξηγήσουμε και στη συνεχεία

Συγκεκριμένα όσον άφορα στην κατανομή Poisson, θα έπρεπε η κάθε ημέρα να είναι

ανεξάρτητη από την προηγούμενη και το κάθε πείραμα τύχης που εκτελούμε να μην επηρεάζει το επόμενο. Στο παράδειγμα μας όμως, υπάρχει η εποχικότητα (seasonality), δηλαδή υπάρχουν περίοδοι που η εμφάνιση των λαθών είναι αρκετά αυξημένη, κάτι που έρχεται σε αντιπαράθεση με τα χαρακτηριστικά της Poisson.

Επίσης, δεν ακολουθούν όλες οι ήμερες την ίδια κατανομή. Δηλαδή δεν είναι όλες οι μέρες ίδιες μεταξύ τους, αφού το Σαββατοκύριακο αποτελεί εξαίρεση, μιας και η εργατικότητα δεν είναι σε τόσο υψηλά επίπεδα σε σχέση με τις άλλες ήμερες. Αυτά τα χαρακτηριστικά δεν μπορούν να προσομοιωθούν από μια κατανομή Poisson, όποτε γίνεται εμφανές πως οι αλγόριθμοι μηχανικής μάθησης θα αποδώσουν καλύτερα σε σχέση με την κατανομή πιθανοτήτων.

Οι αλγόριθμοι μηχανικής μάθησης είναι αλγόριθμοι οι οποίοι προσπαθούν να εκπαιδευτούν και να εκτελέσουν μια εργασία χωρίς να γνωρίζουν αρχικά δεδομένα. Ο σκοπός των αλγορίθμων αυτών είναι να μάθουν να προβλέπουν με βάση το παρελθόν , να χειρίζονται προβλήματα ή να παίρνουν αποφάσεις που θεωρούν πως είναι βέλτιστες. Η διαδικασία εκτέλεσης ενός τέτοιου αλγόριθμου αποτελείται από δύο στάδια. Η εκπαίδευση και η αξιολόγηση. Στο πρώτο στάδιο ο αλγόριθμος παίρνει σαν είσοδο ένα πλήθος δεδομένων και προσπαθεί να ανακαλύψει μοτίβα ή οτιδήποτε άλλο μπορεί να επηρεάσει τις επιδόσεις του. Στο κομμάτι αξιολόγησης ο αλγόριθμος μας, όντας εκπαιδευμένος πλέον, αποφαίνεται για ποια απόφαση πρέπει να πάρουμε ή κάνει προβλέψεις (ανάλογα με την χρήση για την οποία έχει δημιουργηθεί). Σε αυτό το στάδιο αξιολογούμε τον κώδικα που έχουμε δημιουργήσει ανάλογα με τις επιδόσεις του (γνωρίζοντας ποια είναι τα βέλτιστα που θα μπορούσε να είχε αποφανθεί) και κρίνουμε από το αποτέλεσμα. Η διαδικασία που κρύβεται πίσω από έναν τέτοιο αλγόριθμο είναι η ελαχιστοποίηση μιας συνάρτησης κόστους (Loss function minimisation). Ένα απλό παράδειγμα είναι ο διαχωρισμός email σαν ανεπιθύμητη αλληλογραφία ή όχι [37].

Κατά τη διάρκεια αυτής της διπλωματικής εργασίας και για την επίτευξη του σκοπού της πρόβλεψης των λαθών λογισμικού εκπονήθηκαν αρκετοί αλγόριθμοι. Το πρόβλημα της πρόβλεψης λαθών λογισμικού σε δυαδικές τιμές (ημέρα με σφάλμα ή όχι) μπορεί να ενταχθεί σε μια ευρύτερη κατηγορία προβλημάτων. Αφού τα δεδομένα που διαθέτουμε μπορούν να πάρουν δύο τιμές και προσπαθούμε να αποφανθούμε, αν η επόμενη ήμερα θα είναι μέρα με λάθος ή όχι, τότε το πρόβλημα μας εντάσσεται στην κατηγορία της επιβλεπόμενης μάθησης (Supervised Learning).

Επιβλεπόμενη μάθηση (Supervised-learning)

Στην επιβλεπόμενη μάθηση έχουμε δεδομένα με ετικέτες και με βάση αυτά προσπαθούμε να ταξινομήσουμε σε κατηγορίες τα μελλοντικά δεδομένα (εφαρμόζεται σε classification, regression [18]).

Διάσημοι αλγόριθμοι:

•linear regression.

• logistic regression.

• naive Bayes.

• linear discriminant analysis.

• decision trees.

• k-nearest neighbor algorithm.

Η επιλογή του κατάλληλου αλγορίθμου για το πρόβλημα της πρόβλεψης λαθών λογισμικού αποτελεί ένα δύσκολο ζήτημα. Προϋποθέτει κατανόηση των δεδομένων που διαθέτουμε και πιο συγκεκριμένα ποιοι από τους παραπάνω αλγορίθμους αποδίδουν καλύτερα. Ένας τρόπος είναι η μέθοδος της δοκιμής και λάθους (test and error), καθώς δοκιμάζοντας κάθε δυνατό αλγόριθμο με διαφορετικούς τρόπους και αξιολογώντας τον με βάση τις μετρικές που θα ορίσουμε και στη συνεχεία θα μπορούμε να αποφανθούμε σχετικά με την βέλτιστη προσέγγιση.

Αντίθετα, υπάρχουν αλγόριθμοι που μπορούμε να αποφανθούμε πως δεν θα ταίριαζαν στο πρόβλημα μας καθώς είναι αρκετά απλοί για τα σύνθετα δεδομένα και δεν μπορούν να αποδώσουν και να κατανοήσουν τα μοτίβα που εμφανίζονται (κεφάλαιο 4).

ΚΕΦΑΛΑΙΟ 2 Το πλαίσιο που εντάσσεται το πρόβλημα- τρόποι αξιολόγησης της κάθε προσέγγισης

Στο κεφάλαιο αυτό εντάσσουμε το πρόβλημα της πρόβλεψης λαθών λογισμικού σε ένα γενικότερο πλαίσιο, ορίζοντας το μαθηματικό υπόβαθρο που απαιτείται για την κατανόησή του και εξηγώντας με ποιο μαθηματικό πλαίσιο μπορούμε να το προσεγγίσουμε. Επίσης εξηγούμε τους τρόπους αξιολόγησης της κάθε προσέγγισης, όπως και τους δύο βασικούς διαφορετικούς τρόπους, που έχουν εφαρμοστεί στο πρόβλημα αυτό.

2.1. Χρονοσειρές

i) Μια χρονολογική σειρά είναι μια σειρά σημείων στο πεδίο του χρόνου. Συχνότερα, μια χρονολογική σειρά είναι μια ακολουθία που λαμβάνεται σε διαδοχικά ισαπέχουσες χρονικές στιγμές. Έτσι είναι μια ακολουθία δεδομένων διακριτού χρόνου. Παραδείγματα χρονολογικών σειρών είναι τα ύψη των παλιρροιών των ωκεανών, οι μετρήσεις των ηλιακών κηλίδων και η ημερήσια τιμή κλεισίματος του Dow Jones.

Οι χρονολογικές σειρές χρησιμοποιούνται στην στατιστική, στην επεξεργασία σήματος, στην αναγνώριση μοτίβων, στην οικονομετρία, στα μαθηματικά οικονομικών, στην πρόγνωση καιρού, στην σεισμική πρόβλεψη, στο ηλεκτροεγκεφαλογράφημα, στη μηχανική ελέγχου, στην αστρονομία, στην μηχανική επικοινωνιών και σε μεγάλο βαθμό σε οποιοδήποτε τομέα των εφαρμοσμένων επιστημών και της μηχανικής που περιλαμβάνει χρονικές μετρήσεις.

Η ανάλυση χρονολογικών σειρών περιλαμβάνει μεθόδους για την ανάλυση δεδομένων χρονοσειρών προκειμένου να εξαχθούν σημαντικά στατιστικά στοιχεία και άλλα χαρακτηριστικά των δεδομένων. Η πρόβλεψη χρονολογικών σειρών είναι η χρήση ενός μοντέλου για την πρόβλεψη μελλοντικών τιμών βάσει προηγούμενων τιμών [5] [17].

Οι χρονοσειρές αποτελούν ένα εργαλείο για την επίλυση διάφορων προβλημάτων. Ένα από αυτά είναι η προσέγγιση μιας συνάρτησης στο πεδίο του χρόνου. Έχοντας σαν είσοδο διάφορες τιμές της συνάρτησης καθώς και τα ορίσματά της προσπαθούμε να δημιουργήσουμε μια φόρμουλα για να ελαχιστοποιήσουμε την απόκλιση της πρόβλεψης μας από την πραγματική τιμή της χρονοσειράς.

Επίσης μια ακόμα χρησιμότητα των χρονοσειρών είναι η χρήση αυτών για την πρόβλεψη τιμών με την χρήση στατιστικών μεθόδων (όπως υλοποίησα και στο πρόβλημα μας). Ειδικότερα παίρνοντας σαν δεδομένο την έξοδο της συνάρτησης που θέλουμε να προβλέψουμε σε παρελθοντικές χρονικές στιγμές προσπαθούμε να αποφανθούμε για την τιμή της στο μέλλον. Οι τρόποι με τους οποίους γίνεται κάτι τέτοιο αναλύονται στην συνεχεία.

Το πρόβλημα της πρόβλεψης λαθών λογισμικού, με την προσέγγιση χωρίς να λαμβάνουμε υπόψιν μας τις μετρικές του κώδικα, αποτελεί ένα πρόβλημα πρόβλεψης χρονοσειράς, διότι προβλέπουμε την επόμενη τιμή της ημέρας {0, 1} με βάση το παρελθόν της.

Θα μπορούσε το συγκεκριμένο πρόβλημα να ενταχθεί και στην κατηγορία προσέγγισης συναρτήσεων (function approximation). Θεωρώντας ως συνάρτηση f (n) ={0, 1} όπου n θα ήταν η ήμερα που θέλουμε (έχοντας ορίσει ως 0 την πρώτη μέρα που ξεκινάμε να μελετάμε το πρόβλημα) και ως f (n) αν εμφανίστηκε σφάλμα με τα χαρακτηριστικά που αναζητούμε την ήμερα αυτή. Με αυτήν την προσέγγιση έχουμε πρόβλημα χρονοσειρών διακριτού χρόνου [4].

2.2. Χρονοσειρές μαρκοβιανών αλυσίδων

Έχοντας κατανοήσει, γιατί το πρόβλημα της πρόβλεψης λαθών λογισμικού αποτελεί ένα πρόβλημα χρονοσειρών θα πρέπει να εξετάσουμε, αν αποτελεί ένα πρόβλημα Μακροβιανής αλυσίδας η όχι.

Ως Μαρκοβιανή αλυσίδα ορίζεται μια χρονοσειρά στην οποία η επόμενη κατάσταση εξαρτάται αποκλειστικά από την υπάρχουσα και όχι από τον τρόπο που φτάσαμε στο σημείο αυτό. Δηλαδή η πιθανότητα εμφάνισης μιας νέας κατάστασης δεδομένης της τωρινής κατάστασης είναι ίση με την πιθανότητα δεδομένου όλων των προηγούμενων καταστάσεων της. Σε μαθηματική μορφή:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, \dots \dots X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1})$$

Αυτή η προαναφερθείσα ιδιότητα ονομάζεται μαρκοβιανή ιδιότητα και είναι εμφανές πως το πρόβλημα της πρόβλεψης λαθών λογισμικού δεν εντάσσεται σε αυτήν την κατηγορία. Αν ίσχυε η παραπάνω ιδιότητα, τότε θα μπορούσαμε γνωρίζοντας αποκλειστικά την τελευταία ήμερα {0, 1} να προβλέπαμε τις πιθανότητες εμφάνισης της επόμενης {0, 1}.

Μία άλλη κατηγορία Μακροβιανής αλυσίδας είναι αυτή που διαθέτει μνήμη τάξης m. Δηλαδή, οι πιθανότητες εμφάνισης κάθε νέας κατάστασης μπορούν να οριστούν από τις τελευταίες m καταστάσεις της αλυσίδας αυτής. Σε μαθηματική μορφή:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, \dots\dots, X_1 = x_1) = P(X_n = x_n \big| X_{n-1} = x_{n-1}, \dots, X_{n-m} = x_{n-m})$$

$$\forall\, n > m$$

Το πρόβλημα της πρόβλεψης λαθών λογισμικού εντάσσεται σε αυτήν την κατηγορία όπως θα εξηγήσουμε και στην συνεχεία. Οι πιθανότητες εμφάνισης κάθε νέας κατάστασης ορίζονται από τις m τελευταίες καταστάσεις.

Ο αλγόριθμος μας είναι δομημένος με τέτοιο τρόπο ώστε για κάθε μια από τις 2^m πιθανές τελευταίες καταστάσεις να προβλέπει την πιθανότητα η επόμενη ήμερα να είναι bug-free ή όχι.
Η ιδιότητα αυτή βασίζεται στην φύση του προβλήματος, διότι η εποχικότητα επηρεάζει το αποτέλεσμα μας, και, όπως θα εξηγήσουμε και στη συνεχεία, πετυχαίνουμε ιδιαίτερα ικανοποιητικά αποτελέσματα, κάτι που ενισχύει την πεποίθηση μας πως οι τελευταίες m καταστάσεις είναι αρκετές για να προβλέψουμε την τιμή της νέας κατάστασης {0, 1}.

2.3. Προηγούμενες έρευνες

Έχοντας ορίσει, πλέον, μια βάση για το πρόβλημα της πρόβλεψης λαθών λογισμικού είναι απαραίτητο να αναφερθούμε σε δημοσιεύσεις που προϋπήρχαν στον τομέα αυτόν.

Κάθε μια δημοσίευση που έχει γίνει για την πρόβλεψη λαθών λογισμικού εντάσσεται σε μία από δύο μεγάλες κατηγορίες. Είτε προσπαθεί ο αλγόριθμος που έχει δημιουργηθεί να προβλέψει την εμφάνιση λαθών τις επόμενες ήμερες με βάση τον κώδικα και τις μετρικές

που έχει ορίσει είτε προσπαθεί να προβλέπει, αν θα εμφανιστεί κάποιο λάθος ή το πλήθος αυτών που θα βρεθούν με βάση χαρακτηριστικά που θεωρήθηκαν σημαντικά στην εκάστοτε περίπτωση.

Αρχικά, θα κάνουμε μια σύντομη αναφορά στις δημοσιεύσεις που προβλέπουν με βάση τις μετρικές του κώδικα , δηλαδή διαβάζοντας τον κώδικα να αποφαίνεται ο αλγόριθμος, αν περιέχει λάθη ή όχι με βάση διάφορες παραμέτρους [26] [8]

Μια παράμετρος που μπορεί να μας δώσει κώδικα με σφάλματα είναι ένα υψηλό επίπεδο ιεραρχίας μεταξύ των συναρτήσεων που χρησιμοποιούνται στον αλγόριθμο. Κάτι τέτοιο σε αρκετές περιπτώσεις κάνει τον κώδικα πιο δυσνόητο με αποτέλεσμα να αυξάνονται οι πιθανότητες για bugs [25].

Επίσης μια άλλη μετρική που μπορεί να μας δείξει, αν ο κώδικας περιέχει λάθη η όχι, είναι τα διαφορετικά μονοπάτια που μπορούν να εκτελεστούν κατά τη διάρκεια του αλγόριθμου για διαφορετικές εισόδους. Όσο αυξάνεται ο αριθμός αυτός τόσο πιο πιθανό είναι να υπάρξει κάποια είσοδος η οποία να αποδώσει λάθος. Ο όρος αυτός αποδίδεται ως κυκλική πολυπλοκότητα (Cyclomatic Complexity) [19].

Ο όρος χαμηλή σύζευξη (low coupling) είναι επίσης μια παράμετρος που μπορεί να επηρεάσει την απόδοση ενός κώδικα. Οι διάφορες συναρτήσεις που χρησιμοποιούνται για την εκτέλεση του αλγορίθμου οφείλουν να είναι ανεξάρτητες μεταξύ τους, ώστε να μπορούν να χρησιμοποιηθούν και σε άλλους αλγορίθμους με επιτυχία (έτσι διασφαλίζεται και η ακεραιότητα τους). Κάτι που φαίνεται να επηρεάζει την ακεραιότητα του κώδικα είναι το πλήθος γραμμών κώδικα της κάθε συνάρτησης που χρησιμοποιείται. Σε περίπτωση που μια συνάρτηση είναι μεγάλη σε έκταση, τότε είναι πιθανό να προσπαθεί να εκτελεί αρκετά σύνθετους υπολογισμούς με αυξημένη πιθανότητα λάθους

Τέλος, έχει δημιουργηθεί ακόμα μεγάλο πλήθος μετρικών για την επίτευξη αυτού του σκοπού και κάθε προσέγγιση που έχει γίνει μπορεί να χρησιμοποιηθεί με διαφορετικό συνδυασμό αυτών για να υλοποιήσει τον σκοπό της πρόβλεψης λαθών λογισμικού.

Μια άλλη προσέγγιση για το πρόβλημα αυτό, διαφορετική από την προηγούμενη, είναι να προσπαθούμε να κάνουμε ταξινόμηση κάθε αλλαγή στον κώδικα ως λανθασμένη ή όχι. Δηλαδή κάθε φορά που γίνεται κάποια αλλαγή ο αλγόριθμος ενημερώνεται για την

αλλαγή αυτή και αποφαίνεται, αν είναι λανθασμένη ή όχι, λαμβάνοντας υπόψιν τις γραμμές που τροποποιήθηκαν στον κώδικα. Η πρόβλεψη αυτή γίνεται με βάση το ιστορικό αντίστοιχων αλλαγών και πόσες από αυτές κατάφεραν να εξαλείψουν τα λάθη του κώδικα χωρίς να δημιουργήσουν νέα.

Από την άλλη πλευρά αρκετά μεγάλο πλήθος δημοσιεύσεων προσπαθεί να επιλύσει το πρόβλημα αυτό χωρίς να γνωρίζει τον κώδικα αλλά μόνο το ιστορικό των λαθών.

Σε περίπτωση που αναζητούμε το πλήθος των λαθών που θα εμφανιστούν τις επόμενες ημέρες είναι αρκετά σύνηθες στις δημοσιεύσεις να ορίζουν διαστήματα της μορφής [α, β] και να προσπαθούν να βρουν σε ποιο διάστημα θα εντάσσεται η επόμενη ημέρα αντί να προβλέπουν το ακριβές πλήθος λαθών που θα εμφανιστούν.

Σε ήδη υπάρχουσα δημοσίευση η πρόβλεψη γίνεται αποκλειστικά με το πλήθος των εργαζόμενων προγραμματιστών κάθε ήμερα, δηλαδή μια συνάρτηση της μορφής f (N) -> n όποτε N το πλήθος των προγραμματιστών και n το πλήθος των λαθών. Έτσι ο αλγόριθμος εκπαιδεύεται ώστε να καταφέρει να ελαχιστοποιήσει το σφάλμα της συνάρτησης αυτής κατατάσσοντας το n σε κατηγορίες διαστημάτων της μορφής [α, β]. Αν ο αλγόριθμος μας πετύχει την κατηγορία που βρίσκεται το n, τότε θεωρείται επιτυχημένη πρόβλεψη, ενώ σε διαφορετική περίπτωση όχι. Επιπλέον, ο κάθε αλγόριθμος χρησιμοποιείται συνήθως σε διαφορετικά δεδομένα για την πιο ορθολογική αξιολόγησή του [12].

Είναι εμφανές πως ένας συνδυασμός των δύο παραπάνω θα μπορούσε να αποδώσει καλύτερα αποτελέσματα. Κάτι τέτοιο άλλωστε προκύπτει από το γεγονός πως ως μελλοντική εργασία των δημοσιεύσεων ορίζεται η εισαγωγή μετρικών κώδικα σε όσες δημοσιεύσεις δεν είχαν την δυνατότητα να τα συμπεριλάβουν.

Η δικιά μας προσέγγιση στο πρόβλημα της πρόβλεψης λαθών λογισμικού εντάσσεται στην δεύτερη κατηγορία, διότι δεν είχαμε την δυνατότητα να διαθέτουμε μετρικές κώδικα και προσπαθούμε να προβλέψουμε βασιζόμενοι αποκλειστικά σε δεδομένα του παρελθόντος και να εκμεταλλευτούμε την εμφάνιση κάποιων μοτίβων ή άλλων στοιχείων χρονοσειρών που θα συμβάλλουν στην πρόβλεψη μας για την επόμενη μέρα και την κατηγοριοποίηση της σε μέρα με σφάλμα ή όχι.

Κάθε μία από τις ήδη υπάρχουσες δημοσιεύσεις χρησιμοποιεί τις μετρικές αξιολογήσεις

των αλγορίθμων μηχανικής μάθησης που αφορούν την επιβλεπόμενη μάθηση (supervised learning). Οι μετρικές αυτές αναλύονται στη συνεχεία και στο τέλος θα γίνει μια γενικότερη αξιολόγηση αυτών (Κεφάλαιο 4).

2.4. Μετρικές αλγορίθμων μηχανικής μάθησης

Για την αξιολόγηση αλγορίθμων μηχανικής μάθησης (machine learning algorithms) υπάρχουν κάποιες μετρικές που χρησιμοποιούμε για να καταφέρουμε να συγκρίνουμε τα αποτελέσματα μεταξύ διαφορετικών προσεγγίσεων. Η πιο προφανής μετρική που θα μπορούσε να σκεφτεί κάποιος είναι η ακρίβεια (accuracy) , δηλαδή το ποσοστό των σωστά ταξινομημένων παραδειγμάτων (instances) σε σχέση με το συνολικό πλήθος παραδειγμάτων που ταξινόμησε ο αλγόριθμος.

$$ακρίβεια\ (accuracy) = \frac{πλήθος\ σωστών\ προβλέψεων}{πλήθος\ συνολικών\ προβλέψεων}$$

Η μετρική αυτή, αν και εκ πρώτης όψεως φαίνεται ορθή και κατάλληλη για την αξιολόγηση αλγορίθμων, ένα απλό παράδειγμα μπορεί να μας πείσει για το αντίθετο. Η ταξινόμηση email σαν ανεπιθύμητο (spam) ή όχι θα μπορούσε να δώσει ακρίβεια 95%, αν προβλέπαμε πως κάθε email θα είναι ανεπιθύμητο, μιας και το ποσοστό αυτό αντιστοιχεί στο πραγματικό ποσοστό των email που θεωρούνται ανεπιθύμητα (spam) emails. Άρα θα μπορούσαμε να πετύχουμε ένα τέτοιο υψηλό ποσοστό στην μετρική αυτή, χωρίς πρακτικά να έχουμε δημιουργήσει κάποιον αλγόριθμο!

Αυτό το αίτιο αποτελεί ένα από τους βασικούς λόγους που δημιουργήθηκαν και άλλες μετρικές αξιολόγησης αλγορίθμων μηχανικής μάθησης.

Στο παράδειγμα μας ως 1 θεωρούμε τις ημέρες που εμφανίστηκε λάθος, ενώ ως 0 αυτές που δεν εμφανίστηκε κάποιο λάθος.

Πριν εστιάσουμε σε άλλες μετρικές θα ορίσουμε τέσσερις έννοιες που θα χρησιμοποιηθούν στη συνεχεία:

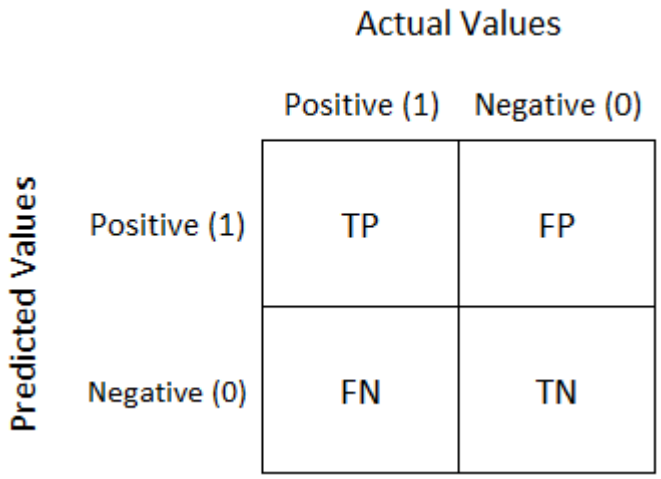
1) true positive: Το πλήθος των προβλέψεων που ήταν 1 δεδομένου πως η πραγματική τιμή ήταν 1.

2) true negative (TN)Το πλήθος των προβλέψεων που ήταν 0 δεδομένου πως η πραγματική τιμή ήταν 0.

3) false positive (FP) Το πλήθος των προβλέψεων που ήταν 1 δεδομένου πως η πραγματική τιμή ήταν 0.

4) false negative (FN) Το πλήθος των προβλέψεων που ήταν 0 δεδομένου πως η πραγματική τιμή ήταν 1.

Οι παραπάνω τέσσερις τιμές φαίνονται σε έναν πίνακα που ονομάζεται πίνακας σύγχυσης (ConfusionMatrix).



Εικόνα 2.1 Πίνακας Σύγχυσης

Έχοντας ορίσει πλέον τις τέσσερις αυτές έννοιες θα δούμε τις μετρικές αξιολόγησης [7]:

1)Precision:

$$precision = \frac{tp}{tp + fp}$$

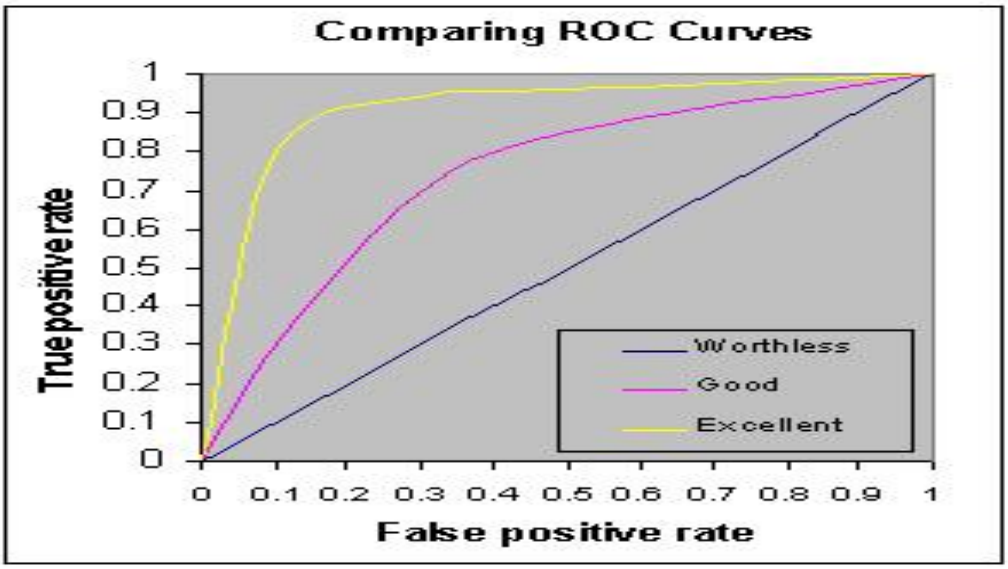
2)Recall:

$$recall = \frac{tp}{tp + fn}$$

3)F1 score:

$$F1\ score = \frac{2 * precision * recall}{precision + recall}$$

4)Roc (Receiver Operating Characteristic Curve) - AUC (Area Under Curve) [35], [24], [10].

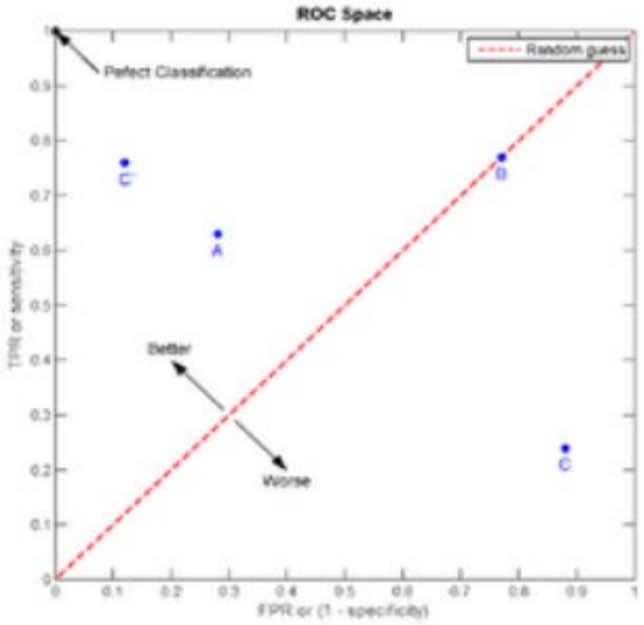


Εικόνα 2.2 Σύγκριση Roc Curve

Η μπλε γραμμή υποδηλώνει έναν τυχαίο ταξινομητή και κάθε σύγκριση γίνεται με αυτόν για να αποφανθούμε πόσο καλύτερος του είναι. Δηλαδή το AUC της μπλε γραμμής είναι 0. 5, όποτε αναζητούμε ταξονομητές με AUC > 0. 5.

Η μπλε γραμμή αποτελεί την γραφική παράσταση ROC ενός προβλεπτή, ενώ με διακεκομμένη ευθεία παρουσιάζεται ο τυχαίος προβλεπτής, ο οποίος βρίσκεται στην ευθεία x=y.΄ Όσο καλύτερο προβλεπτή έχουμε, τόσο μεγαλύτερο εμβαδό δημιουργείται ανάμεσα στην καμπύλη και την ευθεία x= 0.

Επομένως όσο πιο ψηλά βρίσκεται η ROC ενός προβλεπτή έναντι της x=y, τόσο πιο πετυχημένος είναι ο προβλεπτής. Σχηματικά αυτό φαίνεται παρακάτω:



Εικόνα 2.3 Roc Space

Κάθε μια από τις παραπάνω μετρικές θα μπορούσε να δώσει αποτελέσματα αρκετά καλά, χωρίς την ύπαρξη αξιόλογου αλγορίθμου σαν υπόβαθρο. Όμως ο συνδυασμός των μετρικών για να συγκρίνουμε αλγορίθμους είναι η πιο αξιόπιστη λύση. Παρατηρώντας όλες τις μετρικές και συγκρίνοντας τα αποτελέσματα τους στο ίδιο πρόβλημα μας προκύπτει ποια προσέγγιση είναι η πιο κατάλληλη ανάλογα με τον σκοπό που έχουμε για το πρόβλημα που προσπαθούμε να επιλύσουμε.

Οι παραπάνω τύποι χρησιμοποιούνται σε κάθε προσέγγιση που κάνουμε. Στο πρόβλημα μας το 1 αποτελεί την εμφάνιση λάθους την συγκεκριμένη μέρα, ενώ το 0 τη μη εμφάνιση. Με βάση αυτήν την υπόθεση υπολογίζουμε το πλήθος των TP, TN, FP, FN στα δεδομένα αξιολόγησης και βρίσκουμε τις μετρικές αυτές. Στη συνεχεία υπολογίζουμε τις παραπάνω μετρικές ώστε να αξιολογήσουμε την κάθε προσέγγιση.

Μια πιο εκτενής αναφορά σε κάθε μέθοδο που χρησιμοποιήθηκε και στα αποτελέσματα

που απέδωσε γίνεται στο κεφάλαιο 4, όπου παρουσιάζονται οι μετρικές για κάθε μια από τις μεθόδους που εφαρμόσαμε.

2.5. Overfitting – Underfitting

Ένα επιπλέον πρόβλημα που μπορεί να επηρεάσει τις μετρικές του αλγορίθμου μας είναι η εμφάνιση του φαινόμενου overfitting ή underfitting. Πιο συγκεκριμένα για το πρόβλημα της πρόβλεψης λαθών λογισμικού οφείλουμε να βρούμε το κατάλληλο πλήθος παραμέτρων που θεωρούμε πως επηρεάζει την πρόβλεψη μας για την επόμενη ήμερα. Η μη σωστή επιλογή των παραμέτρων μπορεί να μας οδηγήσει στο φαινόμενο του overfitting ή underfitting. [6].

Overfitting

Το overfitting αποτελεί ένα βασικό πρόβλημα στους αλγορίθμους μηχανικής μάθησης και πρέπει να λαμβάνεται υπόψιν από τους προγραμματιστές. Δίνοντας πολλές παραμέτρους σε ένα μοντέλο, αποκτάται η δυνατότητα να <ταιριάξει> πάνω στα δεδομένα με αποτέλεσμα μια μικρή αλλαγή να αποτελεί πρόβλημα στον αλγόριθμο μας ή ακόμα χειρότερα, σε οποιοδήποτε νέο παράδειγμα εκτός αυτού που έχει εκπαιδευτεί να παρουσιάζεται μια παράλογη συμπεριφορά. Για παράδειγμα στη δυαδική ταξινόμηση η διαχωριστική γραμμή μεταξύ των δύο κλάσεων πρέπει να διαθέτει συγκεκριμένα χαρακτηριστικά ανάλογα με την μέθοδο που χρησιμοποιήθηκε για να αποφευχθεί το πρόβλημα του overfitting.

Πιο συγκεκριμένα το πρόβλημα του overfitting αρχίζει να εμφανίζεται, όταν δίνουμε την ευχέρεια σε έναν αλγόριθμο να έχει παραπάνω παραμέτρους από αυτές που δικαιολογεί τα δεδομένα που διαθέτει. Έτσι αντί να εκπαιδεύεται, απλά απομνημονεύει ό,τι βλέπει με αποτέλεσμα η έξοδος του να είναι πιο πολύπλοκη από αυτήν που πραγματικά απαιτεί το πρόβλημα. Στην περίπτωση αυτήν ο αλγόριθμος μας θα έχει πολύ άσχημα αποτελέσματα σε δεδομένα που διαφέρουν λίγο από αυτά της εκπαίδευσης ή σε δεδομένα τα οποία δεν έχει δει (δεδομένα αξιολόγησης). Στα δικό μας δεδομένα και στην βέλτιστη μέθοδο που χρησιμοποιήθηκε, δηλαδή να έχουμε μια παράμετρο m που να μας δείχνει πόσες προηγούμενες ήμερες οφείλουμε να λάβουμε υπόψιν για να αποφασίζουμε, αν η επόμενη μέρα θα εμφανίσει λάθος ή όχι, υπάρχει ο εξής κίνδυνος overfitting: Αν η παράμετρος που ορίζουμε είναι αρκετά μεγάλος αριθμός, τότε στην φάση εκπαίδευσης τα βάρη θα

προσαρμοστούν σε τιμές τέτοιες ώστε να απομνημονεύσουν τα μοτίβο χωρίς ουσιαστικά να υπάρξει κάποια πραγματική εκπαίδευση. Έτσι κάθε φορά που θα αναγνωρίζει ο αλγόριθμός μας κάτι που έχει προϋπάρξει από την φάση της εκπαίδευσης θα αποδίδει ορθά με αρκετά μεγάλη πιθανότητα (τα βάρη μπορεί να μην είναι δυνατόν από μαθηματικής απόψεως να πάρουν τιμές τέτοιες ώστε να μπορούν να δώσουν 100% απόδοση στα δεδομένα εκπαίδευσης), όμως σε κάθε περίπτωση που δεν έχει ξαναδεί (ή σε ένα νέα δεδομένα του επόμενου έτους για παράδειγμα) να υπάρχει αρκετά χαμηλή απόδοση.

Underfitting:
Το underfitting αποτελεί ένα επίσης σύνηθες πρόβλημα στους αλγορίθμους μηχανικής μάθησης. Το πρόβλημα αυτό εμφανίζεται, όταν προσπαθούμε να προσεγγίσουμε ένα δεδομένο με μια πιο απλοϊκή συνάρτηση από αυτήν που πραγματικά απαιτείται (όπως η προσέγγιση του προβλήματος της πρόβλεψης λαθών λογισμικού με κατανομή Poisson). Όπως φαίνεται και στην εικόνα η απόπειρα να χρησιμοποιήσουμε την μέθοδο του γραμμικής παλινδρόμησης (linear regression) (δηλαδή να θεωρήσουμε πως η έξοδος μας μπορεί να είναι της μορφής $\psi = \alpha\chi + \beta$,ενώ στην πραγματικότητα να έπρεπε να εφαρμοστεί μια καμπύλη αρκετά πιο σύνθετη), δημιουργεί το φαινόμενο του underfitting.

Bias:
Το Bias ορίζεται ως E [Y'-Y] δηλαδή η αναμενόμενη τιμή του σφάλματος. Υψηλό bias παρατηρείται όταν έχουμε κάνει υποθέσεις οι οποίες είναι λανθασμένες, όπως για παράδειγμα πως το σετ δεδομένων μας θα μπορούσε να προσομοιωθεί μία απλοϊκή μέθοδος όπως η γραμμική παλινδρόμηση ενώ θα χρειαζόμασταν παραπάνω παραμέτρους.

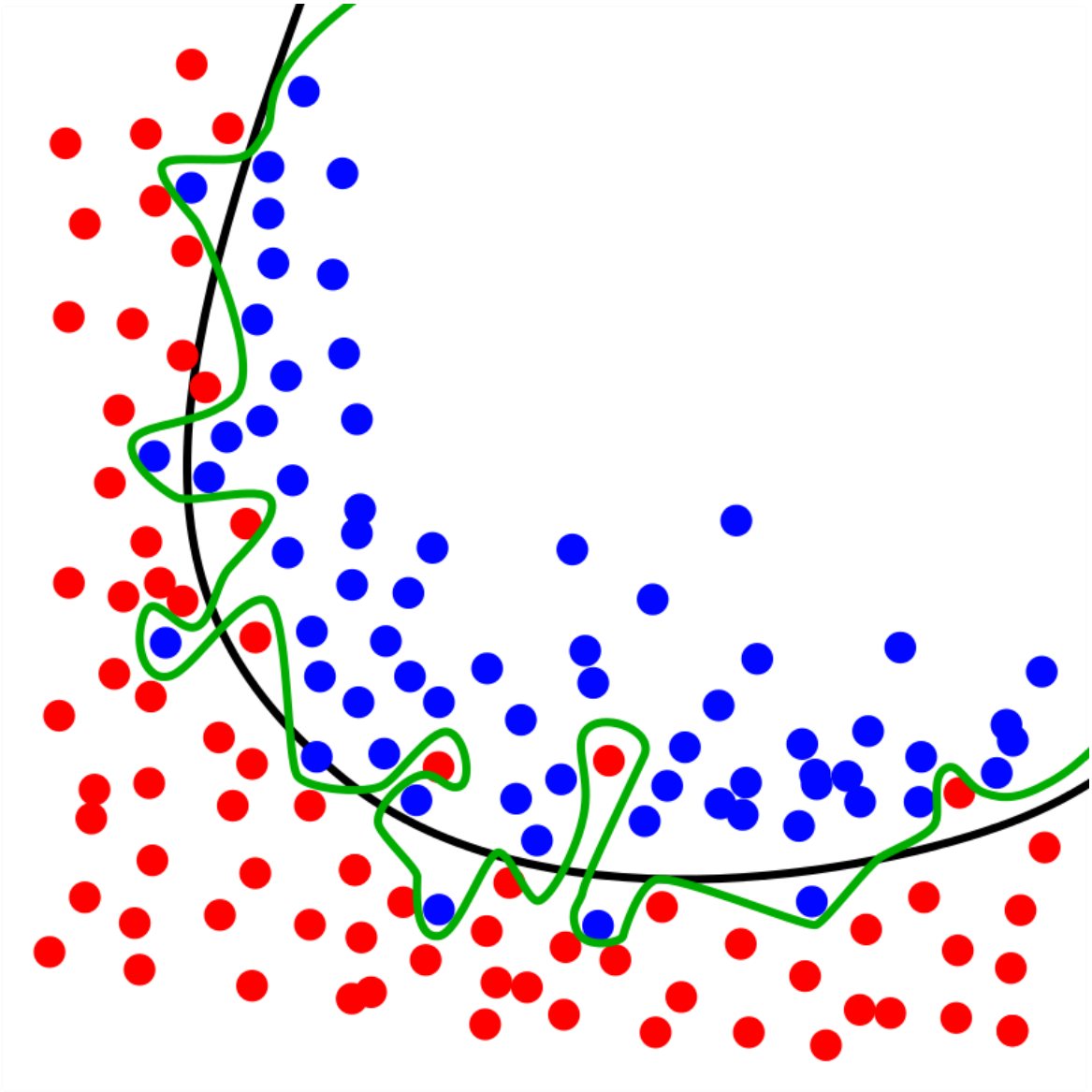
Υψηλό bias -> Underfitting

Διακύμανση (Variance): Η διακύμανση αποτελεί την απόκλιση που θα είχαμε σε νέα δεδομένα. Δηλαδή η υψηλή διακύμανση σημαίνει πως ο εκτιμητής μας έχει ταιριάξει πάνω στα δεδομένα μας, με αποτέλεσμα να μην αποδίδει σε διαφορετικά δεδομένα.

Υψηλή διακύμανση -> Overfitting

Είναι αυτονόητο πλέον πως αυτό που αναζητούμε σε έναν αλγόριθμο είναι η ελαχιστοποίηση των δύο αυτών μεγεθών. Ειδικότερα αναζητούμε τον βέλτιστο αριθμό παραμέτρων που πρέπει να δώσουμε στον αλγόριθμο μας ώστε να εκπαιδευτεί αποφεύγοντας αυτά τα δύο μη θεμιτά φαινόμενα (overfitting, underfitting),

κάτι που φαίνεται και στην παρακάτω εικόνα, η οποία χαρακτηρίζεται από αρκετά υψηλή διακύμανση .



Εικόνα 2 .4 Υψηλή Διακύμανση- Overfitting

Μπορεί μεν η μαύρη γραμμή να μην ταξινομεί 100% ορθά τα δεδομένα εκπαίδευσης, αλλά θα έχει μεγαλύτερη επιτυχία στα νέα δεδομένα σε σχέση με την πράσινη γραμμή η οποία έχει <ταιριάξει> 100% στα δεδομένα μας. Η εικόνα θα ήταν ακόμα πιο εμφανής, αν υπήρχε ένα παράδειγμα με κόκκινο χρώμα σε ένα σημείο πάνω δεξιά στην εικόνα. Στην

περίπτωση αυτή η πράσινη γραμμή θα είχε ακόμα πιο περίεργη συμπεριφορά και θα έδινε σαφώς χειρότερα αποτελέσματα στην διαδικασία αξιολόγησης της. Αυτό το σενάριο θα υπήρχε στην περίπτωση που εμφανιζόταν κάποια εμφανής εξαίρεση στα δεδομένα μας, η οποία όμως πρέπει να μην επηρεάζει εμφανώς τις επιδόσεις του αλγορίθμου μας και να έχει την δυνατότητα από μόνος του να διαπιστώσει το πρόβλημα και να το αποφύγει με την ελάχιστη δυνατή ζημία.

Το πρόβλημα μας για την πρόβλεψη λαθών και συγκεκριμένα η ταξινόμηση της κάθε ημέρας ως ημέρα με σφάλμα ή όχι θεωρώντας το πρόβλημα αυτό πως προσομοιώνεται από Μαρκοβιανή αλυσίδα οφείλει να αποφύγει το φαινόμενο της υψηλής διακύμανσης. Πιο συγκεκριμένα πρέπει να βρεθεί το κατάλληλο πλήθος προηγούμενων ημερών με βάση το οποίο θα προβλέπουμε για την επόμενη (δηλαδή η μνήμη της Μαρκοβιανής αλυσίδας) .

Το ερώτημα που προκύπτει είναι πώς μπορούμε να αποφύγουμε τα δύο παραπάνω προβλήματα στις προσεγγίσεις μας για το πρόβλημα της πρόβλεψης λαθών λογισμικού. Εάν δίναμε την δυνατότητα στον αλγόριθμο μας να εκπαιδεύεται με μεγάλο πλήθος παραμέτρων, τότε θα είχαμε το πρόβλημα της υψηλής διακύμανσης (Overfitting). Πιο συγκεκριμένα, με την προσέγγιση χρονοσειρών μακροβιανής αλυσίδας τάξης m, αν το m ήταν ένα αρκετά μεγάλος αριθμός, τότε θα δινόταν η δυνατότητα να απομνημονεύει ακολουθίες και να <ταιριάζει> πάνω σε αυτές, αντί να εκπαιδεύεται και να δίνει τα κατάλληλα βάρη στις τιμές που θεωρεί πως επηρεάζουν το αποτέλεσμα.

Αντίθετα, κατά την προσέγγιση με τη μορφή Poisson Distribution, θεωρήσαμε πως τα δεδομένα μας ακολουθούν μια απλοϊκή κατανομή και μπορεί να εκφραστούν με μια μοναδική παράμετρο. Ο αλγόριθμος μας προσπαθεί, χωρίς να έχει την δυνατότητα να διαφύγει από την κατανομή που του έχουμε ορίσει, να βρει την βέλτιστη παράμετρο ώστε να προβλέπει, αν οι επόμενες ήμερες είναι χωρίς λάθος ή όχι. Αυτή η προσέγγιση παρουσιάζει το πρόβλημα της υψηλής διακύμανσης, διότι υποθέσαμε πως τα δεδομένα μας ακολουθούν μια πιο απλοϊκή κατανομή από την πραγματική τους κατανομή.

Τέλος, θα πρέπει τόσο να ορίσουμε την κατάλληλη μέθοδο για την επίτευξη των βέλτιστων αποτελεσμάτων αλλά και να προσέξουμε να μη δώσουμε ελευθερία στον αλγόριθμο να μάθει τα δεδομένα και να μην κάνει κάποια πραγματική εκπαίδευση πάνω σε αυτά.

2.6. Γλώσσα προγραμματισμού-Πλατφόρμα

Γνωρίζοντας πλέον τόσο το είδος της μεθόδου που μπορούμε να χρησιμοποιήσουμε όσο και τους πιθανούς κίνδυνους που μπορούμε να συναντήσουμε και πρέπει να αποφύγουμε, μια επιπλέον επιλογή είναι η γλώσσα προγραμματισμού και η πλατφόρμα που θα χρησιμοποιήσουμε.

Η επιλογή θα επηρεάσει το αποτέλεσμα μας καθώς κάθε διαφορετικός συνδυασμός χρησιμοποιεί άλλες μεθόδους εκπαίδευσης (διαφορετικές προσεγγίσεις).

Η γλώσσα προγραμματισμού που χρησιμοποιείται κατά κόρον στους αλγορίθμους μηχανικής μάθησης είναι η python. Η έκδοση που χρησιμοποίησα εγώ για τον κώδικα είναι η python 3 στην πλατφόρμα sklearn.

https: //github. com/scikit-learn/scikit-learn

Scikit-learn (επίσημα scikits. learn) είναι βιβλιοθήκη μηχανικής μάθησης σχεδιασμένη για την Python.
Μπορεί να εφαρμοστεί για κατηγοριοποίηση, παλινδρόμηση και αλγορίθμους ομαδοποίησης (support vector machines, random forests, gradient boosting, k-means and DBSCAN), και μπορεί να λειτουργεί παράλληλα με τη βιβλιοθήκη Numpy, κάτι που υλοποιήθηκε και στον κώδικα του συγκεκριμένου προβλήματος.

Scikit-learn είναι φτιαγμένο σε Python, με κάποιους αλγορίθμους σε Cython για να πετύχουμε καλύτερα αποτελέσματα. Τα Support vector machines έχουν υλοποιηθεί σε Cython. Η logistic regression και linear support vector machines στη βιβλιοθήκη LIBLINEAR [9], η οποία αναπτύχθηκε από το National Taiwan University.

Η πλατφόρμα αυτή μας δίνει την δυνατότητα για εύκολη εκπαίδευση και αξιολόγηση καθώς οι βιβλιοθήκες που υπάρχουν διαθέσιμες δεν καθυστερούν την φάση της εκπαίδευσης και μας επιτρέπουν την εκτέλεση του αλγορίθμου μας σε λιγότερο από ένα λεπτό για οποιαδήποτε προσέγγιση χρησιμοποιήσουμε.

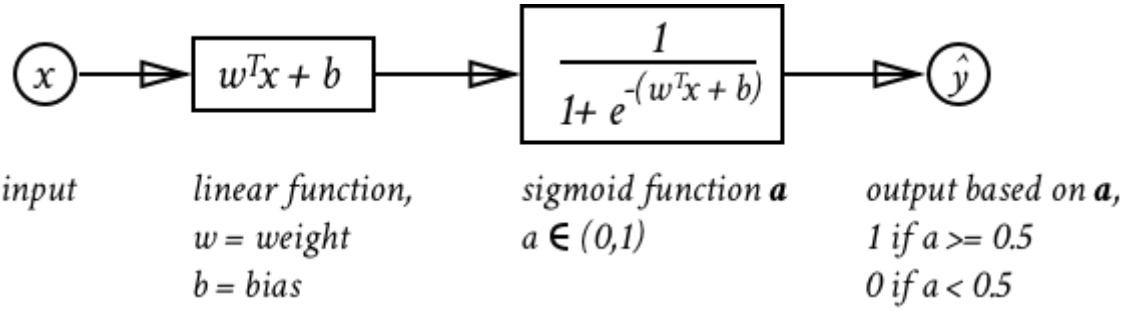
ΚΕΦΑΛΑΙΟ 3 Οι αλγόριθμοι που χρησιμοποιήθηκαν

Έχοντας πλέον ορίσει μια βάση για το πρόβλημα και τους τρόπους αξιολόγησης αυτού, καθώς και την προηγούμενη δουλειά που έχει γίνει πάνω σε αυτό μπορούμε να αναφερθούμε στην καλύτερη δημοσίευση (Prophet Facebook) καθώς και στις δικές μας προσεγγίσεις στο πρόβλημα της πρόβλεψης λαθών λογισμικού χωρίς την γνώση του κώδικα. Τόσο κατά την διάρκεια του πειράματος με βάση τα δικά μας δεδομένα, όσο και με την προσέγγιση της Facebook [33] ο αλγόριθμος που πέτυχε τα καλύτερα αποτελέσματα ήταν η λογιστική παλινδρόμηση(Logistic Regression).

3.1. Λογιστική παλινδρόμηση

Η μέθοδος λογιστική παλινδρόμηση (Logistic Regression) αποτελεί την πιο διαδεδομένη μέθοδο για προβλήματα δυαδικής ταξινόμησης σε θέματα επιβλεπόμενης μάθησης. [23], [29] . Για την μέθοδο αυτή οφείλουμε να έχουμε ορίσει κάποιες παραμέτρους που θεωρούμε πως επηρεάζουν το αποτέλεσμα μας. Πιο συγκεκριμένα στην περίπτωση μας η ερώτηση που πρέπει να απαντηθεί είναι ποιες παράμετροι πρέπει να ληφθούν υπόψιν για την πρόβλεψη αν η επόμενη μέρα θα είναι ημέρα με λάθη ή όχι. Έχοντας ορίσει τις βέλτιστες παραμέτρους, των οποίων οι τιμές είναι είτε πραγματικοί αριθμοί είτε δυαδικές τιμές, ο αλγόριθμος μας υπολογίζει βάρη, μια σταθερά που χρησιμοποιείται ώστε να πετύχουμε καλύτερα αποτελέσματα.

Σε μαθηματική μορφή:



Η τιμή του α που υπάρχει ως έξοδος από την σιγμοειδή συνάρτηση είναι η πιθανότητα η επόμενη ήμερα να αποτελεί ημέρα λάθους. Ο λόγος που το νευρωνικό μας δεν τερματίζει στο σημείο αυτό είναι πως η αρχική ερώτηση είναι αν η επόμενη ήμερα θα εμφανίσει λάθος ή όχι. Για αυτό το λόγο χρησιμοποιείται ένα κατώφλι (συνήθως στο 0. 5) ώστε να στρογγυλοποιεί την πιθανότητα που βρήκαμε στο {0, 1} για να μπορέσουμε να

συγκρίνουμε με την πραγματική τιμή της συνάρτησης που προσπαθούμε να προσεγγίσουμε.

Ο αλγόριθμος μας προσπαθεί να βρει τα βέλτιστα βάρη ελαχιστοποιώντας την συνάρτηση σφάλματος που φαίνεται παρακάτω.

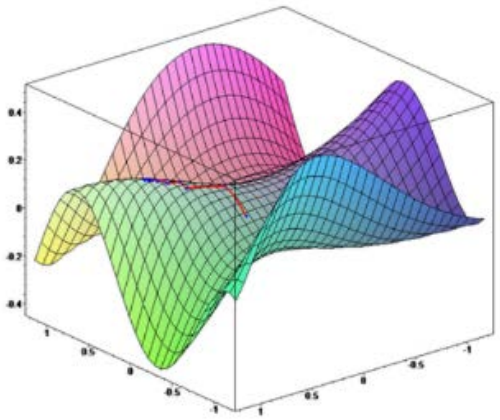
$$J(w) = -\frac{1}{n}\sum_{i=0}^{n-1}\{y^i \log[h(w^t x^i)] + (1-y^i)\log[1-h(w^t x^i)]\}$$

Η διαδικασία εύρεσης των βαρών ονομάζεται φάση εκπαίδευσης (train-phase) και είναι ισοδύναμη με την εύρεση των βαρών που ελαχιστοποιούν την συνάρτηση σφάλματος. Κάτι τέτοιο αποτελεί ένα πλήρως μαθηματικά ορισμένο πρόβλημα το οποίο επιλύεται προσεγγιστικά από τον υπολογιστή με τον κίνδυνο να μην βρεθούν τα βάρη ώστε να πετύχουμε το ολικό ελάχιστο της συνάρτησης αυτής αλλά ένα τοπικό ελάχιστο.

Η ελαχιστοποίηση της συνάρτησης αυτής γίνεται αρκετά πιο απλή διαδικασία, εκμεταλλευόμενοι την κυρτότητα της συνάρτησης αυτής [1]. Το χαρακτηριστικό αυτό μας δίνει την δυνατότητα σε κάθε σημείο εξετάζοντας το πρόσημο της παράγωγου και κινούμενοι αντίθετα σε αυτό να κατευθυνόμαστε προς την σωστή κατεύθυνση κάνοντας κάθε φορά ένα βήμα που εξαρτάται από μια παράμετρο α.

Ένας από τους συνηθέστερους αλγόριθμους εύρεσης του ελαχίστου της συνάρτησης κόστους είναι ο Gradient Descent. Ο αλγόριθμος αυτός ξεκινά από ένα σύνολο τιμών w και επαναληπτικά κινείται προς την εύρεση του ελάχιστου της συνάρτησης κόστους J (w). Αυτό το πετυχαίνουμε κινούμενοι σε κάθε σημείο προς την αντίθετη κατεύθυνση της παράγωγου μέχρι το σημείο που μηδενίζεται. Ο αλγόριθμος ακολουθεί την κατεύθυνση της κλίσης που δημιουργεί η επιφάνεια της συνάρτησης κόστους μέχρι να βρεθεί σε κοιλάδα.

Σχηματικά:



Εικόνα 3.1 Gradient Descent

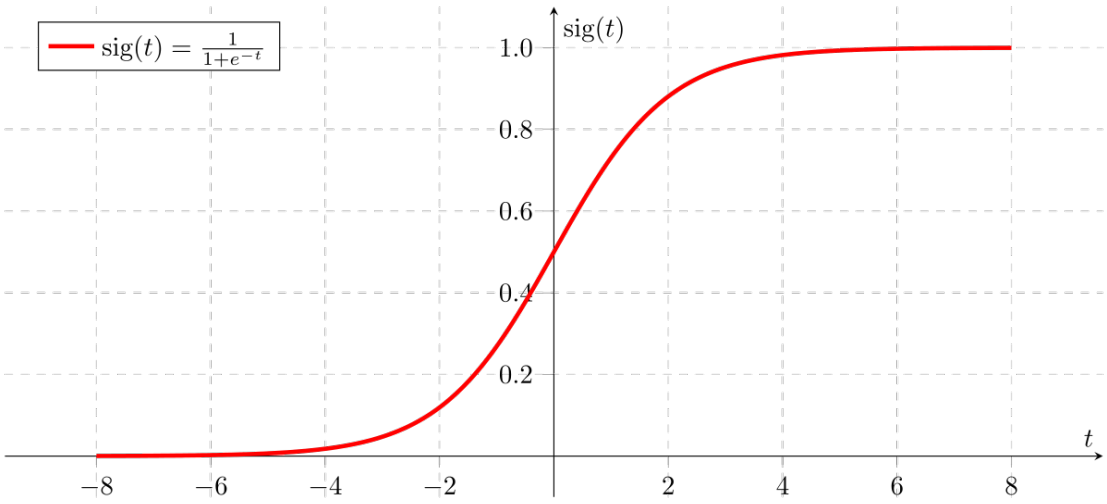
Σε κάθε επανάληψη το βάρος ανανεώνεται με ρυθμό εκμάθησης (learning rate) α.

$$\delta w = -\alpha \nabla_w J(w)$$

Στην διαδικασία αυτή εγκυμονεί ο κίνδυνος να φτάσουμε σε ένα τοπικό ελάχιστο στο οποίο η παράγωγος θα είναι 0 και να μην γνωρίζουμε την κατεύθυνση που πρέπει να κινηθούμε.

Για την αποφυγή του φαινόμενου αυτού εκτελούμε την διαδικασία εκπαίδευσης ξεκινώντας από διαφορετικά σημεία και εξετάζουμε, αν θα καταλήξουμε στο ίδιο σημείο ελάχιστο αυξάνοντας την πιθανότητα να αποτελεί και το ολικό ελάχιστο της συνάρτησης.

Έχοντας πλέον ορίσει τα βέλτιστα βάρη καθώς και τις υπόλοιπες παραμέτρους υπολογίζουμε το άθροισμα που προαναφέραμε και παίρνουμε σαν αποτέλεσμα έναν πραγματικό αριθμό. Ο αριθμός αυτός δίνεται σαν είσοδος σε μια συνάρτηση (sigmoid function) και το αποτέλεσμα αυτής είναι η πιθανότητα που αναζητούμε. Τέλος, η πιθανότητα αυτή στρογγυλοποιείται στο {0, 1} ώστε να είναι η έξοδος του αλγορίθμου μας στην μορφή που αναζητούσαμε εξαρχής.



Εικόνα 3.2 Sigmoid Function

Για να εφαρμόσουμε την λογιστική παλινδρόμηση στο πρόβλημα μας και να καταφέρουμε να προβλέψουμε την πιθανότητα η επόμενη ήμερα είναι χωρίς λάθος ή όχι αρκεί να ορίσουμε τα features που θεωρούμε πως είναι υπεύθυνα για την εμφάνιση ή μη λάθους την επόμενη ήμερα.

Ο πρωταρχικός στόχος για την εφαρμογή της λογιστικής παλινδρόμησης είναι ο σωστός ορισμός των χαρακτηριστικών που θα χρησιμοποιηθούν. Τόσο η εύρεση του κατάλληλου πλήθους που θα χρησιμοποιήσουμε όσο και τι θα συμβολίζει το κάθε ένα από αυτά (ένα χαρακτηριστικό θα μπορούσε να συμβολίζει την περίοδο διακοπών η όχι) (με δυαδική τιμή) αποτελεί ένα ιδιαίτερα σημαντικό ζήτημα. Μια παράμετρος θα μπορούσε να είναι το διάστημα ημερών από την τελευταία μέρα που βρέθηκε σφάλμα. Ένα πρόβλημα που θα μπορούσε να παρουσιαστεί στην περίπτωση αυτή είναι η έλλειψη μεταβλητών που θα μπορούσαν να δρουν συνδυαστικά για το κάθε ένα από τα ανεξάρτητα χαρακτηριστικά. Δηλαδή τα λεγόμενα σταυρωτά χαρακτηριστικά (cross features) που θα αποτελούνταν από το γινόμενο κάποιων χαρακτηριστικών υψωμένα σε κάποια δύναμη που θα αποτελούσε την βέλτιστη επιλογή. Όμως κάτι τέτοιο θα καθιστούσε την εκπαίδευση αρκετά πολύπλοκη διαδικασία και επίσης οι πιθανοί συνδυασμοί των παραμέτρων είναι εκθετικοί σε πλήθος σε σχέση με τον αριθμό αυτών. Σαν αποτέλεσμα η διαδικασία εκπαίδευσης θα απαιτούσε αρκετό χρόνο, χωρίς να είμαστε βέβαιοι πως οι παράμετροι που επιλέξαμε είναι οι βέλτιστες και πως δεν υπάρχει κάποιες άλλες που να μην έχουμε προσθέσει ή κάποιες διασταυρωμένες παραμέτρους (cross-features) που θα βελτίωναν αισθητά την απόδοση του αλγορίθμου μας.

Παρατηρώντας λίγο πιο προσεκτικά τα δεδομένα παρατηρούμε πως η εμφάνιση των λαθών τις επόμενες ήμερες εξαρτάται κατά κύριο λόγο από το εποχικότητα, δηλαδή από

τη εμφάνιση η όχι λαθών τις προηγούμενες ήμερες. Ας προχωρήσουμε σε ένα παράδειγμα για την καλύτερη κατανόηση της προσέγγισης αυτής.

Ας υποθέσουμε πως αναζητούμε έναν τρόπο να προβλέψουμε, αν θα εμφανιστεί αύριο ένα σφάλμα με συγκεκριμένα χαρακτηριστικά με την μέθοδο της λογιστικής παλινδρόμησης. Για την εφαρμογή της μεθόδου αυτής χρειάζεται αρχικά να ορίσουμε τις παραμέτρους με τις οποίες θα εκπαιδεύσουμε τον αλγόριθμο μας μέσω των δεδομένων που διαθέτουμε και στη συνεχεία θα αξιολογήσουμε τα δεδομένα μας με βάση το κομμάτι αξιολόγησης. Το ποσοστό των δεδομένων που θα αφορούν τη φάση εκπαίδευσης κα τη φάση αξιολόγησης είναι ανεξάρτητο από τον τρόπο που θα ορίσουμε τις παραμέτρους της μεθόδου αυτής, αν και στη συνεχεία θα ορίσουμε το βέλτιστο ποσοστό που πρέπει να χρησιμοποιήσουμε για εκπαίδευση.

Ως χαρακτηριστικά θα χρησιμοποιήσουμε τις m τελευταίες μέρες. Δηλαδή σαν κάθε παράδειγμα του αλγορίθμου μας θα χρησιμοποιείται οι τιμές Xi i=1... m και σαν Yi η τιμή την επόμενη ημέρα (δυαδικές τιμές). Πιο συνοπτικά θεωρούμε πως η έξοδος μας την επόμενη ημέρα εξαρτάται από τις m προηγούμενες ημέρες.

Μια πιο προσεκτική σκέψη μας δείχνει πως κάτι τέτοιο δίνει την δυνατότητα στον αλγόριθμο να αντιληφθεί την εποχικότητα καθώς δίνοντας τα κατάλληλα βάρη στις τιμές Xi μας δείχνει ποιες μέρες είναι αυτές που επηρεάζουν πιο πολύ το αποτέλεσμα μας. Πριν προχωρήσουμε σε περαιτέρω ανάλυση των δεδομένων μας και της μεθόδου αυτής θα ασχοληθούμε με ένα ερώτημα που προκύπτει. Ποιο είναι το κατάλληλο m και αν αυτό εξαρτάται από το ποσοστό των δεδομένων που επιλέγουμε για train του αλγορίθμου μας.

Αν είχαμε αρκετά μεγάλο πλήθος δεδομένων τότε οι μετρικές μας θα έπρεπε να είναι αύξουσα συνάρτηση του m καθώς με βάση την εκπαίδευση του αλγορίθμου μας θα μπορούσε να δώσει μηδενικό βάρος σε ήμερες που δεν θα επηρέαζαν το άθροισμα μας. Ειδικότερα ένας αλγόριθμος εκπαιδευμένος με m=50 θα απέδιδε πάντα καλύτερα από ένα με m=X m στο διάστημα [1, X] αφού θα είχε την δυνατότητα να μηδενίσει τα βάρη για τις ημέρες στο διάστημα [X, m] και να δώσει ίδια βάρη για τις ήμερες [1, X].

Με λίγα λόγια οι δυνατότητες ενός αλγορίθμου με m μεγαλύτερο από κάποιον άλλο θα ήταν υπερσύνολο σε σχέση με έναν που θα χρησιμοποιούσε μικρότερο m. Βεβαία αυτή η θεώρηση θα ίσχυε όταν το πλήθος των δεδομένων ήταν αρκετά μεγάλο ώστε να μπορούν να βρεθούν τα βέλτιστα βάρη σε σχέση με το m.

Όμως, όπως γίνεται αντιληπτό οι 6000 μέρες που αποτελούν τα δεδομένα μας, δηλαδή 6000 τιμές από {0, 1} δεν είναι και τόσο μεγάλο πλήθος για να ισχύει η μονοτονία σε συνάρτηση με το m.

Άρα προκύπτει το ερώτημα για το βέλτιστο m. Δοκιμάζοντας διαφορετικούς συνδυασμούς μεταξύ m και ποσοστό δεδομένων για εκπαίδευση του αλγορίθμου βλέπουμε πως η ιδανική τιμή του m κυμαίνεται μεταξύ 25-35 ανάλογα με το πρόβλημα που έχουμε να προσεγγίσουμε. Επιλέγουμε να κρατήσουμε m=30 για όλα τα υποπροβλήματα που προσεγγίζουμε. Επίσης η φάση εκπαίδευσης ασχολείται με το 80% των δεδομένων διότι αυτή η τιμή κατά μέσο όρο μας δίνει τα βέλτιστα αποτελέσματα.

Άρα για την προσέγγιση αυτή παίρνουμε ως κάθε στιγμιότυπο τις 30 τελευταίες μέρες, τις πολλαπλασιάζουμε κάθε φορά με τα αντίστοιχα βάρη και συγκρίνουμε το αποτέλεσμα με το κατώφλι που έχει ορίσει ο αλγόριθμος μας. Ανάλογα με το αποτέλεσμα αποφαίνεται για 0 ή 1. Αυτή η διαδικασία εκτελείται στο στάδιο αξιολόγησης και υπολογίζουμε τις 4 τιμές των TP, TN, FP,FN.

Στη συνεχεία έχοντας βρει αυτές τις τιμές υπολογίζουμε τις μετρικές που έχουμε προαναφέρει και συγκρίνουμε τον αλγόριθμο αυτόν είτε με άλλες προσεγγίσεις είτε με άλλη επιλογή παραμέτρων. Η συγκεκριμένη προσέγγιση με λογιστική παλινδρόμηση, m=30, ποσοστό εκπαίδευσης= 80% των δεδομένων αποτελεί κατά μέσο όρο την καλύτερη προσέγγιση μας στο πρόβλημα της πρόβλεψης λαθών λογισμικού όπως φαίνεται και από τις γραφικές παραστάσεις που παρουσιάζονται σε επόμενο κεφάλαιο.

Ένα ακόμα πλεονέκτημα της προσέγγισης αυτής είναι πως κάποια αλλαγή μιας ημέρας (που μπορεί να θεωρείται ασήμαντη) μεταξύ των τριάντα ήμερων δεν θα επηρεάσει το αποτέλεσμα, καθώς το βάρος της θα είναι αρκετά μικρό είτε ακόμα και μηδέν με αποτέλεσμα να θεωρείται ευσταθής προσέγγιση. Επίσης δεν χρειάζεται να ορίσουμε εμείς ποιες μέρες θεωρούνται πιο <σημαντικές> καθώς η εύρεση των βέλτιστων βαρών είναι κάτι που γίνεται αποκλειστικά κατά τη διάρκεια της εκπαίδευσης.

Ένα γενικό σχόλιο με βάση το αποτέλεσμα του αλγορίθμου μας είναι η παρατήρηση πως έχουμε βάρη τόσο αρνητικά όσο και θετικά, κάτι που σημαίνει πως η εμφάνιση ενός σφάλματος μια συγκεκριμένη ήμερα μπορεί να συμβάλλει αρνητικά και να μειώνει την πιθανότητα εμφάνισης ενός άλλου την επόμενη ημέρα.

Με βάση τα παραπάνω εφαρμόσαμε την μέθοδο της λογιστικής παλινδρόμησης στο πρόβλημα της πρόβλεψης λαθών λογισμικού πετυχαίνοντας αρκετά ικανοποιητικά αποτελέσματα τα οποία προβάλλονται σε γραφικές παραστάσεις στο κεφάλαιο 4.

3.2. Γραμμική παλινδρόμηση

Μια άλλη μέθοδος που θα μπορούσε να χρησιμοποιηθεί για το πρόβλημα πρόβλεψης λαθών λογισμικού είναι η μέθοδος της γραμμικής [28] . Η μέθοδος αυτή έχει αρκετά κοινά με την λογιστική παλινδρόμηση αλλά και μερικές διαφορές.

Και στην περίπτωση αυτήν χρησιμοποιούμε ένα νευρωνικό δίκτυο, μια διαφορετική συνάρτηση ενεργοποίησης (activation function) καθώς και άλλη συνάρτηση σφάλματος (error function).

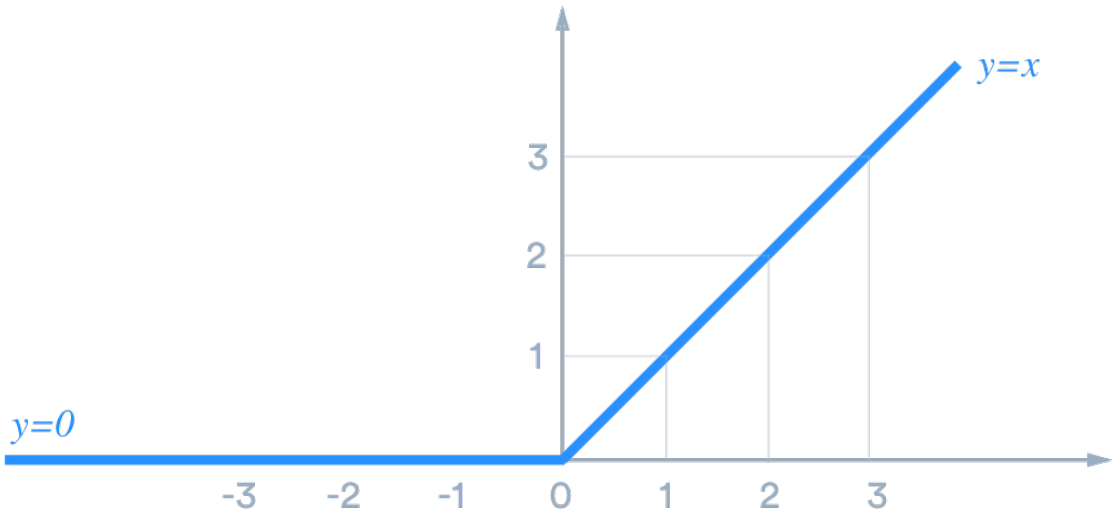
Και στην προσέγγιση αυτή θεωρήσαμε πως τα χαρακτηριστικά μας είναι οι τελευταίες μέρες. Προσπαθούμε να βρούμε τα αντίστοιχα βάρη χρησιμοποιώντας την παρακάτω συνάρτηση σφάλματος (mean square error):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\mathrm{Y_i} - Y_j\right)^2$$

Ειδικότερα προσπαθούμε να μειώσουμε την απόκλιση μας από την πραγματική έξοδο της επόμενης ημέρας {0, 1}.

Το πρώτο στάδιο είναι ίδιο με την λογιστική παλινδρόμηση και στη συνεχεία χρησιμοποιούμε ως συνάρτηση ενεργοποίησης την ReLu. Θα μπορούσαμε να χρησιμοποιήσουμε και κάποιες παραλλαγές αυτής εντάσσοντας παραμέτρους [13].

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

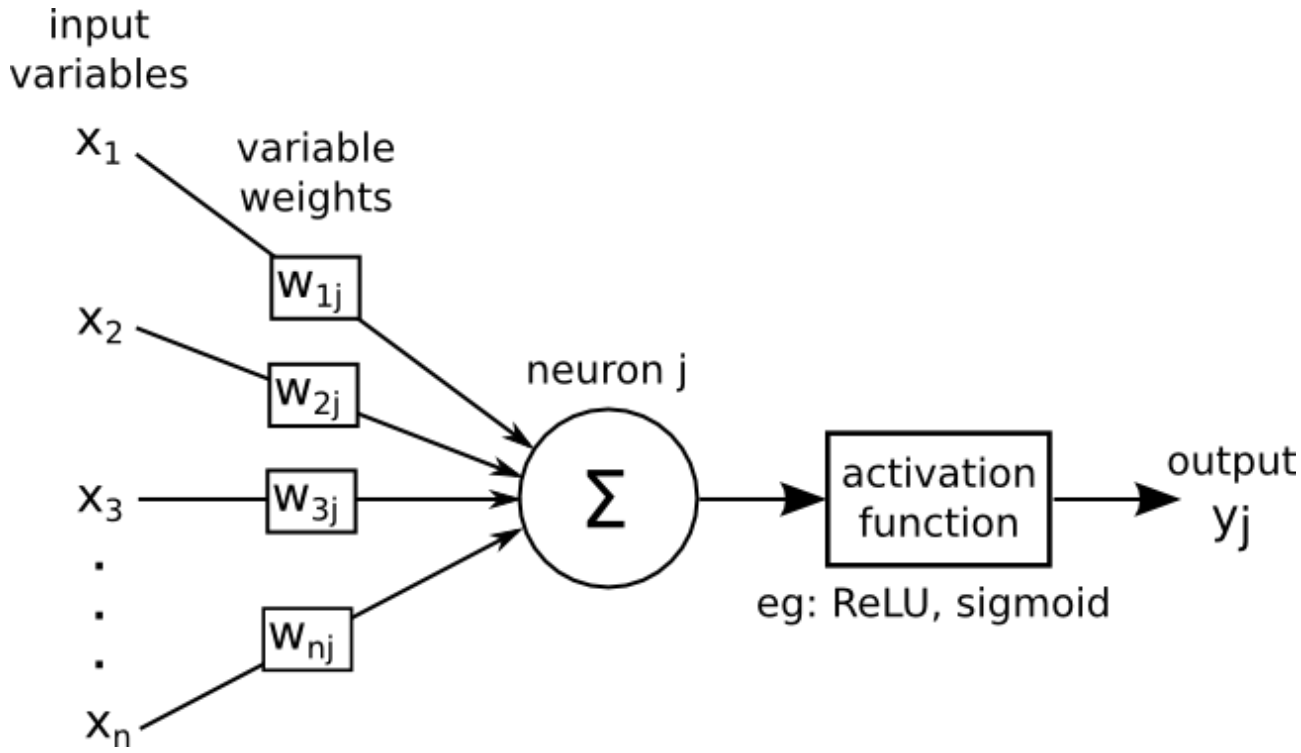


Εικόνα 3.3 Relu Συνάρτηση

Με αυτόν τον τρόπο υπολογίζουμε την πιθανότητα η επόμενη ημέρα να εμφανίσει λάθος.

Πιο συγκεκριμένα, στην περίπτωση μας αφού έχουμε κάνει την υπόθεση πως η επόμενη μέρα προκύπτει από μια μαρκοβιανή αλυσίδα, που καθορίζεται πλήρως από τις τελευταίες m ήμερες, έχουμε ορίσει 30 μεταβλητές (οι τελευταίες 30 ημέρες) στις οποίες ο αλγόριθμος μας βρίσκει τα βέλτιστα βάρη για την ελαχιστοποίηση της συνάρτησης σφάλματος και στη συνεχεία για κάθε νέο παράδειγμα υπολογίζουμε την πιθανότητα η επόμενη ήμερα να είναι χωρίς σφάλμα με βάση την παραπάνω συνάρτηση.

Η βασική διαφορά με την λογιστική παλινδρόμηση που έχουμε ήδη ορίσει είναι η συνάρτηση ενεργοποίησης, καθώς και η συνάρτηση σφάλματος που ελαχιστοποιούμε στις δύο περιπτώσεις. Η τοπολογία του νευρωνικού δικτύου είναι ίδια και φαίνεται και σχηματικά στην παρακάτω εικόνα:



Εικόνα 3.4 Νευρωνικό δίκτυο

3.3. Δέντρα απόφασης

Αρχικά θα αναφέρουμε τι είναι τα δέντρα απόφασης (decision trees) ,τα πλεονεκτήματα τους όπως και τα μειονεκτήματα τους καθώς και για ποιες χρήσεις είναι ιδανικά. Στη

συνεχεία θα δείξουμε πώς τα εφαρμόσαμε στο πρόβλημα μας της πρόβλεψης λαθών λογισμικού.

Τα δέντρα απόφασης χρησιμοποιούνται όταν έχουμε ένα πλήθος από χαρακτηριστικά (είτε πραγματικούς αριθμούς, είτε λογικές τιμές (ναι ,όχι), είτε ακεραίους) και με βάση αυτά προσπαθούμε να αποφανθούμε για την τιμή μιας άλλης μεταβλητής (στο παράδειγμα μας 0-1).

Επιλέγουμε ένα από τα χαρακτηριστικά που επηρεάζουν την πρόβλεψη μας με μια συγκεκριμένη μέθοδο που θα εξηγήσουμε στη συνεχεία και το βάζουμε στην κορυφή του δέντρου. Στη συνεχεία εφαρμόζοντας αναδρομικά την ίδια μέθοδο επιλέγουμε παραμέτρους για να επικοινωνεί με την ρίζα. Συνεχίζοντας αναδρομικά αυτήν την διαδικασία έχουμε κατασκευάσει ένα δέντρο απόφασης. Κάθε ενδιάμεσος κόμβος μας πηγαίνει ένα επίπεδο πιο χαμηλά ανάλογα με την τιμή του αντίστοιχου χαρακτηριστικό που εξετάζεται την συγκεκριμένη χρονική στιγμή. Έτσι εξετάζοντας κάθε φορά όλα τις παραμέτρους με την σειρά που έχουν οριστεί καταλήγουμε σε ένα κόμβο-φύλλο (leaf node). Ο κόμβος αυτός είναι υπεύθυνος για να αποφασίσει, με βάση τη διαδικασία εκπαίδευσης, αν η επόμενη μέρα είναι χωρίς λάθος ή όχι.

Ο χωρισμός των χαρακτηριστικών μπορεί σε μερικές περιπτώσεις να σταματήσει χωρίς να τα εξετάσουμε όλα τα, αν ο αλγόριθμος αποφανθεί πως ένας συγκεκριμένος συνδυασμός αυτών είναι ικανός να μας δώσει το τελικό αποτέλεσμα ανεξάρτητα από τις τιμές των υπολοίπων.

Gini impurity:

Κάθε φορά που επιλέγουμε ένα χαρακτηριστικό για να μπει στην ανώτερη διαθέσιμη θέση υπολογίζουμε το «gini impurity» του κάθε κόμβου-φύλλο.

Ως πιθανότητες που φαίνονται και στον παρακάτω τύπο ορίζουμε το πηλίκο των σωστά ταξινομημένων προς το συνολικό πλήθος που ανατεθήκαν στο συγκεκριμένο φύλλο. Στην περίπτωση μας το J (οπού είναι το πλήθος των πιθανών εξόδων) είναι προφανώς ίσο με δύο.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i\,(1 - p_i) = \sum_{i=1}^{J} (p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2$$

Στη συνεχεία υπολογίζουμε το βεβαρυμένο μέσο (weighted average) μεταξύ των «gini impurity» κάθε φύλλου για να βρούμε το συνολικό. Συγκρίνοντας τις τιμές αυτές για τις διάφορες επιλογές που κάναμε για ανώτερο φύλλο κρατάμε αυτήν με την μεγαλύτερη τιμή και συνεχίζομε αναδρομικά για να βρούμε και τις υπόλοιπες.

Έχοντας κατασκευάσει, πλέον, το δέντρο απόφασης ακολουθούμε την σειρά που έχει οριστεί εξετάζοντας ξεχωριστά κάθε ένα από τα m χαρακτηριστικά (που έχουμε ως είσοδο για να βρούμε, αν η επόμενη ήμερα θα εμφανίσει λάθος ή όχι). Ακολουθώντας την πορεία του δέντρου καταλήγουμε σε ένα κόμβο-φύλλο το οποίο μας δείχνει και την απόφαση {0, 1}.

Η πολυπλοκότητα κατασκευής ενός βέλτιστου δέντρου απόφασης είναι αρκετά μεγάλη (NP-complete πρόβλημα). Ο λόγος που συμβαίνει αυτό είναι πως πρέπει να εξετάζουμε τους n! συνδυασμούς σε κάθε επίπεδο.

Δηλαδή θέλουμε το άθροισμα O (1! +2!... +n!)

$$\sum_{k=0}^{n} k! = \frac{i\pi}{e} + \frac{Ei(1)}{e} - \frac{(-1)^n \Gamma[n+2]\Gamma[-n-1,-1]}{e}$$

Η python χρησιμοποιεί τεχνικές βελτιστοποίησης και προσεγγιστικούς αλγορίθμους ώστε να μην επιλύσει ένα NP-complete πρόβλημα, διότι θα ήταν κάτι που θα καθυστερούσε ιδιαίτερα την εκτέλεση του αλγορίθμου μας, χωρίς να δοκιμάζει όλους τους δυνατούς συνδυασμούς για κάθε επίπεδο.
Έχοντας κατασκευάσει όμως το δέντρο απόφασης η συνεχεία είναι αρκετά απλή καθώς μια διάσχιση του θα έχει πολυπλοκότητα όσο το ύψος του δηλαδή O (m). Έτσι για κάθε νέο παράδειγμα η εκτέλεση του αλγορίθμου μας θα είναι αρκετά γρήγορη.

Ένα βασικό μειονέκτημα των δέντρων απόφασης είναι πως μια αλλαγή σε ένα χαρακτηριστικό μπορεί να μας δώσει τελείως διαφορετικό υπόδεντρο και τελικά διαφορετικό αποτέλεσμα. Κάτι τέτοιο καθιστά τα δέντρα απόφασης αρκετά ασταθή.

Ένα πλεονέκτημα τους είναι πως μπορούν να διαχειριστούν μεγάλο όγκο δεδομένων και πως δέχονται χαρακτηριστικά που δεν είναι αποκλειστικά δυαδικά (κάτι το οποίο δεν μας άφορα στο πρόβλημα μας αφού όλες οι τιμές μας είναι αποκλειστικά {0, 1}. Ένας επιπλέον τρόπος βελτιστοποίησης των δέντρων απόφασης είναι η αντικατάσταση κάθε κόμβου με μια άλλη τεχνική απόφασης (random forest) ώστε να αποδίδει καλύτερα. Κάτι τέτοιο όμως θα καθιστούσε τον αλγόριθμο μας με αυξημένη πολυπλοκότητα και πιο αργό.

Επιπλέον, θα μπορούσε να γίνει και ανθρώπινη παρέμβαση στην κατασκευή του δέντρου ώστε να έχει προοριστεί μια συγκεκριμένη σειρά των χαρακτηριστικά, έχοντας κατανοήσει τα δεδομένα μας, για να βάλουμε το πιο σημαντικό από αυτά σαν κόμβο ρίζα ή ακόμα και να ορίσουμε την ακριβή σειρά των χαρακτηριστικά ώστε να μην πρέπει να βρει ο

αλγόριθμος την σειρά που θα μπουν στο δέντρο καθιστώντας την εκπαίδευση αρκετά πιο γρήγορη.

Έχοντας κατασκευάσει το δέντρο απόφασης το αξιολογούμε με βάση τις μετρικές που έχουμε προαναφέρει και συγκρίνουμε την μέθοδο αυτή με τις υπόλοιπες. Πιο συγκεκριμένα στο παράδειγμα μας στην ρίζα του δέντρου θα βρισκόταν η ημέρα που θεωρήθηκε πιο σημαντική από τον αλγόριθμο μας και θα εξετάζαμε δυο υπόδεντρα ανάλογα με την τιμή της ημέρας αυτής στο στιγμιότυπο που αναζητούμε {0, 1}. Στη συνεχεία αναδρομικά ο αλγόριθμος θα συνέχιζε σε ένα από τα δύο αυτά υπόδεντρα την ίδια διαδικασία μέχρι να καταλήξουμε σε ένα φύλλο του δέντρου. Στο φύλλο αυτό θα γινόταν η τελική πρόβλεψη για την επόμενη μέρα με βάση την διαδικασία που προαναφέραμε και αυτή θα ήταν η εκτίμηση μας για την επόμενη ήμερα.

3.4. Naive Bayes ταξινομητής

Μια άλλη μέθοδος που χρησιμοποιήσαμε με αντίστοιχο τρόπο με τις παραπάνω προσεγγίσεις (δηλαδή με την υπόθεση πως το πρόβλημα μας ακολουθεί μια μαρκοβιανή αλυσίδα με μνήμη τάξης m) είναι με την χρήση του Naive Bayes Classifier[16], [27] [31].

Η μέθοδος αυτή χρησιμοποιήθηκε για την κατηγοριοποίηση στην περίπτωση της πρόβλεψης λαθών λογισμικού. Με την μέθοδο αυτή προβλέπουμε, αν η επόμενη ήμερα θα περιέχει σφάλμα ή όχι. Για την μέθοδο αυτή γίνεται μια βασική υπόθεση. Η ανεξαρτησία μεταξύ των παραμέτρων, δεδομένου πως γνωρίζουμε την κλάση. Δηλαδή η πιθανότητα μία παράμετρος να έχει μια συγκεκριμένη τιμή δεδομένου της κλάσης που εντάσσεται το instance στο οποίο ανήκει δεν επηρεάζεται από τις υπόλοιπες τιμές των χαρακτηριστικών για το ίδιο παράδειγμα.

Σε μαθηματική μορφή:

P (x | C)=P(x | C,y) οπού x, y χαρακτηριστικά του προβλήματος μας.

Ειδικότερα στο πρόβλημα μας, θεωρώντας πως έχουμε μακροβιανή αλυσίδα με μνήμη τάξης m, γνωρίζοντας την τιμή της ημέρες m+1 θα μπορούσαμε να υπολογίσουμε τις πιθανότητες για κάθε παράμετρο να παίρνει κάθε μια από τις τιμές {0, 1}.

Έχοντας ως δεδομένο το πλήθος των παραμέτρων υπολογίζουμε την πιθανότητα

$$P(C_k|x_1, \dots . x_n)$$

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)}$$

Όπου x θεωρούμε το διάνυσμα των παραμέτρων.

$$x = (x_1, \ldots . x_n)$$

Η υπόθεση που έχουμε κάνει πως κάθε ημέρα είναι ανεξάρτητη από την προηγούμενη ημέρα και πως η κάθε μια από τις n μέρες μπορεί να προσδιοριστεί πλήρως από την τελική κατάσταση Ck φαίνεται παρακάτω:

$$p(x_i|x_{i_{+1}}, \ldots x_n, C_k = p(x_i|C_k)$$

Η υπόθεση πως η κατανομή αυτή μπορεί να ακολουθεί την παραπάνω ιδιότητα είναι εν μέρει λανθασμένη. Εάν ίσχυε η υπόθεση αυτή, τότε θα μπορούσαμε να προσδιορίσουμε την πιθανότητα κάθε ημέρας γνωρίζοντας μόνο μια ημέρα στο μέλλον. Κάτι τέτοιο δεν ισχύει, διότι θα είχαμε καταπατήσει την ιδιότητα της μαρκοβιανής αλυσίδας τάξης m που ακολουθούν τα δεδομένα μας, δηλαδή πως κάθε ημέρα προσδιορίζεται πλήρως από τις m τελευταίες.

Έχοντας κάνει την υπόθεση πως ο Naive Bayes ταξινομητής είναι ιδανικός για την πρόβλεψη μας προσπαθούμε να υπολογίσουμε την πιθανότητα η επόμενη ημέρα να εντάσσεται σε κάθε μια από τις κλάσεις που διαθέτουμε {0, 1} και αποφαινόμαστε πως η θα είναι αυτή στην οποία μεγιστοποιείται η πιθανότητα να ανήκει σε αυτήν.

$$y = argmax_{k\,\{1,2,\ldots,K)} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Τα αποτελέσματα και αυτής της προσέγγισης φαίνονται στο κεφάλαιο 4.

3.5. Poisson προσέγγιση

Η πρώτη προσέγγιση που έγινε αφορούσε την χρήση μιας πιθανοτικής κατανομής για την πρόβλεψη σφαλμάτων λογισμικού. Θεωρήσαμε πως η κατανομή των σφαλμάτων ακολουθεί την Poisson κατανομή.

Κατανομή Poisson:

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Όπου λ η παράμετρος της κατανομής.

Για να προσεγγισθεί μια κατανομή από Poisson Distribution και να μπορεί να θεωρηθεί ορθή η απόφαση αυτή θα πρέπει να πληροί κάποιες προϋποθέσεις.

Αρχικά μια βασική προϋπόθεση είναι να μπορεί να προσεγγιστεί η κατανομή μας από κατανομή με μόνο μια παράμετρο (λ στην περίπτωση της Poisson). Αν κάνουμε αυτή την υπόθεση, τότε το επόμενο ερώτημα που προκύπτει είναι ποιο είναι το κατάλληλο λ για να χρησιμοποιήσουμε και πώς θα το επιλέξουμε αυτό.

Ένας τρόπος να επιλέξουμε το ιδανικό λ θα ήταν να θεωρήσουμε ως λ το μέσο πλήθος των λαθών που εμφανίστηκαν ανά μέρα μέχρι την ημέρα που εξετάζουμε. Ένας άλλος τρόπος θα ήταν να παίρναμε ως λ τον βεβαρυμένο μέσο των τελευταίων μηνών ή εβδομάδων. Όμως εκεί το ερώτημα που προκύπτει είναι με ποιό τρόπο θα βρίσκαμε τα ιδανικά βάρη και η διαδικασία εκπαίδευσης θα ήταν αρκετά πιο δύσκολη και θα έμοιαζε αρκετά με την διαδικασία εκπαίδευσης της λογιστικής παλινδρόμησης καθώς θα αναζητούσαμε βάρη που θα ελαχιστοποιούσαν την συνάρτηση σφάλματος.

Ακόμα και αν είχαμε βρει με κάποιο τρόπο από αυτούς που προαναφέραμε ένα ιδανικό λ ή ακόμα ορθότερα μια συνάρτηση λ (t) μέσω της συνάρτησης της Poisson στη συνέχεια

προκύπτει ένα ακόμα ερώτημα. Μπορούμε είτε να στρογγυλοποιήσουμε μέσω ενός κατωφλίου την πιθανότητα που βρήκαμε στο {0, 1} είτε να εκτελέσουμε ένα πείραμα τύχης με πιθανότητα επιτυχίας την πιθανότητα που ορίσαμε προηγουμένως. Αν ακολουθήσουμε την παραπάνω τεχνική και θεωρήσουμε ένα σταθερό κατώφλι το 50% τότε θα εμφανιστεί το παρακάτω πρόβλημα: Αν η πιθανότητα εμφάνισης λάθους με συγκεκριμένα χαρακτηριστικά κυμαίνεται στο διάστημα [10%, 40%] ανάλογα με διάφορα κριτήρια ο αλγόριθμος μας θα αποφαινόταν πάντα μηδέν (δηλαδή μη εμφάνιση λάθους).

Αν από την άλλη δώσουμε την δυνατότητα στον αλγόριθμο να βρει αυτός κατώφλι, τότε η προσέγγιση μας γίνεται μια προσέγγιση με λογιστικής παλινδρόμησης έχοντας ορίσει διαφορετικές παραμέτρους. Η μονή διαφορά θα αποτελεί η συνάρτηση ενεργοποίησης καθώς δεν θα είναι η σιγμοειδής.

Όποτε για να μην προσομοιώσουμε την λογιστική παλινδρόμηση και να αναδείξουμε την διάφορα των δύο προσεγγίσεων θα υλοποιήσουμε απλοϊκά την Poisson θεωρώντας ως λ το μέσο όρο των σφαλμάτων μέχρι τώρα. Στην συνεχεία μέσω της κατανομής Poisson υπολογίζουμε την πιθανότητα και εκτελούμε πείραμα τύχης με την παραπάνω πιθανότητα επιτυχίας.
Έχοντας ορίσει το λ υπάρχουν ακόμα κάποιες επιπλέον προϋποθέσεις που πρέπει να ικανοποιηθούν. Αρχικά η ανεξαρτησία γεγονότων. Δηλαδή έχοντας ορίσει το βέλτιστο λ πρέπει να εξακριβώσουμε πως η εμφάνιση ή όχι ενός σφάλματος μια μέρα δεν επηρεάζει το αποτέλεσμα της επόμενης. Αυτό αποτελεί την ιδιότητα της έλλειψης μνήμης και είναι απαραίτητο για να αποδειχθεί πως η προσέγγιση της Poisson θα έχει θεμιτά αποτελέσματα και θα μπορεί να χαρακτηριστεί ως μια αξιόλογη προσέγγιση.

Κάτι τέτοιο όμως στην περίπτωση μας δεν ισχύει, αφού, όπως εξηγήσαμε και παραπάνω, η βέλτιστη λύση που βρέθηκε ήταν βασισμένη στην υπόθεση πως η εμφάνιση ή όχι λαθών τις επόμενες ήμερες εξαρτάται από τις m προηγούμενες ήμερες και από το αποτέλεσμα αυτών. Σαν συνεπεία αυτού μια απαραίτητη προϋπόθεση για την Poisson προσέγγιση δεν καλύπτεται, με αποτέλεσμα να υπάρχει η υπόνοια πως δεν αποτελεί την βέλτιστη προσέγγιση στο πρόβλημα μας.

Επίσης ένα ακόμα πρόβλημα που υπάρχει είναι πως η «Poisson Regression» είναι μια μέθοδος που εφαρμόζεται όταν αναζητούμε θετικούς ακέραιος αριθμούς. Στην περίπτωση μας όμως αναζητούμε {0, 1} άρα οποιαδήποτε τιμή >= 1 την θεωρούμε 1.

Τα παραπάνω μας οδηγούν στο συμπέρασμα πως ορίζοντας το λ όπως προαναφέραμε δεν θα έχουμε ιδιαίτερα καλά αποτελέσματα στις μετρικές μας, κάτι που φαίνεται και στην συνέχεια με τις γραφικές παραστάσεις.

Άλλοι τρόποι να εφαρμόζαμε την Poisson θα ήταν αφού, είχαμε μελετήσει τα δεδομένα,

να δημιουργούσαμε οικογένειες Poisson ανάλογα με την ημέρα που αναζητούμε την εμφάνιση ή μη ενός σφάλματος. Κάτι τέτοιο θα είχε καλύτερα αποτελέσματα, διότι παρατηρώντας τα δεδομένα κατανοούμε πως οι Κυριακές είναι αρκετά πιο συχνά ημέρες που δεν εμφανίζονται σφάλματα μιας και οι προγραμματιστές δεν εργάζονται ή εργάζονται λιγότερο την ήμερα αυτήν. Γενικότερα η μέθοδος αυτή θα μπορούσε να εφαρμοστεί με αρκετούς τρόπους αλλά επιλέγουμε την απλοϊκή προσέγγιση που θεωρείται και η πιο απλή στην υλοποίηση για να την αξιολογήσουμε με βάση τις μετρικές.

Η υλοποίηση αυτή μοιάζει αρκετά με Bernoulli κατανομής καθώς κάθε μέρα εκτελούμε ένα πείραμα τύχης με μια συγκεκριμένη πιθανότητα και με βάση το αποτέλεσμα αυτού αποφαινόμαστε για το αποτέλεσμα {0, 1}.

Στην περίπτωση της Poisson δεν έχει νόημα να υπολογίσουμε την μετρική της ROC ,διότι το λ δεν αλλάζει ιδιαίτερα κάθε ημέρα με αποτέλεσμα η πιθανότητα με την μέθοδο αυτήν να μην τροποποιείται σε μεγάλο βαθμό , ώστε να έχει νόημα να ορίσουμε ένα κατώφλι για να κάνουμε την στρογγυλοποίηση. Κάθε ημέρα εκτελούμε ένα πείραμα τύχης με αυτήν την πιθανότητα και αποφαινόμαστε για την πρόβλεψη μας. Αν ορίζαμε ένα κατώφλι τότε οι περισσότερες ημέρες θα είχαν την ιδιά πρόβλεψη, αφού ως λ ορίσαμε το μέσο πλήθος m ημέρων με αποτέλεσμα η αλλαγή μιας ημέρας να αλλάζει ελάχιστα το λ μας, αρά και την πιθανότητα που ορίζει αυτή η κατανομή.

3.6. Facebook Algorithm (Prophet)

Η Facebook δημιούργησε ένα μοντέλο γραμμένο σε Python και R το οποίο ονομάστηκε prophet ώστε να προβλέπει μελλοντικές τιμές χρονοσειρών με αλγόριθμους μηχανικής μάθησης. Οι παράμετροι που χρησιμοποιούνται είναι προσαρμοσμένοι ώστε να μπορούν να ταιριάξουν σε κάθε δυνατή χρονοσειρά που θα δοθεί ως είσοδος. Η μαθηματική σχέση που περιγράφει την έξοδο του αλγορίθμου μας φαίνεται παρακάτω:

$$y(t)=g(t)+s(t)+h(t)+\varepsilon(t)$$

Κάθε ένας από τους ορούς αυτούς έχει αρκετές παραμέτρους οι οποίες μπορούν είτε να δοθούν από την χρήστη σαν σταθερές είτε να δοθούν συγκεκριμένα όρια που πρέπει να κινούνται είτε να δοθεί πλήρης ελευθερία στον αλγόριθμο να βρει τις κατάλληλες τιμές αυτών. Στην εκτέλεση που κάναμε στα δικό μας δεδομένα αφήσαμε ελεύθερες όλες τις παραμέτρους δίνοντας την δυνατότητα στον αλγόριθμο να βρει όλες τις παραμέτρους που θεωρεί πως ταιριάζουν στα δεδομένα αυτά.

g (t): περιγράφει μια συνάρτηση η οποία μοντελοποιεί μη περιοδικές αλλαγές της χρονοσειράς.

s(t): αναπαριστά περιοδικές αλλαγές (εβδομαδιαία, ετησία)

h (t): αναπαριστά την επίδραση των διακοπών στο αποτέλεσμα της συνάρτησης

ε (t): όρος σφάλματος που περιγράφει αλλαγές που εξαρτώνται από το μοντέλο

Κάθε ένας από τους τέσσερις παραπάνω όρους είναι υπεύθυνος για να μοντελοποιήσει και να προσομοιώσει μια διαφορετική παράμετρο που συμβάλλει στο αποτέλεσμα μας και την πρόβλεψη για την επόμενη ημέρα της χρονοσειράς. Για περιοδικές αλλαγές (s (t)) χρησιμοποιούνται σειρές Fourier και η εύρεση των κατάλληλων συντελεστών αφήνεται στον αλγόριθμο.

Από την άλλη συναρτήσεις όπως η g(t) προσομοιώνονται με μοντέλα λογιστικής παλινδρόμησης με μικρές αλλαγές καθώς και με την εισαγωγή επιπλέον παραμέτρων που είναι είτε σταθερές είτε συναρτήσεις με όρισμα τον χρόνο.

Η h (t) είναι υπεύθυνη για να μπορέσει να αντιληφθεί ποιες περίοδοι αποτελούν ημέρες διακοπών για την χώρα που εξετάζουμε και να επηρεάσει αντίστοιχα το άθροισμα. Για την πιο ορθή χρήση αυτής της συνάρτησης ο αλγόριθμος δίνει την δυνατότητα στον χρήστη να εισάγει πριν εκτελέσει τον αλγόριθμο τις ημέρες διακοπών στην χώρα που εξετάζουμε.

Ο όρος ε (t) αποτελεί μια συνάρτηση η οποία επηρεάζει το αποτέλεσμα μας ανάλογα με τα δεδομένα. Ειδικότερα, προσπαθεί να αντιληφθεί μοτίβα ή οτιδήποτε άλλο επηρεάζει το αποτέλεσμα μας και δεν μπορεί να επηρεάσει κάποια από τις προηγούμενες τρεις συναρτήσεις.

Οι συναρτήσεις αυτές χαρακτηρίζονται από ένα μεγάλο πλήθος παραμέτρων οι οποίες μπορούν είτε να εισαχθούν από τον χρήση είτε να δοθεί η ελευθερία στον αλγόριθμο να τις υπολογίσει για ελαχιστοποίηση του λάθους των προβλέψεων. Η Facebook έχει ορίσει κάποιες κατανομές τις οποίες ακολουθούν οι παράμετροι αυτοί, κάτι που μπορεί να μην ισχύει σε κάθε χρονοσειρά που προσπαθούμε να εφαρμόσουμε πάνω στον αλγόριθμο αυτόν.

Κατά τη διάρκεια της εφαρμογής του αλγορίθμου αυτού στα δεδομένα μας η εκτίμηση των παραμέτρων αφέθηκε αποκλειστικά στον αλγόριθμο καθώς δεν προκαθορίσαμε καμιά από τις παραμέτρους ούτε δώσαμε επιπλέον δεδομένα (πχ ποια περίοδο θεωρείται περίοδος διακοπών). Ο σκοπός του αλγορίθμου είναι να αντιληφθεί αυτά τα δεδομένα και να εξάγει

τις βέλτιστες παραμέτρους για να κάνει τις προβλέψεις. Στη συνεχεία υπολογίζουμε και για την προσέγγιση αυτή τις μετρικές και τις συγκρίνουμε με τις δίκες μου προσεγγίσεις.

Τον αλγόριθμο αυτόν τον εκμεταλλευτήκαμε και τον χρησιμοποιήσαμε στα δικά μας δεδομένα χωρίς να έχουμε εισάγει τίποτα πριν την εκτέλεση του αλγορίθμου δίνοντας πλήρη ελευθερία στον αλγόριθμο για την εύρεση των βέλτιστων παραμέτρων.

Τέλος, στην δικιά μας προσέγγιση η χρονοσειρά αποτελεί μια υποπερίπτωση της γενικότερης χρήσης που μπορεί να γίνει στον αλγόριθμο αυτών καθώς ο σκοπός μας είναι η δυαδική ταξινόμηση των ήμερων ως ημέρες με σφάλμα ή όχι.

ΚΕΦΑΛΑΙΟ 4 Συμπεράσματα, αξιολόγηση και μελλοντική εργασία

Στο κεφάλαιο αυτό γίνεται αναφορά στην πηγή από την οποία προήλθαν τα δεδομένα της διπλωματικής εργασίας, παρουσιάζονται τόσο τα αποτελέσματα κάθε προσέγγισης που χρησιμοποιήσαμε, γίνεται σύγκριση των αποτελεσμάτων αυτών με βάση τις μετρικές αξιόλογης που έχουμε ορίσει, παρουσιάζονται γραφικά τα αποτελέσματα, εξηγούμε για ποιους λογούς κάποιες προσεγγίσεις απέδωσαν καλύτερα, βγάζουμε τα αντίστοιχα συμπεράσματα .Επίσης, εξηγούμε περιληπτικά την διαδικασία που ακολουθήσαμε από την αρχή της εκπόνησης της διπλωματικής εργασίας και την γενικότερη διαίσθηση που αναπτύξαμε γύρω από το πρόβλημα αυτό. Τέλος, δίνεται ο κώδικας που χρησιμοποιήθηκε για την επίτευξη του παραπάνω σκοπού γραμμένος σε Python 3.

4.1. Τα δεδομένα μας

Τα δεδομένα πάνω στα οποία εκπαιδεύσαμε τις διάφορες προσεγγίσεις μας αποτελούν πραγματικά δεδομένα εταιρίας. Πιο συγκεκριμένα της εταιρίας με την επωνυμία « NOKIA Solutions and Networks Hellas». Τα δεδομένα αφορούν ένα συγκεκριμένο προϊόν και τις διάφορες εκδόσεις του. Η κάθε έκδοση δεν υπήρχε από την πρώτη ήμερα που έχουμε το ιστορικό των δεδομένων, για αυτό και εξετάζοντας πιο πρόσφατες εκδόσεις, έχουν σαν συνέπεια για αρκετά μεγάλο χρονικό διάστημα να μην είχαμε σφάλματα μέχρι να εμφανιστεί το πρώτο.

Επίσης ο κώδικας της κάθε έκδοσης βρίσκεται σε αρκετά αρχεία, είναι γραμμένος κατά κύριο λόγο σε Python, Java, με αποτέλεσμα να είναι αρκετά δύσκολο να βρούμε μετρικών κώδικα τα οποία στη συνεχεία θα μπορούσαμε να τα θεωρήσουμε ως χαρακτηριστικά στις προσεγγίσεις μας.

Επιπλέον, για την ύπαρξη των μετρικών αυτών θα έπρεπε να είχαμε πρόσβαση στον κώδικα της κάθε έκδοσης κάθε ήμερα για να μπορούμε να βρούμε τις μετρικές και για τις 6000 ήμερες των δεδομένων μας. Κάτι τέτοιο δεν ήταν εφικτό όποτε η ιδέα για εισαγωγή των μετρικών κώδικα απορρίφθηκε.

Κάθε σφάλμα που διαθέτουμε έχει τα παρακάτω χαρακτηριστικά:

Ημερομηνία που βρέθηκε

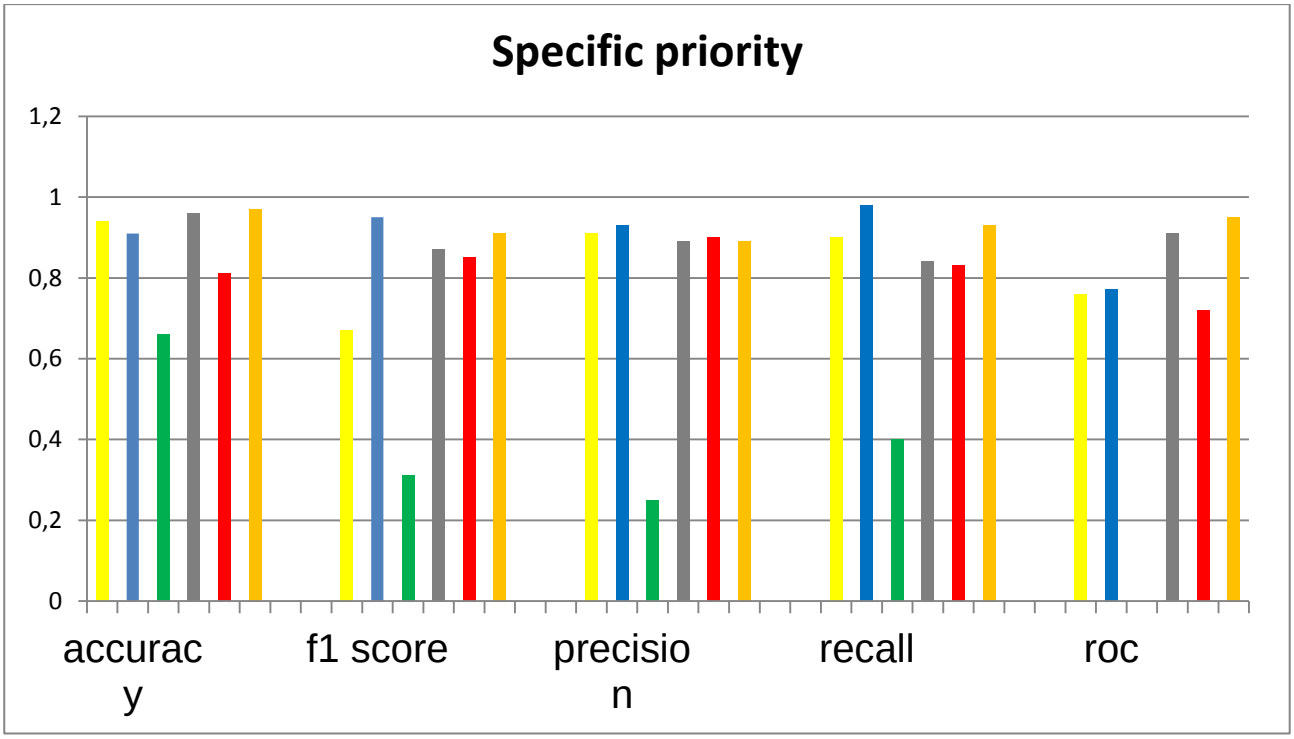
Έκδοση που αφορά.

Το όνομα της εφαρμογής στην συγκεκριμένη έκδοση, όνομα του προγραμματιστή ή της ομάδας που το ανακάλυψε

«Priority» , δηλαδή πόσο σημαντικό θεωρείται (Blocker, Major, Minor οι 3 βασικές κατηγορίες που χρησιμοποιήσαμε).

Επίσης διαθέτουν τα σφάλματα κάποια επιπλέον χαρακτηριστικά αλλά αυτές οι πέντε παράμετροι είναι αυτές που εστιάσαμε για τους αλγορίθμους που εφαρμόσαμε.

4.2. Αποτελέσματα-Γραφικές παραστάσεις



Specific Priority					
	accuracy	f1 score	precision	recall	roc
decision trees	0,94	0,67	0,91	0,9	0,76
logistic	0,91	0,95	0,93	0,98	0,77
poisson	0,66	0,31	0,25	0,4	X
artificial N.N.	0,96	0,87	0,89	0,84	0,91
naïve bayes	0,81	0,85	0,9	0,83	0,72
prophet facebook	0,97	0,91	0,89	0,93	0,95

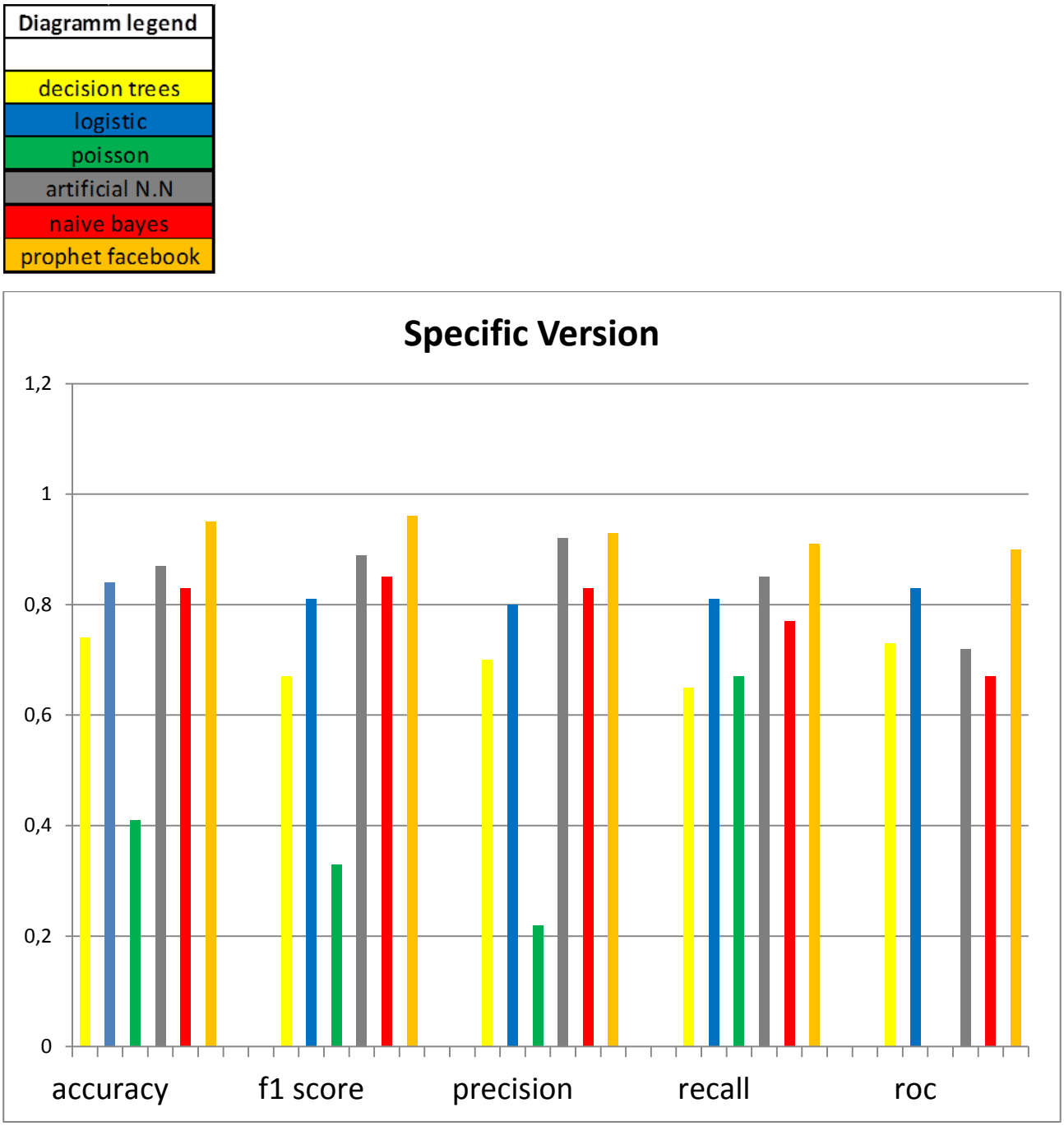
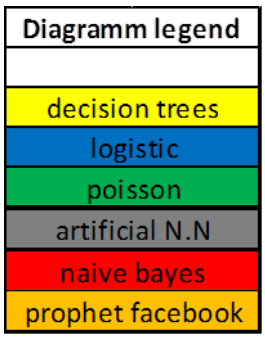
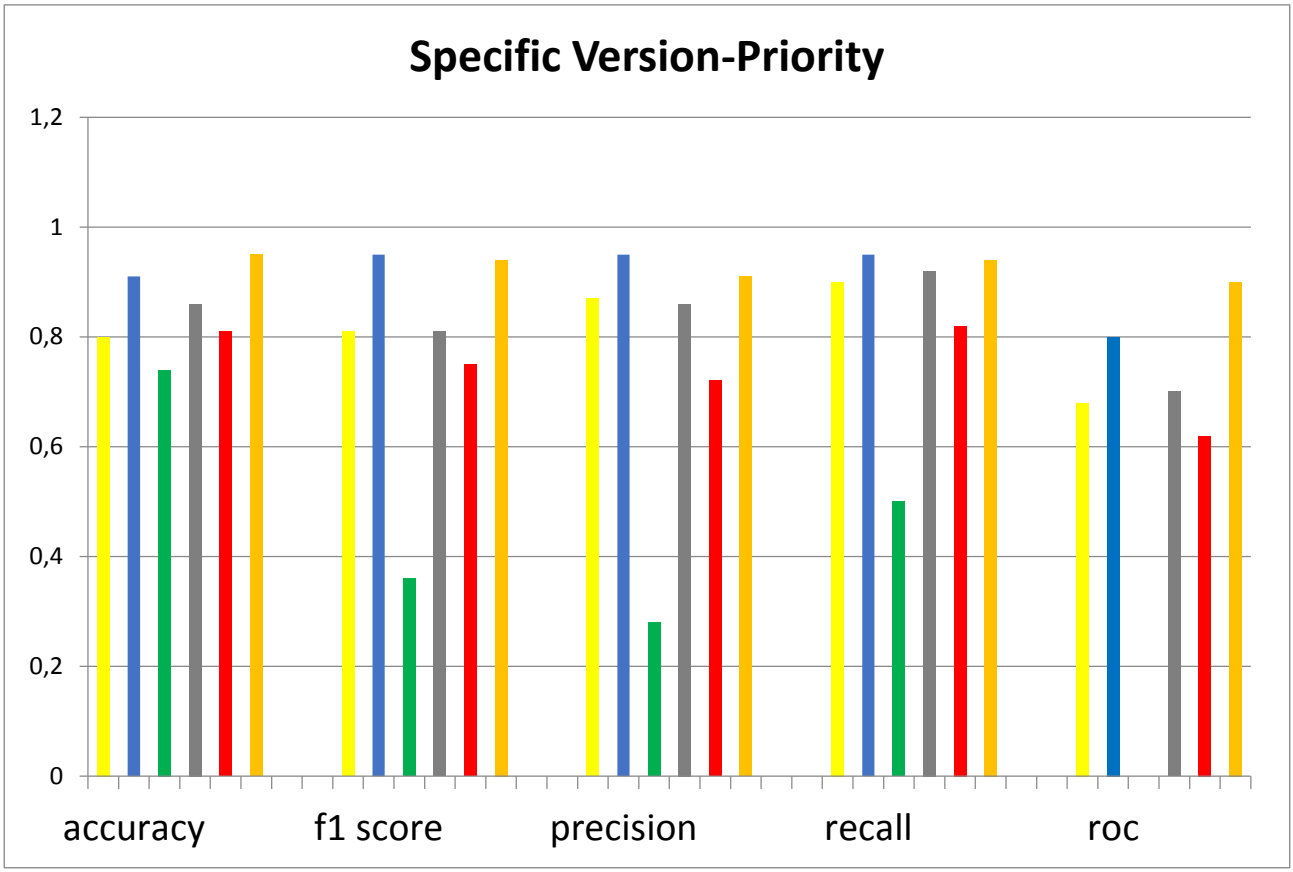


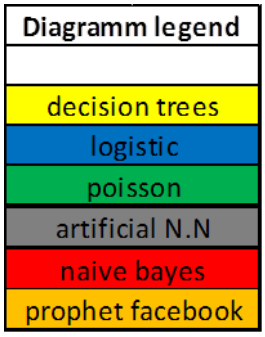
Diagramm legend
decision trees
logistic
poisson
artificial N.N
naive bayes
prophet facebook
Specific Version
1,2
1
0,8
0,6
0,4
0,2
0
accuracy
f1 score
precision
recall
roc

Specific Version					
	accuracy	f1 score	precision	recall	roc
decision trees	0,74	0,67	0,7	0,65	0,73
logistic	0,84	0,81	0,8	0,81	0,83
poisson	0,41	0,33	0,22	0,67	X
artificial N.N.	0,87	0,89	0,92	0,85	0,72
naïve bayes	0,83	0,85	0,83	0,77	0,67
prophet facebook	0,95	0,96	0,93	0,91	0,9





Specific Version-Priority					
	accuracy	f1 score	precision	recall	roc
decision trees	0,88	0,83	0,83	0,9	0,73
logistic	0,92	0,89	0,91	0,95	0,82
poisson	0,7	0,64	0,62	0,76	X
artificial N.N.	0,89	0,85	0,83	0,9	0,76
naïve bayes	0,79	0,72	0,71	0,82	0,65
prophet facebook	0,95	0,93	0,91	0,95	0,91



4.3. Συμπεράσματα

Στο υποκεφάλαιο αυτό θα εξηγήσουμε τους λόγους για τους οποίους κάθε μια από τις προσεγγίσεις μας απέδωσε ικανοποιητικά ή όχι, καθώς και τους λόγους που συνέβη αυτό.

4.3.1 Poisson

Θα αναφερθούμε αρχικά στην προσέγγιση της Poisson.

Είναι εμφανές τόσο με βάση τα αποτελέσματα, όσο και με την διαίσθηση μας πως η « Poisson Distribution» δεν είναι κατάλληλη για την επίλυση του προβλήματος αυτού. Ο βασικότερος λόγος είναι πως τα δεδομένα μας δεν προέρχονται από πειράματα τύχης μιας κατανομής Poisson και ως συνεπεία αυτού δεν μπορούμε να τα προσεγγίσουμε με αυτήν την κατανομή.

Επίσης, ένα ακόμα βασικό πρόβλημα αποτελεί το γεγονός πως η Poisson από όλα τα δεδομένα μας εκπαιδεύεται βρίσκοντας μόνο μία παράμετρο και πιο συγκεκριμένα την μέση τιμή των bugs και προβλέπει με βάση αυτό. Ένας ιδιαίτερα σημαντικός λόγος αποτυχίας είναι πως η συγκεκριμένη έκδοση που εξετάζουμε μπορεί να έχει καλύψει ένα μεγάλο ποσοστό των λαθών της και να μην εμφανίζονται πλέον, αλλά παρ' όλα αυτά λόγω του τρόπου που ορίστηκε το λ να αποφαινόμαστε πως θα υπάρξουν ήμερες με λάθη.

Τέλος, για την Poisson κατανομή κάθε πρόβλεψη γίνεται με τον ίδιο τρόπο ανεξάρτητα από την ημέρα. Δηλαδή ακόμα και αν η επόμενη ημέρα είναι Σαββατοκύριακο το λ είναι κοινό με την περίπτωση που είναι η επόμενη μέρα είναι πχ Δευτέρα κάτι το οποίο αποτελεί μια λανθασμένη υπόθεση, καθώς οι καθημερινές, λόγω φύσης του προβλήματος, έχουν πιο αυξημένη πιθανότητα να είναι ημέρες με λάθη.

Αυτοί είναι οι βασικοί λόγοι που η Poisson Regression δεν αποδίδει ιδιαίτερα ικανοποιητικά αποτελέσματα στο πρόβλημα της πρόβλεψης λαθών λογισμικού.

Ακόμα και αν δεν χρησιμοποιούσαμε την Poisson Distribution, αλλά κάποια άλλη γνωστή κατανομή πιθανοτήτων η απόδοση θα ήταν ιδιαίτερα χαμηλή, καθώς η υπόθεση πως τα δεδομένα μας ακολουθούν την κατανομή αυτή θα ήταν λανθασμένη και η εύρεση των παραμέτρων της κατανομής αυτής θα γινόταν χωρίς να έχουμε τα επιθυμητά αποτελέσματα.

Η προσέγγιση που θα μπορούσε να γίνει με βάση πιθανοτικές κατανομές και να αποδώσει αισθητά καλύτερα αποτελέσματα θα ήταν να ορίζαμε τον σταθμισμένο μέσο μεταξύ

κατανομών με τα κατάλληλα βάρη. Με αυτόν τον τρόπο κάθε κατανομή θα μπορούσε να συμβάλλει στο άθροισμα που αναζητούμε (δηλαδή την πιθανότητα η επόμενη ημέρα να εμφανίσει λάθος) στο βαθμό που ορίζουν τα δεδομένα μας και να αλληλοεξουδετερωθούν πιθανώς κάποια από τα αίτια που δεν επέτρεπαν σε κάθε μια από αυτές να δρα ξεχωριστά και να επιλύει το πρόβλημα με ικανοποιητικά αποτελέσματα.

4.3.2 Naive Bayes ταξινομητής

Αυτή η προσέγγιση βασίζεται στην ανεξαρτησία μεταξύ των παραμέτρων. Αυτός είναι ο βασικός λόγος που δεν καταφέρνει να ανταγωνιστεί τις υπόλοιπες προσεγγίσεις και να αποδώσει τα βέλτιστα αποτελέσματα. Η γενικότερη ιδέα πίσω από αυτήν την μέθοδο είναι πως η τιμή της κάθε παραμέτρου δεν επηρεάζεται από τις τιμές των υπολοίπων και πως ο συνδυασμός αυτός μπορεί να μας δώσει την συνάρτηση εξόδου μέσω του ταξινομητή αυτού.

Αναλύοντας το πρόβλημα της πρόβλεψης λαθών λογισμικού με την προσέγγιση των χρονοσειρών και συγκεκριμένα με Μαρκοβιανή αλυσίδα μνήμης τάξης m είναι εμφανές πως η κάθε ημέρα προσδιορίζεται από τις προηγούμενες m ημέρες. Κάτι τέτοιο έχει σαν αποτέλεσμα η τιμή κάθε μιας από τις m αυτές ημέρες να μην είναι ανεξάρτητη από τις προηγούμενες ήμερες. Δεδομένου πως κάθε μια από αυτές τις μέρες χρησιμοποιείται ως παράμετρος γίνεται αντιληπτό πως η υπόθεση για ανεξαρτησία των παραμέτρων είναι λανθασμένη και πως η προσέγγιση του προβλήματος αυτού με βάση την τεχνική αυτή δεν θα αποδώσει τόσο ικανοποιητικά αποτελέσματα.

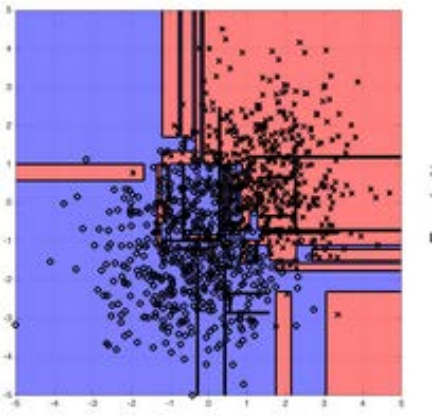
Σε σύγκριση με την «Poisson Regression» η υπόθεση που κάναμε στον Naive Bayes ταξινομητή έχει αισθητά μικρότερο περιθώριο λάθους σε σχέση με την υπόθεση πως τα δεδομένα μας ακολουθούν μια συγκεκριμένη πιθανοτική κατανομή.

Τέλος αναμένουμε καλύτερα αποτελέσματα σε σύγκριση με «Poisson Regression» αλλά όχι ικανοποιητικά ιδιαίτερα για τους παραπάνω λόγους.

4.3.3 Δέντρα απόφασης

Η μέθοδος αυτή δίνει την δυνατότητα να διαχωρίσουμε τον χώρο μας σε αρκετά κομμάτια και να χρωματίσουμε το κάθε ένα από αυτά ώστε να γνωρίζουμε πως στο κομμάτι αυτό η πρόβλεψη μας θα είναι 0 ενώ σε άλλα κομμάτια θα είναι 1. Κάτι τέτοιο φαίνεται να αποδίδει ιδιαίτερα ικανοποιητικά στο πρόβλημα μας, αφού για κάθε έναν από τους 2^m πιθανούς συνδυασμούς θα καταλήγουμε βέλτιστα στον χώρο που ορίζουν τα δέντρα απόφασης και θα αποφαινόμαστε για την επόμενη ημέρα.

Σχηματικά σε δύο διαστάσεις:



Εικόνα 4.1 Decision Trees

Ένα μειονέκτημα των δέντρων απόφασης είναι η ανάγκη μεγάλου πλήθους δεδομένων που χρειάζονται για να εκπαιδευτούν και να διαχωρίσουν κατάλληλα τον χώρο. Στην περίπτωση μας τα 6000 σφάλματα ιστορικό που διαθέτουμε δεν είναι ένας τόσο ικανοποιητικός αριθμός λαθών ώστε να μπορέσει το δέντρο απόφασης να εκπαιδευτεί βέλτιστα.

Επίσης, λόγω του περιορισμένου όγκου δεδομένων που διαθέτουμε ,και της μη βέλτιστης εκπαίδευσης, μια αλλαγή μιας από τις m ημέρες μπορεί να επηρεάσει τελείως τον χώρο κίνησης μας, ενώ στην πραγματικότητα αυτή η αλλαγή δεν θα έπρεπε να επηρεάσει το αποτέλεσμα μας, αν η εκπαίδευση γινόταν βέλτιστα.

Τέλος, ένα ακόμα μειονέκτημα που μπορεί να προκύψει είναι η λανθασμένη ανακάλυψη σχέσεων μεταξύ των m ήμερων που μπορούν να καθορίσουν το τελικό αποτέλεσμα. Για παράδειγμα λόγω του μικρού εύρους dataset που διαθέτουμε για την ταξινόμηση της επόμενης ημέρας ως ημέρα με σφάλμα ή όχι μπορεί να εκπαιδευτεί το δέντρο απόφασης πως ο συνδυασμός 0, 1, 0 των τριών πρώτων ημερών δίνει πάντα αποτέλεσμα 1. Κάτι τέτοιο είναι λανθασμένο αλλά η εμφάνιση αυτού του pattern αρκετά συχνά θα ορίζει τον χώρο με τέτοιο τρόπο ώστε να εγκυμονεί ο κίνδυνος να γίνονται λάθη στη διαδικασία της αξιολόγησης.

4.3.4 Λογιστική παλινδρόμηση – Νευρωνικό Δίκτυο με Relu συνάρτηση ενεργοποίησης

Αρχικά, και οι δύο μέθοδοι αποτελούνται από νευρωνικά δίκτυα χωρίς κρυφό επίπεδο (hidden layer). Στο βασικό τους επίπεδο υπολογίζουν το γινόμενο δύο πινάκων (παράμετροι Χ βάρη + Β) και στη συνεχεία μέσω μιας συνάρτησης ενεργοποίησης προσδιορίζεται η πιθανότητα η επόμενη ημέρα να εμφανίσει λάθος.

Η λογική να προσδιορίζουμε τα κατάλληλα βάρη σε κάθε μια από τις προηγούμενες m ημέρες και στη συνεχεία να αποφαινόμαστε για την πιθανότητα που αναζητούμε έχει αρκετά καλές πιθανότητες επιτυχίας. Αν η υπόθεση μας πως η κάθε μέρα προσδιορίζεται από τις προηγούμενες m ημέρες είναι ορθή, τότε αυτή η μέθοδος θα απέδιδε τα βέλτιστα, όπως και γίνεται εν τέλει.

Μια βασική διάφορα μεταξύ των δύο αυτών μεθόδων είναι το πλεονέκτημα της λογιστικής παλινδρόμησης να μπορεί να διαθέτει μη γραμμική συνάρτηση ενεργοποίησης. Μέσω της Relu συνάρτησης ενεργοποίησης, δεν δίνεται η δυνατότητα στο μοντέλο μας να προσομοιώνει μη γραμμικές αλλαγές και είναι υποχρεωμένο να θεωρεί πως μέσω γραμμικού συνδυασμού των παραμέτρων μπορούμε να πάρουμε βέλτιστα αποτελέσματα. Η υπόθεση αυτή μπορεί να μας οδηγήσει στο φαινόμενο του Underfitting, καθώς προσπαθούσε να προσεγγίσουμε έναν όγκο δεδομένων με πιο απλοϊκό τρόπο από αυτόν που πραγματικά απαιτείται.

Ένα πρόβλημα που θα μπορούσε να υπάρξει είναι αν τα δεδομένα μας ήταν δομημένα με τέτοιο τρόπο ώστε μια ήμερα να μπορεί να προσδιορίσει πλήρως την πρόβλεψη μας. Στην περίπτωση αυτή το βάρος της θα έπρεπε να είναι ιδιαίτερα αυξημένο (ιδανικά άπειρο) και τα υπόλοιπα βάρη αρκετά μικρά. Κάτι τέτοιο όμως θα μπορούσε να μην συμβαίνει λόγω του μικρού όγκου δεδομένων που διαθέτουμε. Αυτό δεν αποτελεί πρόβλημα στην περίπτωση μας, διότι όλες οι m μέρες συμβάλλουν στο αποτέλεσμα μας και τα βάρη των λιγότερων σημαντικών είναι αρκετά μικρότερα όπως θα έπρεπε.

Η διαίσθηση μας είναι πως η λογιστική παλινδρόμηση θα απέδιδε βέλτιστα σε σύγκριση με τις υπόλοιπες τεχνικές που εφαρμόστηκαν στο πρόβλημα αυτό.

4.3.5 Facebook-Prophet

Η περίπτωση αυτή αποτελεί μια ιδιαίτερη περίπτωση που εφαρμόσαμε στα δεδομένα μας, μιας και δεν υλοποιήθηκε από την αρχή απλώς χρησιμοποιήθηκε ο ήδη υπάρχων κώδικας στα δεδομένα μας. Το γεγονός πως δεν ορίσαμε εξαρχής αρκετά δεδομένα, η έλλειψη μεγάλου πλήθους δεδομένων που απαιτείται για την εύρεση των παραμέτρων του αλγορίθμου αυτού, καθώς και το γεγονός πως η χρονοσειρά μας αποτελείτο αποκλειστικά από {0, 1} ήταν κάποια από τα προβλήματα που επηρέασαν αρνητικά την απόδοση της προσέγγισης αυτής.

Παρ' όλα αυτά, τα αποτελέσματα ήταν αρκετά ικανοποιητικά καθώς η δυνατότητα που δίνεται το αποτέλεσμα να αθροίζεται από τέσσερις συναρτήσεις και όχι αποκλειστικά από μια, και η ορθή προσαρμογή των παραμέτρων σε συγκεκριμένες κατανομές (δηλαδή έχει προοριστεί για αρκετές παραμέτρους το είδος της κατανομής που ακολουθεί και το πρόβλημα απλοποιείται στην εύρεση των παραμέτρων των κατανομών αυτών) δίνει ικανοποιητικά αποτελέσματα.

4.4. Επιπλέον δεδομένα

Η ερώτηση που προκύπτει είναι ποια επιπλέον δεδομένα θα θέλαμε να είχαμε στη διάθεση μας ώστε να πετυχαίναμε καλύτερο αποτέλεσμα. Η αλήθεια είναι πως υπάρχουν αρκετοί τρόποι ώστε να βελτιστοποιούσαμε την απόδοση μας ακόμα και αν δεν είχαμε στη διάθεση μας τις μετρικές του κώδικα ούτε τον κώδικα της κάθε εφαρμογής. Μία αρκετά σημαντική παράμετρος που θα θέλαμε να έχουμε στην διάθεση μας είναι μια αξιολόγηση του κάθε προγραμματιστή που εργάζεται πάνω στο πρόβλημα μας και προσπαθεί να ανακαλύψει λάθη λογισμικού και αν αυτός εργάζεται η όχι. Δηλαδή έχουμε ορίσει για κάθε προγραμματιστεί ως απόδοση το μέσο πλήθος λαθών που βρήκε κάθε μέρα δηλαδή έναν αριθμό Xi. Στη συνεχεία γνωρίζοντας κάθε μέρα ποιοι προγραμματιστές εργάζονται πάνω σε ποιους κώδικες θα μπορούσαμε να υπολογίσουμε το Σxi των προγραμματιστών που εργάζονται πάνω στα λάθη που αναζητούμε εμείς. Η μέθοδος αυτή θα μπορούσε να λαμβάνει υπόψιν της, αν βρισκόμαστε σε περίοδο διακοπών ή ακόμα καλύτερα, αν κάποιος προγραμματιστής ιδιαίτερα αποδοτικός βρισκόταν σε άδεια θα το γνωρίζαμε και ο αλγόριθμος μας θα μπορούσε να αξιολογήσει την απουσία αυτήν έχοντας ορίσει το αντίστοιχο βάρος στο χαρακτηριστικό αυτό.

Επίσης κάτι ακόμα που θα επηρέαζε την απόδοση του αλγορίθμου είναι ο τρόπος επίλυσης των σφαλμάτων. Δηλαδή ιδανικά θα θέλαμε κάθε φορά που κάποιο σφάλμα εμφανίζεται να επιλύεται πλήρως και να μην υπάρχει κίνδυνος για μετέπειτα εμφάνιση αυτού ή παρόμοιου του. Αν για παράδειγμα ένα λάθος επιλύθηκε από έναν όχι τόσο ικανό προγραμματιστή τότε είναι αρκετά πιθανό να επανεμφανιστεί τις επόμενες ημέρες ένα λάθος με παρόμοια χαρακτηριστικά. Όποτε ένα ακόμα χαρακτηριστικό που θα θέλαμε να υπάρχει είναι πόσο ορθά επιλύθηκε ένα σφάλμα . Δηλαδή ιδανικά θα θέλαμε να υπάρχει ένας αριθμός που θα μας έδειχνε την πιθανότητα επανεμφάνισης λάθους με αυτά τα χαρακτηριστικά με βάση τον προγραμματιστή ή την ομάδα που θεωρήθηκε υπεύθυνη για την επίλυση του προβλήματος αυτού ανάλογα με τα αντίστοιχα περιστατικά στο παρελθόν. Επιπλέον, ακόμα πιο ιδανικό θα ήταν αντί ο αριθμός αυτός να είναι ένας πραγματικός στο διάστημα [0, 1] να είχαμε μια συνάρτηση f (t) με σύνολο τιμών το [0, 1] που θα μας έδειχνε ποιες η πιθανότητες επανεμφάνισης αντίστοιχου λάθους για κάθε μέρα μετά την επίλυση του ανάλογα με τον προγραμματιστή ή την ομάδα που ανέλαβε να διορθώσει το πρόβλημα αυτό.
Με βάση το παραπάνω, οι πιο ικανές ομάδες θα είχαν συνάρτηση f (t) που θα βρισκόταν πιο κοντά στο 0 και θα ήταν φθίνουσα συνάρτηση του t καθώς τις πρώτες μέρες θα ήταν

πιο πιθανό να βρεθεί επανεμφάνιση αντίστοιχου λάθους. Αν θεωρήσουμε την προσέγγιση που κάναμε και προηγουμένως με τις m ήμερες θα έπρεπε η συνάρτηση να ορίζεται για διακριτές τιμές στο διάστημα [0, m] και θα μπορούσαμε να τις συμπεριλάβουμε σαν χαρακτηριστικά σε κάθε μέθοδο που θα χρησιμοποιούσαμε.

Αυτά οι δύο παράμετροι, κατά την άποψη μου, θα συνέβαλαν στην μεγιστοποίηση των μετρικών καθώς σε συνδυασμό με τις m προηγούμενες ήμερες, την κατάλληλη μέθοδο λογιστικής παλινδρόμησης, το κατάλληλο ποσοστό δεδομένων για διαδικασία εκπαίδευσης- αξιολόγησης και πιθανώς κάποιες μετρικές κώδικα σε συνάρτηση με τον χρόνο θα μπορούσαμε να πετύχουμε ακόμα πιο εξαιρετικά αποτελέσματα για το πρόβλημα της πρόβλεψης λαθών λογισμικού στα υπάρχοντα δεδομένα που διαθέτουμε από τη ΝΟΚΙΑ.

4.5. Η διαίσθηση πίσω από το πρόβλημα της πρόβλεψης λαθών λογισμικού

Αρχικά το πρόβλημα της πρόβλεψης λαθών λογισμικού χωρίς την γνώση του κώδικα φαίνεται άλυτο εκ πρώτης όψεως. Πώς είναι δυνατόν ένας αλγόριθμος να κατανοήσει κάποια μοτίβα που αφορούν την απόδοση εργαζόμενων προγραμματιστών; Δυο βασικά ερωτήματα για την προσέγγιση του προβλήματος αυτού είναι αρχικά ποιες παράμετροι επηρεάζουν την απόδοση των προγραμματιστών και στη συνεχεία με ποια μέθοδο θα ελαχιστοποιήσουμε την συνάρτηση σφάλματος που έχουμε ορίσει. Ποιος είναι ο ιδανικός τρόπος για να ορίσουμε τα χαρακτηριστικά που θα συμπεριλάβουμε στο πρόβλημα μας; Έχοντας κατανοήσει το πρόβλημα και λαμβάνοντας υπόψιν μας τις ήδη υπάρχουσες προσεγγίσεις που έχουν γίνει αντιλαμβανόμαστε πως η εποχικότητα είναι ίσως η πιο βασική παράμετρος για το πρόβλημα μας. Ακόμα και αν δεν έχουμε ορίσει τα βέλτιστα χαρακτηριστικά τότε πρέπει η μέθοδος μας να κατανοήσει πως κάποιες από αυτές τις παραμέτρους δεν έχουν επιρροή στην πρόβλεψη μας και να ορίσει μηδενικό βάρος σε αυτές αν είχαμε το κατάλληλο πλήθος δεδομένων και την υπολογιστική ισχύ ώστε να βρούμε τα βέλτιστα βάρη.

Αν είχαμε το κατάλληλο όγκο δεδομένων τότε η συνάρτηση της κάθε μετρικής μας θα έπρεπε να είναι αύξουσα καθώς προσθέτουμε επιπλέον παραμέτρους. Άρα η αναζήτηση παραμέτρων (features) θα γινόταν πιο απλή διαδικασία καθώς ακόμα και λανθασμένες παράμετροι δεν θα μείωναν την απόδοση στο πρόβλημα μας.

Πριν καταλήξω στην εύρεση των παραμέτρων που χρησιμοποίησα σε συνδυασμό με τη μέθοδο λογιστικής παλινδρόμησης ώστε να πετύχω τα βέλτιστα αποτελέσματα σε σχέση με τις άλλες προσεγγίσεις προσπάθησα να κατανοήσω τα δεδομένα. Η γραφική αναπαράσταση αυτών είναι ένα πρώτο στάδιο για την κατανόηση αυτών. Στη συνεχεία η εφαρμογή της διαίσθησης είναι επίσης απαραίτητη. Δηλαδή αν παρατηρήσουμε πως για

κάποιες μέρες δεν εμφανίζονται λάθη τότε μπορούμε να υποθέσουμε πως βρισκόμαστε σε κάποια περίοδο υπολειτουργίας (πχ διακοπές). Από την άλλη πλευρά αν είχαμε συνεχόμενες ημέρες με λάθη, τότε υπάρχει μεγάλη πιθανότητα να συνεχιστεί το μοτίβο αυτό και την επόμενη ήμερα λόγω αυξημένης εργατικότητας των εργαζόμενων.

Συνοψίζοντας, έχοντας μελετήσει αρκετές προσεγγίσεις στο πρόβλημα αυτό θα έθετα ως απαραίτητη προϋπόθεση να έχουμε μετρικές κώδικα κάθε ήμερα από την έναρξη της μελέτης του dataset που διαθέτουμε ώστε να βρούμε τα κατάλληλα βάρη και για αυτά. Επίσης επειδή θεωρούμε πως είναι αρκετά πιθανό σενάριο η εισαγωγή παραμέτρων που δεν ταιριάζουν, θα έπρεπε να έχουμε αρκετά δεδομένα για την εύρεση των βέλτιστων βαρών για κάθε ένα από αυτά. Επιπλέον, η υπολογιστική ισχύς είναι απαραίτητη διότι το στάδιο εκπαίδευσης θα έπρεπε να γίνει αρκετές φόρες ώστε να μειωθεί η πιθανότητα να σταματήσουμε σε κάποιο τοπικό ελάχιστο, αντί για την εύρεση του ολικού ελάχιστου στην συνάρτηση σφάλματος. Σε κάποιες προσεγγίσεις (Δέντρα απόφασης) η σωστή κατασκευή του δικτύου είναι NP πρόβλημα και αν δεν υπάρχει η κατάλληλη υπολογιστική ισχύς τότε η φάση εκπαίδευσης θα διαρκέσει αρκετά μεγάλο χρονικό διάστημα, παρότι γίνεται με προσεγγιστικούς τρόπους ώστε να μην χρειαστούμε εκθετικό χρόνο για τη διαδικασία εκπαίδευσης.

Τα αποτελέσματα μας είναι διαφορετικά ανάλογα και τόσο με την γλώσσα προγραμματισμού όσο και με τις βιβλιοθήκες της συγκεκριμένης γλώσσας που χρησιμοποιήσαμε αφού κάθε μια προσέγγιση επιτυγχάνει την ελαχιστοποίηση της συνάρτησης σφάλματος με διαφορετικό τρόπο.

Η ορθή επιλογή όλων των παραπάνω μας οδηγεί στην βέλτιστη απόδοση για το πρόβλημα αυτό.

4.6. Μελλοντική εργασία

Πριν ολοκληρώσουμε την παρουσίαση μας, θα ήθελα να ορίσω μια μελλοντική εργασία για οποιονδήποτε ήθελε να ασχοληθεί με το πρόβλημα της πρόβλεψης λαθών λογισμικού ή γενικότερα με την πρόβλεψη τιμών χρονοσειρών που αποτελούνται από binary values. Αρχικά θα μπορούσαν να γίνουν άλλες προσεγγίσεις για δυαδική ταξινόμηση είτε με πιθανοτικές κατανομές (Gaussian, Weibull) είτε αλγορίθμους μηχανικής μάθησης όπως SVM είτε άλλες προσεγγίσεις [26] .Κάτι τέτοιο όμως δεν θεωρώ πως θα απέδιδε ιδιαίτερα καθώς οι προσεγγίσεις που επιλέξαμε αποτελούν τις πιο ισχυρές για το συγκεκριμένο πρόβλημα.

Αντίθετα, θα μπορούσαμε να ορίσουμε κάποιες κατανομές και να παίρνουμε τον σταθμισμένο μέσο αυτών προσπαθώντας να εξαλείψουμε τα μειονεκτήματα κάθε μιας από τις κατανομές αυτές μέσω της μεθόδου αυτής. Για κάθε κατανομή ο αλγόριθμος μας θα έπρεπε πέρα από να προσδιορίσει τις βέλτιστες παραμέτρους της κατανομής αυτής να βρει

και το κατάλληλο βάρος με το οποίο θα πολλαπλασιαστεί το αποτέλεσμα της (δηλαδή πόσο θα συνεισφέρει το αποτέλεσμα αυτής στην τελική πρόβλεψη μας).

Κάτι που θα αποτελούσε ακόμα πιο ενδιαφέρον πρόβλημα και θα μπορούσε να συνδυαστεί με το δικό μας θα ήταν να προσπαθήσουμε να το προσεγγίσουμε χωρίς να μετατρέψουμε τα δεδομένα μας σε δυαδικές τιμές. Δηλαδή αντί να αναζητούμε κάθε μέρα αν θα εμφανιστεί λάθος με συγκεκριμένα χαρακτηριστικά θα μπορούσαμε να εξετάζουμε τα λάθη αθροιστικά. Δηλαδή θα είχαμε χρονοσειρά ακεραίων αριθμών και με βάση αυτούς να προσπαθήσουμε να προβλέψουμε τον επόμενο.

Ειδικότερα θα είχαμε ένα Deep Νευρωνικό Δίκτυο κατάλληλης τοπολογίας το οποίο θα έπαιρνε σαν είσοδο ακεραίους αριθμούς που θα συμβόλιζαν το συνολικό πλήθος σφαλμάτων που εμφανίστηκαν κάθε ημέρα και κάποια ακόμα χαρακτηριστικά που θα είχαμε ορίσει (όπως αυτά που προαναφέραμε) και θα προσπαθούσαμε να προβλέψουμε τον επόμενο αριθμό. Προφανώς στην περίπτωση αυτήν δεν θα υπολογίζαμε αυτές τις μετρικές αλλά θα έπρεπε να μετράμε την απόσταση της πρόβλεψης μας από την πραγματική τιμή της συνάρτησης.

Αυτή η προσέγγιση μοιάζει αρκετά ανεξάρτητο πρόβλημα από το δικό μας, μιας και χρησιμοποιούνται άλλου είδους νευρωνικά άλλες μετρικές ακόμα και αλλά δεδομένα. Όμως υπάρχει ένας τρόπος να τα συνδυάσουμε για να πετύχουμε το βέλτιστο συνολικό αποτέλεσμα. Αν είχε δημιουργηθεί ένα νευρωνικό, το οποίο θα μπορούσε να βρει το συνολικό πλήθος των λαθών την επόμενη ημέρα, τότε θα μπορούσαμε να υπολογίζουμε δεσμευμένες πιθανότητες στο πρόβλημα μας. Δηλαδή, αντί να υπολογίζαμε την πιθανότητα εμφάνισης ενός λάθους με συγκεκριμένα χαρακτηριστικά, θα υπολογίζαμε την πιθανότητα αυτή δεδομένο ως το συνολικό πλήθος λαθών την επόμενη ήμερα θα είναι Ν. Το νόημα αυτής της προσέγγισης είναι πως η βέλτιστη μέθοδος μας (logistic regression) θα είχε μία ακόμα παράμετρο, αρκετά ισχυρή ώστε με το κατάλληλο βάρος να αποδίδει ακόμα καλύτερα. Δηλαδή η έξοδος του πρώτου νευρωνικού που θα ήταν το πλήθος των λαθών την επόμενη ήμερα θα ήταν μια παράμετρος, όπως όλα τα υπόλοιπα, στον αλγόριθμο μας.

Τέλος, ακόμα και στην βέλτιστη προσέγγιση της λογιστικής παλινδρόμησης ως μελλοντική εργασία θα μπορούσαμε να ορίσουμε την εισαγωγή διασταυρωμένων παραμέτρων (cross features) στην συνάρτηση του νευρωνικού δικτύου που μέσω της διαδικασίας εκπαίδευσης θα έπαιρναν το αντίστοιχο βάρος ώστε να εκμεταλλευτούμε κάποιους συνδυασμούς που μπορούν να αποδώσουν καλύτερα γνωρίζοντας πως κάποια μοτίβα συγκεκριμένων ημερών θα μπορούσαν να καθορίσουν το αποτέλεσμα.

4.7. Εναλλακτικά δεδομένα

Οι αλγόριθμοι που χρησιμοποιήθηκαν καθ' όλη τη διάρκεια της διπλωματικής εργασίας καθώς και η λογική πίσω από αυτούς θα μπορούσαν να εφαρμοστούν σε αλλά δεδομένα είτε παρομοίου προβλήματος είτε και τελείως διαφορετικού περιεχόμενου.

Στην πρώτη περίπτωση θα έπρεπε να έχουμε ημέρες με λάθη ή χωρίς λάθη και να εκπαιδεύουμε τον αλγόριθμο μας με βάση αυτά προσδιορίζοντας το κατάλληλο m, ώστε να κάνουμε τις προβλέψεις μας για τις επόμενες ήμερες. Κάτι τέτοιο περιέχει την υπόθεση πως και τα νέα δεδομένα μας θα μπορούν να θεωρηθούν Μαρκοβιανή αλυσίδα τάξης m και πως κάθε μια από τις νέες ημέρες θα προκύπτει από τις m τελευταίες ήμερες. Αυτή είναι μια αρκετά ισχυρή υπόθεση που θα θέλαμε και δοκιμάσουμε και σε αλλά δεδομένα για να αποφανθούμε την εγκυρότητα της. Θα θέλαμε να μπορέσουμε να έχουμε και αντίστοιχα δεδομένα άλλης εφαρμογής ώστε να κατανοήσουμε πως οι αλγόριθμοι μας, και συγκεκριμένα ο τρόπος επιλογής των παραμέτρων, δεν ταίριαξαν στα δεδομένα μας αλλά έχουν γενική εφαρμογή και ισχύ.

Τέλος, οι παραπάνω αλγόριθμοι θα μπορούσαν να εφαρμοστούν σε οποιαδήποτε δεδομένα με δυαδικές ετικέτες που θέλουμε να γνωρίζουμε σε ποια από τις δύο κατηγορίες εντάσσεται η επόμενη ήμερα {0, 1}. Στην περίπτωση αυτή δεν θα υπήρχαν ικανοποιητικά αποτελέσματα, διότι η υπόθεση της μαρκοβιανής αλυσίδας τάξης m δεν είναι εφικτό να ισχύει σε κάθε πλήθος δεδομένων και μπορεί να χρειαζόταν τροποποίηση των παραμέτρων που έχουμε ορίσει στις μεθόδους που χρησιμοποιήσαμε.

4.8. Κώδικας Εφαρμογής

Στο κεφάλαιο αυτό θα αναφερθούμε στον κώδικα που έχουμε δημιουργήσει για να εκτελέσει τα παραπάνω. Δέχεται σαν είσοδο ένα αρχείο σε μορφή Json το οποίο περιγράφει το dataset που διαθέτουμε, δηλαδή το ιστορικό εμφάνισης των bugs. Αυτό, είναι ένα dictionary και το ανοίγουμε χρησιμοποιώντας μια συνάρτηση open. Στη συνέχεια δημιουργούμε τους πίνακες X, Y οπου X η είσοδος μας και Y η έξοδος μας για κάθε instance. Επίσης το πλήθος των ημερών που θέλουμε να προβλέπουμε αφήνεται ανοιχτό για τροποποίηση αν και είναι ορισμένο στο 30 , μικρές παραλλαγές αυτού θα μπορούσαν να αλλάξουν το αποτέλεσμα. Χρησιμοποιούνται και αρκετές βιβλιοθήκες για συναρτήσεις όπως η train_test_split η οποία χωρίζει τα δεδομένα μας σε train και test data.

Από όλο το dataset, ο αλγόριθμος μας απομονώνει τα χαρακτηριστικά των bugs που θέλουμε και ταξινομεί τις ημέρες ως bug free ή όχι ανάλογα με την εμφάνιση ή όχι bug την ημέρα εκείνη. Στη συνέχεια εκπαιδεύεται , κάνει τις προβλέψεις μας, αξιολογείται με βάση

τις μετρικές και τέλος εμφανίζει και τα αποτελέσματα τόσο αυτά που προέβλεψε όσο και τα πραγματικά .

Οι κώδικες είναι ανεβασμένοι στο github και λινκ σε αυτό παρατίθεται παρακάτω:

https://github.com/odydro4/Software-Bug-Prediction

References

1) *Ahmadi A.A., Hall G., 2016, ORF 523 Lecture 7, Spring, Princeton University*
2) *Ambros, M.D.; Lanza, M.; Robbes, R. An Extensive Comparison of Bug Prediction Approaches. In Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories (MSR2010), Cape Town, South Africa, 2–3 May 2010.*
3) *Arora H.D., Kumar V., Sahni R., 2014, Study of bug prediction modeling using various entropy measures- a theoretical approach, Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*
4) *Banerjee A., 2006, Binary time series prediction using recurrent networks, Smart Engineering System Design: Neural Networks, Evolutionary Programming, Data Mining, and Artificial Life, Volume 16*
5) *Brockwell, P. J., & Davis, R. A., 2002, Introduction to Time Series and Forecasting, Springer Texts in Statistics 2nd Edition*
6) *Bullinaria J. A., 2015, Bias and Variance, Under-Fitting and Over-Fitt, Neural Computation : Lecture 9*
7) *Canbek G., 2017, Binary Classification Performance Measures/Metrics, A comprehensive visualized roadmap to gain new insights, 2017 International Conference on Computer Science and Engineering (UMBK), At Antalya Turkey*
8) *Dam H.K, et al., 2018, A deep tree-based model for software defect prediction, https://arvix.org*
9) *Fan R.E., et al., 2008, Liblinear: A library for large linear classification, J. Mach. Learn. Res., p. 1871–1874*
10) *Fawcett T., 2006 An introduction to ROC analysis. Pattern Recognition Letters, p.861–874*
11) *Gupta, Dharmendra Lal, and Kavita Saxena. "Software bug prediction using object-oriented metrics." Sādhanā (2017): 1-15..*
12) *Hammouri A., et al., 2018, Software Bug Prediction using Machine Learning Approach, International Journal of Advanced Computer Science and Applications, Vol. 9, No. 2*
13) *He Hu, 2018, vReLU Activation Functions for Artificial Neural Networks, 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*
14) *Huanjing Wang, Taghi M. Khoshgoftaar, Jason Van Hulse. (2010). A Comparative Study of Threshold-based Selection Techniques, Proceeding of IEEE International Conference: Granular Computing (pp.499-504).*
15) *Kalai Magal. R, Shomona Gracia Jacob. (2015). Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques, International Journal of Computer Applications (0975-8887), 117(23), 18-22.*

16) *Kavitha S., Varuna S., Ramya R., 2016, A comparative analysis on linear re-gression and support vector regression, Online International Conference on Green Engineering and Technologies (IC-GET)*
17) *Kedem, B., & Fokianos, K. 2002, Regressions Models for Time Series Analysis, Wiley Series in Probability and Statistics*
18) *Kotsiantis S.B., 2007, Supervised Machine Learning: A Review of Classification Technique, Informatica Journal*
19) *Sarwar M.M.S., Shahzad S., Ahmad I., 2013, Cyclomatic complexity: The nesting problem, Department of Computer Science University of Peshawar*
20) *Maimon O. & Rokach L., 2005, Data Mining and Knowledge Discovery Handbook, Springer-Verlag, Berlin, Heidelberg*
21) *Okutan, Ahmet, and Olcay Taner Yıldız.(2014) "Software defect prediction using Bayesian networks." Empirical Software Engineering 154-181.*
22) *Olsen, David L. and Delen,,(2008) " Advanced Data Mining Techniques ", Springer, 1st edition, page 138, ISBN 3-540-76016-1, .*
23) *Peng C.-Y. J., Lee K.L., and Ingersoll G.M., 2002, An introduction to logistic regression analysis and reporting, The Journal of Educational Research*
24) *Powers D.M.W., 2011, Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation, Journal of Machine Learning Technologies, p.37–63*
25) *Punitha K., Chitra S., 2013, Software defect prediction using software metrics - A survey, Conference: Information Communication and Embedded Systems (ICICES)*
26) *Rizwan S., et al., 2017, Empirical Study on Software Bug Prediction, School of Computer Science and Technology, Harbin, International Conference on Software and e-Business, p.55-59*
27) *Shalev-Shwartz S.and Ben-Davi S., 2014, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, p.347*
28) *Shalev-Shwartz S.and Ben-Davi S., 2014, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, Pages 123-125*
29) *Shalev-Shwartz S.and Ben-Davi S., 2014, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, Pages 126-128*
30) *Sharma and P. Chandra, "Software Fault Prediction Using MachineLearning Techniques," Smart Computing and Informatics. Springer, Singapore, 2018. 541-549.*
31) *Taheri S., Mammadov M., 2013 LEARNING THE NAIVE BAYES CLASSIFIER WITH OPTIMIZATION MODELS, Int. J. Appl. Math. Comput. S, Vol. 23, No. 4, 787–7*
32) *Tashman, L. J. & Leach, M. L. (1991), 'Automatic forecasting software: a survey and evaluation', International Journal of Forecasting 7, 209–230.*
33) *Taylor S.J. & Letham B., 2018, Forecasting at Scale, The American Statistician, Volume 72*

34) *Tim Menzies, Jeremy Greenwald, Art Frank. (2007). Data Mining Static Code Attributes to Learn Defect Predictors, IEEE Transactions on Software Engineering, 32(11), 1-12.*

35) *U. o. U. Department of Mathematics, 2017, Using the receiver operating characteristic (roc)curve to analyze a classification model [pdf]*
36) *Webb G.I., et al., 2011, Genetic and evolutionary algorithms, Encyclopedia of Machine Learning, Springer US, pages 456–457*
37) *Wijaya A., Bisri A., 2016, Hybrid decision tree and logistic regression classifier for email spam detection, 8th International Conference on Information Technolo*
38) *Zhou, Y.; Hareton, L. Empirical analysis of object-oriented design metrics for predicting high and low severity faults Software Engineering. IEEE Trans. 2006, 32, 771–789*