Π.Μ.Σ. ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΚΑΤΕΥΘΥΝΣΗ - ΕΠΙΣΤΗΜΗ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ Η/Υ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Υπολογιστική σύγκριση αλγορίθμων κατηγοριοποίησης

Ονοματεπώνυμο σπουδαστή: Φίλιππος Βαγγέλης

Αριθμός Μητρώου: MAI20006

Επιβλέπων Καθηγητής: Σαμαράς Νικόλαος

Θεσσαλονίκη, Φεβρουάριος 2022

Περίληψη

Η συγκεκριμένη διπλωματική εργασία έχει ως σκοπό την μελέτη, την ανάλυση αλλά και την υπολογιστική σύγκριση των βασικότερων αλγορίθμων που πραγματοποιούν κατηγοριοποίηση.

Αρχικά, θα πραγματοποιηθεί μια εισαγωγική περιγραφή της διαδικασίας της κατηγοριοποίησης. Στην συνέχεια, θα περιγραφούν και θα αναλυθούν εκτενέστερα κάποιοι από τους βασικότερους αλγόριθμους που επιτυγχάνουν την παραπάνω διαδικασία. Οι αλγόριθμοι αυτοί είναι οι εξής: Logistic Regression, Decision Trees, Boosted Trees, Random Forest, Neural Networks και K-Nearest Neighbor.

Τέλος, η υπολογιστική τους σύγκριση θα πραγματοποιηθεί μέσω διάφορων συνόλων δεδομένων που συγκεντρώθηκαν από το διαδίκτυο και αναλύθηκαν από όλους τους παραπάνω αλγόριθμους με την βοήθεια κάποιων έτοιμων βιβλιοθηκών στην προγραμματιστική γλώσσα Python.

Περιεχόμενα

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

1. Εισαγωγή

1.1 Αντικείμενο της Εργασίας

Η εποχή που ζούμε διακρίνεται για τον όγκο των δεδομένων και πληροφοριών που παράγονται και συλλέγονται σε ολόκληρο τον κόσμο μέσω του διαδικτύου. Η αξιοποίηση των παραπάνω με σκοπό την βελτίωση συγκεκριμένων τομέων και την λύση πολλών προβλημάτων πραγματοποιείται από την επιστήμη της Πληροφορικής και συγκεκριμένα μέσω του πεδίου της Μηχανικής Μάθησης, το οποίο έχει την ικανότητα να μετατρέψει αυτά τα δεδομένα σε σημαντική και ιδιαίτερα χρήσιμη γνώση. Η μηχανική μάθηση δημιουργήθηκε από την ένωση δύο διαφορετικών πεδίων: το πεδίο της στατιστικής και το πεδίο της επιστήμης των υπολογιστών και συγκεκριμένα της τεχνητής νοημοσύνης.

Η μηχανική μάθηση στην ουσία, χρησιμοποιώντας μεγάλες συλλογές από δεδομένα, έχει την ικανότητα να αναγνωρίζει διαφορετικά μοτίβα και στην συνέχεια έχοντας ως βάση τα παραπάνω, παρέχει την δυνατότητα να προβλέπει μελλοντικά αποτελέσματα ή να παίρνει αποφάσεις.

Μερικά από τα πλεονεκτήματα που παρέχει ο τομέας αυτός είναι τα εξής:

- Διαρκής βελτίωση αποφάσεων
- Ευρύς τομέας εφαρμογής
- Καινοτομία

- Ταχύτητα και Ικανότητα Προσαρμογής

Εικόνα 1. Μηχανική μάθηση(Πηγή: https://becominghuman.ai/an-introduction-to-machine-learning-33a1b5d3a560)

Η μηχανική μάθηση περιέχει αλγόριθμους πολλών κατηγοριών αλλά κυρίως χωρίζεται σε 3 βασικούς τομείς, την επιβλεπόμενη μάθηση, την μη επιβλεπόμενη μάθηση και την ενισχυτική μάθηση. Η επιβλεπόμενη μάθηση χρησιμοποιεί τις τεχνικές της κατηγοριοποίησης και της παλινδρόμηση, ενώ η μη επιβλεπόμενη τις τεχνικές του μετασχηματισμού και της συσταδοποίησης. Η ενισχυτική μάθηση πραγματοποιείται μέσω των πρακτόρων, οι οποίοι λαμβάνουν αποφάσεις θετικές ή αρνητικές ανάλογα το περιβάλλον.

Η συγκεκριμένη εργασία ασχολείται με την τεχνική της κατηγοριοποίησης και με κάποιους από τους βασικότερους αλγόριθμους της κατηγορίας αυτής. Οι αλγόριθμοι αυτοί είναι οι εξής:

- Λογιστική Παλινδρόμηση(Logistic Regression)
- Δέντρα Απόφασης (Decision Trees)
- Ενισχυμένα Δέντρα (Boosted Trees)
- Τυχαία Δάση (Random Forest)
- Νευρωνικά Δίκτυα (Neural Networks)
- Κ-Εγγύτεροι Γείτονες (K-Nearest Neighbors)

Οι παραπάνω αλγόριθμοι θα αναλυθούν τόσο θεωρητικά, όσο και πρακτικά σε αυτή την εργασία και θα χρησιμοποιηθούν πάνω σε διαφορετικά σύνολα δεδομένων με σκοπό την εύρεση της υπολογιστικής μελέτης τους και αργότερα την σύγκριση μεταξύ τους.

Αναφέροντας την έννοια της εύρεσης της υπολογιστικής μελέτης των αλγορίθμων, εννοείται η εύρεση της πολυπλοκότητας των αλγορίθμων και η υπολογιστική τους συμπεριφορά. Η διαδικασία αυτή πραγματοποιείται με την πολλαπλή εκτέλεση των αλγορίθμων σε διάφορες συλλογές δεδομένων, κάτι που θα εκτελεσθεί και στην συγκεκριμένη εργασία με σκοπό αργότερα την σύγκριση των αποτελεσμάτων μεταξύ τους. Όσων αφορά την δημιουργία έγκυρων αποτελεσμάτων σχετικά με την συμπεριφορά των αλγορίθμων, τα δεδομένα χρειάζεται να πληρούν συγκεκριμένες προϋποθέσεις. Μία από τις βασικότερες προϋποθέσεις έχει να κάνει με το μέγεθος των δεδομένων, αφού από δεδομένα μικρού μεγέθους δεν θα παραχθούν ασφαλή αποτελέσματα. Στο παρελθόν, πολλές εργασίες έχουν δημιουργηθεί για το πως πρέπει να πραγματοποιούνται τα πειράματα για την εύρεση της πολυπλοκότητας των αλγορίθμων και για τον τρόπο διενέργειας των υπολογιστικών μελετών.

1.2 Διάρθρωση της Εργασίας

Η συνέχεια της εργασίας προκύπτει με την λεπτομερή ανάλυση της τεχνικής της κατηγοριοποίησης στο Κεφάλαιο 2. Ακόμα, παρουσιάζονται οι τομείς στους οποίους η συγκεκριμένη τεχνική βρίσκει εφαρμογή. Στο 3ο Κεφάλαιο, παρουσιάζονται οι βασικότεροι, γνωστότεροι αλλά και αποτελεσματικότεροι αλγόριθμοι κατηγοριοποίησης και εξετάζονται τόσο θεωρητικά όσο και πρακτικά.

Το 4ο Κεφάλαιο ασχολείται με την υπολογιστική μελέτη των παραπάνω αλγορίθμων, αφού αρχικά παρουσιαστεί και εξηγηθεί όλη η θεωρία πίσω από αυτόν τον τομέα. Η πρακτική ανάλυση πραγματοποιείται πάνω σε διαφορετικά σύνολα δεδομένων και ο κώδικας που χρησιμοποιήθηκε παρατίθεται στο τέλος της εργασίας.

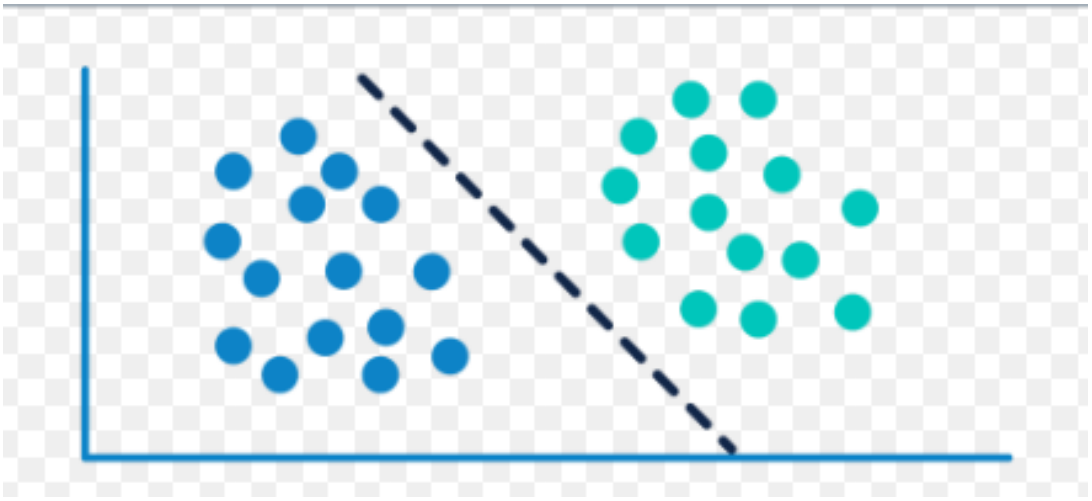
Το 5ο Κεφάλαιο ασχολείται με την σύγκριση των αποτελεσμάτων που έχουν παραχθεί από το προηγούμενο κεφάλαιο και την εξαγωγή συμπερασμάτων πάνω σε αυτή την διεργασία.

2. Κατηγοριοποίηση

2.1 Εισαγωγή

Η κατηγοριοποίηση είναι μια από τις πιο αποτελεσματικές και δημοφιλής τεχνικές που χρησιμοποιούνται στην ανάλυση δεδομένων και στην μηχανική μάθηση. Στόχος της είναι η παραγωγή μοντέλων τα οποία θα περιγράφουν και θα κατηγοριοποιούν διαφορετικές κλάσεις των δεδομένων με βάση κοινά τους χαρακτηριστικά. Αυτά τα μοντέλα ονομάζονται κατηγοριοποιητές και έχουν την ικανότητα να διακρίνουν αλλά και να προβλέπουν τάξεις που περιέχουν ονομαστικές μεταβλητές. Στις μέρες μας, χρησιμοποιείται σε καθημερινή βάση η τεχνική της κατηγοριοποίησης από πολλές διαφορετικές εταιρείες και επιχειρήσεις. Για παράδειγμα, δημιουργούνται μοντέλα κατηγοριοποίησης για την διάγνωση ιατρικών πορισμάτων, για την ασφαλή απόφαση έγκρισης κάποιου δανείου ή για την αναγνώριση κάποιας απάτης(Han, Kamber & Pei,2012).

Απαραίτητη προϋπόθεση πριν την εκκίνηση δημιουργίας κάποιου μοντέλου κατηγοριοποίησης είναι η καλή γνώση και κατανόηση των δεδομένων. Έπειτα, από την στιγμή που βρίσκεται διαθέσιμος ένας ικανοποιητικός αριθμός δεδομένων, τα δεδομένα χωρίζονται σε δεδομένα που χρησιμοποιούνται στην εκπαίδευση και σε δεδομένα που χρησιμοποιούνται στον έλεγχο. Τα δεδομένα που χρησιμοποιούνται στην εκπαίδευση καθορίζουν και ρυθμίζουν τις απαραίτητες παραμέτρους του μοντέλου κατά την διαδικασία εκπαίδευσης, ενώ με την χρήση των υπολοίπων πραγματοποιείται ο έλεγχος και ο υπολογισμός της ακρίβειας του παραχθέντος μοντέλου.



Εικόνα 2. Διάγραμμα Μοντέλου Κατηγοριοποίησης(Πηγή: https://machinelearningmind.com/2019/11/02/machine-learning-classification/)

Ένα από τα αρχικά στάδιο της κατηγοριοποίησης είναι η προεπεξεργασία των δεδομένων, αφού η μη πραγματοποίησης της ή η λάθος εφαρμογή της έχει ως αποτέλεσμα την παραγωγή μεγάλων σφαλμάτων και ανακριβειών στο μοντέλο. Πολύ συχνά παρατηρείται από τους ανθρώπους που ασχολούνται με αυτόν τον τομέα, το φαινόμενο της δημιουργίας συνόλων δεδομένων που περιέχουν ασυνέπειες, ελλιπή και λανθασμένα δεδομένα. Καθ'αυτόν τον τρόπο, κρίνεται αναγκαίο το λεγόμενο 'καθάρισμα' των δεδομένων με την χρήση διάφορων τεχνικών πρωτού χρησιμοποιηθούν. Κάποιες από αυτές είναι η εξής:

- Αγνόηση των δεδομένων που λείπουν εάν είναι λίγα στον αριθμό
- Γέμισμα των δεδομένων που λείπουν με την χρήση του μέσου όρου των τιμών του χαρακτηριστικού
- Γέμισμα των δεδομένων που λείπουν με μια χαρακτηριστική λέξη(συνήθως N/A)

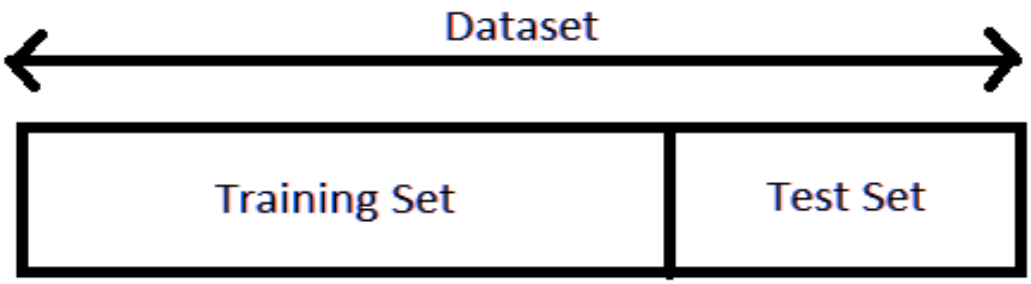
2.2 Γενική προσέγγιση προβλημάτων κατηγοριοποίησης

Η τεχνική της κατηγοριοποίησης είναι κατά κάποιον τρόπο μια διαρκής εκτίμηση δημιουργίας μοντέλων από σύνολα δεδομένων με σκοπό την δημιουργία κλάσεων. Για την πραγματοποίηση του προηγούμενου χρησιμοποιούνται αλγόριθμοι μερικοί εκ των οποίων είναι τα Δέντρα Απόφασης, τα Νευρωνικά Δίκτυα και η Λογιστική Παλινδρόμηση. Μέσω των αλγορίθμων επιδιώκεται η δημιουργία του μοντέλου που ταιριάζει και αποδίδει καλύτερα σε ένα συγκεκριμένο σύνολο δεδομένων και παράγει αποτελέσματα με καλύτερη ακρίβεια(Han, Kamber & Pei,2012).

Αρχικά, η κατηγοριοποίηση των δεδομένων αποτελείται από δύο επίπεδα. Το πρώτο είναι το επίπεδο της εκπαίδευσης και δημιουργίας του μοντέλου και το δεύτερο το επίπεδο της εφαρμογής του μοντέλου σε άγνωστα δεδομένα με σκοπό την εύρεση της αποδοτικότητας και ακρίβειάς του. Από την στιγμή που η κλάση που ανήκει κάθε στιγμιότυπο του συνόλου δεδομένων είναι γνωστή πριν την ολοκλήρωση του μοντέλου και το μοντέλο εκπαιδεύεται με βάση αυτήν, η τεχνική της κατηγοριοποίησης που χρησιμοποιείται λέγεται ότι βρίσκεται εντός του πεδίου της εποπτευόμενης μάθησης. Η εποπτευόμενη μάθηση έρχεται σε αντίθεση με την μη εποπτευόμενη μάθηση, στην οποία τα δεδομένα πριν την δημιουργία του μοντέλου

δεν έχουν πληροφορία για την κλάση που βρίσκονται και οι κλάσεις δημιουργούνται από το ίδιο το μοντέλο με βάση τα κοίνα χαρακτηριστικά των στιγμιοτύπων του συνόλου.

Από το σύνολο δεδομένων ξεχωρίζεται ένας αριθμός δεδομένων, όπου συνήθως αυτός είναι κοντά στο 60-70% ολόκληρου του συνόλου. Αυτά τα δεδομένα ονομάζονται σύνολο εκπαίδευσης και χρησιμοποιούνται στην εκπαίδευση του μοντέλου, ενώ τα υπόλοιπα ονομάζονται σύνολο ελέγχου και χρησιμοποιούνται με το πέρας της δημιουργίας του μοντέλου με σκοπό την εύρεση της αποτελεσματικότητας και ακρίβειας που παράγει το μοντέλο(Han, Kamber & Pei,2012).



Εικόνα 3. Διαμερισμός του συνόλου δεδομένων(Πηγή: https://data-flair.training/blogs/train-test-set-in-python-ml/).

2.3 Απόδοση της κατηγοριοποίησης

Το τελικό αποτέλεσμα και εάν αυτό είναι ικανοποιητικό εξετάζει η ακρίβεια του αλγορίθμου. Η ακρίβεια ενός αλγορίθμου σε ένα γνωστό σύνολο δεδομένων είναι το ποσοστό επί της εκατό του συνόλου ελέγχου που κατηγοριοποιήθηκε σωστά από το μοντέλο. Η ακρίβεια του μοντέλου εν τέλει φανερώνει το πόσο καλά ανταποκρίθηκε ο αλγόριθμος πάνω στο συγκεκριμένο σύνολο δεδομένων στο οποίο και εκπαιδεύτηκε. Εάν η εκτίμηση αυτή βρίσκεται πάνω από το επίπεδο που κρίνεται αναγκαίο, το μοντέλο έχει την ικανότητα να εκτελεί προβλέψεις σε διαφορετικά σύνολα δεδομένων για τα οποία χρειάζεται να πραγματοποιηθεί κατηγοριοποίηση(Han, Kamber & Pei,2012). Ακόμα μια εφαρμογή της ακρίβειας των αλγορίθμων βρίσκεται στην μεταξύ τους σύγκριση μέσω αυτής, αφού η ακρίβεια αποτελεί μαζί με την ταχύτητα, την ανθεκτικότητα, την επεκτασιμότητα και την ερμηνευσιμότητα, διάφορα μέτρα σύγκρισης των αλγορίθμων. Ο μαθηματικός τύπος της ακρίβειας δίνεται παρακάτω:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Εικόνα 4. Ακρίβεια(Πηγή: https://developers.google.com/machine-learning/crash-course/classification/accuracy).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

Εικόνα 5. Μαθηματικός τύπος ακρίβειας(Πηγή: https://developers.google.com/machine-learning/crash-course/classification/accuracy).

3. Αλγόριθμοι κατηγοριοποίησης

3.1 Λογιστική Παλινδρόμηση

Οι τεχνικές της γραμμικής η μη γραμμικής παλινδρόμησης μπορούν να χρησιμοποιηθούν τόσο για παλινδρόμηση και πρόβλεψη μιας συνεχής μεταβλητής, όσο και για την διαδικασία της ταξινόμησης. Ο τρόπος που αυτό πραγματοποιείται είναι ο εξής: για κάθε κλάση πραγματοποιείται μια παλινδρόμηση για την οποία θέτονται τα αποτελέσματα ίσα με ένα για τα στιγμιότυπα που θα χρησιμοποιηθούν κατά την εκπαίδευση και ανήκουν στην κλάση και μηδέν για αυτά που δεν ανήκουν. Το παραπάνω σημαίνει μια γραμμική έκφραση της κλάσης, η οποία όταν χρησιμοποιηθεί σε άγνωστα δεδομένα θα έχει την ικανότητα να υπολογίζει και να εντοπίζει εάν τα δεδομένα αυτά ανήκουν στην κλάση ή όχι. Η μέθοδος αυτή ονομάζεται Γραμμική Παλινδρόμηση με πολλαπλή απόκριση(Witten, Eibe & Hall,2011).

Συνήθως, η μέθοδος αυτή παρουσιάζει καλά αποτελέσματα στην πράξη, εμφανίζοντας δύο κύρια μειονεκτήματα. Αρχικά, οι τιμές που παράγει δεν μπορούν να θεωρηθούν σωστές πιθανότητες, αφού δεν βρίσκονται εντός των ορίων 0-1. Επίσης, η παλινδρόμηση ελαχίστων τετραγώνων προϋποθέτει ότι τα σφάλματα είναι στατιστικά ανεξάρτητα και ακολουθούν μια κανονική κατανομή με την ίδια τυπική απόκλιση, κάτι που παραβιάζεται στην συγκεκριμένη περίπτωση, αφού η τεχνική αυτή χρησιμοποιείται στην κατηγοριοποίηση και οι παρατηρήσεις παίρνουν μόνο τις τιμές 0 και 1 (Witten, Eibe & Hall,2011).

Χρησιμοποιώντας μια παρόμοια τεχνική που ονομάζεται Λογιστική Παλινδρόμηση όλα τα παραπάνω μειονεκτήματα αποφεύγονται, αφού χρησιμοποιείται ένα γραμμικό μοντέλο βασισμένο στην μετασχηματισμένη μεταβλητή στόχου και με αυτό τον τρόπο δεν γίνεται προσπάθεια προσέγγισης των τιμών 0 και 1 απευθείας. Η εκτίμηση των παραμέτρων στην συγκεκριμένη μέθοδο πραγματοποιείται με την επιλογή των πιο πιθανοφανών τιμών των παραμέτρων(λόγος πιθανοφάνειας), κάτι διαφορετικό από την γραμμική παλινδρόμηση, η οποία πραγματοποιούσε το παραπάνω με την μέθοδο των ελάχιστων τετραγώνων.

Η ανάπτυξη του μοντέλου της λογιστικής παλινδρόμησης οδηγεί στην πρόβλεψη της πιθανότητας εμφάνισης κάποιου γεγονότος, όταν τα δεδομένα του γεγονότος χρησιμοποιηθούν στην εξίσωση της λογιστικής καμπύλης, η οποία είναι η εξής:

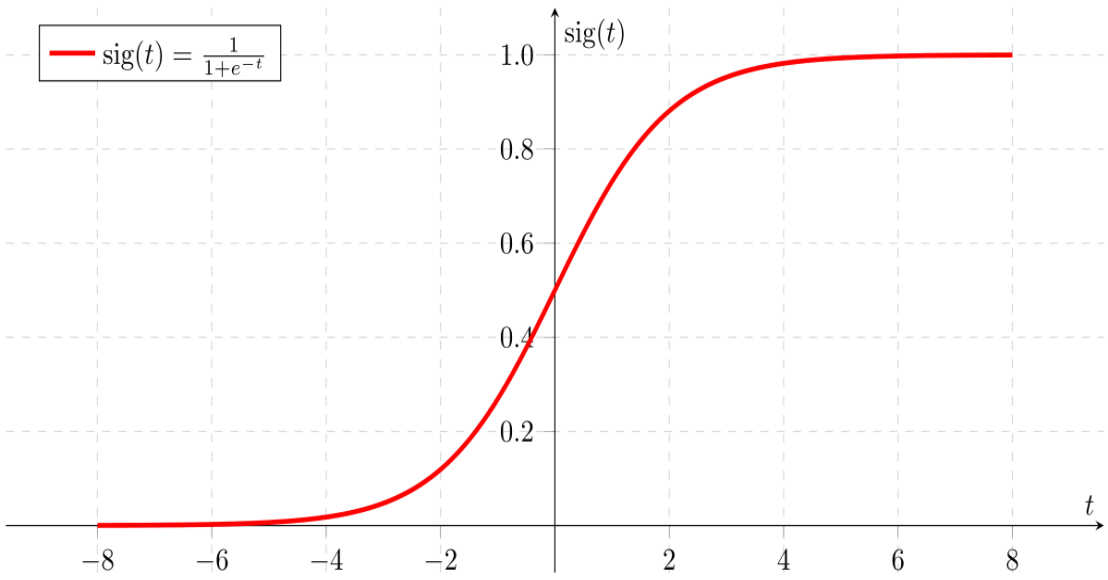
$$f(z)=\frac{e^{z}}{1+e^{z}}=\frac{1}{1+e^{-z}}$$

όπου το z είναι η μεταβλητή και το f(z) αποτέλεσμα που παράγεται από αυτήν.

Ο λογιστικός μετασχηματισμός ο οποίος χρησιμοποιείται είναι ο εξής:

$$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$$

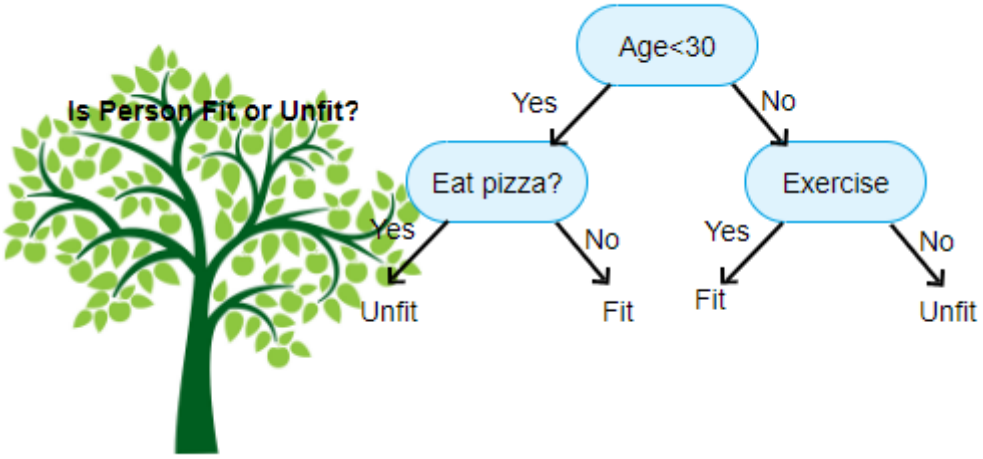
Το β_0 είναι το ύψος της κλίσης της γραμμικής παλινδρόμησης και τα β_i είναι οι συντελεστές των μεταβλητών που συμμετέχουν στην εξίσωση, εκφράζοντας το επίπεδο που συνεισφέρει η κάθε μεταβλητή στο συνολικό αποτέλεσμα. Οι θετικές τιμές των συντελεστών συνεισφέρουν στην αύξηση της πιθανότητας της επιτυχημένης έκβασης του γεγονότος, ενώ οι αρνητικές στην μείωση.



Εικόνα 6. Διάγραμμα Λογιστικής Παλινδρόμησης(πηγή: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc)

3.2 Δέντρα Απόφασης

Ένας από τους πιο γνωστούς αλγόριθμους κατηγοριοποίησης που προέρχεται από τους αλγόριθμους της κατηγορίας διαίρει και βασίλευε(divide and conquer) είναι τα δέντρα απόφασης. Ένα δέντρο απόφασης είναι μια δομή παρόμοια με αυτή του διαγραμμάτος ροής, με κάθε εσωτερικό κόμβο (node) να δηλώνει μια δοκιμή σε ένα χαρακτηριστικό, κάθε κλαδί (branch) το αποτέλεσμα της δοκιμής και κάθε φύλλο (leaf node) μια ετικέτα της κλάσης που ανήκει.



Εικόνα 7. Δέντρο Απόφασης (πηγή : https://www.aitimejournal.com/@akshay.chavan/a-comprehensive-guide-to-decision-tree-learning)

Το παραπάνω δέντρο απόφασης παρουσιάζει την περίπτωση που ένα άτομο κρατιέται σε φόρμα ή όχι. Κάθε εσωτερικός κόμβος παρουσιάζει την δοκιμή των χαρακτηριστικών, τα οποία είναι το εάν τρώει πίτσα και εάν ασκείται. Κάθε φύλλο παρουσιάζει από μια κλάση, δηλαδή εάν κρατιέται σε φόρμα ή όχι στο συγκεκριμένο παράδειγμα.

Ένα από τα σημαντικά πλεονεκτήματα των δέντρων αποφάσεων είναι ότι μπορούν να επεξεργαστούν δεδομένα πολλών διαστάσεων. Η κατασκευή ενός δέντρου δεν απαιτεί κάποια εξειδικευμένη γνώση στον τομέα αυτό ή την ρύθμιση κάποιας ασυνήθιστης ή δυσνόητης παραμέτρου και συνήθως πετυχαίνει καλή ακρίβεια, κάτι που κάνει την τεχνική αυτή ιδιαίτερα δημοφιλή για την εξόρυξη γνώσης από τα δεδομένα. Σήμερα, η τεχνική αυτή χρησιμοποιείται σε πολλούς τομείς όπως η ιατρική, η οικονομία, η αστρονομία και η μοριακή βιολογία(Han, Kamber & Pei,2012).

Ο αλγόριθμος περιγράφεται παρακάτω:

- **Δεδομένα Εισόδου**: παραδείγματα, ένα σύνολο παραδειγμάτων
  χαρακτηριστικά, ένα σύνολο χαρακτηριστικών
  προεπιλογή, προεπιλεγμένη κατηγορία

-**if** σύνολο είναι κενό **then return** προεπιλογή

-**else if** όλα στο παραδείγματα έχουν την ίδια κατηγορία **then return** κατηγορία

-**else if** χαρακτηριστικά είναι κενό **then return** Majority-Class(παραδείγματα)

-**else**

- best ← Choose-Attribute(χαρακτηριστικά, παραδείγματα)
- tree ← νέο δέντρο αποφάσεων με έλεγχο ρίζας το χαρακτηριστικό best
- m ← Majority-Value(παραδείγματα)

-**for each** τιμή u_i του best **do**

- παραδείγματα$_i$ ← {στοιχεία από τα παραδείγματα με best = u_i}
- subtree ← Decision-Tree-Learning(παραδείγματα$_i$, χαρακτηριστικά-best,m)
- προσθήκη διακλάδωσης στο tree με ετικέτα u_i και υποδέντρο = subtree
- **return** tree

Η διαδικασία κατασκευής ενός δέντρου απόφασης είναι επαναληπτική. Στην αρχή, ένα χαρακτηριστικό επιλέγεται και αναφέρεται ως ρίζα του δέντρου (root). Στην συνέχεια, κατασκευάζεται μια ακμή και ένας κόμβος για κάθε διακριτή τιμή του χαρακτηριστικού. Τα παραπάνω βήματα επαναλαμβάνονται διαρκώς, ώσπου όλα τα χαρακτηριστικά να εισαχθούν στους κόμβους του δέντρου.

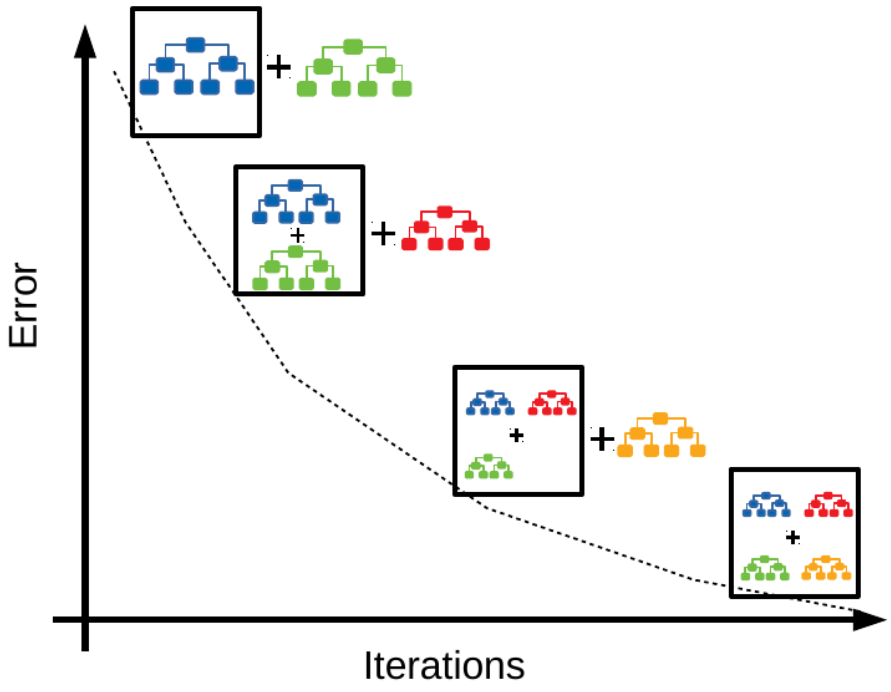
Η επιλογή των καταλληλότερων χαρακτηριστικών με σκοπό την δημιουργία του δέντρου με την μεγαλύτερη ακρίβεια πραγματοποιείται μέσω της εύρεσης της Εντροπίας. Η Εντροπία (Entropy) παρουσιάζει τον βαθμό αβεβαιότητας που δημιουργεί ένα χαρακτηριστικό, δηλαδή τον βαθμό ανομοιογένειας που εντάσσεται στον κόμβο. Ο υπολογισμός της προκύπτει από την παρακάτω εξίσωση:

$$E(N) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

Όπου με N παρουσιάζεται ο υπό εξέταση κόμβος και με c ο αριθμός των κλάσεων. Το p_i είναι ίσο με τον αριθμό των αντικειμένων που ανήκουν στον κόμβο N και στην κλάση i διά τον συνολικό αριθμό των αντικειμένων που ανήκουν στον κόμβο N ($p_i = n_i/n$).

3.3 Ενισχυμένα Δέντρα

Η ενίσχυση(boosting) είναι μια μέθοδος που χρησιμοποιείται στην κατηγοριοποίηση και έχει να κάνει με μια επαναληπτική διαδικασία κατά την οποία στην διάρκεια των επαναλήψεων η κατανομή του συνόλου εκπαίδευσης αλλάζει και προσαρμόζεται έτσι ώστε οι ταξινομητές να εστιάζουν στα παραδείγματα που είναι δύσκολο να ταξινομηθούν. Σε κάθε γύρο, διάφορα βάρη μοιράζονται σε κάθε παράδειγμα και αυτά αλλάζουν ή προσαρμόζονται με σκοπό την βελτίωση της ακρίβειας. Η μέθοδος αυτή χρησιμοποιεί πολλούς αδύναμους ταξινομητές με σκοπό να δημιουργήσει έναν ισχυρό.

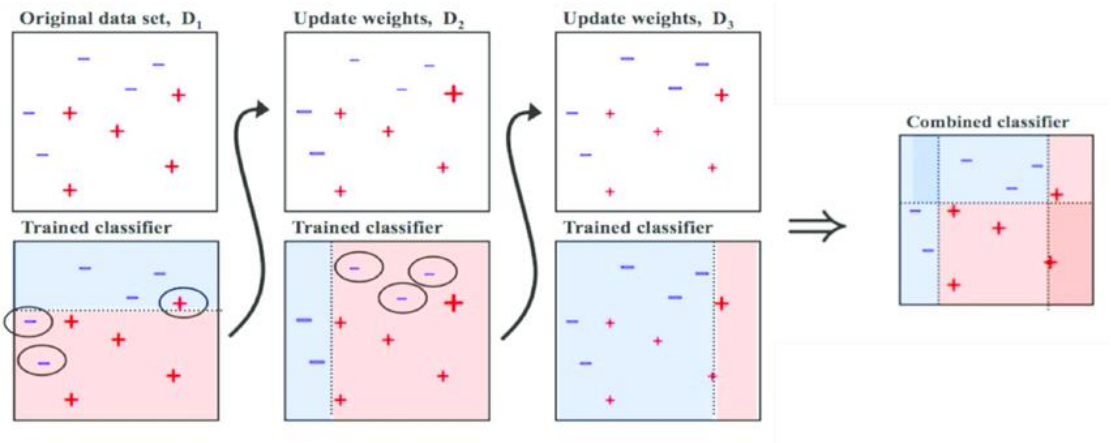


Εικόνα 8. Διάγραμμα επαναλήψεων ενισχυμένων δέντρων(πηγή: http://tvas.me/articles/2019/08/26/Block-Distributed-Gradient-Boosted-Trees.html)

Όπως παρατηρείται και στην παραπάνω εικόνα, με το πέρας των επαναλήψεων το σφάλμα μειώνεται και το μοντέλο γίνεται πιο ισχυρό, δηλαδή πιο ακριβές.

Ένα παράδειγμα ενός αλγόριθμου που χρησιμοποιεί αυτή την μέθοδο είναι ο αλγόριθμος Adaboost. Ο αλγόριθμος αυτός παράγει μια ακολουθία αδύναμων ταξινομητών και σε κάθε επανάληψη διαλέγει τον καλύτερο ταξινομητή έχοντας ως μέτρο σύγκρισης τα βάρη του κάθε δείγματος. Τα δείγματα που έχουν κατηγοριοποιηθεί λανθασμένα σε κάθε επανάληψη δέχονται περισσότερο βάρος στην επόμενη και τα δείγματα που έχουν κατηγοριοποιηθεί σωστά δέχονται λιγότερο. Το παραπάνω έχει ως αποτέλεσμα τα δείγματα που είναι δύσκολο να κατηγοριοποιηθούν να αυξάνουν ολοένα το βάρος τους μέχρι ο αλγόριθμος να επιλέξει ένα μοντέλο που θα έχει την ικανότητα να ταξινομεί σωστά τα συγκεκριμένα δείγματα. Ο αλγόριθμος Adaboost παρουσιάζεται παρακάτω:

1. Ορίστε μια κλάση με τιμή +1 και μια ακόμα με τιμή -1
2. Το βάρος κάθε δείγματος αρχικοποιείται με την ίδια τιμή(1/n)
3. Για k = 1 εώς K κάνε
4. Δημιουργία ενός ασθενούς ταξινομητή χρησιμοποιώντας τα σταθμισμένα δείγματα και υπολογισμός του k^{th} σφάλματος της λανθασμένης ταξινόμησης του μοντέλου (misclassification error-errk)
5. Υπολογισμός της k^{th} μεταβλητής βάρους μέσω ln((1-errk)/errk)
6. Αναβάθμιση των βαρών δίνοντας περισσότερο βάρος στα δείγματα που κατηγοριοποιήθηκαν λανθασμένα και λιγότερο σε αυτά που κατηγοριοποιήθηκαν σωστά
7. Τέλος
8. Υπολογισμός της πρόβλεψης του ενισχυμένου ταξινομητή για κάθε δείγμα πολλαπλασιάζοντας την k^{th} μεταβλητή βάρους με την k^{th} πρόβλεψη του μοντέλου και προσθέτοντας αυτές. Εάν αυτό που θα προκύψει είναι θετικό, τότε το δείγμα ανήκει στην +1 τάξη, αλλιώς στην -1 τάξη.



Εικόνα 9. Adaboost αλγόριθμος(πηγή: https://towardsdatascience.com/understanding-adaboost-for-decision-tree-ff8f07d2851)

3.4 Τυχαία Δάση

Μια διαφορετική μέθοδος από την ενίσχυση(boosting) που παρουσιάστηκε παραπάνω, είναι το bagging(bootstrap aggregation). Η συγκεκριμένη μέθοδος δουλεύει ως εξής: Δεδομένου ενός συνόλου D από d πλειάδες. Για επαναλήψεις i(i = 1,2,…,k),ένα σύνολο εκπαίδευσης Di από d πλειάδες δειγματίζεται με αντικατάσταση από το αρχικό σύνολο D. Κάθε σύνολο εκπαίδευσης ονομάζεται bootstrap σύνολο και μερικά από τις αρχικές πλειάδες του D μπορεί να μην συμπεριλαμβάνονται στο Di,ενώ άλλες να συμπεριλαμβάνονται, αφού χρησιμοποιείται δειγματοληψία με αντικατάσταση. Ένα μοντέλο κατηγοριοποιητή εκπαιδεύεται από κάθε σύνολο εκπαίδευσης Di και στην περίπτωση που χρειάζεται να πραγματοποιηθεί κάποια πρόβλεψη σε άγνωστο σύνολο X, το κάθε μοντέλο κάνει την πρόβλεψή του και την επιστρέφει σαν ψήφο στο X. Η διαδικασία του bagging ολοκληρώνεται με την δημιουργία της πρόβλεψης από τον μέσο όρο όλων των ψήφων

των μοντέλων και αρκετά συχνά πετυχαίνει εξαιρετικά αποτελέσματα (Han, Kamber & Pei,2012).

Κάθε βήμα του αλγορίθμου παρουσιάζεται παρακάτω:

Αλγόριθμος Bagging

Δεδομένα Εισόδου:

- D, ένα σύνολο από d πλειάδες εκπαίδευσης
- k, ο αριθμός των μοντέλων που θα χρησιμοποιηθούν
- ένας τρόπος κατηγοριοποιήσης(π.χ. Δέντρα Απόφασης)

Δεδομένα Εξόδου:

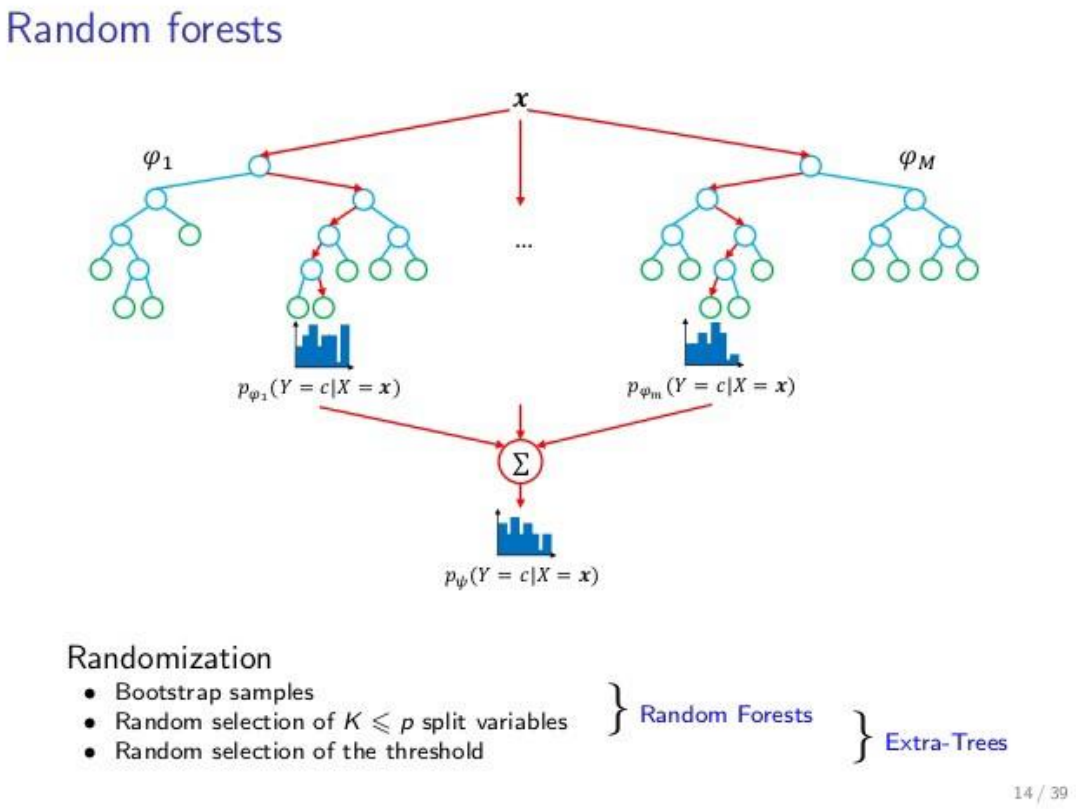
- M, ένα σύνθετο μοντέλο κατηγοριοποιητής

Μέθοδος:

1. Για i = 1 εώς k κάνε:
2. Δημιουργία δείγμα boostrap,Di,δειγματίζοντας το D με αντικατάσταση;
3. Χρησιμοποίησε το Di και τον τρόπο κατηγοριοποίησης για την δημιουργία του μοντελου M;
4. Τέλος Για

Για την πρόβλεψη ενός συνόλου X, κάθε μοντέλο k θα κάνει μια πρόβλεψη(ψήφος) και θα επιστραφεί το μοντέλο M με την πλειοψηφία των ψήφων(Han, Kamber & Pei,2012).

Τα τυχαία δάση δεν είναι τίποτα άλλο από την παραπάνω μέθοδο, χρησιμοποιώντας σαν τρόπο κατηγοριοποίησης τα Δέντρα Απόφασης. Δηλαδή, το αποτέλεσμα δεν θα προέκυπτε από έναν κατηγοριοποιητή-δέντρο, αλλά από πολλούς κατηγοριοποιητές-δέντρα ή αλλιώς ένα δάσος. Η παρακάτω εικόνα παρουσιάζει τον τρόπο που λειτουργεί η συγκεκριμένη μέθοδος.



Εικόνα 10. Random Forest (πηγή: https://www.kdnuggets.com/2017/10/random-forests-explained.html)

3.5 Μέθοδος Κ-Κοντινότερων Γειτόνων

Η μέθοδος των Κ-Κοντινότερων Γειτόνων χρησιμοποιείται ευρέως σήμερα, κυρίως στον τομέα της αναγνώρισης μοτίβων, αλλά έκανε την εμφάνισή της πρώτα στις αρχές του 1950. Ο κατηγοριοποιητής αυτός είναι βασισμένος στην εκμάθηση κατ' αναλογία, αφού συγκρίνει ένα σύνολο εκπαίδευσης με ένα σύνολο ελέγχου, προκειμένου να βρει ομοιότητες. Οι πλειάδες εκπαίδευσης περιγράφονται με n χαρακτηριστικά και κάθε πλειάδα αντιπροσωπεύει ένα σημείο σε ένα χώρο n-διαστάσεων. Στην συνέχεια, όταν δοθεί μια άγνωστη πλειάδα, ο κατηγοριοποιητής ψάχνει για k-πλειάδες εκπαίδευσης που παρουσιάζουν ομοιότητες με αυτήν(Han, Kamber & Pei,2012). Η μέθοδος αυτή ανήκει στους lazy learners κατηγοριοποιητές, αφού καθυστερούν την διαδικασία δημιουργίας μοντέλου μέχρι η κατηγοριοποίηση ενός παραδείγματος είναι αναγκαία, εν αντιθέσει με τα δέντρα απόφασης που ανήκουν στους eagers learners και δημιουργούν κατευθείαν ένα μοντέλο ανάλογα με τα δεδομένα που έχουν.

Ο συγκεκριμένος τρόπος κατηγοριοποίησης χρησιμοποιεί την Ευκλείδεια απόσταση με σκοπό να βρει ομοιότητες μεταξύ των στοιχείων.

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}.$$

Όπου X_1 και X_2 είναι δύο πλειάδες που για την καθεμία αντίστοιχα ισχύει $X_1 = (x_{11},x_{12},\ldots,x_{1n})$ και $X_2 = (x_{21},x_{22},\ldots,x_{2n})$.

Παρακάτω παρουσιάζεται ο αλγόριθμος:

KNN Αλγόριθμος:

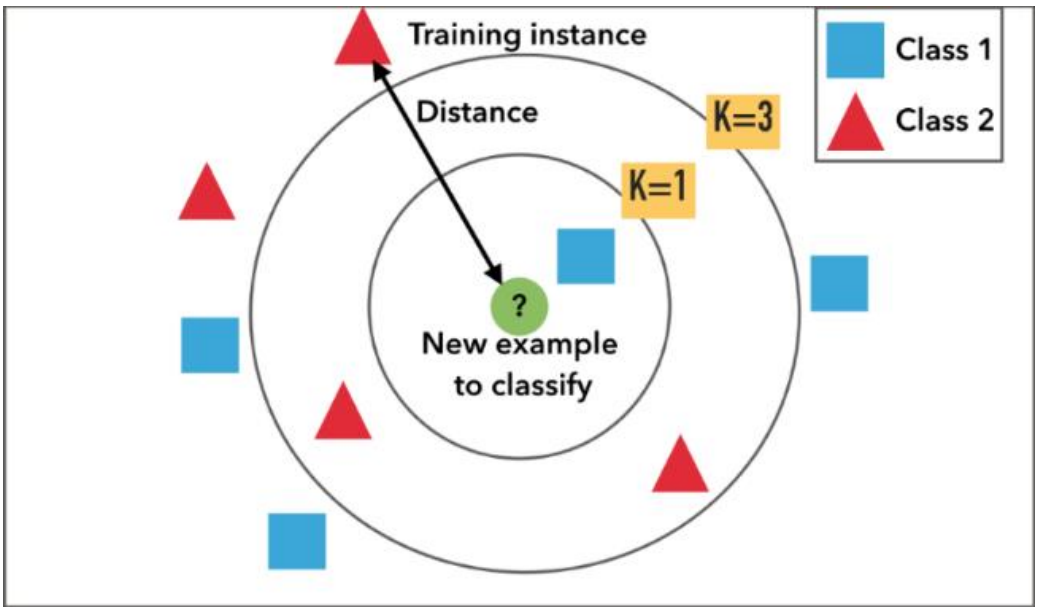
1. K = αριθμός των κοντινότερων γειτόνων και το D = σύνολο των παραδειγμάτων εκπαίδευσης

2. Για κάθε παράδειγμα ελέγχου z, z = (x',y') κάνε

3. Υπολόγισε d(x',x), την απόσταση μεταξύ των z και κάθε παραδείγματος (x,y) που ανήκει στο D

4. Επέλεξε το σύνολο των k κοντινότερων παραδειγμάτων στο z που είναι υποσύνολο του D

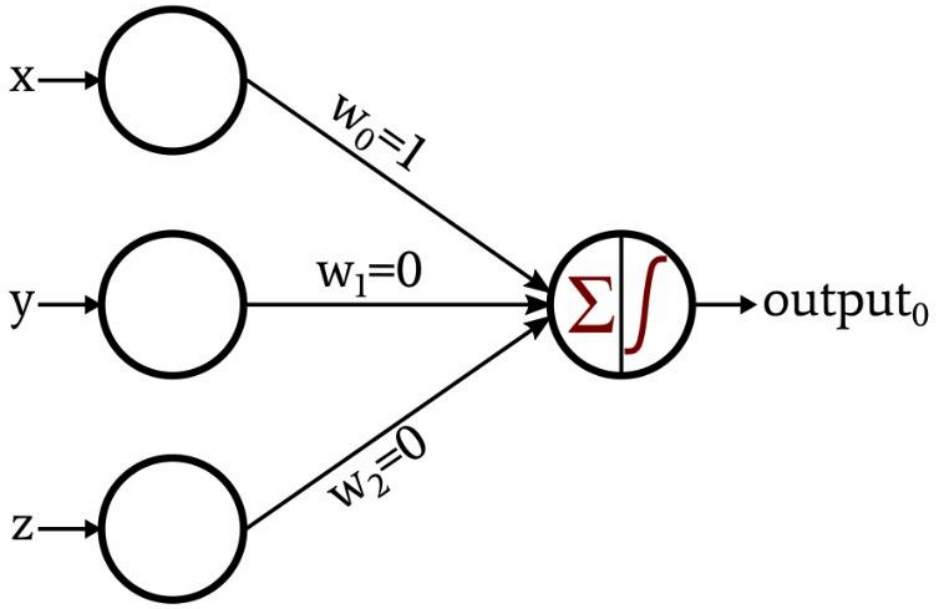
5. Το y' θα πάρει την τιμή που έχουν τα περισσότερα y των κοντινότερων γειτόνων



Εικόνα 11. KNN Αλγόριθμος (πηγή: https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7)

3.6 Νευρωνικά Δίκτυα

Ο τομέας των Τεχνητών Νευρωνικών Δικτύων δημιουργήθηκε με βάση την γνώση που υπάρχει γύρω από την λειτουργία του βιολογικού νευρικού συστήματος των ζώντων οργανισμών. Οι ζώντες οργανισμοί περιέχουν ένα νευρικό σύστημα, το οποίο ευθύνεται για πολλές διεργασίες όπως η μνήμη και η μάθηση. Το ανθρώπινο νευρικό σύστημα, όπως και ο εγκέφαλος που θεωρείται η κεντρική μονάδα του νευρικού συστήματος, περιέχουν έναν μεγάλο αριθμό νευρωνικών δικτύων. Κάθε νευρωνικό δίκτυο αποτελείται από νευρώνες ή νευρώνια(neurons), τα οποία αποτελούν το πιο μικρό και ανεξάρτητο δίκτυο που υπάρχει. Η δουλειά των νευρώνων είναι η διαρκής επεξεργασία πληροφοριών, μοιράζοντας ηλεκτρικά σήματα μεταξύ τους. Η προσπάθεια των επιστημόνων να δημιουργήσουν κάποια μοντέλα που να μιμούνται την λειτουργία του ανθρώπινου εγκεφάλου και να περιέχουν τα χαρακτηριστικά που είναι γνωστά μέχρι και σήμερα με σκοπό την επεξεργασία πληροφοριών και την λήψη αποφάσεων, οδήγησε στην δημιουργία των Τεχνητών Νευρωνικών Δικτύων.



Εικόνα 12. Τεχνητός Νευρώνας (πηγή: https://www.allaboutcircuits.com/technical-articles/how-to-train-a-basic-perceptron-neural-network/)

Η παραπάνω εικόνα παρουσιάζει έναν απλό τεχνητό νευρώνα, δηλαδή την πιο απλή μορφή ενός τεχνητού νευρωνικού δικτύου. Ο κάθε νευρώνας αποτελείται από τους κόμβους που δέχονται τα δεδομένα εισόδου και τους κόμβους από τους οποίους εξέρχονται τα δεδομένα εξόδου. Όπως φαίνεται και στην παραπάνω εικόνα, οι κόμβοι που δέχονται τα δεδομένα εισόδου έχουν μια τιμή βάρους, η οποία δηλώνει το πόσο συνδεδεμένοι είναι οι δύο νευρώνες που συνδέονται με το βάρος αυτό και μπορεί να πάρει τιμές από -1 εώς +1.

Με την ενεργοποίησή του, ένας νευρώνας υπολογίζει μια συνάρτηση όλων των δεδομένων που έχει και συγκρίνει την τιμή που υπολόγισε με την τιμή του κατωφλιού που έχει ο συγκεκριμένος νευρώνας. Το μοντέλο στην παραπάνω φωτογραφία έχει τρεις κόμβους εισαγωγής δεδομένων και ο κάθε κόμβος έχει το δικό του βάρος. Τα δεδομένα εξαγωγής που μπορούν να προκύψουν από αυτό δίνονται από την παρακάτω εξίσωση:

$$\mathbf{output_0 = w0x + w1y + w2z}$$

εάν το παραπάνω αποτέλεσμα είναι μεγαλύτερο από το μηδέν το $output_0$ θα είναι ίσο με το 1, ενώ εάν είναι μικρότερο από το μηδέν θα είναι είσο με 0(Tan, Steinbach, Anuj & Vipin,2019). Η διαφορά μεταξύ των κόμβων που δέχονται δεδομένα εισόδου με τους κόμβους που παράγουν δεδομένα εξόδου είναι ότι οι πρώτοι λαμβάνουν δεδομένα από εξωτερικές πηγές χωρίς καμιά δική τους επέμβαση ή επεξεργασία, οι δεύτεροι επεξεργάζονται με την παρακάτω μαθηματική σχέση τα δεδομένα που δέχονται.

$$\mathbf{Output = sign[w_d x_d + w_{d-1} x_{d-1} + \ldots + w_1 x_1 + w_0 x_0] = sign(w \cdot x)}$$

Όπου w είναι το βάρος του κάθε κόμβου και το x το δεδομένο εισόδου. Το sign έχει να κάνει με την συνάρτηση ενεργοποίησης του νευρώνα,

στην οποία εισάγεται το Output και ευθύνεται για την παραγωγή του τελικού αποτελέσματος(-1 ή 1).

Κατά την διάρκεια εκπαίδευσης του μοντέλου, τα βάρη μεταβάλλονται και ρυθμίζονται μέχρι τα δεδομένα εξόδου του νευρώνα ταιριάξουν με τα δεδομένα εξόδου του συνόλου εκπαίδευσης. Η σχέση που είναι υπεύθυνη για την ρύθμιση των βαρών είναι η εξής:

$$\mathbf{W_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \tilde{y}_i^{(k)})x_{ij},}$$

όπου $w^{(k)}$ είναι το βάρος που αντιστοιχεί σε κάθε κόμβο μετά την k^{th} επανάληψη, το λ είναι γνωστό σαν ποσοστό μάθησης(learning rate) και το x_{ij} είναι η τιμή του j^{th} χαρακτηριστικού από το παράδειγμα εκπαίδευσης x. Παρακάτω παρουσιάζεται ο αλγόριθμος ενός Τεχνητού Νευρώνα.

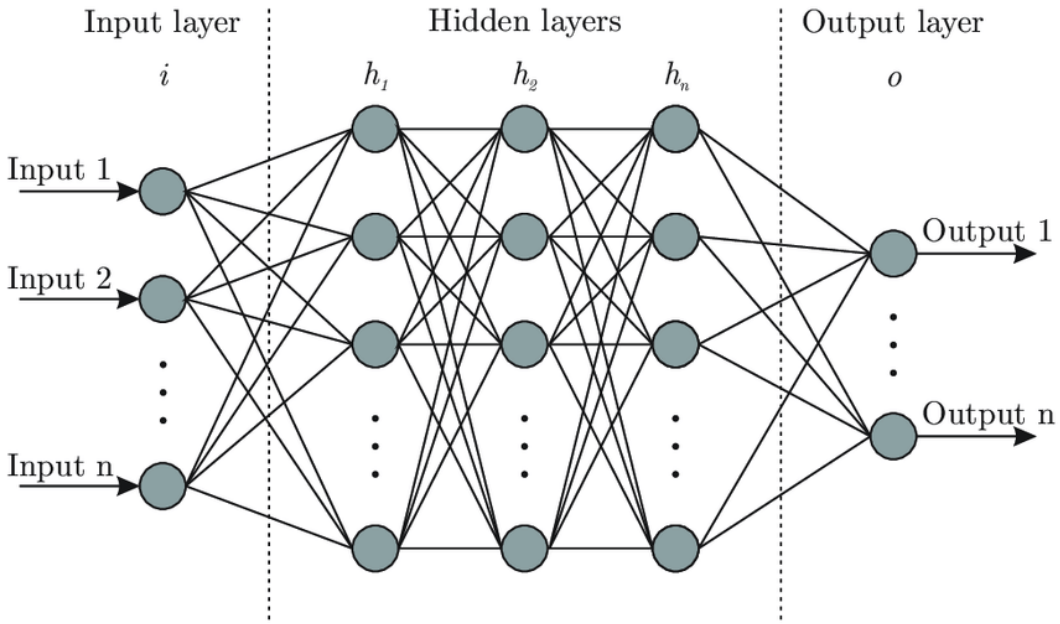
Αλγόριθμος Τεχνητού Νευρώνα

1. Έστω $D = \{(x_i,y_i) \mid i = 1,2, \dots , N \}$ ένα σύνολο εκπαίδευσης
2. Αρχικοποίηση των βαρών w με τυχαίες τιμές
3. Επανάληψη:
4. Για κάθε (x_i,y_i) που ανήκει στο D:
5. Υπολόγισε την πρόβλεψη $\tilde{y}_i^{(k)}$
6. Για κάθε w_j:
7. Αναβάθμισε τα βάρη $W_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \tilde{y}_i^{(k)})x_{ij}$
8. Τέλος Για
9. Τέλος Για
10. Μέχρι την σχέση τερματισμού

Τα τεχνητά νευρωνικά δίκτυα συνήθως έχουν πιο περίπλοκη δομή από έναν τεχνητό νευρώνα. Μερικές από τις πιο σημαντικές διαφορές παρουσιάζονται παρακάτω:

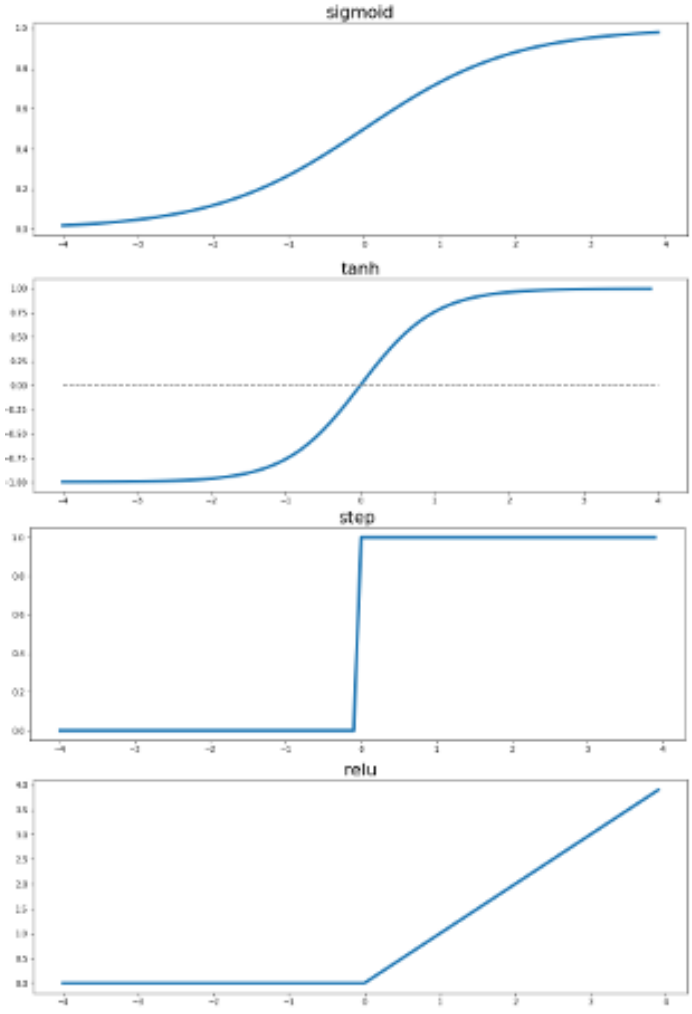
1. Το κάθε δίκτυο μπορεί να περιλαμβάνει πολλά ενδιάμεσα επίπεδα που ονομάζονται κρυμμένα επίπεδα και οι κόμβοι που βρίσκονται σε αυτά κρυμμένοι κόμβοι. Τα δίκτυα αυτά ονομάζονται Πολυεπίπεδα Νευρωνικά Δίκτυα. Στα feed forward δίκτυα, οι κόμβοι κάθε επιπέδου συνδέονται με τους κόμβους του επόμενου επιπέδου.

2. Τα δίκτυα νευρώνα αποτελούν ένα feed forward νευρωνικό δίκτυο με 1 μονό επίπεδο, αφού έχει μόνο ένα επίπεδο από κόμβους εξαγωγής, εν αντιθέσει με τα αναδρομικά νευρωνικά δίκτυα, τα οποία πολλές φορές περιέχουν συνδέσεις των κόμβων μεταξύ τους εντός του ίδιου επιπέδου ή κόμβων του ενός επιπέδου με το προηγούμενο.



Εικόνα 13. Πολυεπίπεδο Νευρωνικό Δίκτυο(πηγή: https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed)

3. Τα πιο σύνθετα δίκτυα μπορεί να χρησιμοποιούν συναρτήσεις ενεργοποίησης διαφορετικές από την sign. Μερικές από αυτές είναι η γραμμική, η σιγμοειδής, η ReLu ή οι υπερβολικές συναρτήσεις(Tan, Steinbach, Anuj & Vipin,2019).



Εικόνα 14. Συναρτήσεις ενεργοποίησης(πηγή: http://www.snee.com/bobdc.blog/2017/09/understanding-activation-funct.html)

Ο στόχος των νευρωνικών δικτύων είναι η ρύθμιση των βαρών w, έτσι ώστε να ελαχιστοποιείται το συνολικό άθροισμα του τετραγώνου των σφαλμάτων. Στην παρακάτω σχέση αποδεικνύεται αυτό:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

Το άθροισμα των τετραγώνων των σφαλμάτων εξαρτάται από τα βάρη, αφού οι προβλέψεις που δημιουργούνται, προκύπτουν από την συνάρτηση των βαρών που έχουν ανατεθεί στους κρυφούς κόμβους και κόμβους εξαγωγής.

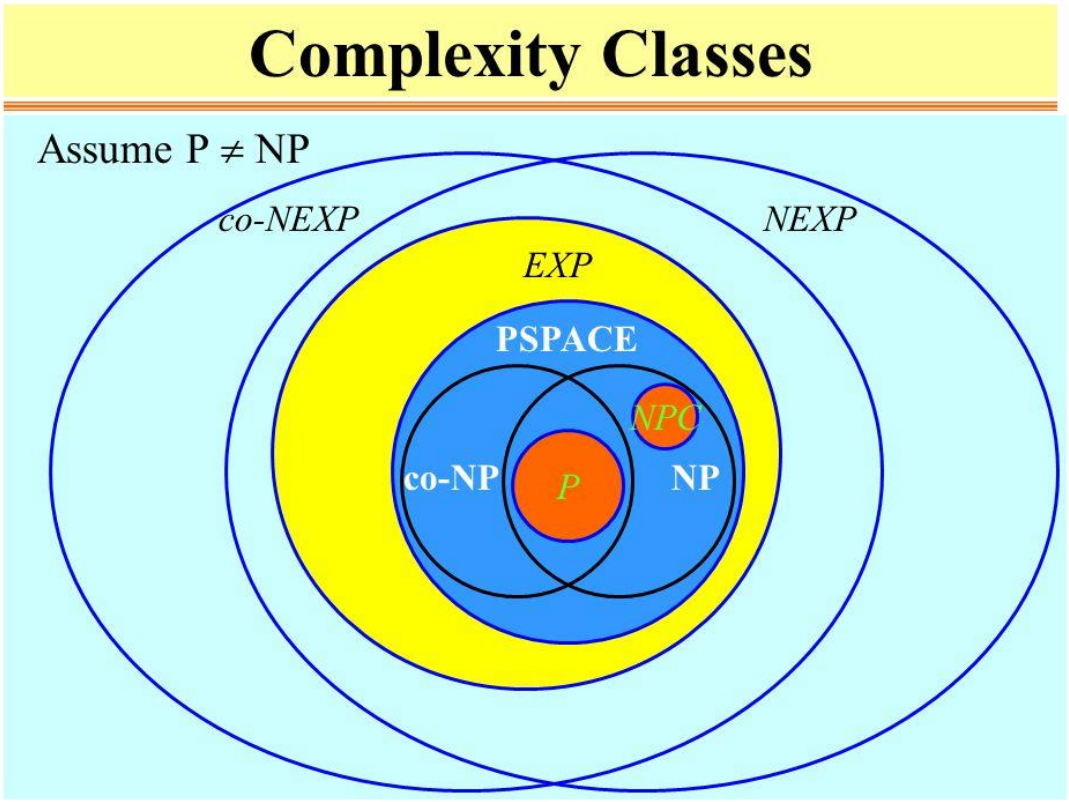
Στις περισσότερες των περιπτώσεων, τα αποτελέσματα που παράγονται από ένα τεχνητό νευρωνικό δίκτυο είναι μια μη γραμμική συνάρτηση των παραμέτρων του εξαιτίας της επιλογής μιας συνάρτησης ενεργοποίησης(Tan, Steinbach, Anuj & Vipin,2019).

4 Υπολογιστική Μελέτη και Σύγκριση

4.1 Σύγκριση αλγορίθμων μέσω της χρονικής πολυπλοκότητας

Το πεδίο της υπολογιστικής πολυπλοκότητας ανήκει στο ευρύτερο πεδίο της θεωρητικής επιστήμης των υπολογιστών και κύριο στόχο έχει την σύγκριση αλλά και ταξινόμηση της πρακτικής δυσκολίας που αντιμετωπίζουν οι αλγόριθμοι στην επίλυση προβλημάτων. Ένας ακόμα στόχος είναι η πρόταση επίσημων και ακριβών κριτηρίων για το πότε ένα μαθηματικό πρόβλημα είναι εφικτό να λυθεί, δηλαδή να μπορεί να υπολογιστεί από μια συμβατική(ντετερμινιστική) μηχανή Turing σε μια σειρά βημάτων ανάλογη με μια πολυωνυμική συνάρτηση του μεγέθους εισόδου του.

Υπάρχει μια ολόκληρη κατηγορία προβλημάτων που ακολουθούν την παραπάνω ιδιότητα και χαρακτηρίζονται ως P (polynomial time). Τα P προβλήματα μπορούν να αποδειχτούν επίσημα και να διαφοροποιηθούν από τις υπόλοιπες κατηγορίες όπως παραδείγματος χάρη την εκθετική EXP, που περιλαμβάνει προβλήματα που χρειάζονται εκθετικό χρόνο επίλυσης. Ένα ακόμα σύνολο προβλημάτων που αποτελεί κατηγορία είναι τα NP (non deterministic polynomial time) και περιέχουν τα προβλήματα στα οποία μπορεί να δοθεί μια απόφαση μέσω μιας μη ντετερμινιστικής μηχανής Turing σε μια σειρά βημάτων ανάλογη με μια πολυωνυμική συνάρτηση του μεγέθους εισόδου του και να ελεγχθεί η απόφαση αυτή σε πολυωνυμικό χρόνο. Στην θεωρητική επιστήμη των υπολογιστών θεωρείται πλέον πως τα P προβλήματα είναι υποκατηγορία των NP, δηλαδή $P \subsetneq NP$ (Walter ,2016).



Εικόνα 15. Κατηγορίες πολυπλοκότητας (πηγή: https://slideplayer.com/slide/6105206/)

Το πεδίο της υπολογιστικής πολυπλοκότητας αναλύει τους αλγορίθμους με βάση την χρονική και χωρική πολυπλοκότητά τους. Η χωρική πολυπλοκότητα είναι στην ουσία η ποσότητα του χώρου μνήμης που απαιτείται για την επίλυση ενός προβλήματος σε σχέση με το μέγεθος των δεδομένων εισόδου. Από την άλλη, η χρονική πολυπλοκότητα έχει να κάνει με τον χρόνο που απαιτείται για την επίλυση ενός προβλήματος σε σχέση με το μέγεθος εισόδου των δεδομένων.

Κατά την διαδικασία ανάλυσης της χρονικής πολυπλοκότητας περιγράφονται 3 περιπτώσεις, οι οποίες είναι οι εξής:

- Η καλύτερη περίπτωση(best case), στην οποία επιλύεται ο αλγόριθμος για την καλύτερη κατάσταση των δεδομένων εισόδου.

- Η περίπτωση μέσου όρου(average case), στην οποία τα δεδομένα εισόδου ακολουθούν την κατανομή όπου εμφανίζονται τις περισσότερες φορές και διαφέρουν από την προηγούμενη περίπτωση.
- Η χειρότερη περίπτωση(worst case), στην οποία τα δεδομένα εισόδου ακολουθούν την χειρότερη κατανομή στην οποία θα μπορούσαν να ακολουθήσουν.

Στην ανάλυση των αλγορίθμων λαμβάνεται υπόψη συνήθως η χειρότερη περίπτωση και χρησιμοποιείται το λεγόμενο Big-O Notation. Το Big-O Notation είναι στην ουσία ένα μαθηματικό διάγραμμα που δείχνει την συμπεριφορά μιας συνάρτησης, όταν το αποτέλεσμά της τείνει σε μια συγκεκριμένη τιμή ή και το άπειρο. Οι πιο συχνές κατηγορίες που χρησιμοποιούνται για την χρονική πολυπλοκότητα χρησιμοποιώντας το Big-O Notation παρουσιάζονται στον παρακάτω πίνακα:

Όνομα	Χρονική Πολυπλοκότητα
Σταθερός χρόνος	O(1)
Λογαριθμικός χρόνος	O(log n)
Γραμμικός χρόνος	O(n)
Quasilinear χρόνος	O(n log n)
Τετραγωνικός χρόνος	O(n^2)
Εκθετικός χρόνος	O(2^n)
Παραγοντικός χρόνος	O(n!)

Πίνακας 1. Κατηγορίες χρονικής πολυπλοκότητας

Σταθερός χρόνος - O(1)

Ένας αλγόριθμος έχει σταθερό χρόνο όταν δεν εξαρτάται από τον αριθμό των δεδομένων εισόδου.

Λογαριθμικός χρόνος – O(log n)

Ένας αλγόριθμος έχει λογαριθμικό χρόνο όταν μειώνει το μέγεθος των δεδομένων εισόδου σε κάθε βήμα που πραγματοποιεί (πχ binary trees)

Γραμμικός χρόνος – O(n0)

Ένας αλγόριθμος έχει γραμμικό χρόνο όταν ο χρόνος εκτέλεσής του αυξάνεται γραμμικά σε σχέση με τον αριθμό των δεδομένων εισόδου.

Quasilinear χρόνος – O(n log n)

Ένας αλγόριθμος έχει Quasilinear χρόνο όταν κάθε βήμα επεξεργασίας των δεδομένων εισόδου έχει λογαριθμικό χρόνο πολυπλοκότητας(πχ mergesort algorithm). Για παράδειγμα κάθε τιμή στο dataset1 με O(n) χρησιμοποιεί O(log n) για να ψάξει την ίδια τιμή στο dataset2.

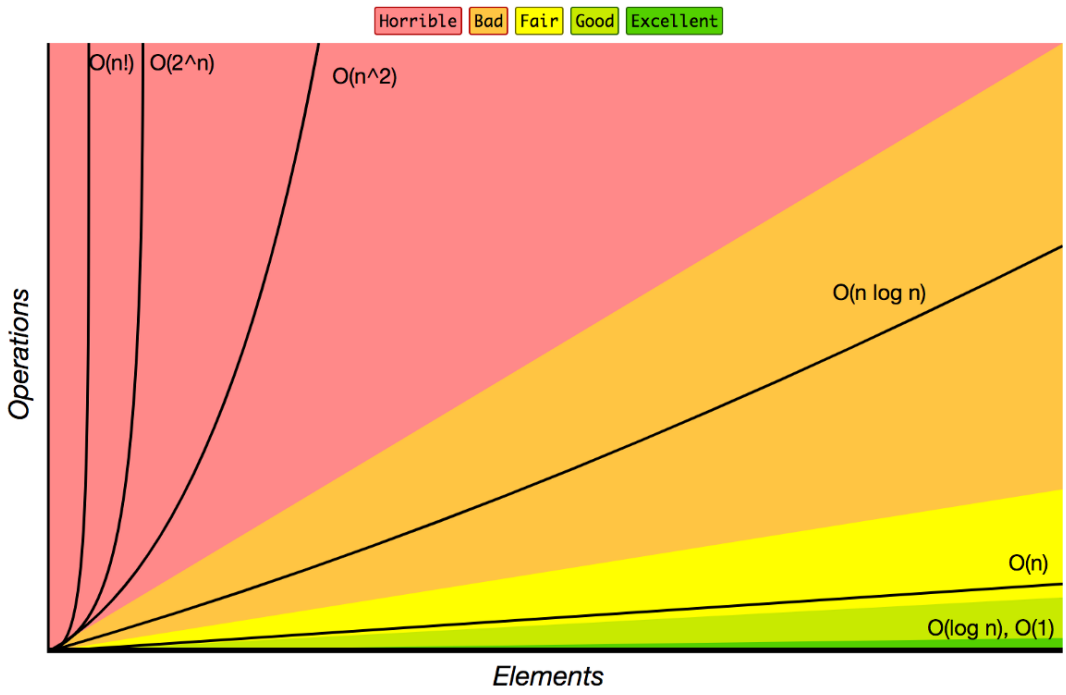
```
for value in dataset1:
        result.append(binary_search(dataset2, value))
```

Τετραγωνικός χρόνος – $O(n^2)$

Ένας αλγόριθμος έχει τετραγωνικό χρόνο όταν χρειάζεται γραμμικό χρόνο για να εκτελέσει κάθε τιμή στο σύνολο των δεδομένων εισόδου(πχ bubblesort algorithm).

Εκθετικός χρόνος – $O(2^n)$

Ένας αλγόριθμος έχει εκθετικό χρόνο όταν η ανάπτυξή του διπλασιάζεται σε κάθε μια πρόσθεση στο σύνολο δεδομένων εισόδου.



Εικόνα 16. Big-O Notation (πηγή: https://www.freecodecamp.org/news/all-you-need-to-know-about-big-o-notation-to-crack-your-next-coding-interview-9d575e7eec4/)

Η χρονική πολυπλοκότητα των αλγορίθμων που θα αναλυθούν στην συγκεκριμένη εργασία παρουσιάζεται στον παρακάτω πίνακα (n είναι ο αριθμός των στιγμιότυπων, n_{li} είναι ο αριθμός των νευρώνων στο επίπεδο i σε ένα νευρωνικό δίκτυο,k είναι ο αριθμός των δέντρων και d είναι οι διαστάσεις):

Μέθοδος	Χρονική πολυπλοκότητα εκπαίδευσης	Χρονική πολυπλοκότητα πρόβλεψης
Λογιστική Παλινδρόμηση	O(nd)	O(d)
Δέντρα Απόφασης	$O(n^2d)$	O(d)
Ενισχυμένα Δέντρα	O(n*d*k)	O(d*k)
Τυχαία Δάση	O(n*log(n)*d*k)	O(d*k)
K-Κοντινότερων Γειτόνων	O(knd)	O(nd)
Τεχνητά Νευρωνικά Δίκτυα	-	$O(dn_{l1} + n_{l1}n_{l2}+ ...)$

Πίνακας 2. Χρονική Πολυπλοκότητα μεθόδων κατηγοριοποίησης

-Χρονική πολυπλοκότητα στην Λογιστική Παλινδρόμιση

Η χρονική πολυπλοκότητα στην Λογιστική Παλινδρόμιση για την εκπαίδευση ειναι O(nd), όπου n είναι ο αριθμός των στιγμιοτύπων και d οι διαστάσεις του συγκεκριμένου συνόλου εκπαίδευσης που επεξεργάζεται. Στην Λογιστική Παλινδρόμιση η εξαρτημένη μεταβλητή είναι δυαδική που σημαίνει ότι παίρνει τιμές 0 ή 1, άρα το μοντέλο προβλέπει για P(Y=1) ως συναρτηση του X.

Έστω ένα σημείο A = (x1, y1, z1) στις 3 διαστάσεις με διανύσματα n = (a, b, c) ορίζονται ως εξής:

a(x-x1) + b (y-y1) + c(z-z1) + b = 0

ax + by + cz + b = 0 όπου b = -(ax1 + by1 + cz1)

Το παραπάνω μπορεί να γραφτεί ως

w1x1 + w2x2 + w3x3 + b = 0 και ακολούθως

w^t * xi + b = 0 με xi να είναι η i παρατήρηση.

Όπως φαίνεται από τα παραπάνω, προκειμένου να βρεθεί η διαχωριστική γραμμή ή μοντέλο μεταξύ των κλάσεων, αρκεί να βρεθούν το w και το b. Η τιμή των w και b πρέπει να είναι αυτές που μεγιστοποιούν το yi*w^t*xi>0. Το w είναι ένα διάνυσμα μεγέθους d. Το yi*w^t*xi χρειάζεται O(d) για να εκτελεσθεί. Η επανάληψή του σε n σημεία των δεδομένων και η εύρεση του μέγιστου αθροίσματος χρειάζονται n βήματα. Καθ΄ αυτόν τον τρόπο, η χρονική πολυπλοκότητα της Λογιστικής Παλινδρόμισης είναι n(O(d)) = O(nd).

-Χρονική πολυπλοκότητα στον Κ-Κοντινότερων Γειτόνων

Έστω ότι δίνεται ένα σημείο από το σύνολο δεδομένων x, ο KNN αναζητά τα k πιο κοντινά σημεία στο x σημείο που δόθηκε στα δεδομένα εκπαίδευσης και επιστρέφει την πιο κοινή ταμπέλα. Για να πραγματοποιηθεί το παραπάνω πρέπει να συγκριθεί η απόσταση μεταξύ του x και κάθε σημείου από το σύνολο δεδομένων και αυτό αντιστοιχεί σε n πράξεις. Η εκτέλεση αυτή ολοκληρώνεται σε O(d) για τις γνωστές αποστάσεις όπως η Ευκλείδια,η Μανχαταν και άλλες. Για κάθε σημείο εντός του συνόλου ο αλγόριθμος ψάχνει να βρεί την μικρότερη απόσταση και αυτό εκτελείται σε O(kn). Ολόκληρη η εκτέλεση του αλγορίθμου πραγματοποιείται σε O(knd).

-Χρονική πολυπλοκότητα στα Δένδρα Απόφασης

Η χρονική πολυπλοκότητα στα Δένδρα Απόφασης είναι O(p*n*n). Στην διαδικασία εκπαίδευσης ενός δένδρου, πρέπει να βρεθεί ο διαχωρισμός μέχρι το μέγιστο βάθος d. Η μέθοδος για να βρεθεί το παραπάνω είναι η εύρεση της κάθε μεταβλητής όπου αυτές είναι p, στα διαφορετικά κατώφλια όπου αυτά είναι n και να υπολογισθεί το κέρδος(Information Gain) όπου αυτο χρειάζεται O(n).

-Χρονική πολυπλοκότητα στα Τυχαία Δάση

Έστω ότι τα δέντρα μπορούν ελεύθερα να φτάσουν ένα μέγιστο ύψος O(logn), η εκπαίδευση χρειάζεται O(kdlogn), όπου k είναι ο αριθμός των δέντρων και d είναι ο αριθμός των χαρακτηριστικών για τα οποία θα πραγματοποιηθεί διαχωρισμός. Η πρόβλεψη των καινούργιων δειγμάτων χρειάζεται O(klogn).

-Χρονική πολυπλοκότητα στον XGBoost

Η εκπαίδευση στον παραπάνω αλγόριθμο είναι O(kdxlogn)(σε ταξινομημένα δεδομένα είναι O(kdx) αφού ο αλγόριθμος χρειάζεται O(logn) προκειμένου να πραγματοποιήσει τον διαχωρισμό κάθε κόμβου), όπου k είναι ο αριθμός των δέντρων, d είναι το ύψος των δέντρων, και x είναι ο αριθμός των στιγμιοτύπων στο σύνολο δεδομένων εκπαίδευσης. Οι προβλέψεις για τα καινούργια στιγμιότυπα χρειάζονται O(kd).

-Χρονική πολυπλοκότητα στα Νευρωνικά Δίκτυα

Η χρονική πολυπλοκότητα για τα Νευρωνικά Δίκτυα όσων αφορά τον έλεγχο είναι $O(dn_{l1} + n_{l1}n_{l2}+ \ldots)$ όπου n είναι ο αριθμός των στιγμιότυπων, n_{li} είναι ο αριθμός των νευρώνων στο επίπεδο i σε ένα νευρωνικό δίκτυο,k είναι ο αριθμός των δέντρων και d είναι οι διαστάσεις

4.2 Σύγκριση αλγορίθμων μέσω μετρικών επίδοσης

Εκτός από την ανάλυση και σύγκριση των αλγορίθμων ως προς τον χρόνο εκτέλεσης των διαδικασιών, οι αλγόριθμοι αναλύονται και ως προς την ακρίβεια που επιτυγχάνουν στην διαδικασία της κατηγοριοποίησης. Συνήθως, ο αλγόριθμος εκπαιδεύεται με την χρήση του συνόλου δεδομένων εκπαίδευσης και ελέγχεται ως προς την ακρίβεια πάνω στο σύνολο των δεδομένων ελέγχου. Στην προκειμένη διαδικασία ελέγχου δημιουργείται το λεγόμενο confusion matrix που περιέχει τιμές με ονόματα όπως TP, TN, FP, FN και υπολογίζει μετρικές όπως το accuracy, το recall, το precision και το F-score(Han, Kamber & Pei,2012). Το περιεχόμενο κάθε τιμής παρουσιάζεται παρακάτω:

- True positives(TP) – είναι οι θετικές τιμές που σωστά κατηγοριοποιήθηκαν από τον κατηγοριοποιητή

- True negatives(TN) – είναι οι αρνητικές τιμές που σωστά κατηγοριοποιήθηκαν από τον κατηγοριοποιητή

- False positives(FP) – είναι οι αρνητικές τιμές που λανθασμένα κατηγοριοποιήθηκαν ως θετικές από τον κατηγοριοποιητή

- False negatives(FN) – είναι οι θετικές τιμές που λανθασμένα κατηγοριοποιήθηκαν αρνητικές από τον κατηγοριοποιητή

Παρακάτω παρουσιάζονται οι σχέσεις των παραπάνω τιμών με τις μετρικές απόδοσης των αλγορίθμων:

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total}$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

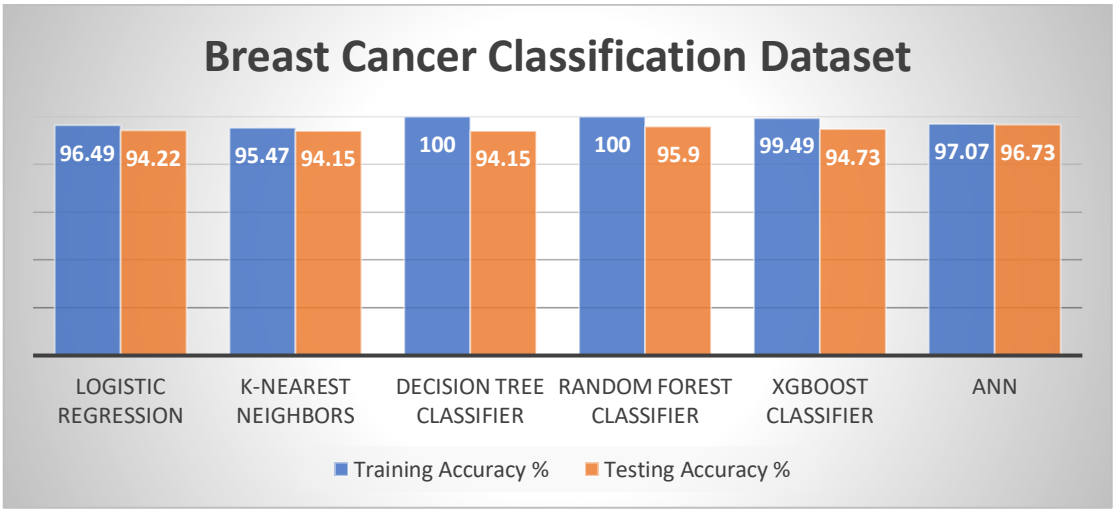
$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F1 - score = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

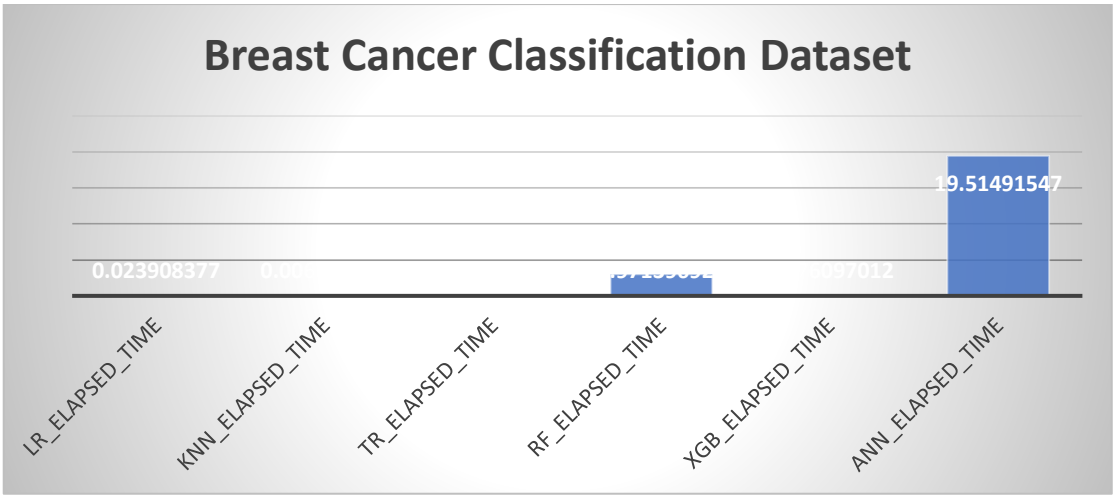
4.3 Εκτέλεση του πειράματος

Λαμβάνοντας υπόψη τα παραπάνω, πραγματοποιήθηκε η επεξεργασία 4 διαφορετικών συνόλων δεδομένων για τους αλγόριθμους που ήδη αναλύθηκαν και στην συνέχεια τα αποτελέσματα θα παρουσιαστούν, θα αναλυθούν και θα συγκριθούν μεταξύ τους, τόσο ως προς την χρονική πολυπλοκότητα, όσο και ως προς τις μετρικές που ήδη αναφέρθηκαν.

Το 1^{ο} σύνολο δεδομένων αποτελείται από οχτώ χαρακτηριστικά(τύπου float) και την στήλη με το δυαδικό αποτέλεσμα. Το θέμα του είναι η διάγνωση του καρκίνου του μαστού και παρακάτω παρουσιάζονται οι επιδόσεις.

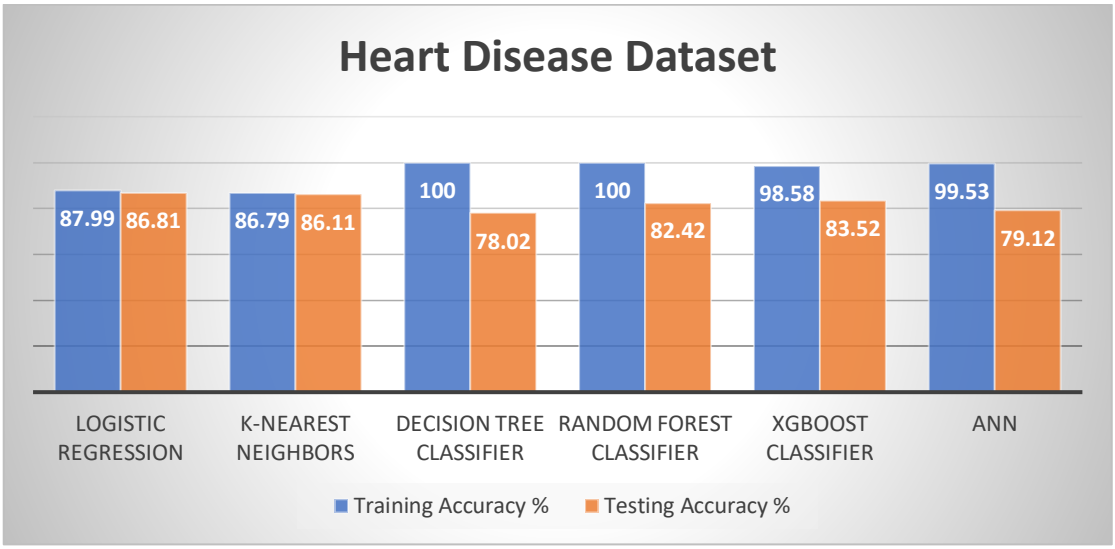


Εικόνα 17. Ακρίβεια αλγορίθμων

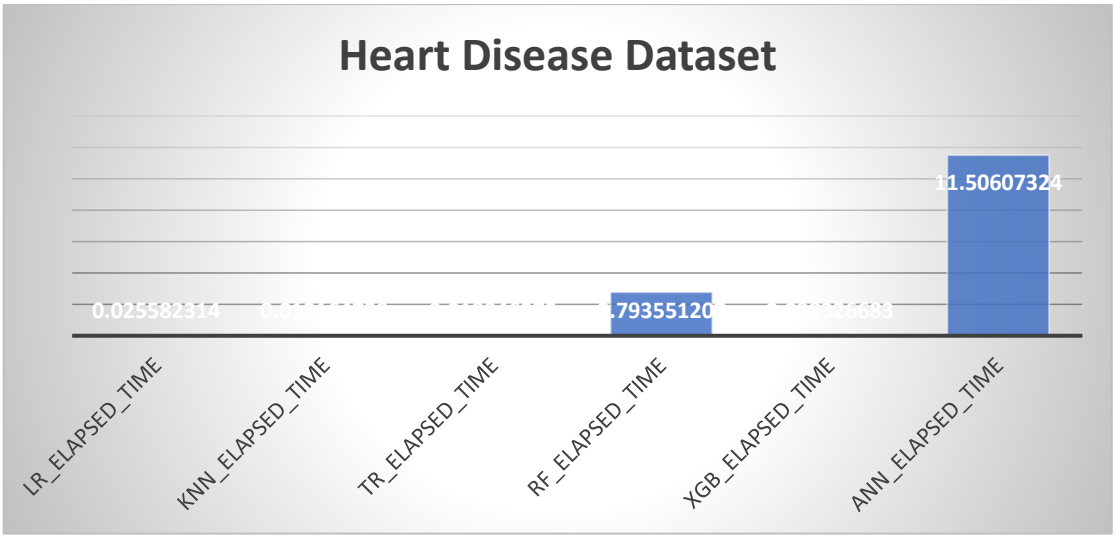


Εικόνα 18. Χρόνος εκτέλεσης αλγορίθμων

Το 2ο σύνολο δεδομένων αποτελείται από 12 χαρακτηριστικά(τύπου integer) και την στήλη με το δυαδικό αποτέλεσμα. Το θέμα του είναι η διάγνωση της καρδιακής προσβολής και παρακάτω παρουσιάζονται οι επιδόσεις.

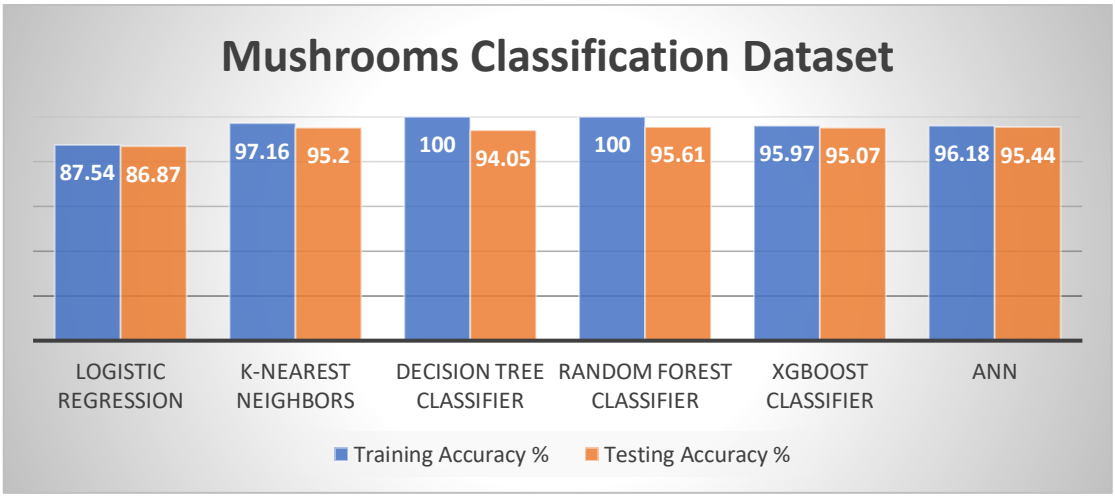


Εικόνα 19. Ακρίβεια αλγορίθμων

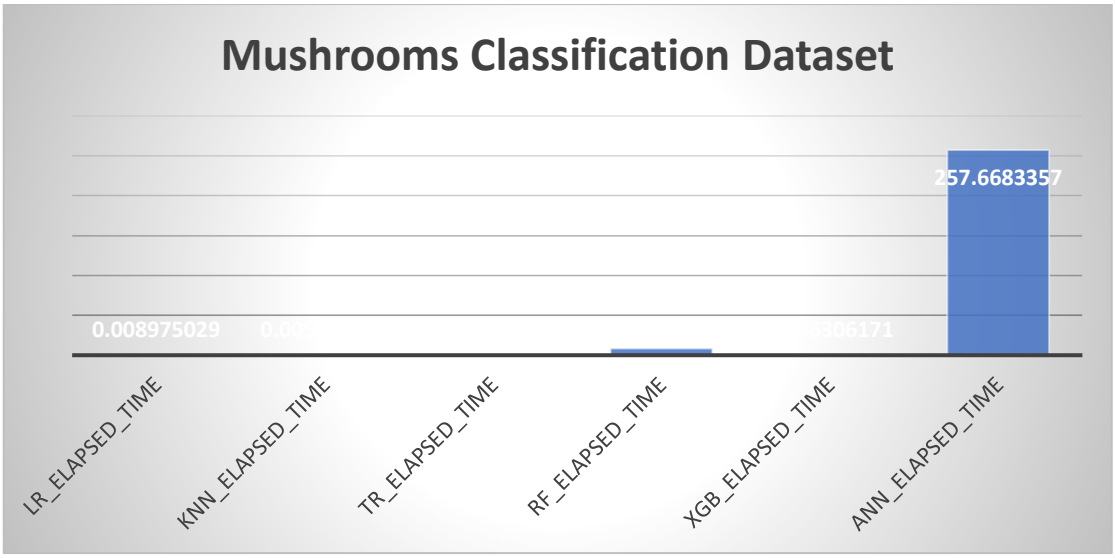


Εικόνα 20. Χρόνος εκτέλεσης αλγορίθμων

Το 3ο σύνολο δεδομένων αποτελείται από 12 χαρακτηριστικά(τύπου string) και την στήλη με το δυαδικό αποτέλεσμα. Το θέμα του είναι η ταξινόμηση μανιταριών σε 2 κατηγορίες(δηλητηριώδη-μη δηλητηριώδη) και παρακάτω παρουσιάζονται οι επιδόσεις.

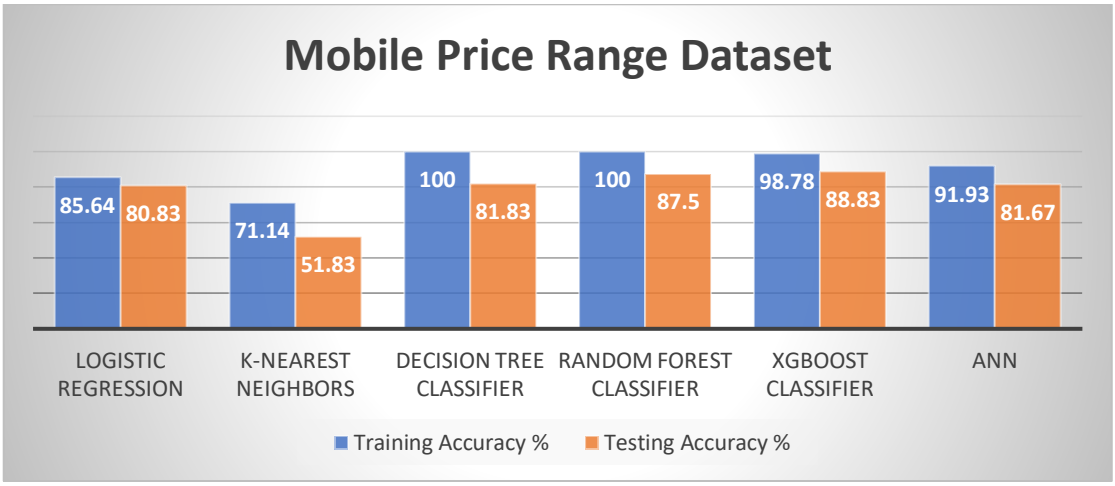


Εικόνα 21. Ακρίβεια αλγορίθμων

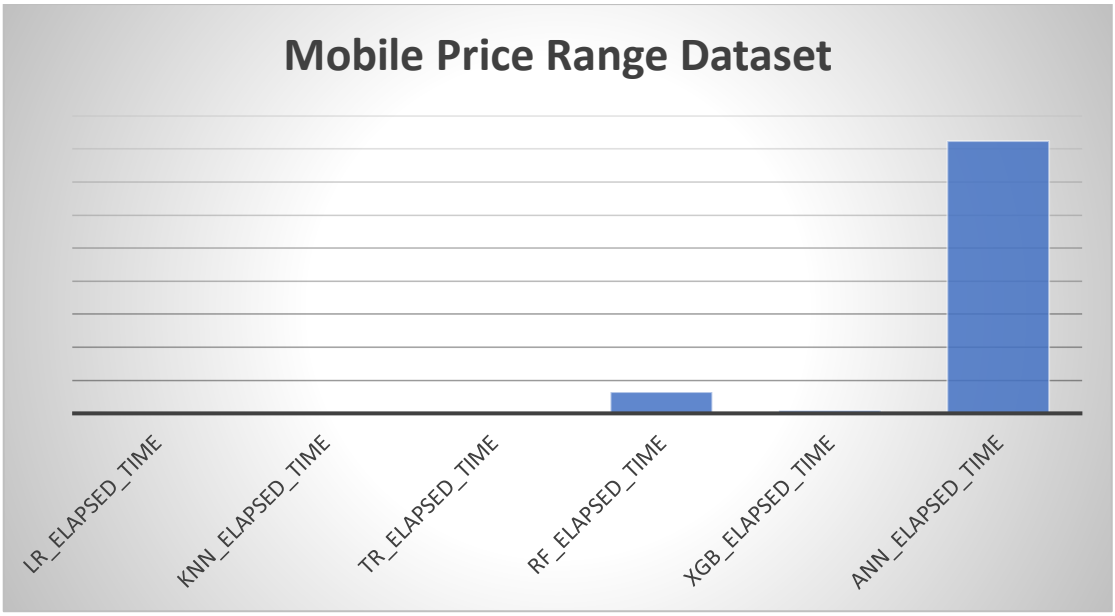


Εικόνα 22. Χρόνος εκτέλεσης αλγορίθμων

Το 4ο σύνολο δεδομένων αποτελείται από 17 χαρακτηριστικά(τύπου float και string) και την στήλη με το αποτέλεσμα. Το θέμα του είναι η ταξινόμηση κινητών τηλεφώνων σε 4 κατηγορίες τιμής και παρακάτω παρουσιάζονται οι επιδόσεις.



Εικόνα 23. Ακρίβεια αλγορίθμων



Εικόνα 24. Χρόνος εκτέλεσης αλγορίθμων

5 Συμπεράσματα

Παρατηρώντας τα παραπάνω διαγράμματα προκύπτει το εξής συμπέρασμα: ο κάθε αλγόριθμος αποδίδει καλύτερα κάτω από συγκεκριμένες συνθήκες. Τα Νευρωνικά Δίκτυα, όπως και τα Τυχαία Δάση είναι οι πιο αργοί στην εκπαίδευση αλγόριθμοι, αλλά πετυχαίνουν μεγαλύτερη ακρίβεια σε περίπλοκα και μεγάλα σύνολα δεδομένων. Από την άλλη μεριά, η Λογιστική Παλινδρόμηση είναι ο πιο

γρήγορος αλγόριθμος με βάση τα παραπάνω, αλλά αποδίδει αποτελεσματικά σε μικρά και απλά σύνολα δεδομένων. Εάν ένας χρήστης επιδιώκει γρήγορα αποτελέσματα θα μπορέσει να τα έχει χρησιμοποιώντας τον αλγόριθμο Λογιστικής Παλινδρόμησης, ενώ εάν χρειάζεται ακρίβεια θα χρησιμοποιήσει αλγορίθμους Δέντρων και Νευρωνικά Δίκτυα.

Βιβλιογραφία

-Han, J.W., Kamber, M. and Pei, J. (2012) Data Mining Concepts and Techniques. 3rd Edition, Morgan Kaufmann Publishers, Waltham

-Ian H. Witten, Frank Eibe, Mark A. Hall. (2011) Data mining : practical machine learning tools and techniques. 3rd Edition, Morgan Kaufmann Publishers, Waltham

-Tan Pang-Ning., Steinbach Michael., Karpatne Anuj., Kumar Vipin. (2019) Introduction to Data Mining. Second edition, Pearson Education, New York

-Walter Dean. (2016) Computational Complexity Theory, Stanford University

Παράρτημα Κώδικα

-Heart-disease dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import time

dt = pd.read_csv('heart.csv')
dt.head()
dt['target'].unique()
dt.describe()
sns.pairplot(dt)
dt.isnull().sum()
for x in dt.columns:
  plt.hist(dt[x])
  plt.show()
  plt.title(x)
dt['cp'].unique()
dt.info()
dt.shape
pd.set_option("display.float","{:.2f}".format)
dt.describe()
df.target.value_counts()
```

```
categorical_val = []
continous_val = []
for column in dt.columns:
    print('==============================')
    print(f"{column} : {dt[column].unique()}")
    if len(dt[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
categorical_val
corr_matrix = dt.corr()
fig, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
dt.drop('target', axis=1).corrwith(df.target).plot(kind='bar', grid=True, figsize=(12, 8),
title="Correlation with target")
categorical_val.remove('target')
print(df.columns)
from sklearn.preprocessing import StandardScaler
s_sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[col_to_scale] = s_sc.fit_transform(dataset[col_to_scale])
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
```

```
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
from sklearn.model_selection import train_test_split

X = dataset.drop('target', axis=1)
y = dataset.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
from sklearn.linear_model import LogisticRegression

alltimes = {}
lr_clf = LogisticRegression(solver='liblinear')
lr_start_time = time.time()
lr_clf.fit(X_train, y_train)
lr_elapsed_time = time.time() - lr_start_time
alltimes['lr_elapsed_time'] = lr_elapsed_time

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_start_time = time.time()
knn_clf.fit(X_train, y_train)
knn_elapsed_time = time.time() - knn_start_time
alltimes['knn_elapsed_time'] = knn_elapsed_time

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.svm import SVC

svm_clf = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)

test_score = accuracy_score(y_test, svm_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_clf.predict(X_train)) * 100
```

```
results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.tree import DecisionTreeClassifier



tree_clf = DecisionTreeClassifier(random_state=42)
tr_start_time = time.time()
tree_clf.fit(X_train, y_train)
tr_elapsed_time = time.time() - tr_start_time
alltimes['tr_elapsed_time'] =tr_elapsed_time

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, tree_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rf_clf = RandomForestClassifier(n_estimators=1000, random_state=42)
rf_start_time = time.time()
rf_clf.fit(X_train, y_train)
rf_elapsed_time = time.time() - rf_start_time
alltimes['rf_elapsed_time'] = rf_elapsed_time

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, rf_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_start_time = time.time()
xgb_clf.fit(X_train, y_train)
xgb_elapsed_time = time.time() - xgb_start_time
alltimes['xgb_elapsed_time'] = xgb_elapsed_time

print_score(xgb_clf, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, xgb_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(12, input_dim=30, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
ann_start_time = time.time()
model.fit(X_train, y_train, epochs=150, batch_size=10)
ann_elapsed_time = time.time() - ann_start_time
alltimes['ann_elapsed_time'] =ann_elapsed_time

_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))

test_score = accuracy_score(y_test, model.predict_classes(X_test)) * 100
train_score = accuracy_score(y_train, model.predict_classes(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["ANN", train_score, test_score]],
                    columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

- Mobile dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import time
dt1 = pd.read_csv('mobile_class/train.csv', sep=',')
dt2 = pd.read_csv('mobile_class/test.csv', sep=',')
dt2.head()

dt1.describe()

dt2.describe()
```

```
dt1.info()

dt1.isnull().sum()

dt2.isnull().sum()

dt1['price_range'].unique()

dt1.drop(columns=['price_range']).corrwith(dt1['price_range']).plot(kind='bar', grid=True, figsize=(12,
8), title="Correlation with price")


from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split


X = dt1.drop('price_range', axis=1)

y = dt1['price_range']


s_sc = StandardScaler()

X_stand = s_sc.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X_stand, y, test_size=0.3, random_state=42)

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")


    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
```

```
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
from sklearn.linear_model import LogisticRegression

alltimes = {}
lr_clf = LogisticRegression()
lr_start_time = time.time()
lr_clf.fit(X_train, y_train)
lr_elapsed_time = time.time() - lr_start_time
alltimes['lr_elapsed_time'] = lr_elapsed_time

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_start_time = time.time()
knn_clf.fit(X_train, y_train)
knn_elapsed_time = time.time() - knn_start_time
alltimes['knn_elapsed_time'] = knn_elapsed_time

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.svm import SVC


svm_clf = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, svm_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(random_state=42)
tr_start_time = time.time()
tree_clf.fit(X_train, y_train)
tr_elapsed_time = time.time() - tr_start_time
alltimes['tr_elapsed_time'] =tr_elapsed_time

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, tree_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rf_clf = RandomForestClassifier(n_estimators=1000, random_state=42)
rf_start_time = time.time()
rf_clf.fit(X_train, y_train)
rf_elapsed_time = time.time() - rf_start_time
alltimes['rf_elapsed_time'] = rf_elapsed_time

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, rf_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_start_time = time.time()
xgb_clf.fit(X_train, y_train)
xgb_elapsed_time = time.time() - xgb_start_time
alltimes['xgb_elapsed_time'] = xgb_elapsed_time
```

```
print_score(xgb_clf, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, xgb_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(20, input_dim=20, kernel_initializer='he_uniform', activation='relu'))
model.add(Dense(1, kernel_initializer='he_uniform'))
model.compile(loss='mae', optimizer='adam',metrics=['accuracy'])

ann_start_time = time.time()
model.fit(X_train, y_train, epochs=150, batch_size=10)
ann_elapsed_time = time.time() - ann_start_time
alltimes['ann_elapsed_time'] =ann_elapsed_time

_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
train_score = 91.93
test_score = 81.67
results_df_2 = pd.DataFrame(data=[["ANN", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
alltimes
```

-Mushrooms dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
dt = pd.read_csv('mushrooms.csv')
dt.head()
for x in dt.columns:
    print(x)
dt.loc[dt['class']=='p','class'] = 1
dt.loc[dt['class']=='e','class'] = 0
dt.head()
dt.isnull().sum()
dt['class'].value_counts().plot(kind='bar',color=["salmon", "lightblue"])
d3 = dt.copy()
dt.info()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in dt.columns:
    dt[col] = le.fit_transform(dt[col])
dt.info()
dataset = pd.get_dummies(dt, columns = dt.drop('class',axis=1).columns)
dataset.head()
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
```

```
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n==========")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_____")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
from sklearn.model_selection import train_test_split

X = dataset.drop('class', axis=1)
y = dataset['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
from sklearn.linear_model import LogisticRegression

alltimes = {}
lr_clf = LogisticRegression(solver='liblinear')
lr_start_time = time.time()
lr_clf.fit(X_train, y_train)
lr_elapsed_time = time.time() - lr_start_time
alltimes['lr_elapsed_time'] = lr_elapsed_time

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)

test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100
```

```
results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_start_time = time.time()
knn_clf.fit(X_train, y_train)
knn_elapsed_time = time.time() - knn_start_time
alltimes['knn_elapsed_time'] = knn_elapsed_time

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.decomposition import PCA
pca = PCA(n_components=2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
from sklearn.linear_model import LogisticRegression

alltimes = {}
lr_clf = LogisticRegression(solver='liblinear')
lr_start_time = time.time()
lr_clf.fit(X_train, y_train)
lr_elapsed_time = time.time() - lr_start_time
alltimes['lr_elapsed_time'] = lr_elapsed_time
```

```
print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
knn_clf = KNeighborsClassifier()
knn_start_time = time.time()
knn_clf.fit(X_train, y_train)
knn_elapsed_time = time.time() - knn_start_time
alltimes['knn_elapsed_time'] = knn_elapsed_time

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.svm import SVC



svm_clf = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, svm_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(random_state=42)
tr_start_time = time.time()
tree_clf.fit(X_train, y_train)
tr_elapsed_time = time.time() - tr_start_time
alltimes['tr_elapsed_time'] =tr_elapsed_time

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, tree_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rf_clf = RandomForestClassifier(n_estimators=1000, random_state=42)
rf_start_time = time.time()
rf_clf.fit(X_train, y_train)
rf_elapsed_time = time.time() - rf_start_time
alltimes['rf_elapsed_time'] = rf_elapsed_time
```

```
print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, rf_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_start_time = time.time()
xgb_clf.fit(X_train, y_train)
xgb_elapsed_time = time.time() - xgb_start_time
alltimes['xgb_elapsed_time'] = xgb_elapsed_time

print_score(xgb_clf, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, xgb_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(12, input_dim=2, activation='relu'))
model.add(Dense(8, activation='relu'))
```

```
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
ann_start_time = time.time()
model.fit(X_train, y_train, epochs=150, batch_size=10)
ann_elapsed_time = time.time() - ann_start_time
alltimes['ann_elapsed_time'] =ann_elapsed_time


_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
test_score = accuracy_score(y_test, model.predict_classes(X_test)) * 100
train_score = accuracy_score(y_train, model.predict_classes(X_train)) * 100


results_df_2 = pd.DataFrame(data=[["ANN", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
alltimes
```

-Breast cancer dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
dt = pd.read_csv('data.csv')
dt.head()
dt.drop(['Unnamed: 32','id'],axis = 1,inplace= True)
dt.head()
dt.isnull().sum()
dt.describe()
y = dt.diagnosis
X = dt.drop(['diagnosis'], axis = 1)
dt.diagnosis.value_counts()
dt.diagnosis.value_counts().plot(kind = 'bar')
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.decomposition import PCA
pca = PCA(n_components=4)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_____________________________________________")
```

```
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
from sklearn.preprocessing import StandardScaler

s_sc = StandardScaler()
X_train = s_sc.fit_transform(X_train)
X_test = s_sc.fit_transform(X_test)
from sklearn.linear_model import LogisticRegression

alltimes = {}
lr_clf = LogisticRegression(solver='liblinear')
lr_start_time = time.time()
lr_clf.fit(X_train, y_train)
lr_elapsed_time = time.time() - lr_start_time
alltimes['lr_elapsed_time'] = lr_elapsed_time

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)

test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100
```

```
results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_start_time = time.time()
knn_clf.fit(X_train, y_train)
knn_elapsed_time = time.time() - knn_start_time
alltimes['knn_elapsed_time'] = knn_elapsed_time

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.svm import SVC



svm_clf = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)

test_score = accuracy_score(y_test, svm_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, test_score]],
```

```
                columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(random_state=42)
tr_start_time = time.time()
tree_clf.fit(X_train, y_train)
tr_elapsed_time = time.time() - tr_start_time
alltimes['tr_elapsed_time'] =tr_elapsed_time

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, tree_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, test_score]],
                columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rf_clf = RandomForestClassifier(n_estimators=1000, random_state=42)
rf_start_time = time.time()
rf_clf.fit(X_train, y_train)
rf_elapsed_time = time.time() - rf_start_time
alltimes['rf_elapsed_time'] = rf_elapsed_time

print_score(rf_clf, X_train, y_train, X_test, y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test, y_test, train=False)
```

```
test_score = accuracy_score(y_test, rf_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from xgboost import XGBClassifier

xgb_clf = XGBClassifier()
xgb_start_time = time.time()
xgb_clf.fit(X_train, y_train)
xgb_elapsed_time = time.time() - xgb_start_time
alltimes['xgb_elapsed_time'] = xgb_elapsed_time

print_score(xgb_clf, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_clf, X_train, y_train, X_test, y_test, train=False)
test_score = accuracy_score(y_test, xgb_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
from keras.models import Sequential
from keras.layers import Dense

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in dt.columns:
    y_train = le.fit_transform(y_train)
    y_test = le.fit_transform(y_test)
model = Sequential()
```

```
model.add(Dense(12, input_dim=4, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
ann_start_time = time.time()
model.fit(X_train, y_train, epochs=150, batch_size=10)
ann_elapsed_time = time.time() - ann_start_time
alltimes['ann_elapsed_time'] =ann_elapsed_time

_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
test_score = accuracy_score(y_test, model.predict_classes(X_test)) * 100
train_score = accuracy_score(y_train, model.predict_classes(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["ANN", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
alltimes
```