Επιστήμη των Δεδομένων

Δημοκρίτειο Πανεπιστήμιο Θράκης

Αλεξάκης Γεώργιος 58093

Κομνηνός Βασίλειος 58051

Ηλεκτρονικές Μετρήσεις 11 Μαΐου 2023

Περίληψη

Η συγκεκριμένη εργασία γίνεται με σκοπό την εξοικείωση μας με την επιστήμη των δεδομένων και συγκεκριμένα τις τεχνικές, μεθόδους και τους αλγορίθμους αυτής και της μηχανικής μάθησης. Στα πλαίσια της εργασίας α αναλυθούν τα διάφορα προβλήματα που προσπαθεί να λύσει η επιστήμη των δεδομένων όπως είναι η ταξινόμηση, η παλινδρόμηση και η ομαδοποίηση.

Η ταξινόμηση είναι και το πιο γνωστό πρόβλημα όταν κάποιος σκέφτεται την επιστήμη των δεδομένων. Θα δούμε διάφορα ήδη ταξινόμησης όπως είναι η δυαδική και η κατηγορική. Θα αναλύσουμε επίσης διάφορους από τους πιο διάσημους αλγόριθμους ταξινόμησης κάποιοι από τους οποίους μπορεί να είναι παραμετρικοί ενώ άλλοι όχι, και γενικότερα θα δούμε τις χρήσεις και τα πλεονεκτήματα του κάθε αλγορίθμου στον τομέα αυτόν.

Η παλινδρόμηση είναι επίσης ένα σημαντικό πρόβλημα που έρχεται να λύσει η επιστήμη των δεδομένων το οποίο είναι πολύ παρόμοιο με την ταξινόμηση. Μάλιστα υπάρχουν πάρα πολλοί αλγόριθμοι που είναι κατάλληλοι και για τα δύο προβλήματα. Η διαφορά της ταξινόμησης με την παλινδρόμηση είναι ότι ενώ στην ταξινόμηση προσπαθούμε να κατηγοριοποιήσουμε μία είσοδο σε κάποιες διακριτές κατηγορίες και να της δώσουμε μία ¨ταμπέλα¨, στην παλινδρόμηση προσπαθούμε να αντιστοιχίσουμε μια είσοδο σε μία διακριτή ή και συνεχή έξοδο. Το πιο απλό παράδειγμα θα ήταν η προσέγγιση μια συνάρτησης χρησιμοποιώντας μόνο τις εισόδους και εξόδους αυτής.

Η ομαδοποίηση είναι μια μέθοδος η οποία ίσως δεν έχει τόσες χρήσεις σε σύγκριση με τις δύο προηγούμενες. Παρ΄ όλα αυτά είναι και αυτή μια εξίσου σημαντική μέθοδος καθώς και είναι μία μή παραμετρική μέθοδος (κάτι το οποίο είναι χρήσιμο στη μηχανική μάθηση). Αυτό που προσπαθεί να πετύχει είναι να χωρίσει ένα σύνολο δεδομένων σε ομάδες ανάλογα με την ομοιότητα των στοιχείων. Παρόλα αυτά δεν μπορούμε να ελέγξουμε αυτές τις ομάδες αλλά δημιουργούνται αποκλειστικά από τον αλγόριθμο.

Επίσης θα αναλύσουμε σε βάθος την κυρίαρχη δομή που μπορεί να λύσει και τα τρία παραπάνω προβλήματα με διάφορες παραλλαγές της και αυτή είναι το νευρωνικό δίκτυο. Θα δούμε αναλυτικά τον τρόπο λειτουργίας τους και τι τα κάνει να ξεχωρίσουν από τις υπόλοιπες τεχνικές. Επιπλέον στα πλαίσια αυτης της εργασίας θα υλοποιήσουμε το δικό μας νευρωνικό δίκτυο από την αρχή.

Περιεχόμενα

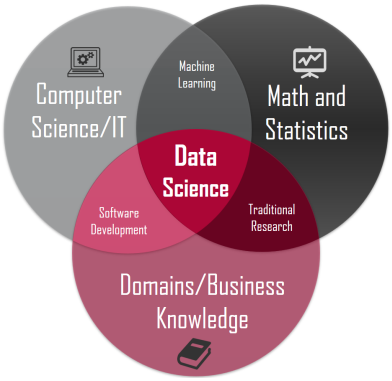
Κατάλογος Σχημάτων

1 Εισαγωγή

Η επιστήμη των δεδομένων (Data science) είναι μια επιστήμη που αφορά την εξαγωγή γνώσης από δεδομένα (δομημένα ή αδόμητα) και για να το πετύχει αυτό συνδυάζει γνώσεις από διάφορες άλλες επιστήμες όπως:

- Μαθηματικά
- Στατιστική
- Προγραμματισμός
- Προγνωστική ανάλυση (Predictive analysis)
- Εξόρυξη δεδομένων (Data mining)
- Τεχνητή Νοημοσύνη (Artificial intelligence)
- Μηχανική μάθηση (Machine learning)

Επιπλέον, η επιστήμη των δεδομένων συνδυάζει τα παραπάνω με τη γνώση κάποιου ειδικού τομέα (όπως η ιατρική) με σκοπό να δώσει λύση σε ένα συγκεκριμένο πρόβλημα[1]



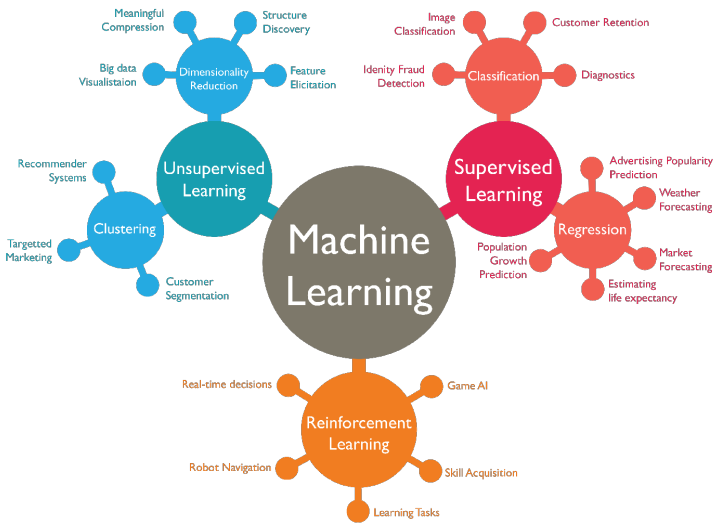
Σχήμα 1: Γνώσεις που συνδυάζει η επιστήμη των δεδομένων

Σε αυτή την εργασία θα ασχοληθούμε κυρίως με την τεχνητή νοημοσύνη και τη μηχανική μάθηση και πιο συγκεκριμένα με τους αλγόριθμους που εφαρμόζονται στην επιστήμη των δεδομένων. Θα δοθεί αναλυτική εξήγηση για πολλούς χρήσιμους αλγόριθμους και θα υλοποιηθούν οι πιο σημαντικοί από αυτούς με τη χρήση της γλώσσας προγραμματισμού C++ η οποία θα μας δώσει τη δυνατότητα να φτιάξουμε γρήγορα προγράμματα.

Οι αλγόριθμοι που θα εξετάσουμε μπορούν να χωριστούν σε πέντε ομάδες με βάση τη χρήση τους και αυτές είναι:

- Αλγόριθμοι ταξινόμησής (Classification)
- Αλγόριθμοι παλινδρόμησης (Regression)
- Αλγόριθμοι ομαδοποίησης (Clustering)
- Αλγόριθμοι μείωσης διαστάσεων (Dimentionality reduction)
- Αλγόριθμοι ενισχυτικής μάθησης (Reinforcement learning)
- Αλγόριθμοι ανίχνευσης ανωμαλίας (Anomaly detection)

Μας ενδιαφέρουν ιδιαίτερα οι πρώτες τρεις ομάδες καθώς είναι αυτές που εμφανίζονται πιο συχνά.Οι παλινδρομητές και οι ταξινομητές είναι πολύ παρόμοιοι αλγόριθμοι οι οποίοι ανήκουν στους αλγόριθμους εποπτευόμενης μάθησης (Supervised learning) και χρησιμοποιούνται για να κάνουν προβλέψεις σύμφωνα με τα δεδομένα που τους έχουμε δώσει. Η διαφορά τους είναι ότι η ταξινόμηση προσπαθεί να προβλέψει την κλάση των καινούριων δεδομένων και να τα κατηγοριοποιήσει σύμφωνα με τις υπάρχουσες κατηγορίες, ενώ η παλινδρόμηση προσπαθεί να προβλέψει την τιμή κάποιου στοιχείου για τα καινούρια δεδομένα σύμφωνα με μια συνάρτηση που έφτιαξε από τα υπάρχοντα δεδομένα. Οι περισσότεροι αλγόριθμοι ταξινόμησης είναι και αλγόριθμοι παλινδρόμησης και το αντίστροφο. Από την άλλη οι αλγόριθμοι ομαδοποίησης είναι μη εποπτευόμενης μάθησης και χρησιμοποιούνται σε αδόμητα δεδομένα για να τα χωρίσουν σε ομάδες.



Σχήμα 2: Αλγόριθμοι μηχανικής μάθησης

2 Ταξινόμηση

Σε αυτή την ενότητα θα αναλύσουμε τους αλγορίθμους ταξινόμησης και θα δούμε τη χρήση τους. Για τους αλγορίθμους αυτούς χωρίζουμε το σύνολο των δεδομένων μας σε δυο μέρη:

- Δεδομένα εκπαίδευσης (training dataset)
- Δεδομένα επαλήθευσής (testing dataset)

Τα πρώτα τα χρησιμοποιούμε έτσι ώστε ο αλγόριθμος να βρει κάποιο μοτίβο στα δεδομένα με το οποίο θα μπορεί να κατατάσσει τα καινούρια δεδομένα που δέχεται σε κάποια από τις υπάρχουσες κλάσεις. Αυτή είναι και η διαδικασία εκπαίδευσης του μοντέλου. Αφού η εκπαίδευση τελειώσει τότε θα χρησιμοποιήσουμε τα δεδομένα επαλήθευσής για να επιβεβαιώσουμε την ορθή λειτουργία του μοντέλου. Υπάρχουν τρεις βασικοί τύποι ταξινομητών:

- Δυαδικοί (binary)
- Πολλαπλών κλάσεων (multy-class)
- Πολλαπλών ετικετών (multy-label)

Οι δυαδικοί ταξινομητές χρησιμοποιούνται όταν έχουμε μόνο δύο κλάσεις στις οποίες θέλουμε να εντάξουμε τα δεδομένα, ή όταν η απάντηση που θέλουμε να πάρουμε από το μοντέλο είναι δυαδικής φύσης. Για παράδειγμα, ένα πρόβλημα δυαδικής φύσης θα ήταν να προβλέψουμε εάν ένας ασθενής έχει ή δεν έχει μια ασθένειά σύμφωνα με τις εξετάσεις του.

Οι ταξινομητές πολλαπλών κλάσεων από την άλλη είναι ικανοί να αναγνωρίσουν περισσότερες από δύο κλάσεις και είναι πολύ χρήσιμοι για την αναγνώριση προτύπων. Συνεχίζοντας με το προηγούμενο παράδειγμα θα θέλαμε χωρίσουμε τους ασθενείς σύμφωνα με την κατάσταση τους σε:

- υγιείς
- ήπια ασθένειά
- σοβαρή ασθένεια

΄Ετσι οι γιατροί θα μπορούν αν δράσουν ανάλογα.

Οι ταξινομητές πολλαπλών ετικετών δεν έχουν κάποια καινούρια λογική αλλά εφαρμόζουν τις λογικές των προηγούμενων προβλημάτων. Δηλαδή θα μπορούσαμε να θέλουμε να υλοποιήσουμε ένα μοντέλο το οποίο να κάνει και τις δύο

προηγούμενες προβλέψεις που συζητήσαμε. Αυτά τα μοντέλα συνήθως δε χρησιμοποιούν ξεχωριστούς αλγορίθμους αλλά συνδυάζουν πολλούς ήδη γνωστούς αλγορίθμους για να φτάσουν στο αποτέλεσμα.

Σχήμα 3: Τύποι ταξινομητών

Στη συνέχεια θα αναλύσουμε τους διασημότερους αλγορίθμους και τη χρήση τους. Οι αλγόριθμοι αυτοί είναι[2·3·4]:

- Naive Bayes
- LDA (Linear Discriminant Analysis)
- QDA (Quadratic Discriminant Analysis)
- SVM (Support Vector Machine)
- k-NN (k Nearest Neighbors)
- Decision trees
- Random Forest
- Νευρωνικά δίκτυα (τα οποία θα αναλύσουμε σε ξεχωριστή ενότητα)

2.1 Naive Bayes

Σύμφωνα με το παρακάτω άρθρο[5] ο αλγόριθμος Naive Bayes είναι ένας αλγόριθμος που κάνει την υπόθεση ότι τα χαρακτηριστικά είναι υπό όρους ανεξάρτητα της δεδομένης κλάσης. Αυτή η υπόθεση στην πραγματικότητα δεν ισχύει αλλά

ο αλγόριθμος πετυχαίνει πολύ υψηλή ακρίβεια και ταυτόχρονα μεγάλη υπολογιστική απόδοση. Αυτός είναι και ο λόγος που χρησιμοποιείται τόσο συχνά στην επιστήμη των δεδομένων.

Τα πλεονεκτήματα του αλγορίθμου είναι[6]:

- Η χρονική πολυπλοκότητα αυξάνεται γραμμικά με το πλήθος των δειγμάτων και των στοιχείων τους
- Χαμηλό variance (η αλλαγή των δεδομένων δεν επηρεάζει πολύ το μοντέλο)
- Μπορούμε εύκολα να προσθέσουμε και άλλα δεδομένα και το μοντέλο θα συνεχίσει την εκπαίδευση χωρίς πρόβλημα
- Δεν είναι επιρρεπής στον θόρυβο
- Δεν επηρεάζεται από έλλειψη τιμών στα δεδομένα

Τα μειονεκτήματά είναι:

- Η υπόθεση ότι τα δεδομένα είναι ανεξάρτητα τον κάνει ακατάλληλο για ορισμένα προβλήματα
- Οι συνδυασμοί στοιχείων που δεν εμφανίζονται στο σύνολο δεδομένων για κάποια πρόβλεψη θα έχει πάντα μηδενική πιθανότητα.
- Οι πιθανότητες που υπολογίζει ενδέχεται να είναι λάθος

Ο αλγόριθμος χρησιμοποιεί τον κανόνα του Bayes:

$$P(y|X) = \frac{P(y) \times P(X|y)}{P(X)}$$

Ο παραπάνω τύπος θα μας δώσει την πιθανότητα ενός δείγματος με στοιχεία X να ανήκει στην κλάση y. Το X είναι διάνυσμα με όλα τα στοιχεία του δείγματος μας. Στο παρακάτω παράδειγμα το X για το πρώτο δείγμα θα είναι $\langle 40, 85 \rangle$ και η κλάση θα είναι $y = 0$. Σκοπός μας είναι όταν πάρουμε τα στοιχεία από έναν ασθενή να μπορούμε να προβλέψουμε αν έχει διαβήτη. Θα πρέπει δηλαδή να υπολογίσουμε τις πιθανότητες $P\left(y = 0|X = \langle x_1, x_2 \rangle\right)$ και $P\left(y = 1|X = \langle x_1, x_2 \rangle\right)$ για τον καινούριο ασθενή με μετρήσεις $x1, x2$ και η πρόβλεψη θα είναι αυτή με τη μεγαλύτερη πιθανότητα.

Γλυκόζη	Αρτηριακή πίεση	Διαβήτης
40	85	0
40	92	0
45	63	1
45	80	0
40	73	1
45	82	0
40	85	0
30	63	1
65	65	1
45	82	0
35	73	1
45	90	0
50	68	1
40	93	0
35	80	1
50	70	1

Πίνακας 1: Δείγματα για πρόβλεψη διαβήτη

Μέχρι εδώ ήταν η γνωστή στατιστική. Αλλά όσα περισσότερα στοιχεία έχουμε ανά δείγμα τόσο πιο πολύπλοκος γίνεται ο υπολογισμός του παράγοντα $P(X|y)$. Εδώ δίνει λύση στο πρόβλημα ο αλγόριθμος Naive Bayes με την ανεξαρτησία των στοιχείων. Ανεξαρτησία σημαίνει ότι

$$P(\langle x_1, x_2, \ldots, x_n \rangle | y) = \prod_{i=1}^{n} P(x_i|y)$$

Επίσης το $P(X)$ είναι πάντα σταθερό για τα δεδομένα στοιχεία που έχουμε άρα τελικά πρέπει να υπολογίσουμε το

$$P(y) \times \prod_{i=1}^{n} P(x_i|y)$$

για κάθε y και στο τέλος να διαλέξουμε εκείνο το y που έδωσε τη μεγαλύτερη τιμή (που θα είναι και η κλάση στην οποία ανήκει το δείγμα). Βέβαια, αν τα στοιχεία δεν είναι υπό όρους ανεξάρτητα δεν ισχύει αυτή η απλοποίηση. Παρ΄όλα αυτά μπορεί να χρησιμοποιηθεί σε πολλά σύνολα δεδομένων παρόλο που κάποιες φορές δεν ισχύει και για αυτόν τον λόγο καλείται και αφελής Bayes.

Ο αλγόριθμος είναι ιδανικός για προβλήματα πολλαπλών κλάσεων λόγω του εύκολου υπολογισμού. Πιο συγκεκριμένα ο Naive Bayes χρησιμοποιείται:

- Στην ανίχνευση spam. Μπορεί διαβάζοντας τις λέξεις ενός mail να βρει την πιθανότητα να είναι spam. Κάθε λέξη που διαβάζει θα αυξάνει ή θα μειώνει την πιθανότητα με κάποιον προκαθορισμένο κανόνα μέχρι που στο τέλος θα έχουμε πάρει μια ικανοποιητική απάντηση.

- Στη συναισθηματική ανάλυση. Οι εταιρίες μπορούν να κατατάξουν τα σχόλια των καταναλωτών τους σε θετικά και αρνητικά χρησιμοποιώντας τον αλγόριθμο. Μετά μπορούν να πράξουν ανάλογα έτσι ώστε να ευνοήσουν την εταιρία.

Υπάρχουν πολλές άλλες χρήσεις πέρα από αυτές. Για μια πιο αναλυτική εξήγηση των μαθηματικών του αλγορίθμου μπορείτε να διαβάσετε την εξής έρευνα[7].

Οι αλγόριθμοι QDA και QDA χρησιμοποιούν επίσης τον τύπο του Bayes. Η διαφορά είναι στον τρόπο υπολογισμού του παράγοντα $P(X|y)$. Όπως και Ο Αλγόριθμος Naive Bayes κάνει μια υπόθεση για να τον υπολογίσει, το ίδιο κάνουν και αυτοί οι αλγόριθμοι. Για να δείτε αυτούς τους τύπους με πλήρη εξήγηση μπορείτε να δείτε το παράρτημα Α΄.

2.2 Support Vector Machine

Οι μηχανές διανυσμάτων υποστήριξης (SVM) είναι ένα σύνολο μεθόδων εποπτευόμενης μάθησης μέθοδοι που χρησιμοποιούνται για ταξινόμηση και παλινδρόμηση. Ανήκουν σε μια οικογένεια γενικευμένων γραμμικών ταξινομητών. Ένα βασικό χαρακτηριστικό του είναι ότι είναι κάνει προβλέψεις που χρησιμοποιούν τη θεωρία μηχανικής μάθησης για να μεγιστοποιήσει την προγνωστική ακρίβεια ενώ ταυτόχρονα αποφεύγει την υπερβολική προσαρμογή στα δεδομένα (over-fitting)[8].

Τα πλεονεκτήματα του SVM είναι[9]:

- Δουλεύει πολύ καλά όταν υπάρχει καθαρή διαφοροποίηση στα δεδομένα

- Δουλεύει καλύτερα σε μεγαλύτερες διστάσεις (πολλά στοιχεία ανά δείγμα)

- Είναι αποδοτικός όταν τα στοιχεία ανά δείγμα είναι περισσότερα από τα δείγματα

- Δεν κάνει μεγάλη χρήση μνήμης

Μειονεκτήματά:

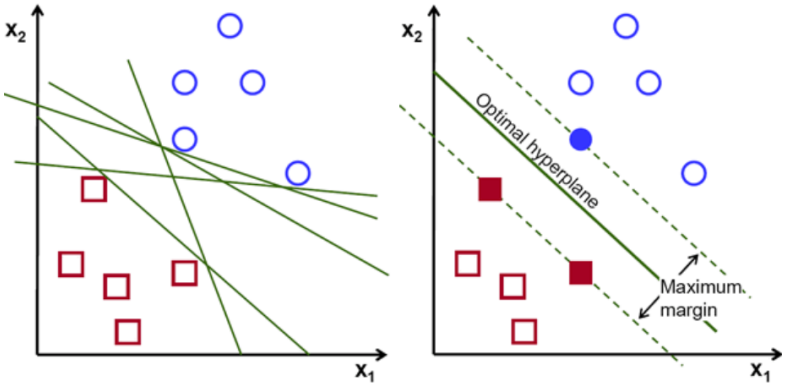
- Δεν είναι κατάλληλος για μεγάλο σύνολο δεδομένων

- Είναι επιρρεπής στον θόρυβο

- Λόγω του τρόπου υλοποίησης του δεν μπορούμε να πάρουμε την πιθανότητα αλλά μόνο την πρόβλεψη

Ο αλγόριθμος αυτός είναι αρκετά απλός και χρησιμοποιείται κυρίως για δυαδική ταξινόμηση. Αρχικά για να τον καταλάβουμε θα πρέπει να αναπαραστήσουμε τα δεδομένα μας σαν σημεία σε έναν χώρο N διαστάσεων, όπου N τα στοιχεία που έχουμε ανά δείγμα. Το κάθε στοιχείο θα ανήκει σε μια από τις δύο κατηγορίες τις οποίες θέλουμε να αναγνωρίσουμε. Ο αλγόριθμος λοιπόν προσπαθεί να φέρει ένα υπερεπίπεδο τέτοιο ώστε να χωρίζει τα σημεία ανάλογα με την κατηγορία τους. Έτσι τα σημεία της μιας κατηγορίας θα βρίσκονται από τη μια μεριά και της δεύτερης από την άλλη.

Για να κατανοήσουμε καλύτερα τον αλγόριθμο θα κάνουμε ένα παράδειγμα σε έναν χώρο δύο διαστάσεων που μπορεί εύκολα να αντιληφθεί το ανθρώπινο μάτι.

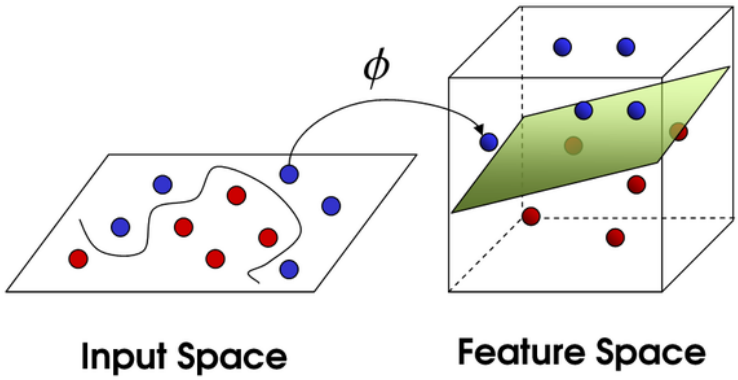


Σχήμα 4: Παράδειγμα SVM

Στον χώρο δύο διαστάσεων το υπερεπίπεδο είναι απλώς μια ευθεία. Εμείς λοιπόν θέλουμε να βρούμε μια ευθεία που να χωρίζει τα τετράγωνα με τους κύκλους. Όπως μπορούμε να δούμε από το πρώτο διάγραμμα υπάρχουν πολλές ευθείες που μπορεί να χωρίζουν τα σημεία μας αλλά εμείς θέλουμε να διαλέξουμε την ευθεία που να έχει τη μέγιστη απόσταση και από τις δύο μεριές. Επομένως, όταν δώσουμε ένα καινούριο δείγμα στον αλγόριθμο και θέλουμε να το κατατάξει σε μια από τις δύο ομάδες πρέπει απλά να το τοποθετήσει σε αυτόν τον χώρο και μετά να δει από ποια μεριά της ευθείας βρίσκεται. Ανάλογα με τη μεριά αυτή θα αποφασίσουμε και την κλάση του αντικειμένου.

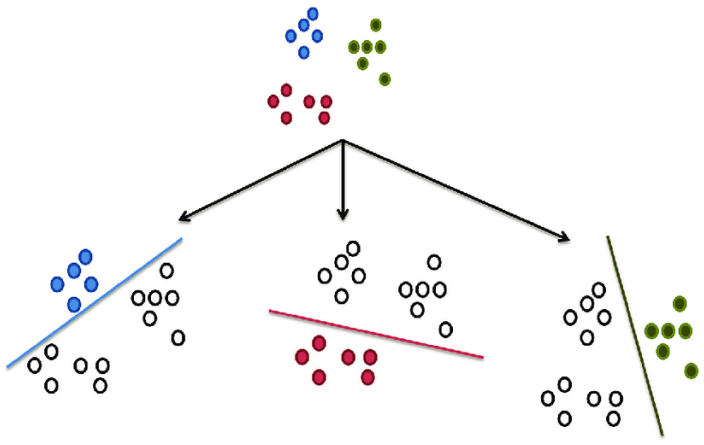
Λύσαμε το πρόβλημα όπου είχαμε πάνω από μία ευθεία αλλά υπάρχει ένα μεγαλύτερο πρόβλημα. Αυτό είναι η περίπτωση που δεν μπορούμε να χωρίσουμε τα στοιχεία μας με μια ευθεία. Εδώ έρχεται το λεγόμενο ¨kernel trick¨ για να

λύσει το πρόβλημα. Το kernel είναι μια συνάρτηση σύμφωνα με την οποία επαυξάνουμε τη διάσταση του χώρου μας έτσι ώστε με την καινούρια διάσταση που θα δημιουργηθεί τα σημεία μας να είναι διαχωρίσιμα από πλέον ένα επίπεδο. Όταν γυρίσουμε πίσω στην αρχική διάσταση θα φαίνεται σαν τα σημεία μας να είναι διαχωρισμένα με μια καμπύλη.



Σχήμα 5: Επαύξηση διάστασης με SVM

Ένα τελευταίο πρόβλημα που θα μπορούσαμε να λύσουμε θα ήταν να τροποποιήσουμε τον αλγόριθμο έτσι ώστε να δουλεύει για πάνω από δύο κλάσεις. Ένας τρόπος να το κάνουμε αυτό θα ήταν με τη μέθοδο ONE versus ALL. Αυτό που κάνουμε είναι να φέρουμε μια ευθεία για κάθε κλάση που έχουμε η οποία θα χωρίζει τον χώρο στα στοιχεία που ανήκουν σε αυτήν την κλάση και σε όλα τα υπόλοιπα. Έτσι συνδυάζοντάς την πληροφορία και από τις τρεις ευθείες μπορούμε να πάρουμε τη σωστή απάντηση.



Σχήμα 6: Ταξινόμηση πολλαπλών κλάσεων με SVM

Βέβαια, κάνοντας αυτή τη μέθοδο θα εμφανιστουν σημεία στον χώρο τα οποία μπορεί να ανήκουν σε πάνω από μια κλάση και και άλλα που δεν θα ανήκουν σε καμία. Θα πρέπει λοιπόν να λάβουμε υπόψη μας όχι μόνο από ποια μεριά της κάθε ευθείας βρισκόμαστε αλλά και την απόσταση από αυτές για να μπορέσουμε να καταλήξουμε σε ένα σωστό συμπέρασμα.

Μερικές χρήσεις του αλγορίθμου είναι[10]:

- Ανίχνευση προσώπου
- Κατηγοριοποίηση κειμένου
- Ταξινόμηση εικόνων
- Βιοπληροφορική (πχ. ανίχνευση καρκίνου)
- Αναγνώριση γραφικού χαρακτήρα
- Γενικευμένος προγνωστικος έλεγχος (έλεγχος χαοτικών συστημάτων)

2.3 K Nearest Neighbors

Ο αλγόριθμος k-NN είναι ένας μη παραμετρικός αλγόριθμος ο οποίος χρησιμοποιεί την εγγύτητα για να κατατάξει τα καινούρια στοιχεία σε μια από τις υπάρχουσες κλάσεις[11]. Αυτό σημαίνει ότι το κάθε καινούριο στοιχείο θα παίρνει την κλάση του βλέποντας τις κλάσεις των πιο κοντινών γειτόνων του διαλέγοντας την κλάση της πλειοψηφίας.

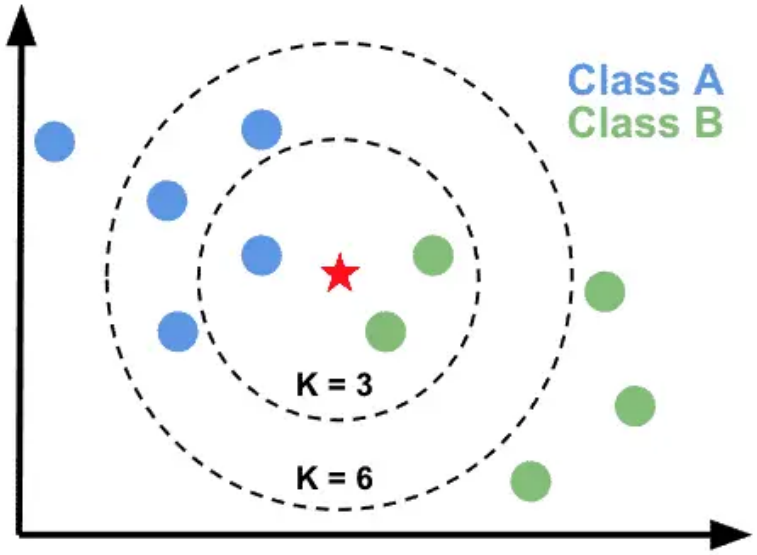
Τα πλεονεκτήματα του k-NN είναι:

- Εύκολη υλοποίηση
- Εύκολη προσαρμογή κατά την εμφάνιση καινούριων δεδομένων
- Δε χρειάζεται παραμέτρους παρά μόνο το k και μια μετρική απόστασης

Κάποια μειονεκτήματά του είναι:

- Δεν έχει καλή επεκτασιμότητα
- Δε δουλεύει καλά με μεγάλες διαστάσεις
- Μπορεί να γίνει εύκολα over fitting

Για να ξεκινήσουμε τον αλγόριθμο πρέπει πρώτα να ορίσουμε ένα k. Αυτό θα μας λέει πόσα γειτονικά δείγματα να λάβουμε υπόψη μας για τον προσδιορισμό της κλάσης του καινούριου δείγματος. Έπειτα βρίσκουμε τις αποστάσεις του σημείου από όλα τα σημεία με μια από τις μετρικές που θα αναλύσουμε παρακάτω και διαλέγουμε τα k σημεία με την κοντινότερη απόσταση στο δικό μας. Τέλος, η κλάση του σημείου μας θα είναι η κλάση που έχει η πλειοψηφία των k σημείων.



Σχήμα 7: Αλγορίθμος k-NN

Για της μετρικές απόστασης γενικότερα στην επιστήμη των δεδομένων και στη μηχανική μάθηση έχουμε τέσσερις βασικές επιλογές:

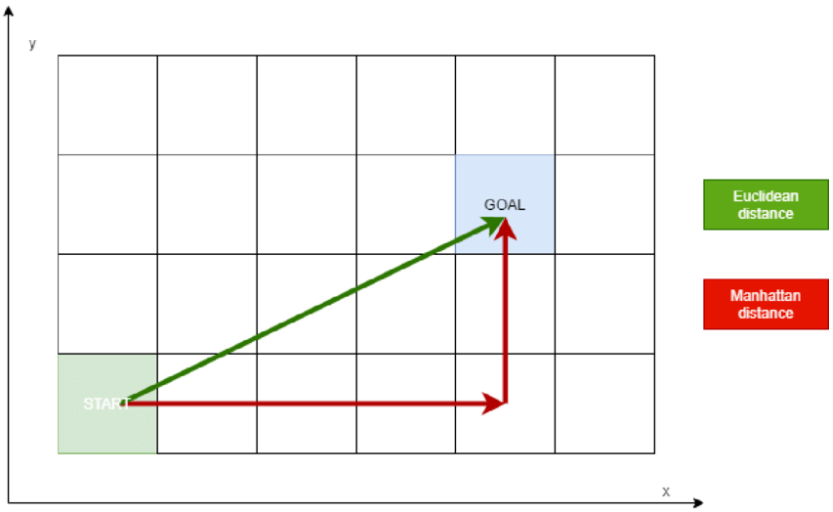
- Ευκλείδεια απόσταση
- Απόσταση Manhattan
- Απόσταση Minkowski
- Απόσταση Hamming

Η Ευκλείδεια απόσταση είναι η πιο γνωστή μετρική και γραφικά είναι το μήκος της ευθείας γραμμής που ενώνει τα δύο σημεία μεταξύ τους. Δίνεται από τον γνωστό τύπο:

$$D = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Η απόσταση Manhattan είναι επίσης πολύ γνωστή την οποί μπορούμε να παραστήσουμε γραφικά όπως το παρακάτω σχήμα και έχει τύπο:

$$D = \sum_{i=1}^{n} |x_i - y_i|$$

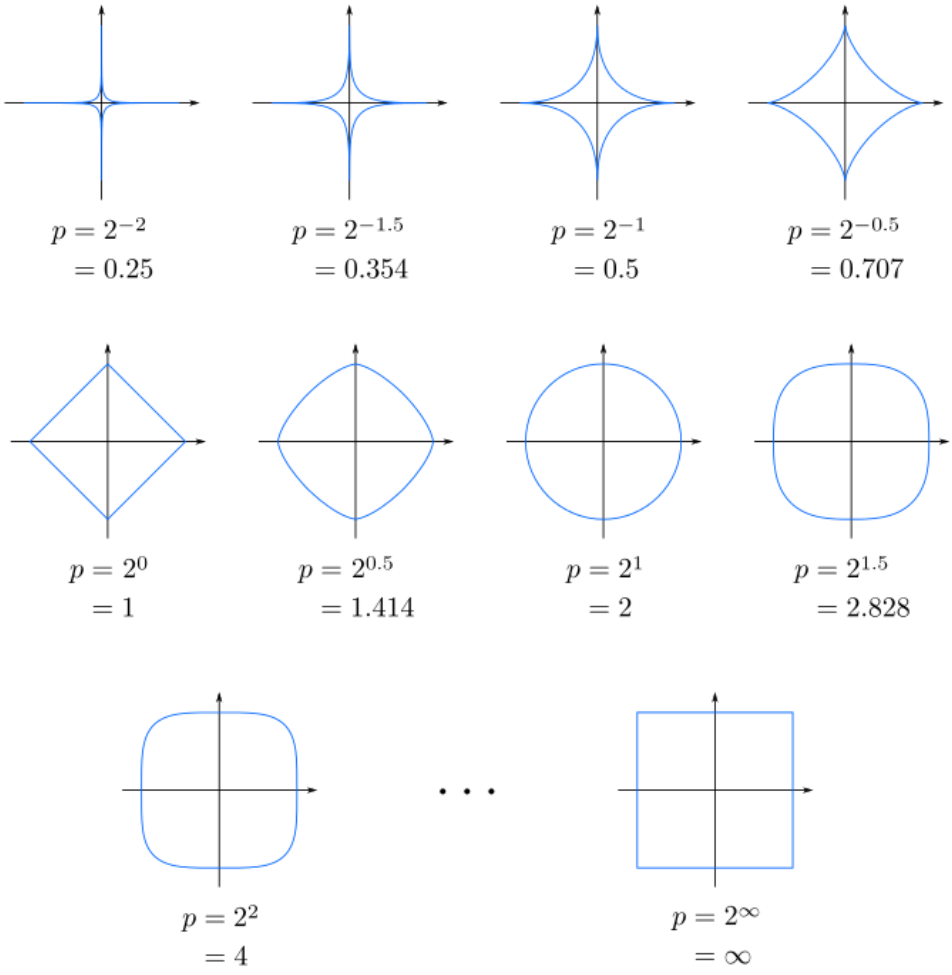


Σχήμα 8: Ευκλείδεια απόσταση και Απόσταση Manhattan

Η απόσταση Minkowski είναι ένα σύνολο αποστάσεων που αποτελεί μια γενίκευση των παραπάνω δύο με τύπο:

$$D = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Όπως μπορούμε αν δούμε πως για $p = 1$ και $p = 2$ παίρνουμε την απόσταση Manhattan και την ευκλείδεια απόσταση αντίστοιχα. Δεν μπορούμε να την αναπαραστήσουμε γραφικά αλλά για περαιτέρω κατανόησή μπορούμε να δούμε το παρακάτω σχήμα όπου όλα τα σημεία στην μπλε γραμμή ισαπέχουν από το κέντρο τους με μετρική την απόσταση Minkowski για διαφορετικά p



Σχήμα 9: Απόσταση Minkowski

Τέλος, η απόσταση Hamming είναι πολύ απλή και εφαρμόζεται σε δυαδικά διανύσματα και είναι το πλήθος των συνιστωσών που είναι διαφορετικές μεταξύ τους. Για παράδειγμα, αν έχουμε τα διανύσματα $x = \langle 0, 1, 1, 0, 1 \rangle, y = \langle 1, 1, 0, 0, 0 \rangle$ η απόσταση τους θα είναι 3 γιατί οι πρώτες, τρίτες και πέμπτες συνιστώσες είναι διαφορετικές μεταξύ τους.

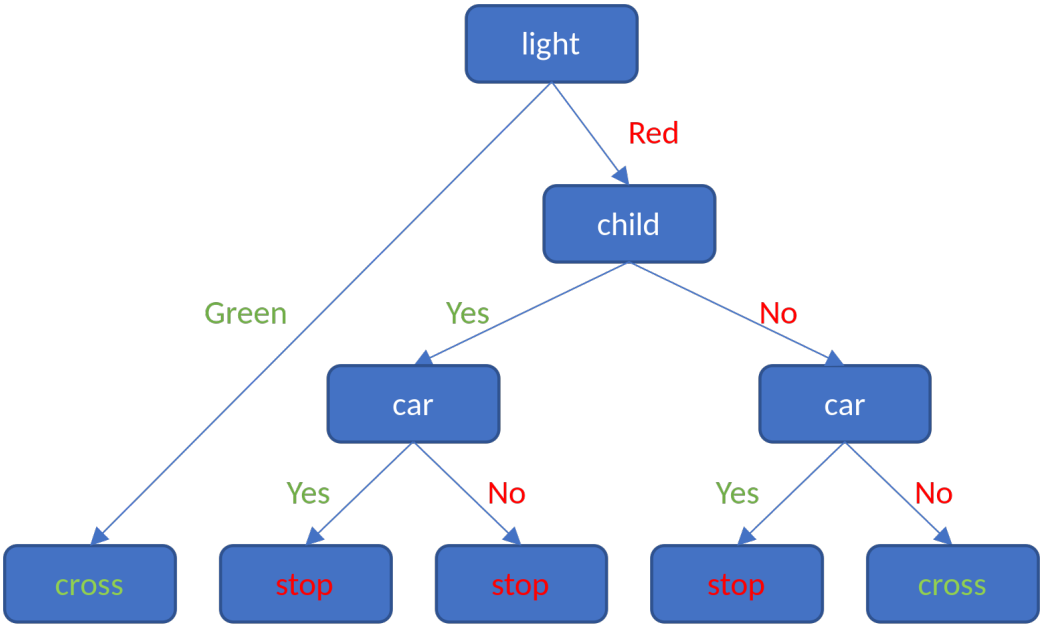
Οι χρήσεις του k-NN είναι πολύ παρόμοιες με αυτές του SVM. Μια ακόμα χρήση του είναι στην επιλογή προτεινόμενων που υπάρχουν σε πολλές μηχανές

αναζήτησης. Μπορεί να βρει τις προτάσεις που θα ταίριαζαν στον κάθε χρήστη ανάλογα με τα ¨κλικ¨ που κάνει βρίσκοντας έτσι τα ενδιαφέροντα του.

2.4 Decision Trees

Τα δέντρα αποφάσεων είναι μη παραμετρικοί αλγόριθμοι που χρησιμοποιούνται για ταξινόμησή και παλινδρόμηση. Η αναπαράσταση τους γίνεται με τη μορφή δέντρου με κόμβους που χωρίζονται σε κόμβους αποφάσεων και στους τελικούς κόμβους.

Κάθε δέντρο ξεκινάει από έναν κόμβο απόφασης με δύο ή περισσότερους δρόμους που οδηγούν σε άλλους. Ο κάθε κόμβος απόφασης παρουσιάζει μια συνθήκη για τα δεδομένα και ο δρόμος που θα διαλέξουμε εξαρτάτε από το αν η συνθήκη είναι αληθείς ή όχι. Αν συνεχίσουμε αυτή τη διαδικασία κάποια στιγμή θα καταλήξουμε σε έναν τελικό κόμβο ο οποίος θα μας δίνει σαν απάντηση μία από τις κλάσεις που έχουμε ορίσει για το σύνολο δεδομένων.



Σχήμα 10: Παράδειγμα δέντρου απόφασης

Για παράδειγμα στο σχήμα 10 θέλουμε ο υπολογιστής να πάρει την απόφαση για το αν ο άνθρωπος πρέπει να περάσει τον δρόμο ή όχι. Για να το κάνει αυτό περνάει από πολλά στάδια αποφάσεων πριν καταλήξει στην τελική του απόφαση.

Στην πραγματικότητα όμως τα δεδομένα δε θα είναι τόσο απλά. Τα δεδομένα μπορεί να μη διαχωρίζονται με τις συνθήκες που έχουμε επιλέξει για τους κόμβους απόφασης. Αυτό όμως είναι ένα πρόβλημα γιατί ένας τελικός κόμβος θα

πρέπει να μας υποδεικνύει μόνο μία κλάση που σημαίνει ότι το σύνολο των συνθηκών που περάσαμε για να φτάσουμε εκεί θα πρέπει να ισχύει μόνο για εκείνη την κλάση.

Για να μπορέσει ο υπολογιστής να βρει τις βέλτιστες συνθήκες για κάθε κόμβο πρέπει να χρησιμοποιήσουμε τη θεωρεία της πληροφορίας. Θα κάνουμε ένα παράδειγμα για καλύτερη κατανόηση του αλγορίθμου.

Έστω ότι έχουμε ένα πρόβλημα δυαδικής ταξινόμησης και έναν κόμβο απόφασης που στον οποίο έχουμε 20 στοιχεία και τα σημεία είναι διαχωρισμένα στις 2 κλάσεις στη μέση ($50\% - 50\%$). Τώρα θέλουμε να επιλέξουμε ανάμενα σε 2 συνθήκες:

- Συνθήκη Α: Χωρίζει τα σημεία σε 2 κόμβους όπου:
  - κόμβος 1: Έχει 14 σημεία με διαχωρισμό $42.8\% - 57.2\%$
  - κόμβος 2: Έχει 6 σημεία με διαχωρισμό $33.3\% - 66.7\%$
- Συνθήκη Β: Χωρίζει τα σημεία σε 2 κόμβους όπου:
  - κόμβος 1: Έχει 4 σημεία με διαχωρισμό $0\% - 100\%$
  - κόμβος 2: Έχει 16 σημεία με διαχωρισμό $37.5\% - 62.5\%$

Μπορούμε να υπολογίσουμε την εντροπία από τον τύπο:

$$E = \sum_{i=1}^{n} -p_i \log_2 p_i$$

όπου p_i η πιθανότητα της κλάσης i. Άρα έχουμε:

$$E_{parent} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

$$E_{A1} = -0.428 \log_2 0.428 - 0.572 \log_2 0.572 = 0.99$$

$$E_{A2} = -0.333 \log_2 0.333 - 0.667 \log_2 0.667 = 0.91$$

$$E_{B1} = -0 \log_2 0 - 1 \log_2 1 = 0$$

$$E_{B2} = -0.375 \log_2 0.375 - 0.625 \log_2 0.625 = 0.95$$

Παρατηρούμε ότι όσο καλύτερος είναι ο διαχωρισμός των στοιχείων, τόσο μικρότερη είναι η εντροπία. Το κέρδος πληροφορίας για τις 2 συνθήκες είναι:

$$I_A = E_{parent} - \frac{14}{20} E_{A1} - \frac{6}{20} E_{A2} = 0.034$$

$$I_B = E_{parent} - \frac{4}{20} E_{B1} - \frac{16}{20} E_{B2} = 0.24$$

Ισχύει ότι $I_B > I_A$ και άρα θα προτιμήσουμε τη συνθήκη Β. Αυτός ο έλεγχος θα πρέπει να γίνει για όλες τις πιθανές συνθήκες και για όλους τους κόμβους ξεκινώντας από τον αρχικό κόμβο. Είναι σημαντικό να αναφέρουμε ότι αυτός είναι ένας άπληστος αλγόριθμος διότι επιλέγει την καλύτερη συνθήκη για έναν κόμβο αλλά μπορεί μια χειρότερη επιλογή να οδηγούσε σε καλύτερο αποτέλεσμα αν βλέπαμε τη συνολική εικόνα.

΄Ενα πρόβλημα με τα δέντρα είναι ότι έχουν μεγάλο variance και γι΄ αυτό τον λόγο πολλές φορές δεν μπορούν να γενικευτούν για καινούρια δεδομένα. ΄Ετσι σαν μια εξέλιξη του αλγορίθμου δημιουργήθηκε ο αλγόριθμος του τυχαίου δάσους (Random Forest). Ο αλγόριθμος αυτός είναι συνδυασμός μεθόδων και όπως φαίνεται από το όνομα του αποτελείται από πολλά decision trees. Μετά για την τελική απόφαση αν έχουμε πρόβλημα ταξινόμησης θα επιλέξουμε την κλάση που δείχνει η πλειοψηφία, ενώ αν έχουμε πρόβλημα παλινδρόμησης θα πάρουμε τον μέση τιμή των απαντήσεων[12].

Το παραπάνω εξηγεί τη λέξη δάσος όμως δεν εξηγεί τη λέξη τυχαίο. Ο λόγος που είναι τυχαίο είναι επειδή το κάθε δέντρο δε συμπεριλαμβάνει όλα τα δείγματα αλλά ένα τυχαίο τμήμα αυτών και επιπλέον για αυτά τα δείγματα δε θα έχουν όλα τα χαρακτηριστικά τους αλλά θα επιλεχθούν κάποια τυχαία. Παρακάτω βλέπουμε ένα παράδειγμα για το πώς θα χωρίζαμε τα δεδομένα μας για ένα τυχαίο δάσος που θα αποτελείται από τέσσερα δέντρα.

id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

Πίνακας 2: Σύνολο δεδομένων για τον αλγόριθμο random forest

id
2
0
2
4
5
5

id
2
1
3
1
4
4

id
4
1
3
0
0
2

id
3
3
2
5
1
2

Πίνακας 3: Σύνολο δεδομένων για κάθε δέντρο

Επίσης, όπως είπαμε το κάθε υποσύνολο δεδομένων δε θα περιέχει όλα τα στοιχεία. Για παράδειγμα, το πρώτο θα μπορούσε να έχει μόνο τα στοιχεία x_1, x_2, το δεύτερο τα x_1, x_4 κ.ο.κ.

Τα πλεονεκτήματα των δέντρων είναι ότι δε χρειάζεται προ επεξεργασία των δεδομένων. Στα δεδομένα πολλές φορές λείπουν στοιχεία που πρέπει ή μπορεί να χρειάζεται κάποια κανονικοποίηση για να εφαρμόσουμε κάποιους αλγόριθμους. Τα δέντρα όμως δεν είναι ένας από αυτούς και αυτό τα κάνει πολύ καλή επιλογή σε τέτοιες περιπτώσεις. Επιπλέον, όπως είπαμε είναι μη παραμετρικοί και επίσης δεν είναι γραμμικοί που είναι κάτι που θέλουμε πολλές φορές για καλύτερο διαχωρισμό των κλάσεων[13]. Το βασικό πρόβλημα του αλγορίθμου είναι το overfitting το οποίο λύνει ο αλγόριθμος random forest.

3 Παλινδρόμηση

3.1 Γραμμική Παλινδρόμηση

Η ποιο απλή και ευρέως χρησιμοποιούμενη τεχνική παλινδρόμησης είναι η γραμμική. Όπως υποδεικνύει το όνομα της η γραμμική παλινδρόμηση συσχετίζει μία ανεξάρτητη με μία εξαρτημένη μεταβλητή με μία γραμμική σχέση προσαρμόζοντας μία ευθεία γραμμή παλινδρόμησης πάνω στα δεδομένα. Η ευθεία δίνεται από την σχέση:

$$y = ax + b + c$$

όπου:

y **:** εξαρτημένη μεταβλητή (dependent variablε)

x **:** ανεξάρτητη μεταβλητή (Independent / explanatory variable or regressor)

a **:** κλίση της ευθείας (slope)

b **:** σημείο τομής με τον άξονα y (y – intercept)

c **:** σφάλμα παλινδρόμησης (regression residual / error)

Η ευθεία αυτή προσαρμόζεται στα δεδομένα με την χρήση της τεχνικής του μικρότερου αθροίσματος των τετραγώνων (των residuals). Τα residuals(σφάλματα) είναι η απόκλιση, ως προς το y, των δειγμάτων από την γραμμή παλινδρόμησης και βρίσκεται από τον τύπο:

$$c = y_i - \widehat{y}_i$$

Παίρνοντας τώρα το άθροισμα των τετραγώνων των σφαλμάτων (Sum of Squared Residuals):

$$\text{SSR} = \sum_{i=1}^{n} e_i^2$$

Ψάχνουμε μία ευθεία γραμμή (παλινδρόμησης) η οποία θα ελαχιστοποιεί το άθροισμα. Στην περίπτωση της απλής γραμμικής παλινδρόμησης το άθροισμα ελαχιστοποιείται για:

$$a = \frac{\sum(x_i - \widetilde{x})(y_i - \widetilde{y})}{\sum(x_i - \widetilde{x})^2}$$
$$b = \widetilde{y} - a\widetilde{x}$$

Το τετράγωνο των σφαλμάτων επιλέχτηκε έτσι ώστε να μην αλληλοαναιρούνται τα θετικά με τα αρνητικά σφάλματα. Το πλεονέκτημα έναντι της απόλυτης τιμής είναι ότι το τετράγωνο είναι μία συνεχής και παραγωγίσιμη συνάρτηση. Για ποιο

περίπλοκες μορφές γραμμικής παλινδρόμησης χρησιμοποιούνται και επαναληπτικές μέθοδοι προσέγγισης της καλύτερης ευθείας (line of best fit), εξετάζοντας επαναληπτικά τιμές για την κλίση και την μετατόπιση μέχρι να βρούμε το ζευγάρι που ελαχιστοποιεί το άθροισμα των τετραγώνων.

Μια μετρική της απλής γραμμικής παλινδρόμησης είναι το μέσο τετραγωνικό σφάλμα (Mean Square Error), όπου για μία σταθερή διακύμανση του σφάλματος το MSE υπολογίζεται ως:

$$\sigma_\epsilon^2 = \frac{\text{SSR}}{n - p - 1}$$

όπου:

n **:** πλήθος δειγμάτων

p **:** πλήθος ανεξάρτητων μεταβλητών (regressors)

Για την περίπτωση της απλής γραμμικής παλινδρόμησης που εξετάζουμε τώρα το $p = 1$

Μια άλλη μετρική που χρησιμοποιείται κατά κόρον είναι το R^2 και στη συνέχεια θα δούμε πως υπολογίζεται. Αρχικά υπολογίζουμε τον μέσο όρο των δειγμάτων (ως προς τον άξονα της εξαρτημένης μεταβλητής). Η τιμή αυτή θα είναι το σημείο τομής του άξονα της εξαρτημένης μεταβλητής και μίας ευθείας παράλληλης ως προς τον άξονα της ανεξάρτητης μεταβλητής. Αυτή η ευθεία είναι μία ευθεία παλινδρόμησης (mean) όταν δεν παίρνουμε υπόψιν μας την εξάρτηση από την ανεξάρτητη μεταβλητή. Υπολογίζουμε το SSR και την διακύμανση γύρο από αυτή την γραμμή παλινδρόμησης. Έπειτα υπολογίζουμε την διακύμανση και γύρο από την γραμμή παλινδρόμησης που εξετάζουμε (fit). Το R^2 υπολογίζεται από τον τύπο:

$$R^2 = \frac{\text{var}(\text{mean}) - \text{var}(\text{fit})}{\text{var}(\text{mean})}$$

Η σχέση αυτή μας δείχνει την μείωση της διακύμανσης όταν παίρνουμε υπόψιν μας την ανεξάρτητη μεταβλητή ή αλλιώς το ποσοστό της μεταβολής της εξαρτημένης μεταβλητής που εξηγεί η μεταβολή της μεταβλητής που εξετάζουμε ή θα μπορούσαμε να πούμε ότι προσδιορίζει το **ποσοστό της βεβαιότητας** που έχουμε όταν κάνουμε μια πρόβλεψη της τιμής της εξαρτημένης για μία συγκεκριμένη τιμή της ανεξάρτητης. Το ίδιο αποτέλεσμα θα είχαμε και εάν υπολογίζαμε το R^2 με τα SSR αντί της διακύμανσης λόγω του ότι απαλείφονται οι παρονομαστές.

Το R^2 είναι προφανές ότι παίρνει τιμές από 0 ως 1 και όσο το μεγαλύτερο τόσο το καλύτερο. Είναι μια μετρική μεγάλης σημασίας για τον προσδιορισμό της ακαταλληλότητας μιας ανεξάρτητης μεταβλητής, καθώς για μεγάλα δείγματα ο προσδιορισμός αυτός και η απόρριψη μεταβλητών μικρότερης σημασίας οδηγεί σε μεγάλη μείωση του χρόνου υπολογισμού και πόρων. Ωστόσο χρειαζόμαστε

έναν τρόπο για να προσδιορίζουμε το πόσο σημαντικό είναι στατιστικά (πόσο ακριβές είναι), καθώς όπως φαίνεται και παρακάτω ένα με την προσθήκη έξτρα δεδομένων το R^2 αυξάνεται χωρίς να έχει βελτιωθεί απαραίτητα το μοντέλο μας.

Το **p-value** είναι μια μετρική που κάνει ακριβός αυτή την δουλειά. Το p-value για το R^2 υπολογίζεται από έναν άλλον όρο το F. Το F μοιάζει πολύ με το R^2, η μόνη διαφορά τους βρίσκεται στον παρονομαστή με το R^2 να είναι:

$$R^2 = \frac{\text{διακύμασνη του } y \text{ που εξηγείται από το } x}{\text{διακύμασνη του } y \text{ όταν δεν παίρνουμε υπόψιν το } x}$$

Ενώ το F:

$$F = \frac{\text{διακύμασνη του } y \text{ που εξηγείται από το } x}{\text{διακύμασνη του } y \text{ που δεν εξηγείται από το } x}$$

Ή αλλιώς:

$$F = \frac{\text{SS(mean)} - \frac{\text{SS(fit)}}{p_{\text{fit}}} - p_{\text{mean}}}{\frac{\text{SS(fit)}}{n} - p_{\text{fit}}}$$

Όπου:

p_{fit} **:** παράμετροι (βαθμοί ελευθερίας) για την ευθεία που εξετάζουμε (2 για την απλή γραμμική παλινδρόμηση)

p_{mean} **:** παράμετροι (βαθμοί ελευθερίας) για την ευθεία αναφοράς = 1 (αφού η μόνη παράμετρος που εξετάζουμε είναι η μετατόπιση b)

n **:** αριθμός των δεδομένων

Ο υπολογισμός τώρα του p-value από το F γίνεται μέσω της παραγωγής τυχαίων δεδομένων και υπολογίζοντας το F για αυτό το σετ. Αυτό γίνεται για αρκετές επαναλήψεις (όσο περισσότερες τόσο το καλύτερο) και τα αποτελέσματα (F) που παίρνουμε τα τοποθετούμε σε ένα ιστόγραμμα. Έπειτα βρίσκουμε το F για το σετ δεδομένων που εξετάζουμε και το τοποθετούμε και αυτό στο ιστόγραμμα.

Το π-αλυε υπολογίζεται από τον αριθμό των τιμών που είναι μεγαλύτερες από το F του σετ που εξετάζουμε διαιρεμένο με τον αριθμό όλων των τιμών. Στην πράξη το γράφημα του ιστογράμματος προσεγγίζεται από μία γραμμή F κατανομής, έτσι ώστε να μην χρειαστεί να παράξουμε τόσο μεγάλο όγκο τυχαίων δεδομένων. Το σχήμα της γραμμής εξαρτάται από τον αριθμό των δεδομένων και τους βαθμούς ελευθερίας. Το p-value παίρνει μικρότερες τιμές όσο περισσότερα δείγματα έχουμε ανά παράμετρο. Όσο μικρότερο το p-value τόσο καλύτερο το R^2 και κατά συνέπεια η γραμμή παλινδρόμησης

3.2 Πολλαπλή Παλινδρόμηση

Η πολλαπλή παλινδρόμηση είναι παρόμοια με την απλή γραμμική μόνο που αντί να εξετάζουμε την σχέση μεταξύ μίας ανεξάρτητης και μιας εξαρτημένης μεταβλητής, εξετάζουμε την σχέση αυτή παίρνοντας υπόψιν πάνω από μία ανεξάρτητη (π.χ. για την πρόβλεψη εκδήλωσης διαβήτη σε έναν ασθενή, λαμβάνουμε υπόψιν εκτός από το επίπεδο ινσουλίνης στο αίμα του, το βάρος, την πίεση και άλλες βιολογικές μετρικές για τις οποίες μπορεί να εξεταστεί). Ένα από τα μεγαλύτερα πλεονεκτήματα της πολλαπλής παλινδρόμησης είναι ότι η μέθοδος ελαχίστων τετραγώνων που χρησιμοποιεί, από την φύση του απορρίπτει ή μειώνει την επιρροή των μεταβλητών που δεν σχετίζονται ή δεν είναι τόσο καλές για το μοντέλο.

Με αυτή την λογική δεν υπάρχει περίπτωση να χειροτερέψει ένα μοντέλο από την πρόσθεση οποιασδήποτε μεταβλητής. Ωστόσο υπάρχουν δύο προβλήματα με αυτό. Το πρώτο σχετίζεται με το κόστος σε χρόνο και πόρους που θα χρησιμοποιηθούν σε μία μεταβλητή η οποία δεν θα βελτιώσει πολύ ή ενδεχόμενος και καθόλου το μοντέλο. Και το δεύτερο είναι μια μεταβλητή που δεν σχετίζεται με το μοντέλο λόγω τυχαιότητας μπορεί να αποδώσει λανθασμένα σε καλύτερη προσαρμογή της παλινδρόμησης.

Ο τρόπος υπολογισμού της δεν διαφέρει από αυτόν της απλής γραμμικής, το μόνο που προσθέτουμε είναι επιπλέον άξονες (διαστάσεις) στο μοντέλο. Αυτό προφανώς έχει αντίκτυπο στην μορφή που θα έχει η παλινδρόμηση, όπου για δυο ανεξάρτητες μεταβλητές θα έχουμε ένα επίπεδο αντί για ευθεία (2 διαστάσεις) και όσο περισσότερες μεταβλητές (διαστάσεις) προσθέτουμε θα προσαρμόζεται ανάλογα και η παλινδρόμηση, χωρίς να χάνεται η γραμμικότητα της.

Μία άλλη διαφορά είναι ότι σε αυτή την περίπτωση είναι αναγκαίο να χρησιμοποιήσουμε το προσαρμοσμένο R^2 αντί για το R^2 με το πρώτο να είναι απλά μία παραλλαγή του δεύτερου λαμβάνοντας υπόψιν και τις επιπλέον διαστάσεις στον υπολογισμό του. Το προσαρμοσμένο R^2 δίνεται από την σχέση:

$$R_{\text{adj}} = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$

όπου:

n **:** το πλήθος των δειγμάτων

p **:** το πλήθος των ανεξάρτητων μεταβλητών (regressors)

Το πλεονέκτημα με το προσαρμοσμένο R^2 είναι ότι μειώνει την αδυναμία του R^2 να αυξάνεται με την πρόσθεση επιπλέον μεταβλητών που δεν προσαρμόζονται καλά στα δεδομένα. Το προσαρμοσμένο R^2 λύνει αυτό το πρόβλημα ¨τιμωρώντας¨ το αποτέλεσμα όταν προσθέτονται τέτοιες μεταβλητές που δεν είναι αρκετά καλές. Έτσι τι προσαρμοσμένο R^2 θα είναι πάντα μικρότερο ή ίσο του R^2.

Χρησιμοποιώντας το νέο προσαρμοσμένο R^2 στην θέση του R^2 υπολογίζουμε το p-value.

Με τα παραπάνω συγκρίνουμε την παλινδρόμηση στην μέση τιμή. Το δυνατό σημείο της πολλαπλής παλινδρόμησης ωστόσο είναι στην σύγκριση της απλής γραμμικής με μία πολλαπλή ή μιας πολλαπλής με μία πολλαπλή με παραπάνω μεταβλητές. Η σύγκριση αυτή είναι πολύ σημαντική, καθώς μας δείχνει το πόσο βελτιώνεται η πρόβλεψη μας εάν προσθέσουμε την μεταβλητή (ή μεταβλητές) αυτή και το εάν αξίζει να αφιερώσουμε πολύτιμο χρόνο και πόρους για τον υπολογισμό της.

Η σύγκριση αυτή μπορεί να γίνει είτε συγκρίνοντας τα 2 προσαρμοσμένο R^2 και τα p-values τους ή (αυτό που προτείνεται) να υπολογίσουμε εκ νέου τις μετρικές αυτές αλλά αντί για mean να βάλουμε τις τιμές του simple (απλούστερου) μοντέλου και αντί για το fit να βάλουμε το ποιο πολύπλοκο (πολλαπλό) , έτσι θα έχουμε:

$$R^2 = \frac{\text{Var(simple)} - \text{Var(complex)}}{\text{Var(simple)}}$$

$$F = \frac{\text{SS(simple)} - \frac{\text{SS(complex)}}{p_{\text{complex}}} - p_{\text{simple}}}{\frac{\text{SS(complex)}}{n} - p_{\text{complex}}}$$

Εάν το προσαρμοσμένο R^2 είναι μεγάλο και το p-value μικρό τότε αξίζει να υπολογίσουμε την (τις) επιπλέον μεταβλητή.

3.3 Πολυωνυμική Παλινδρόμηση

Η πολυωνυμική παλινδρόμηση είναι η γενικότερη κατηγορία παλινδρόμησης στην οποία ανήκουν τα διάφορα είδη της γραμμικής παλινδρόμησης που είδαμε προηγουμένως και κατά ακολουθία ανήκει στα τυπικά γραμμικά μοντέλα (Standard Linear Models).

Η ειδοποιός διαφορά που μας κάνει να τα ξεχωρίζουμε σαν δύο διαφορετικές κατηγορίες είναι το γεγονός ότι αντί η γραμμή παλινδρόμησης να χαρακτηρίζεται από μία ευθεία γραμμή μπορεί να έχει και μία κυρτότητα ώστε να ¨ακολουθάει¨ καλύτερα τα δεδομένα, όπου με την σειρά τους δεν έχουν γραμμική εξάρτηση από την ή τις ανεξάρτητες μεταβλητές. Η γραμμή πολυωνυμική παλινδρόμησης δίνεται από τον τύπο:

$$y_i = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n + \epsilon$$

Πολύ εύκολα μπορούμε να δούμε ότι η απλή γραμμική παλινδρόμηση είναι μία πολυωνυμική $1^{ου}$ βαθμού. Ο βαθμός (τάξη) του πολυωνύμου όπως ξέρουμε είναι η μεγαλύτερη δύναμη στην οποία υψώνεται η (ανεξάρτητη) μεταβλητή.

Θα ρωτούσε κανείς πώς γίνεται να είναι η γενικότερη κατηγορία όλων των γραμμικών παλινδρομήσεων όταν υπάρχουν όροι υψωμένοι σε δύναμη μεγαλύτερη του 1. Η απάντηση σε αυτό το ερώτημα θα ήταν ότι παρόλο που οι ανεξάρτητες μεταβλητές είναι μη γραμμικές, οι συντελεστές είναι και αποκλειστικά τα δεδομένα είναι αυτά που υψώνονται σε μεγαλύτερες δυνάμεις.

Ο τρόπος υπολογισμού των συντελεστών γίνεται με χρήση πινάκων, για αυτό και θα πρέπει να αναπαραστήσουμε την πολυωνυμική παλινδρόμηση σε μορφή πινάκων ως εξής:

$$A_k = f_A\left(\begin{bmatrix} W_{11} & \dots & W_{1n^{[k-1]}} \\ \vdots & \ddots & \vdots \\ W_{n^{[k]}1} & \dots & W_{n^{[k]}n^{[k-1]}} \end{bmatrix}^{[k]} \times \begin{bmatrix} a_1 \\ \vdots \\ a_{n^{[k-1]}} \end{bmatrix}^{[k]} + b_1\right) = \begin{bmatrix} a_1 \\ \vdots \\ a_{n^{[k]}} \end{bmatrix}^{[k]}$$

Όπου απλοποιείται σε:

$$\overrightarrow{y} = X\overrightarrow{a} + \overrightarrow{\epsilon}$$

Επομένως με την χρήση της μεθόδου ελαχίστων τετραγώνων έχουμε:

$$\widehat{\overrightarrow{b}} = (X^T X)^{-1} X^T \overrightarrow{y}$$

Ή ένας εναλλακτικός τρόπος είναι να λύσουμε το παρακάτω σύστημα:

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^k y_i \end{bmatrix}$$

Έπειτα χρησιμοποιώντας την μέθοδο του Κραμερ βρίσκουμε τους συντελεστές ως εξής:

$$a_k = \frac{\det(M_i)}{\det(M)}$$

$$M_0 = \begin{bmatrix} \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i & \dots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i y_i & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{N} x_i^k y_i & \sum_{i=1}^{N} x_i^{k+1} & \dots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix}$$

Ενώ έχοντας πλέον την πολυωνυμική παλινδρόμηση με τους συντελεστές της ο τρόπος υπολογισμού του SSR παραμένει ίδιος. Εφόσον έχουμε το SSR μπορούμε να εφαρμόσουμε όλα όσα αναφέραμε και στην απλή γραμμική παλινδρόμηση.

Επίσης όσο μεγαλύτερη η τάξη του πολυωνύμου, τόσο πιο ¨στενά¨ θα προσαρμόζεται η παλινδρόμηση στα δεδομένα. Αναφερόμενοι στις δυνάμεις, ο μόνος περιορισμός που υπάρχει για το μέγιστο βαθμό που πολυωνύμου είναι ότι δεν πρέπει να ξεπερνάει τον αριθμό των δεδομένων, γεγονός που δεν θα μας επιτρέπει να προσαρμόσουμε την ευθεία στα δεδομένα. Κάτι τέτοιο προφανώς (η τάξη να πλησιάσει τον αριθμό των δεδομένων) σπανίως είναι επιθυμητό, εάν όχι καθόλου, καθώς είναι σχεδόν σίγουρο ότι θα οδηγήσει σε over-fitting.

Πως υπολογίζουμε την κατάλληλη τάξη του πολυωνύμου· Το πρόβλημα της εύρεσης κατάλληλου βαθμού για το πολυώνυμο το λύνει το Bayes Information Criteria (BIC):

$$\text{BIK}_k = n \log(\text{SSR}) + K \log(n)$$

Όπου:

k **:** βαθμός πολυωνύμου

n **:** βαθμός αριθμός δεδομένων

Η ποσότητα αυτή υπολογίζεται για κάθε βαθμό του πολυωνύμου και επιλέγεται η μικρότερη ως η καλύτερη για το εκάστοτε μοντέλο. Αν αναπαραστήσουμε τις ποσότητες που βρήκαμε, για ένα μοντέλο όπου τα δεδομένα ακολουθούν μία συνάρτηση $3^{ου}$ βαθμού παρατηρούμε ότι ο υπολογισμός της βέλτιστης τάξης είναι σωστός.

3.4 T-test

Το τ-τεστ ένα στατιστικό τεστ και χρησιμοποιείται κυρίως για την σύγκριση 2 ομάδων, στο πεδίο των δοκιμών υποθέσεων (hypothesis testing), για να βρούμε τι επιρροή θα έχει η αλλαγή ενός χαρακτηριστικού της ομάδας.

Για παράδειγμα ένα τέτοιο τεστ στον τομέα της βιολογίας μπορεί να αφορούσε την αλλαγή ενός γονιδίου ή την χορήγηση ενός φαρμάκου έχοντας ένα control

group (χωρίς να παίρνει το φάρμακο ή placebo) και ένα που να χορηγείται. Λόγο της ομαδοποίησης τους με τέτοιο τρόπο τα δεδομένα έχουν διακριτές τιμές στον άξονα της ανεξάρτητης μεταβλητής (με ή χωρίς / ομάδα 1 ή ομάδα 2). Φυσικά στις δύο ομάδες που ελέγχουμε, μπορούμε να χρησιμοποιήσουμε τεχνικές παλινδρόμησης για να αναλύσουμε τις ομοιότητες και τις διαφορές των ομάδων και να χαρακτηρίσουμε την επίδραση της αλλαγής. Για δείξουμε τον τρόπο εφαρμογής των τεχνικών θα αναλύσουμε ένα παράδειγμα.

΄Εστω ότι έχουμε 2 ομάδες μία με το σύνηθες γονίδιο A και μία με το ποιο σπάνιο μεταλλαγμένο γονίδιο B. Το γονίδιο αυτό (μεταλλαγμένο και μη) επηρεάζει την εμφάνιση της στεφανιαίας νόσου που είναι μια πάθηση καρδιάς (heart disease) και θέλουμε να αναλύσουμε την επίδραση μου έχει το καθένα στην εκδήλωση της ασθενείας. Αρχικά πρέπει να βρούμε ένα μέσο(mean) για όλα τα δείγματα (A και B) και να βρούμε το SSR(mean). Αυτό το κάνουμε ακριβώς όπως στην απλή γραμμική.

Τώρα θα πρέπει να προσαρμόσουμε 2 γραμμές μία σε κάθε ομάδα. Μπορεί να αποδειχθεί ότι την καλύτερη προσαρμογή (best fit) με την μέθοδο των μικρότερου αθροίσματος τετραγώνων (least squares fit) θα έχει ο μέσος όρος τους. Οι γραμμές που βρήκαμε θα είναι γραμμές παράλληλες στον άξονα x και τα τέμνουν το y στην μέση τιμή των δεδομένων του κάθε γκρουπ. Τώρα πρέπει να βρούμε έναν τρόπο να ενώσουμε τις δύο γραμμές σε μία συνάρτηση ώστε να μπορούμε να το περάσουμε σε ένα υπολογιστικό σύστημα για να κάνει όλους τους δύσκολους υπολογισμούς (πχ μετρικών R^2, p-value).

Για χάρη απλότητας θα έχουμε ένα πολύ μικρό δείγμα για να εξηγήσουμε την τεχνική. ΄Εστω λοιπών ότι έχουμε 3 δείγματα από κάθε ομάδα (σύνολο 6) τότε η σχέση που υπολογίζει και τις δύο γραμμές είναι η εξής:

$$
\begin{aligned}
y &= 1 \times (\textbf{meanA}) + 0 \times (\text{meanB}) + (\text{residual\_1}) \\
y &= 1 \times (\textbf{meanA}) + 0 \times (\text{meanB}) + (\text{residual\_2}) \\
y &= 1 \times (\textbf{meanA}) + 0 \times (\text{meanB}) + (\text{residual\_3})
\end{aligned}
$$

$$
\begin{aligned}
y &= 0 \times (\text{meanA}) + 1 \times (\textbf{meanB}) + (\text{residual\_4}) \\
y &= 0 \times (\text{meanA}) + 1 \times (\textbf{meanB}) + (\text{residual\_5}) \\
y &= 0 \times (\text{meanA}) + 1 \times (\textbf{meanB}) + (\text{residual\_6})
\end{aligned}
$$

Παρατηρούμε πως η σχέση είναι ίδια για όλα τα δείγματα, το μόνο που αλλάζει είναι οι συντελεστές μπροστά από τα 2 mean οι οποίοι μάλιστα παίρνουν τιμές 1 και 0. Οι συντελεστές που μόλις αναφέραμε λειτουργούν σαν διακόπτες για το ποια ομάδα διαλέγουμε. Τους συντελεστές μπορούμε να τους έχουμε σε έναν πίνακα που ονομάζεται design matrix. Με την παραλλαγή αυτή η σχέση

μετατρέπεται σε:

$$y = \text{column1} \times \text{meanA} + \text{column2} \times \text{meanB}$$

Όπου:

column1	column2
1	0
1	0
1	0
0	1
0	1
0	1

Αφού έχουμε τον τύπο και για τις 2 γραμμές βρίσκουμε το SSR(fit) για κάθε ομάδα (απόσταση δείγματος και γραμμής ομάδας). Έχοντας και το SSR(fit) μπορούμε να υπολογίσουμε το R^2 και το p-value για τις 2 γραμμές παλινδρόμησης. Σε αυτό το σημείο θα αλλάξουμε λίγο την σχέση σε μία που χρησιμοποιείται πιο συχνά, ωστόσο και οι δύο βγάζουν το ίδιο σωστό αποτέλεσμα. Η νέα σχέση θα είναι:

$$y = \text{column1} \times \text{meanA} + \text{column2} \times \text{difference}(\text{meanB} - \text{meanA})$$

Και το design matrix θα γίνει:

column1	column2
1	0
1	0
1	0
1	1
1	1
1	1

Με λίγα λόγια η νέα σχέση υπολογίζει και τις δύο γραμμές, αλλά τώρα η δεύτερη είναι σε μορφή offset από την πρώτη. Τέτοιου είδους design matrix μπορεί να χρησιμοποιηθεί και για απλή γραμμική παλινδρόμηση με την μόνη διαφορά ότι αντί για διακριτές τιμές μπορούμε να έχουμε οποιαδήποτε τιμή θέλουμε κρατώντας την πρώτη στήλη στο 1 για το intercept.

Τώρα που ξέρουμε τι είναι και από τι αποτελείται ένα t-test θα τα αναμείξουμε με τις τεχνικές της παλινδρόμησης που αναφέραμε πιο πάνω για αναλύσουμε το t-test και για να βγάλουμε κάποια χρήσιμα συμπεράσματα. Για να δείξουμε πως

ενώνουμε τα t-test με την παλινδρόμηση θα χρησιμοποιήσουμε το παράδειγμα που είχαμε παραπάνω με τα δύο γονίδια A(κόκκινο) και B(πράσινο) με τον άξονα y να είναι η πιθανότητα να εμφανιστεί η νόσος και το x να είναι η ηλικία.

Βλέπουμε πως τα άτομα με το B γονίδιο έχουν μεγαλύτερη πιθανότητα να εμφανίσουν την νόσο από τα άτομα με το A γονίδιο και βλέπουμε ότι οι δύο ομάδες ακολουθούν δύο διαφορετικές παλινδρομήσεις.

Εάν κάναμε μία παλινδρόμηση στο σύνολο των δειγμάτων θα βγάζαμε μία γραμμή που θα περνούσε ανάμεσα από τις 2 γραμμές που βλέπουμε και θα είχε πολύ μεγάλη διακύμανση κάτι που είναι εντελώς λάθος να κάνουμε. Επίσης ένα απλό t-test δεν θα λάμβανε υπόψιν την συσχέτιση της ηλικίας με την πιθανότητα εμφάνισης της νόσου και η παλινδρόμηση όπως είδαμε δεν θα λάμβανε υπόψιν την διαφορά στο γονίδιο.

΄Ετσι για τον συνδυασμό τους θα έχουμε 2 διαφορετικές παλινδρομήσεις μία για κάθε ομάδα και θα τις συγκρίνουμε μεταξύ τους, κάτι που θα έκανε ένα απλό t-test στα 2 mean. Για να κάνουμε απλά τα πράγματα στο συγκεκριμένο παράδειγμα οι δύο παλινδρομήσεις έχουν την ίδια κλίση και η μόνη διαφορά τους είναι το σημείο τομής με τον y άξονα.

Στην πραγματικότητα θα μπορούσαμε να έχουμε και διαφορετικές κλίσεις προσθέτοντας στον τύπο που θα δώσουμε παρακάτω μερικές επιπλέον μεταβλητές και στο Design matrix μερικές στήλες για τις αντίστοιχες μεταβλητές. ΄Ετσι λοιπόν θα έχουμε:

$$y = \text{column1} \times \text{intersectA} + \text{column2} \times \text{intersect\_offset} + \text{column3} \times \text{slope}$$

Και το design matrix θα γίνει:

column1	column2	column3
1	0	$X(1)$
1	0	$X(2)$
1	0	$X(3)$
1	1	$X(4)$
1	1	$X(5)$
1	1	$X(6)$

Υπολογίζοντας τις αποστάσεις από την γραμμή παλινδρόμησης (residual) για την κάθε ομάδα ξεχωριστά (B γονίδιο από την πράσινη γραμμή και A αντίστοιχα από την κόκκινη) μπορούμε να βρούμε το R^2 και το p-value για το μοντέλο χρησιμοποιώντας το απλό μοντέλο του μέσου όρου όλων των τιμών ή να το συγκρίνουμε με οποιαδήποτε άλλο μοντέλο μέσω των παρακάτω τύπων.

Το δεύτερο γίνεται για να δούμε εάν κάποια μεταβλητή ό ομαδοποίηση είναι

χρήσιμη για το μοντέλο και αν αξίζει να την υπολογίσουμε:

$$R^2 = \frac{\text{Var}(\text{simple}) - \text{Var}(\text{complex})}{\text{Var}(\text{simple})}$$

$$F = \frac{\text{SS}(\text{simple}) - \frac{\text{SS}(\text{complex})}{p_{\text{complex}}} - p_{\text{simple}}}{\frac{\text{SS}(\text{complex})}{n} - p_{\text{complex}}}$$

Επίσης υπάρχουν και τα ANOVA τα οποία είναι σαν τα τ-τεστ αλλά με περισσότερες ομάδες, πχ για το προηγούμενο παράδειγμα θα μπορούσαμε να έχουμε το γονίδιο A και B σε κανονική διατροφή και ομάδες των 2 γονιδίων αλλά να είναι μέσα ειδικό πρόγραμμα διατροφής ή να έχουν κάποια φαρμακευτική αγωγή που θέλουμε να ελέγξουμε και να έχουμε 2 control groups(A και B), 2 πλαςεβο και 2 που θα παίρνουν την αγωγή. Ο υπολογισμός τέτοιων μοντέλων είναι ο ίδιος με τα t-test μόνο που για συνολική γραμμή υπολογίζουμε τις παλινδρομήσεις όλων των ομάδων και για p_fit για τον υπολογισμού του F θα βάλουμε τον αντίστοιχο αριθμό των ομάδων.

3.5 Λογαριθμική Παλινδρόμηση

Η λογαριθμική παλινδρόμηση ενώ δουλεύει με συνεχής μεταβλητές όπως το βάρος αλλά και διακριτές όπως το φύλλο. Σαν έξοδο παρέχει μία Boolean τιμή (true, false ή έχει/δεν έχει κάποια ασθένεια) προδίδοντας παράλληλα και μία πιθανότητα (σωστής) πρόβλεψης. Παρατηρώντας κάποιος αυτή την ιδιότητα καταλαβαίνει ότι είναι πολύ χρήσιμη σε εφαρμογές ταξινόμησης όπου και χρησιμοποιείται κατά κύριο λόγο.

Η ικανότητα αυτή για ταξινόμηση την κάνει ευρέως γνωστή τον τομέα της μηχανικής μάθησης. Για παράδειγμα χρησιμοποιώντας αυτού του είδους την παλινδρόμηση θα μπορούσαμε να προβλέψουμε με τα αποτελέσματα κάποιων αιματολογικών εξετάσεων και γνωρίζοντας την ηλικία και το φύλο του εξεταζόμενου εάν πάσχει από κάποια ασθένεια και ποια είναι η πιθανότητα, αυτή η πρόβλεψη να είναι ακριβής.

Μια διαφορά με την γραμμική παλινδρόμηση είναι ο τρόπος που προσαρμόζουμε την παλινδρόμηση στα δεδομένα. Στην λογαριθμική επειδή η παλινδρόμηση έχει μία σιγμοειδή μορφή δεν μπορούμε να χρησιμοποιήσουμε την μέθοδο ελαχίστων τετραγώνων, αλλά χρησιμοποιείται η μέθοδος της μέγιστης πιθανότητας (maximum likelihood).

Κάτι που δεν αναφέρθηκε πριν για την απλή γραμμική παλινδρόμηση είναι ότι δεν έχει φραγμένα όρια για τις μεταβλητές τις κάτι που την κάνει ποιο εύκολη

στον υπολογισμό αλλά και επιρρεπή σε λάθη ως προς την φυσική ορθότητα του μοντέλου. Για παράδειγμα εάν είχαμε ένα μοντέλο όπου θα συσχετίζαμε ορθά το μέγεθος (όγκο) ενός ζώου ή ανθρώπου με το βάρος του, όπου ζώα με μεγαλύτερο βάρος θα είχαν και μεγαλύτερο μέγεθος, μπορούμε να υπολογίσουμε το μέγεθος ζώου με μηδενικό ή και αρνητικό βάρος, κάτι που προφανώς δεν είναι δυνατόν να γίνει στον φυσικό κόσμο. Αντίθετα στην λογαριθμική παλινδρόμηση δεν γίνεται κάτι τέτοιο και ο άξονας της εξαρτημένης μεταβλητής είναι φραγμένος στο πεδίο $[0, 1]$. Για να λύσουμε αυτό το θέμα μετατρέπουμε την πιθανότητα (του άξονα y) σε λογάριθμο της πιθανότητας ώστε ο άξονας να παίρνει τιμές από $-\infty$ ως ∞. Η μετατροπή αυτή γίνεται με την συνάρτηση logit που ορίζεται από τον τύπο:

$$\log(\text{odds}) = \log\left(\frac{p}{1-p}\right)$$

Όπου:

p **:** Η πιθανότητα του παλιού άξονα της εξαρτημένης μεταβλητής

$$\lim_{p\to 0}\left[\log\left(\frac{p}{1-p}\right)\right] = -\infty$$
$$\log\left(\frac{0.5}{1-0.5}\right) = 0$$
$$\lim_{p\to 1}\left[\log\left(\frac{p}{1-p}\right)\right] = +\infty$$

Όλες οι άλλες ενδιάμεσες τιμές υπολογίζονται αντίστοιχα. Το αποτέλεσμα του μετασχηματισμού είναι από την κυρτή (σιγμοειδή) γραμμή που είχαμε πριν, να έχουμε μία ευθεία γραμμή χωρίς φραγμένο πεδίο ορισμού.

Τώρα ενώ τα δεδομένα και η λογαριθμική παλινδρόμηση σχετίζονται και παρουσιάζονται στην προηγούμενη μορφή (γράφημα πιθανοτήτων) όλες οι πράξεις και αναπαραστήσεις των συντελεστών θα γίνεται στο μετασχηματισμένο μοντέλο (λογάριθμοι πιθανοτήτων). Έχοντας τώρα ένα γραμμικό μοντέλο για τους συντελεστές μπορούμε να εφαρμόσουμε όλους τους υπολογισμούς και να υπολογίσουμε τις μετρικές που είχαμε στην απλή γραμμική παλινδρόμηση με κάποιες αλλαγές. Αντί για το R^2 εδώ χρησιμοποιείται το z-value ή Wald's test που δίνεται από τον τύπο:

$$\text{z-value} = \frac{\text{εκτίμιση}}{\text{τυπικό σφάλμα}}$$

και μας δείχνει πόσα τυπικά σφάλματα μακριά από το 0 είναι η εκτίμησή μας.

Όλα αυτά αφορούν τις συνεχές μεταβλητές. Για διακριτές μεταβλητές θα δουλέψουμε περίπου με τον ίδιο τρόπο που δουλέψαμε στα t-test αλλά πρώτα πρέπει

να κάνουμε την μετατροπή του άξονα και μετά προσαρμόζουμε 2 γραμμές. Για παράδειγμα έστω ότι είχαμε να μελετήσουμε την παλινδρόμηση για το πόσο μία μετάλλαξη σε ένα γονίδιο επηρεάζει την πιθανότητα να αποκτήσει κάποιος διαβήτη. Για να βρούμε την γραμμή παλινδρόμησης για τα άτομα που δεν έχουν την μετάλλαξη, θα πάρουμε τα άτομα αυτά και θα υπολογίσουμε την τιμή του λογαρίθμου της πιθανότητας να έχουν διαβήτη (λογάριθμος πιθανότητας κανονικό):

$$\log\left(\frac{\text{άτομα με διαβήτη}}{\text{άτομα χωρίς διαβήτη}}\right)$$

Μετά θα υπολογίσουμε την ίδια τιμή (λογάριθμος πιθανότητας μετάλλαξης) και για τα άτομα με την μετάλλαξη. Αυτές οι δύο τιμές θα είναι οι οριζόντιες γραμμές που είχαμε και στα t-test και ANOVA. Για να δημιουργήσουμε την γραμμή της παλινδρόμησης θα συνδυάσουμε τις δύο τιμές σε μία εξίσωση όπου τα είναι οι συντελεστές της:

$$y = \log(\text{πιθανότητα κανονικό})x_1 + \log\left(\frac{\text{πιθανότητα μετάλλαξης}}{\text{πιθανότητα κανονικό}}\right)x_2$$

Ο πρώτος όρος είναι το σημείο τομής και ο δεύτερος όρος μας λέει σε λογαριθμική κλίμακα πως επηρεάζει το γονίδιο την πιθανότητα να έχεις διαβήτη.

Ο υπολογισμός της ευθείας έχει γίνει ποιο γρήγορος και εύκολος λόγω του μετασχηματισμού, ωστόσο έχει δημιουργηθεί ένα πρόβλημα στον τρόπο προσαρμογής της στα δεδομένα, καθώς τα δεδομένα αυτά έχουν πάρει τιμές $-\infty$ και ∞ για τα αντίστοιχα 0 και 1. Αυτό θέτει αδύνατο τον υπολογισμό των ελάχιστο τετραγώνων αφού για όλα τα δεδομένα η απόσταση από την ευθεία θα είναι ∞. Για αυτό όπως αναφέρθηκε πιο πάνω αντί για την μέθοδο των ελάχιστων τετραγώνων χρησιμοποιείται η μέθοδος της μέγιστης πιθανότητας. Η μέθοδος αυτή λειτουργεί προβάλλοντας τα δεδομένα στην εκάστοτε ευθεία που εξετάζουμε, σύμφωνα με την θέση τους στον άξονα ξ, δίνοντάς τα μία τιμή λογαρίθμου πιθανότητας(άξονας ψ). Έπειτα μετατρέπουμε αυτές τις τιμές λογαρίθμου πιθανότητας σε πιθανότητες μέσω του αντίστροφου μετασχηματισμού που δίνεται από τον τύπο:

$$p = \frac{e^{\log(\text{πιθανότητα})}}{1 + e^{\log(\text{πιθανότητα})}}$$

Τώρα που είμαστε στο αρχικό μοντέλο με την λογαριθμική παλινδρόμηση (φραγμένες τιμές από 0 ως 1) και χρησιμοποιώντας τις πιθανότητες που βρήκαμε και την γνωστή κατάσταση των δειγμάτων (πχ έχει ή δεν έχει διαβήτη) μπορούμε να υπολογίσουμε την μέγιστη πιθανότητα. Αρχίζοντας με τα άτομα πού έχουν κατάσταση 1 (έχει διαβήτη) η πιθανότητα του να έχει διαβήτη (να είναι στην κατάσταση 1) σύμφωνα με την παλινδρόμηση που εξετάζουμε είναι η τιμή του άξονα ψ για το συγκεκριμένο σημείο. Αντίστοιχα μετράμε και την πιθανότητα να μην

έχουν διαβήτη για τα άτομα που δεν έχουν διαβήτη (πιθανότητα τα άτομα που είναι στην κατάσταση 0 να είναι σε αυτή την κατάσταση). Έτσι η πιθανότητα του μοντέλου δίνεται από το γινόμενο των πιθανοτήτων αυτών:

$$\prod_{i=1}^{n} p_i \times \prod_{i=1}^{k} (1 - p_i)$$

Όπου:

n **:** αριθμός ατόμων στην κατάσταση 1

k **:** αριθμός ατόμων στην κατάσταση 0

Πολλοί υπολογίζουν τον λογάριθμο αυτής της ποσότητας όπου απλά αντί να πολλαπλασιάζεις τις πιθανότητες προσθέτεις τους λογαρίθμους των πιθανοτήτων ωστόσο και τα δύο είναι ορθά αφού η ευθεία που μεγιστοποιεί τις πιθανότητες μεγιστοποιεί και τους λογαρίθμους αυτωνών στο αντίστοιχο μετασχηματισμένο μοντέλο.

$$\sum_{i=1}^{n} \log p_i + \sum_{i=1}^{k} \log(1 - p_i)$$

Για να βρεθεί η καλύτερη ευθεία με την βοήθεια ενός υπολογιστικού συστήματος δοκιμάζουμε επαναληπτικά με ¨έξυπνους¨ αλγορίθμους που σε κάθε επανάληψη βελτιώνουν την ευθεία (πχ γενετικοί αλγόριθμοι) πολλές τέτοιες ευθείες και απεικονίζουμε τις πιθανότητες των μοντέλων (ευθειών). Η ευθεία με την μέγιστη πιθανότητα είναι η καλύτερα προσαρμοσμένη.

Αφού βρήκαμε την ευθεία που προσαρμόζεται στα δεδομένα μας, το επόμενο βήμα (όπως κάναμε και στην απλή γραμμική παλινδρόμηση) είναι να ελέγξουμε πόσο καλή είναι η μεταβλητή που χρησιμοποιούμε βρίσκοντας το R^2 και το p-value. Ενώ στην απλή γραμμική ο τρόπος για να βρούμε το R^2 είναι ένας και μοναδικός για την λογαριθμική παλινδρόμηση τα πράγματα δεν είναι τόσο ξεκάθαρα. Υπάρχουν πάνω από 10 μέθοδοι εύρεσης του R^2 για την λογαριθμική παλινδρόμηση. Εδώ θα εξετάσουμε μόνο έναν από αυτούς

Αρχικά στην θέση του SSR θα υπολογίζουμε το LL ($\log($likelihood» λογάριθμος πιθανότητας. Για την τιμή LL της ευθείας που προσαρμόζεται (fit) ακολουθάμε την παραπάνω μεθοδολογία. Για την μέση (mean) βρίσκουμε τον λογάριθμο της ολικής πιθανότητας να είναι σε κατάσταση 1, δηλαδή:

$$\log\left(\frac{n}{k}\right)$$

Αυτό μας δίνει μία τιμή, η οποία θα είναι το σημείο τομής του άξονα της εξαρτημένης μεταβλητής και μίας ευθείας παράλληλη στον άξονα της ανεξάρτητης.

Προβάλλοντας τα δεδομένα σε αυτή την γραμμή και κάνοντας τον αντίστροφο μετασχηματισμό logit. Ένας άλλος τρόπος να φτάσουμε στο ίδιο αποτέλεσμα είναι να βρούμε απευθείας την ολική πιθανότητα να βρίσκεται κάποιος στην κατάσταση 1:

$$\frac{n}{n+k}$$

Αυτό μας δίνει έναν τρόπο να επαληθεύουμε τα αποτελέσματά μας. Για να τελειώσουμε την διαδικασία αυτή πρέπει να βρούμε τον ολικό λογάριθμο πιθανότητας, ακολουθώντας τα βήματα που δείξαμε παραπάνω για να βρούμε το $\mathrm{LL(mean)}$. Για το τελευταίο βήμα της εύρεσης του R^2 κάνουμε μια παραλλαγή στον τύπο της απλής γραμμικής:

$$R^2 = \frac{\mathrm{LL(mean)} - \mathrm{LL(fit)}}{\mathrm{LL(mean)}}$$

Κάνοντας το απλό πείραμα του να βάλουμε τιμές οι οποίες είναι εντελώς τυχαίες και η μεταβλητή που εξετάζουμε δεν έχει καμία εξάρτηση από την ανεξάρτητη και υπολογίσουμε το R^2, θα δούμε ότι παίρνει τιμή 0. Αντίστοιχα για έναν τέλειο προβλεπτή (predictor) θα βρίσκαμε $R^2 = 1$. Άρα το R^2 ορθά είναι φραγμένο ανάμεσα στο 0 και το 1.

Για να βρούμε το p-value έχουμε τον εξής τύπο:

$$\chi^2 = 2(\mathrm{fit}) - \mathrm{LL(mean)}$$

Όπου χ^2 είναι μία τιμή με:

$$\text{Βαθμοί ελευθερίας} = \text{Βαθμοί ελευθερίας}(\mathrm{fit}) - \text{Βαθμοί ελευθερίας}(\mathrm{mean})$$

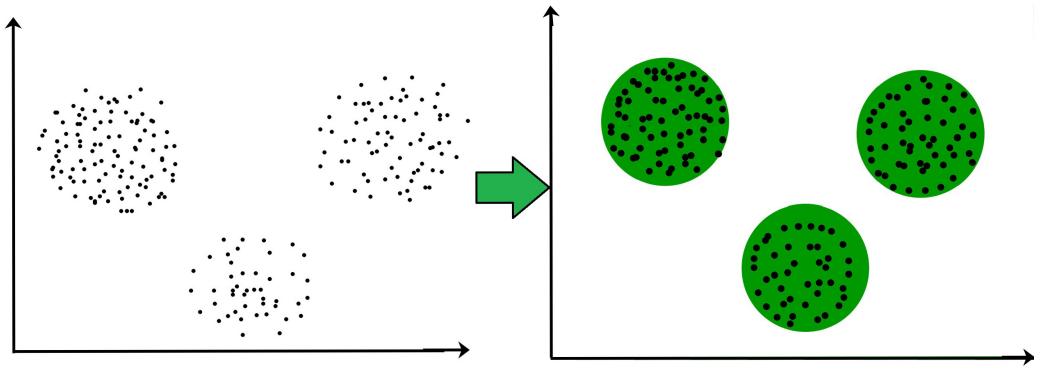
Για μία απλή λογαριθμική παλινδρόμηση ο βαθμός ελευθερίας είναι $2-1=1$ και αυτό βγαίνει από τον αριθμό των συντελεστών για τις παλινδρομήσεις των δύο μοντέλων. Η τιμή χ^2 παρουσιάζει μία κατανομή από το γράφημα της οποίας μπορούμε να βρούμε το p-value. Άμα παρατηρήσουμε το γράφημα και αντικαταστήσουμε στον παραπάνω τύπο για την χειρότερη περίπτωση που η ανεξάρτητη μεταβλητή μας δεν είναι καθόλου καλή στην πρόβλεψη της εξαρτημένης το $\mathrm{LL(mean}$ θα ισούται με το μεαν και έτσι θα έχουμε 0 στο χ^2 άρα p-value $= 1$. Αντίστοιχα για πολύ καλή προσαρμογή το p-value $<< 1$.

Όπως και για την απλή γραμμική μικρό p-value σημαίνει ότι η σχέση μεταξύ εξαρτημένης και ανεξάρτητης μεταβλητής δεν είναι λόγω τύχης και ένα μεγάλο R^2 μας δίνει το πόσο μεγάλη είναι αυτή η συσχέτιση.

4 Ομαδοποίηση

Η ενότητα αυτή αφορά τους αλγορίθμους ομαδοποίησης(clustering). Η δουλειά των αλγορίθμων αυτών είναι να χωρίσουν το σύνολο των δεδομένων σε ομάδες τέτοιες ώστε τα δεδομένα μιας ομάδας να εμφανίζουν περισσότερες ομοιότητες μεταξύ τους από ότι με τα δεδομένα των άλλων ομάδων. Αυτός είναι και ο βασικός στόχος της διερευνητικής και στατιστικής ανάλυσης και χρησιμοποιείται για[14]:

- Αναγνώριση προτύπων
- Ανάλυση εικόνων
- Εξαγωγή πληροφορίας
- Συμπίεση δεδομένων
- Γραφικά υπολογιστών
- Μηχανική μάθηση



Σχήμα 11: Γραφική αναπαράσταση ομαδοποίησης

Υπάρχουν εκατοντάδες αλγόριθμοι ομαδοποίησης οι οποίοι χρησιμοποιούν διαφορετικές μεθόδους και τεχνικές. Δεν υπάρχει κάποιος που να υπερτερεί πλήρως των υπολοίπων αλλά η καταλληλότητα του Κάθε αλγορίθμου εξαρτάται από το σύνολο δεδομένων. Παρά τις διαφορές τους μπορούμε να τους εντάξουμε σε κατηγορίες συμφωνά με τον τρόπο που κάνουν την ομαδοποίηση. Οι τέσσερις πιο βασικές κατηγορίες είναι:

- Centroid based clustering

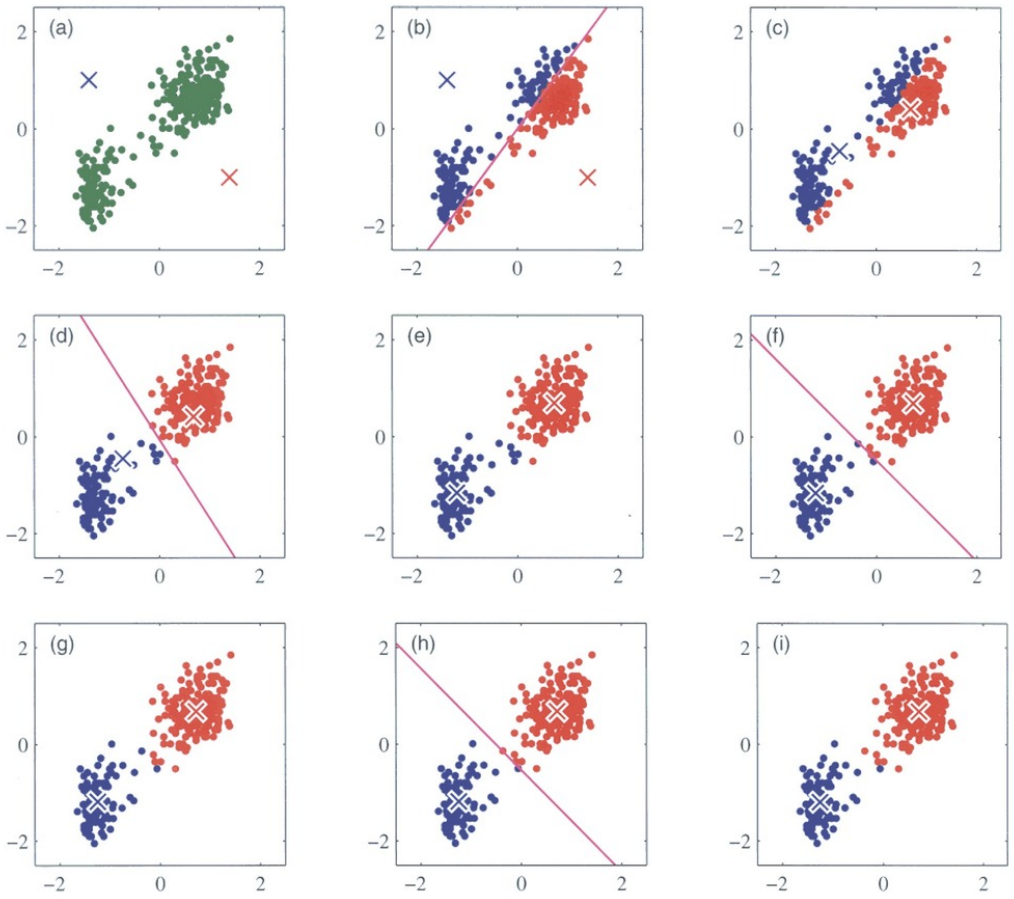
- Density based clustering
- Distribution based clustering
- Hierarchical clustering

Οι κατηγορίες αυτές θα αναλυθούν περισσότερο στη συνέχεια. Επιπλέον, θα δούμε και τους πιο διάσημους αλγορίθμους που ανήκουν σε αυτές της κατηγορίες και θα αναλύσουμε και τον τρόπο λειτουργίας τους.

4.1 Centroid based clustering

Οι αλγόριθμοι τύπου Centroid based clustering όπως μπορούμε να καταλάβουμε κάνουν ομαδοποίηση με βάση τα κέντρα. Αυτό σημαίνει ότι θα έχουμε τόσα κέντρα όσες και οι ομάδες που επιλέξαμε. Έπειτα για να ομαδοποιήσουμε όλα τα σημεία του χώρου θα βλέπουμε σε ποιο κέντρο βρίσκονται πιο κοντά και θα τα εντάσσουμε στην ομάδα αυτού του κέντρου. Το πρόβλημα εδώ είναι πως θα βρούμε τα κατάλληλα κέντρα. Αν δεν επιλέξουμε τα κέντρα σωστά τότε η ομαδοποίηση πυ θα κάνουμε θα απέχει πολύ από την πραγματικότητα. Γι΄ αυτό και οι συγκεκριμένοι αλγόριθμοι έχουν ως βασική δουλειά να προσεγγίσουν αυτά τα κέντρα.

Για να καταλάβουμε με ποιον τροπο συμβαίνει αυτό θα δούμε τον αλγόριθμο k-Means που είναι και ο πιο διάσημος αλγόριθμος ταξινόμησης με βάση τα κέντρα. Το k είναι το πλήθος των ομάδων που θέλουμε. ξεκινάμε τον αλγόριθμο επιλέγοντας k τυχαία κέντρα στον χώρο και χωρίζουμε τα υπόλοιπα σημεία σε ομάδες ανάλογά με την εγγύτητα τους στα τυχαία κέντρα. Έτσι έχουμε καταλήξει σε μια αρχική προσέγγιση. Έπειτα θα βρούμε το πραγματικό κέντρο των ομάδων που δημιουργήθηκαν βρίσκοντας τη μέση τιμή των σημείων που ανήκουν στην ομάδα. Αυτή η διαδικασία επαναλαμβάνεται και έτσι κάποια στιγμή τα τελικά κέντρα θα μπορούν να κάνουν καλή ομαδοποίηση.



Σχήμα 12: Επαναλήψεις του αλγορίθμου k-Means

΄Ενα από τα πλεονεκτήματα του αλγορίθμου είναι η απλότητα του. Επιπλέον λειτουργεί καλά σε μεγάλο σύνολο δεδομένων και εξασφαλίζει ότι θα συγκλίνει σε κάποια λύση. Επίσης μπορεί αν δημιουργήσει ομάδες διαφορετικού μεγέθους και σχήματος. Παρ΄ όλα αυτά τα αποτελέσματα τα τελικά αποτελέσματα του αλγορίθμου εξαρτώνται από τις αρχικές συνθηκες. ΄Αρα εφόσον αρχικοποιούμε τα κέντρα τυχαία, αν ξανά τρέξουμε τον αλγόριθμο θα πάρουμε διαφορετικά αποτελέσματα. Επίσης πρέπει να ορίσουμε μόνοι μας το πλήθος των ομάδων που είναι κάτι που μπορεί να μην θέλουμε πάντα[15].

4.2 Density based clustering

Ο επόμενος τύπος αλγορίθμου που θα δούμε είναι η ομαδοποίηση με βάση την πυκνότητα. Οι συγκεκριμένοι αλγόριθμοι ανιχνεύουν τις περιοχές με μεγάλη συγκέντρωση σημείων ο ποίες διαχωρίζονται μεταξύ τους από περιοχές που τα ση-

μεία είναι πολύ αραιά. Αυτές οι περιοχές θα είναι και οι ομάδες που θα δημιουργηθούν και τα σημεία στις αραιές περιοχές θα θεωρηθούν θόρυβος[16].

Ας δούμε πως λειτουργεί ο αλγόριθμος DBSCAN (Density-based spatial clustering of applications with noise) ο οποίος είναι από τους πιο διάσημους αλγορίθμους ομαδοποίησης. Για αυτόν τον αλγόριθμο πρέπει να ορίσουμε μια παράμετρο ελάχιστης απόστασης και μία παράμετρο ελάχιστων σημείων. Έπειτα θεωρούμε τις εξής κατηγορίες σημείων:

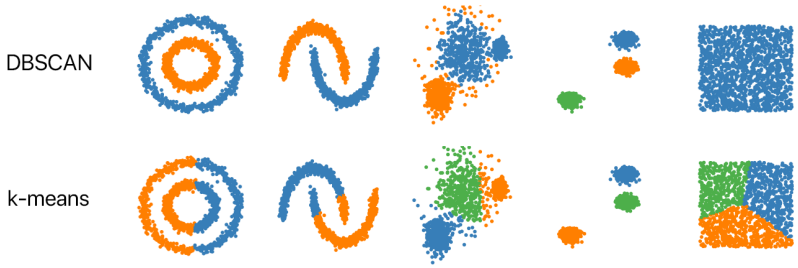
Κύρια σημεία Έχει γειτονικά σημεία τουλάχιστον όσα τα ελάχιστα σε απόσταση μικρότερη από την ελάχιστη.

Άμεσα προσιτά σημεία Για ένα κύριο σημείο ένα άλλο σημείο είναι προσιτό αν είναι μέσα στην ελάχιστη απόσταση.

Προσιτά σημεία Ένα σημείο είναι προσιτό αν υπάρχει ένα μονοπάτι από σημεία που το κάθε ένα είναι άμεσα προσιτό με το διπλανό του. Το σημείο εκκίνησης πρέπει να είναι κύριο σημείο.

Θόρυβος Όλα τα υπόλοιπα σημεία που περίσσεψαν.

Έπειτα κάθε κύριο σημείο δημιουργεί μία ομάδα με όλα τα προσιτά σημεία σε αυτό[17].



Σχήμα 13: Αποτελέσματα ομαδοποίησης με DBSCAN vs k-Means

Από το παραπάνω σχήμα μπορεί να καταλάβει κανείς τις περιπτώσεις που ο αλγόριθμος DBSCAN υπερισχύει του k-Means. Όπως φαίνεται ο DBSCAN είναι κατάλληλος όταν οι ομάδες δεν έχουν απλά σχήματα και επιπλέον μπορεί να διαλέξει μόνος του τον κατάλληλο αριθμό ομάδων που πρέπει να δημιουργήσει κάθε φορά. Συνοψίζοντας τα πλεονεκτήματα του αλγορίθμου είναι[18]:

- Δεν χρειάζεται να ορίσουμε τον αριθμό τον ομάδων
- Δουλεύει καλά σε ομάδες με αυθαίρετα σχήματα

- Μπορεί να καταλάβει ποια σημεία αποτελούν θόρυβο και να μην τα λάβει υπόψη του

Παρ΄ όλα αυτά υπάρχουν και μειονεκτήματα:

- Δέν μπορεί να ομαδοποιήσει δεδομένα που έχουν μεγάλες διαφορές μεταξύ τους γιατί δεν μπορούμε να επιλέξουμε τις παραμέτρους του συστήματος ώστε να ευνοούν όλες τις ομάδες
- Επίσης δεν μπορούμε να διαλέξουμε μια καλή τιμή για την ελάχιστη απόσταση αν δεν έχουμε καλή κατανόηση των δεδομένων
- Δίνει διαφορετικά αποτελέσματα κάθε φορά διότι ξεκινάει την ανάλυση από τυχαία σημεία

4.3 Hierarchical clustering

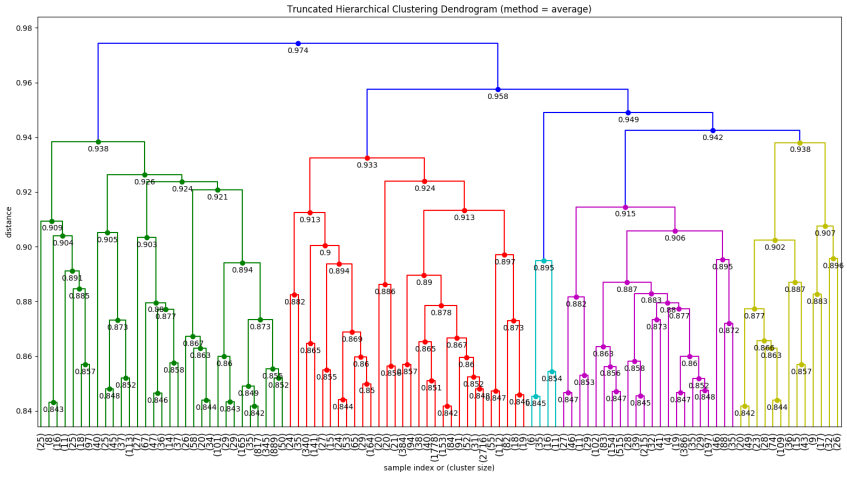
Το Hierarchical clustering είναι μια μέθοδος ομαδοποίησης η οποία προσπαθεί να δημιουργήσει μια ιεραρχία από ομάδες. Για αυτούς τους αλγόριθμους μπορούμε να διακρίνουμε δύο μεγάλες κατηγορίες[19]:

Συσσωματωτική Προσεγγίζει το πρόβλημα από κάτω προς τα πάνω. Το κάθε δείγμα φτιάχνει τη δικιά του ομάδα και όταν ανεβαίνουμε στην ιεραρχία οι ομάδες αυτές συγχωνεύονται σε μεγαλύτερες

Διαιρετική Προσεγγίζει το πρόβλημα από κάτω προς τα πάνω. Ξεκινάμε από μια μεγάλη ομάδα η οποία διαχωρίζεται σε μικρότερες αναδρομικά καθώς κατεβαίνουμε στην ιεραρχία.

Για να αποφασίσουμε πότε δύο ομάδες πρέπει να ενωθούν ή να διαχωριστούν πρέπει να βρούμε μια μετρική ανομοιότητας μεταξύ των σημείων. Αυτή η μετρική τις περισσότερες φορές είναι η απόσταση η οποία μπορεί να είναι η ευκλείδεια η κάποια άλλη που έχουμε ήδη αναλύσει.

Τα αποτελέσματα του αλγορίθμου μπορούν να αναπαρασταθούν με τη μορφή δέντρου



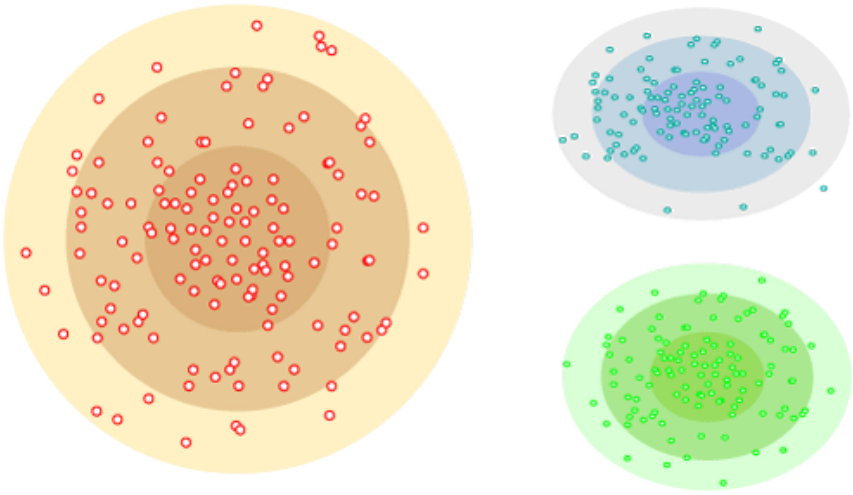
Σχήμα 14: Hierarchical clustering

4.4 Distribution based clustering

Το Distribution based clustering είναι μια υποκατηγορία του Hierarchical clustering και χρησιμοποιείται στις περιπτώσεις που τα δεδομένα έχουν κάποια γνωστή κατανομή όπως είναι η κανονική ή η Γκαουσιανή. Το Distribution based clustering μας δίνει τη δυνατότητα να διαχειριστούμε μεγάλα σύνολα δεδομένων και να βρούμε μη γραμμικές σχέσεις ανάμεσα στα στοιχεία[20].

Από τους πιο διάσημους αλγορίθμους είναι ο αλγόριθμος Gaussian mixture models. Όπως μπορούμε να καταλάβουμε από το όνομά του ο αλγόριθμος αυτός κάνει την υπόθεση ότι τα δεδομένα ακολουθούν Γκαουσιανή κατανομή. Στην στατιστική όταν μιλάμε για Γκαουσιανή κατανομή ξέρουμε ότι τα δεδομένα μας θα έχουν μέσο όρο και απόκλιση. Τώρα που τα δεδομένα μας όμως είναι πολλών διαστάσεων θα πρέπει να γενικεύσουμε αυτες τις παραμέτρους. Έτσι κάθε ομάδα που θα δημιουργείται από τον αλγόριθμο θσ έχει ένα κέντρο και έναν πίνακα συνδιακύμανσης. Αυτούς τους όρους τους έχουμε ξαναδεί σε προηγούμενη ενότητα και οι τύποι βρίσκονται στο παράρτημα Α΄. Μια επιπλέον αναγκαία παράμετρος του συστήματος είναι και η πιθανότητα της κάθε ομάδας. Έτσι με αυτές τις τρεις παραμέτρους μπορούμε να ομαδοποιήσουμε τα δεδομένα.

Η γραφική απεικόνιση των ομάδων στις δύο διαστάσεις θα έχουν τη μορφή ελλείψεων οι οποίες θα περιγράφονται από τις παραμέτρους που αναφέραμε προηγουμένως.



Σχήμα 15: Γραφική αναπαράσταση Distribution based clustering

5 Νευρωνικά Δίκτυα

Τα νευρωνικά δίκτυα είναι στις μέρες μας το κυρίαρχο μοντέλο στη μηχανική μάθηση και μπορεί να λύσει και τους τρεις τύπους προβλημάτων που έχουμε δει (ταξινόμηση, παλινδρόμηση και ομαδοποίηση). Τα τεχνητά νευρωνικά δίκτυα δημιουργήθηκαν με σκοπό να μιμηθούν τον τρόπο που λειτουργεί ο εγκέφαλος του ανθρώπου. Οι νευρώνες του εγκεφάλου μας λειτουργούν δημιουργώντας συνδέσεις με άλλους νευρώνες και όλοι μαζί συμβάλουν στην επεξεργασία ενός ηλεκτρικού σήματος που έρχεται από τον εγκέφαλο το οποίο τελικά καταλήγει να είναι απλή καθημερινή πράξη για τον άνθρωπο (όπως για παράδειγμα να κουνήσει το χέρι του)[21].

Τα τεχνητά νευρωνικά δίκτυα προσπαθούν λοιπόν να αντιγράψουν αυτη ακριβώς την ιδιότητα. Ο σκοπός τους είναι να παίρνουν μια είσοδο και να παράγουν μία απάντηση. Για παράδειγμα θα μπορούσαμε σαν είσοδο να δώσουμε τις αιματολογικές εξετάσεις ενός ανθρώπου και η έξοδος να είναι 0 ή 1 ανάλογα αν έχει μια ασθένεια ή όχι. Αυτό είναι ένα πρόβλημα δυαδικής ταξινόμησης αλλά θα μπορούσαμε να λύσουμε και πολλά άλλα προβλήματα και θα αναλύσουμε στην συνέχεια τις αλλαγές που πρέπει να κάνουμε σε ένα νευρωνικό δίκτυο ανάλογα το πρόβλημα. Σε αυτή την ενότητα θα αναλύσουμε διάφορες έννοιες τις οποίες πρέπει να γνωρίζει κανείς αν θέλει να κατανοήσει τα νευρωνικά δίκτυα. Αυτές είναι[22]:

- Νευρώνας
- Επίπεδο
- Οπισθοδρόμηση (back propagation)
- Συναρτήσεις ενεργοποίησης
- Συναρτήσεις σφάλματος

5.1 Επίπεδα

Το απλούστερο δίκτυο νευρώνων αποτελείται από 3 ήδη επιπέδων, το επίπεδο εισόδου(input layer), το κρυφό επίπεδο(hidden layer) και το επίπεδο εξόδου(output layer).

Στο επίπεδο εισόδου ο κάθε νευρώνας έχει τις τιμές της εισόδου (π.χ. για εικόνα τις RGB τιμές κάθε pixel) και έχει όσους νευρώνες όσες και οι χαρακτηριστικές τιμές της εισόδου. Στο επίπεδο εξόδου συνήθως υπάρχουν τόσοι νευρώνες όσα και τα αντικείμενα που θέλουμε να αναγνωρίζει, αλλά αυτό εξαρτάται από την κωδικοποίηση που θα χρησιμοποιήσουμε.

Το κρυφό επίπεδο μπορεί να έχει αυθαίρετο αριθμό νευρώνων και είναι κάτι που αποφασίζει ο χρήστης σύμφωνα με το πρόβλημα και με τον όγκο πληροφορίας που έχει στην διάθεσή του. Συνήθως οι περισσότεροι νευρώνες οδηγούν σε πιο ακριβή προβλέψεις, αλλά ταυτόχρονα αυξάνουν την πολυπλοκότητα και το υπολογιστικό κόστος.

Σε μεγαλύτερα δίκτυα ο μεγάλος αριθμός νευρώνων σε ένα επίπεδο μπορεί να οδηγήσει σε ένα φαινόμενο που λέγεται over-fitting το οποίο θα αναλυθεί αργότερα. Παράλληλα με τον αριθμό των νευρώνων μπορεί να υπάρχουν πολλαπλά κρυφά επίπεδα, με διαφορετικό αριθμό νευρώνων ο καθένας, και ο αριθμός τους βελτιστοποιείται και αυτός σύμφωνα με το μέγεθος του προβλήματος και τον όγκο των δεδομένων.

Συνήθως στα πιο απλά δίκτυα οι νευρώνες κάθε επιπέδου συνδέονται με όλους τους νευρώνες του επόμενου, με ξεχωριστά βάρη για κάθε σύναψη, σχηματίζοντας ένα πλήρως συνδεδεμένο(fully connected) ή πυκνό (dense) επίπεδο. Με αυτό τον τρόπο η έξοδος των προηγούμενων επιπέδων αποτελεί την είσοδο των επόμενων. Ανάλογα με την υλοποίηση μπορούν να χρησιμοποιηθούν και άλλα είδη επιπέδων αλλά δεν θα τα αναλύσουμε στην παρούσα εργασία. Αναφορικά μερικά από τα επίπεδα που μπορούμε να χρησιμοποιήσουμε είναι:

- Convolutional
- Max Pooling
- Flatten
- Batch Normalization
- Dropout
- Residual

Τα βάρη είναι οι τιμές που εκπαιδεύουμε και λέγονται έτσι καθώς μας δείχνουν πόσο βάρος (έμφαση) πρέπει να δώσουμε στην κάθε είσοδο (έξοδο του προηγούμενου επιπέδου). Η συνολική είσοδος που ¨βλέπει¨ ο νευρώνας υπολογίζεται από το σταθμισμένο άθροισμα όλων των εισόδων και μίας τιμής βιας που επηρεάζει την πρόβλεψη του νευρώνα:

$$S_{in} = \sum_{i=0}^{n} a_i w_i + b$$

Η τιμή του νευρώνα προκύπτει από τιμή της συνάρτησης ενεργοποίησης όταν έχει για είσοδο το σταθμισμένο άθροισμα:

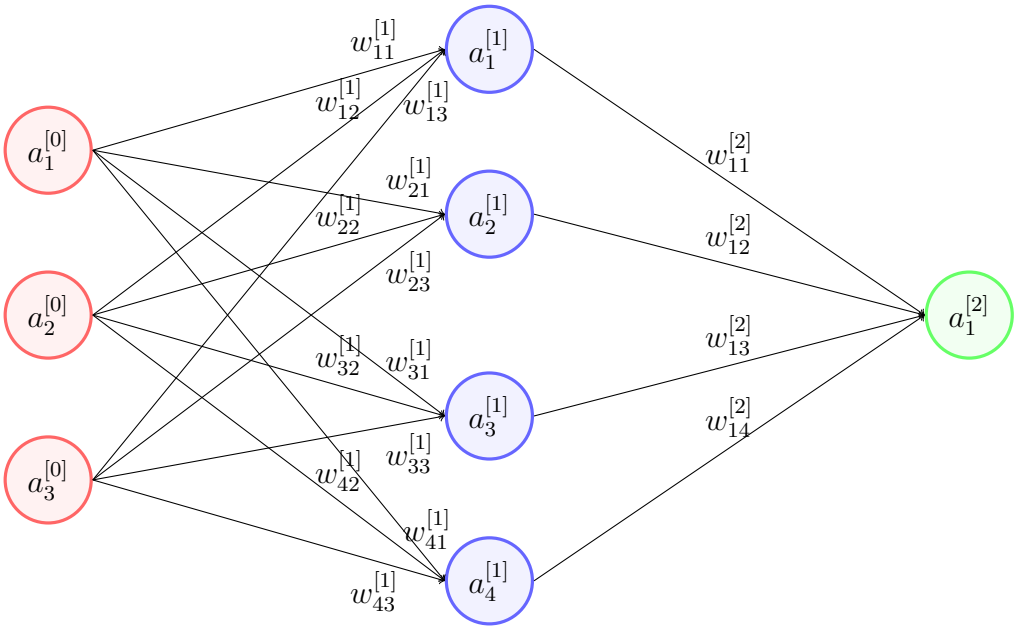
$$a = f(S_{in})$$

Η αρχικοποίηση των βαρών γίνεται με τυχαίο τρόπο και με την πάροδο του χρόνου, μέσω της διαδικασίας της εκπαίδευσης, συγκλίνουν στις κατάλληλες τιμές.

Για να εκπαιδεύσουμε το δίκτυο χρησιμοποιούμε δύο μεθόδους που λέγονται forward και back propagation και θα αναλυθούν λεπτομερώς παρακάτω. Με τις δύο μεθόδους περνάμε τα δεδομένα μας πολλές φορές από το δίκτυο μέχρι αυτό να εκπαιδευτεί επαρκώς. Σύμφωνα με κάποια κριτήρια μπορεί να τερματίσουμε την εκπαίδευση πρόωρα για να διασφαλίσουμε την ακρίβεια των προβλέψεων που κάνουμε.

5.2 Διάδοση προς τα εμπρός

Έχοντας λοιπόν το απλό νευρωνικό δίκτυο με 1 input, 1 hidden και 1 output layer, όπου όλα μεταξύ τους είναι φυλλψ ςοννεςτεδ, θα αναλύσουμε τον τρόπο με τον οποίο διαδίδεται η πληροφορία από τα πίσω προς τα μπροστά επίπεδα, δηλαδή από την είσοδο προς την έξοδο.



Σχήμα 16: Παράδειγμα νευρωνικου δικτύου

Για να ξεκινήσουμε θα αριθμήσουμε όλους τους νευρώνες του δικτύου. Οι εκθέτες δείχνουν το επίπεδο που είμαστε με τα input να είναι το επίπεδο 0. Οι δείκτες δίνουν τον αριθμό του νευρώνα στο εκάστοτε επίπεδο. Έτσι λοιπόν θα ξεκινήσουμε από το επίπεδο 1 (1ο hidden layer), εφόσον στο πρώτο δεν γίνεται κάποια διαδικασία υπολογισμού. Για τον νευρώνα $a_1^{[1]}$ η τιμή του υπολογίζεται

περνώντας το σταθμισμένο άθροισμα όλων των εισόδων του από την συνάρτηση ενεργοποίησης ως εξής:

$$a_1^{[1]} = f_A\left(\sum_{i=1}^{3}\left[w_{1i}^{[1]} \times a_i^{[0]}\right] + b_1\right) = f_A\left(w_{11}^{[1]} \times a_1^{[0]} + w_{12}^{[1]} \times a_2^{[0]} + w_{13}^{[1]} \times a_3^{[0]} + b_1\right)$$

Αντίστοιχα για το $a_2^{[1]}$ θα είχαμε:

$$a_2^{[1]} = f_A\left(\sum_{i=1}^{3}\left[w_{2i}^{[1]} \times a_i^{[0]}\right] + b_2\right) = f_A\left(w_{21}^{[1]} \times a_1^{[0]} + w_{22}^{[1]} \times a_2^{[0]} + w_{23}^{[1]} \times a_3^{[0]} + b_2\right)$$

Και ούτω καθεξής. Η σχέση αυτή γενικεύεται για k επίπεδα και n νευρώνες του κάθε επιπέδου:

$$a_j^{[k]} = f_A\left(\sum_{i=1}^{n}\left[w_{ji}^{[k]} \times a_i^{[k-1]}\right] + b_j\right)$$

Όπου j ο νευρώνας του επιπέδου που υπολογίζουμε την τιμή του με $j \in [1, n]$

Για την ευκολία υπολογισμού των πράξεων και αποθήκευσης από ένα υπολογιστικό σύστημα θα μετατρέψουμε την παραπάνω σχέση σε μορφή πινάκων. Έτσι για την γενικευμένη περίπτωση θα έχουμε:

$$A^{[k]} = f_A\left(\begin{bmatrix} w_{11} & \dots & w_{1n^{[k-1]}} \\ \vdots & \ddots & \vdots \\ w_{n^{[k]}1} & \dots & w_{n^{[k]}n^{[k-1]}} \end{bmatrix}^{[k]} \times \begin{bmatrix} a_1 \\ \vdots \\ a_{n^{[k-1]}} \end{bmatrix}^{[k]} + \begin{bmatrix} b_1 \\ \vdots \\ b_{n^{[k]}} \end{bmatrix}^{[k]}\right) = \begin{bmatrix} a_1 \\ \vdots \\ a_{n^{[k]}} \end{bmatrix}^{[k]}$$

Και απλοποιείται σε:

$$A^{[k]} = f_A(W^{[k]} \times A^{[k-1]} + B^{[k]})$$

Όπως βλέπουμε οι έξοδοι των νευρώνων από κάθε επίπεδο είναι είσοδοι για το επόμενο, όπου με τον πολλαπλασιασμό των βαρών του επόμενου και το σταθμισμένο άθροισμα αυτών (+ το bias του επιπέδου), διαδίδουμε την πληροφορία προς τα μπροστά επίπεδα. Η διαδικασία συνεχίζεται μέχρι να φτάσουμε στο τελευταίο επίπεδο (επίπεδο εξόδου), όπου και παίρνονται οι αποφάσεις (προβλέψεις) με τον τρόπο που θα δούμε παρακάτω. Λόγο της μετάδοσης αυτής προς τα εμπρός επίπεδα η διαδικασία ονομάζεται forward propagation.

5.3 Οπισθοδρόμηση

Όπως αναφέραμε προηγουμένως η ανάθεση βαρών γίνεται τυχαία στην αρχή και επικαιροποιούνται αυτόματα, ανά κάποια χρονικά διαστήματα που εξαρτώνται από τον όγκο των δεδομένων και το batch size (θα αναλυθεί παρακάτω). Προφανώς ακόμα και για ένα μικρό δίκτυο σαν αυτό του παραδείγματος, πόσο μάλλον για ένα μεγαλύτερο, το να επικαιροποιηθούν τα βάρη χωρίς κάποια αυτοματοποίηση καθιστά πρακτικά άχρηστο το δίκτυο.

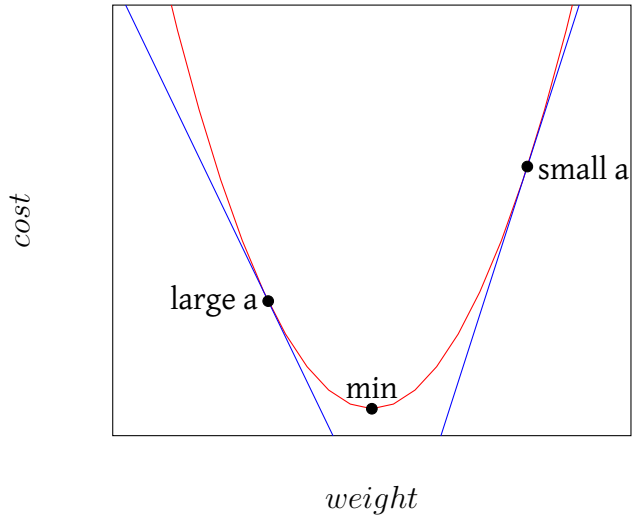
Η διαδικασία αυτή θα ήταν αδύνατη λόγο του μεγάλου όγκου πληροφορίας και των αναγκών ταχύτητας που θα απαιτούσαν κάποιες εφαρμογές. Η μέθοδος αυτόματης επικαιροποίησης των βαρών με σκοπό την βελτιστοποίησή τους, ακούει στο όνομα back propagation και αντικατοπτρίζει την διαδικασία της μάθησης.

Για να μπορέσουμε να βελτιστοποιήσουμε τα βάρη χρειαζόμαστε ένα μέτρο σύγκρισης, όπου θα παίρνει μικρές τιμές όταν κάνουμε καλές προβλέψεις και μεγάλες όταν κάνουμε άστοχες προβλέψεις. Χρησιμοποιούμε λοιπόν την συνάρτηση σφάλματος (cost function), όπου cost είναι η αναπαράσταση του σφάλματος.

Για κάθε πρόβλημα χρησιμοποιούμε συγκεκριμένες συναρτήσεις κόστους οι οποίες προσαρμόζονται κατάλληλα στο κάθε πρόβλημα. Μπορεί κάποιος να καταλάβει ότι στην αρχή που τα βάρη αρχικοποιούνται τυχαία το κόστος θα είναι ψιλό, καθώς οι προβλέψεις μας θα είναι σε μεγάλο βαθμό λανθασμένες.

Για να βελτιστοποιήσουμε τα βάρη πρέπει να ελαχιστοποιήσουμε το κόστος. Εφόσον οι προβλέψεις (έξοδοι) εξαρτώνται από τα βάρη, και το κόστος εξαρτάται από τις εξόδους, με την σειρά του το κόστος εξαρτάται και αυτό από τα βάρη. Λόγω του forward propagation καταλήγουμε σε μία εξάρτηση του κόστους από τα βάρη όλου του δικτύου και όχι μόνο του τελευταίου, αφού η έξοδος του κάθε επιπέδου εξαρτάται από τα βάρη και τις εισόδους του.

Εάν παράξουμε την γραφική weight – cost για ένα συγκεκριμένο βάρος (οποιοδήποτε από τα βάρη του δικτύου) θα έχουμε ένα αποτέλεσμα που θα μοιάζει με το παρακάτω:



Σχήμα 17: Γραφική παράσταση σφάλματος

Όπως είπαμε παραπάνω στην αρχή λόγω της τυχαίας αρχικοποίησης το κόστος πιθανότατα δεν θα έχει την βέλτιστη τιμή που ελαχιστοποιεί το κόστος. Μπορεί για παράδειγμα να βρίσκεται στο δεξιά σημείο που βλέπουμε στην γραφική, όπου το κόστος είναι μεγάλο.

Για να φτάσουμε ή να προσεγγίσουμε στο ολικό ελάχιστο θα πρέπει να ανανεώνουμε τα βάρη κάθε φορά που περνάμε τα δεδομένα για την διαδικασία της μάθησης, με τον κανόνα Δέλτα που φαίνεται παρακάτω. Για συνεχές συναρτήσεις ενεργοποίησης, θα θέλαμε να αλλάξουμε περισσότερο τα βάρη εκείνα, στις μεταβολές των οποίων το συνολικό σφάλμα είναι πιο ¨ευαίσθητο¨. Για έναν νευρώνα του επιπέδου εξόδου και έστω ότι έχουμε το τετραγωνικό σφάλμα(κόστος):

$$E = (a_i - o_i)^2$$

Με o_i να είναι η επιθυμητή έξοδος και a_i η πραγματική. Ο ρυθμός αύξησης του σφάλματος σε σχέση με το βάρος w_{ji} είναι:

$$\frac{\partial E}{\partial w_{ji}} = 2(a_i - o_i)\frac{\partial (a_i - o_i)}{\partial w_{ji}} = 2(a_i - o_i)\frac{\partial}{\partial w_{ji}}\left[f\left(\sum_j (w_{ji}a_j) - o_i\right)\right]$$

$$\frac{\partial E}{\partial w_{ji}} = 2(a_i - o_i)f'(S_i)a_j$$

Όπου S_i είναι το σταθμισμένο άθροισμα των εισόδων. Άρα το βάρος θα αλλάξει ως εξής:

$$w_{new} = w_{old} - alpha\frac{\partial E}{\partial w_{ji}}$$

$$\Delta w_{ji} = -alpha \frac{\partial E}{\partial w_{ji}}$$

Παρατηρούμε πως με την σχέση αυτή, κάθε φορά που επαναυπολογίζουμε το w αφαιρούμε από το προηγούμενο την κλίση της καμπύλης πολλαπλασιασμένη με έναν συντελεστή $alpha$. Ο συντελεστής $alpha$ λέγεται ρυθμός μάθησης (learning rate) και είναι ένας μικρός θετικός αριθμός. Βλέπουμε πως για μεγάλο $alpha$ το βάρος συγκλίνει γρηγορότερα στην βέλτιστη τιμή.

Επίσης παρατηρούμε πως τα βάρη ενός νευρώνα επηρεάζονται το ίδιο από την τιμή της παραγώγου $f'(S_i)$, ενώ τα βάρη διαφορετικών νευρώνων επηρεάζονται σε διαφορετικό βαθμό από τις αντίστοιχες παραγώγους. Από αυτές τις παρατήσεις συμπεραίνουμε πως η παράγωγος κατά κάποιο τρόπο τροποποιεί τον ρυθμό μάθησης και μάλιστα ξεχωριστά για κάθε νευρώνα και για κάθε είσοδο.

Αφαιρώντας την κλίση της ευθείας εξασφαλίζουμε ότι θα συγκλίνει στο ελάχιστο, εφόσον για τα σημεία όπου το βάρος είναι μικρότερο από το βάρος του ελαχίστου η παράγωγος είναι αρνητική άρα το βάρος θα αυξηθεί στην επόμενη επανάληψη. Το αντίστοιχο γίνεται και για τα θετικότερα βάρη. Η διαδικασία αυτή ονομάζεται gradient descent και πρέπει να γίνει για όλα τα βάρη του δικτύου αλλά και για τα bias του κάθε νευρώνα με όμοιο τρόπο.

Όλα αυτά που αναφέραμε ήταν για το πρώτο επίπεδο(επίπεδο εξόδου)του back propagation. Τώρα πρέπει να διαδώσομε το σφάλμα προς το επίπεδο εισόδου μέσω των κρυφών επιπέδων και από το σφάλμα αυτό να επικαιροποιούμε παράλληλα τα βάρη και τα bias. Έτσι το προσαρμοσμένο σφάλμα των κρυφών επιπέδων υπολογίζεται από το σφάλμα των αμέσως επόμενων επιπέδων σύμφωνα με την σχέση:

$$\delta_i = f'(S_i) \sum w_{ik} d_k$$

Όπου :

k **:** επόμενο επίπεδο (προς την έξοδο)

i **:** το επίπεδο που εξετάζουμε

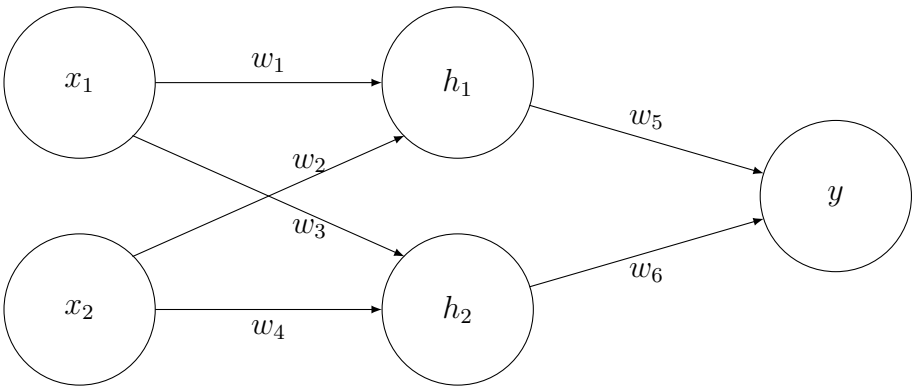
Έχοντας υπολογίσει για κάθε νευρώνα το προσαρμοσμένο σφάλμα η αλλαγή στα βάρη εισόδου θα γίνει με τον εξής τρόπο:

$$\Delta w_{ji} = -alpha \times \delta_i \times a_j$$

Δηλαδή η αλλαγή στο βάρος από τον νευρώνα j (προηγούμενο) στον νευρώνα i (επόμενο) εξαρτάται από το σφάλμα του νευρώνα i την έξοδο του νευρώνα j και την σταθερά μάθησης.

5.4 Συναρτήσεις ενεργοποίησης

Οι συναρτήσεις ενεργοποίησης είναι ένα πολύ σημαντικό κομμάτι των νευρωνικών δικτύων διότι προσφέρουν μη γραμμικότητα (επιθυμητή) στο σύστημά μας. Για να καταλάβουμε καλύτερα τι σημαίνει αυτό θα δούμε ένα παράδειγμα[23]:



Σχήμα 18: Παράδειγμα νευρωνικού δικτύου χωρίς ενεργοποίηση

Βλεπουμε ότι η είσοδος με την έξοδο συνδέονται μέσω ενός κρυφού επιπέδου (dense) όπου:

$$h_1 = w_1 \times x_1 + w_2 \times x_2$$
$$h_2 = w_3 \times x_1 + w_4 \times x_2$$
$$y = w_5 \times h_1 + w_6 \times h_2$$

Αν αναλύσουμε λοιπόν τον τύπο του y θα πάρουμε:

$$y = w_1(w_5 \times x_1 + w_2 \times x_2) + w_6(w_3 \times x_1 + w_4 \times x_2)$$

Που τελικά γίνεται:

$$y = (w_1 w_5 + w_3 w_6) \times x_1 + (w_2 w_5 + w_4 w_6) \times x_2$$

Τα w είναι σταθεροί αριθμοί οπότε τελικά ενώ έχουμε ένα κρυφό επίπεδο ανάμεσα στην είσοδο και στην έξοδο αυτές συνδέονται και πάλι με γραμμική σχέση. Το ίδιο θα συνέβαινε και για πολλαπλά κρυφά επίπεδα με πολλούς νευρώνες. Αυτό για το δίκτυο μας σημαίνει ότι μπορεί να προσεγγίσει δεδομένα που ακολουθούν επίσης μια γραμμική σχέση. Τα δεδομένα όμως συνήθως δεν έχουν τέτοια συμπεριφορά. Για να λύσουμε αυτό το πρόβλημα δημιουργήθηκαν οι συναρτήσεις

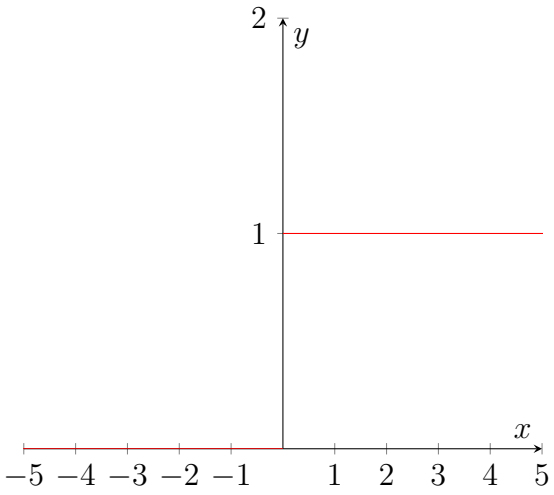
ενεργοποίησης. Στο παράδειγμά μας μπορούμε να προσθέσουμε μις συνάρτηση ενεργοποίησης αμέσως μετά το κρυφό επίπεδο. Έτσι τελικά η έξοδος θα είναι:

$$y = w_5 \times f_A(h_1) + w_6 \times f_A(h_2)$$

Όπου f_A η συνάρτηση ενεργοποίησης. Παρατηρούμε ότι η είσοδος δεν μπορεί πλέον να συνδεθεί γραμμικά με την έξοδο και άρα έχουμε πετύχει τον στόχο μας.

Υπάρχουν πολλές συναρτήσεις ενεργοποίησης και όλες έχουν τα πλεονεκτήματα και τα μειονεκτήματα τους. Εμεις θα αναλύσουμε κάποιες από τις πιο γνωστές οι οποίες χρησιμοποιούνται σχεδόν πάντα στα νευρωνικά δίκτυα. Από τις πρώτες συναρτήσεις ενεργοποίησης ήταν η βηματική συνάρτηση:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x >= 0 \end{cases}$$

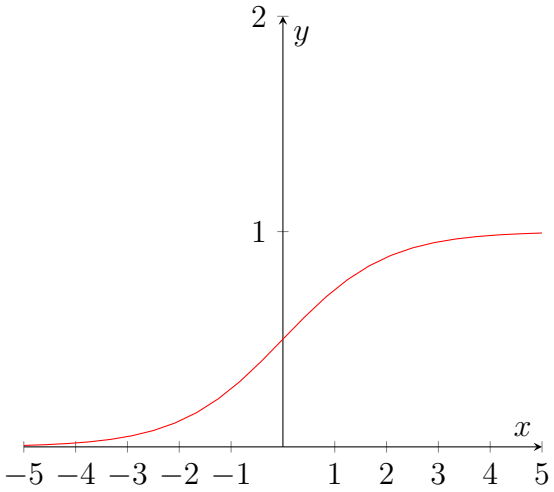


Σχήμα 19: step function

Η βηματική συνάρτηση είναι χρήσιμη όταν έχουμε ένα πρόβλημα δυαδικής ταξινόμησης. Μπορούμε δηλαδή να την εφαρμόσουμε στο επίπεδο εξόδου έτσι ώστε η απάντηση που θα παίρνουμε να είναι πάντα 0 ή 1. Όμως έχει ένα μεγάλο μειονέκτημά το οποίο είναι ότι δεν είναι συνεχής. Η τεχνική της Οπισθοδρόμησης βασίζεται στις παραγώγους των συναρτήσεων κάτι το οποίο την κάνει πολύ δύσχρηστη. Θα μπορούσαμε θεωρητικά να αγνοήσουμε το 0 στο οποίο η συνάρτηση δεν είναι παραγωγίσιμη και να του δώσουμε μια τιμή (όπως 0) αλλά υπάρχει ένα ακόμα πρόβλημα. Το πρόβλημα είναι ότι η παράγωγός της είναι παντού 0 και γνωρίζουμε από την οπισθοδρόμηση ότι πρέπει να πολλαπλασιάσουμε το σφάλμα με αυτή την παράγωγο. Άρα τελικά το σφάλμα που θα διαδίδεται στα επίπεδα θα είναι μηδενικό και άρα δεν θα γίνεται καμία διόρθωση.

Μια καλύτερη συνάρτηση που διατηρεί τα χαρακτηριστικά της βηματικής είναι η σιγμοειδής:

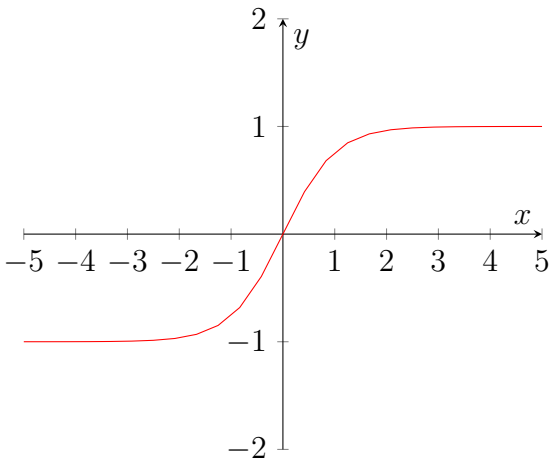
$$f(x) = \frac{1}{1 + e^{-x}}$$



Σχήμα 20: sigmoid function

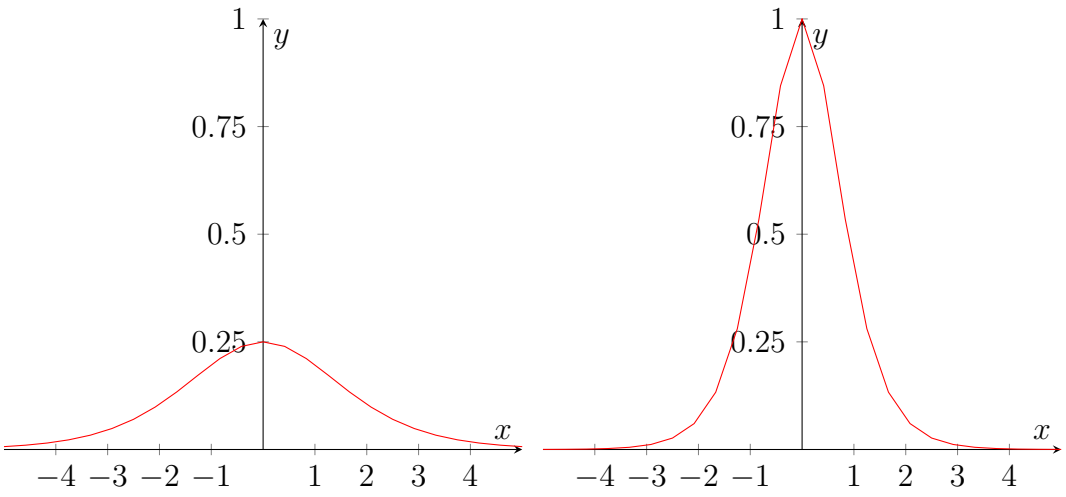
Αυτή η συνάρτηση είναι επίσης φραγμένη από το 0 ως το 1 και για αυτό μπορεί να χρησιμοποιηθεί επίσης στο επίπεδο εξόδου για δυαδική ταξινόμηση. Η σιγμοειδής όμως έχει ποιο ομαλή και συνεχή μετάβαση από το 0 στο 1 άρα μπορούμε να εφαρμόσουμε τους κανόνες της οπισθοδρόμησης. Χρησιμοποιείται λοιπόν με αυτόν τον τρόπο μέχρι και σήμερα. Παρ΄ όλα αυτά για τα κρυφά επίπεδα, τα οποία δεν είναι απαραίτητο να κυμαίνονται απο 0 μέχρι 1, υπάρχει μια ακόμα καλύτερη συνάρτηση. Αυτή είναι η συνάρτηση της υπερβολικής εφαπτομένης:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Σχήμα 21: tanh function

Η συνάρτηση αυτή είναι καλύτερη επειδή αντιμετωπίζει το λεγόμενο vanishing gradient problem. Για να καταλάβουμε τι σημαίνει αυτό θα πρέπει να συγκρίνουμε τις παραγώγους των δύο συναρτήσεων:



(α′) sigmoid derivative (β′) tanh derivative

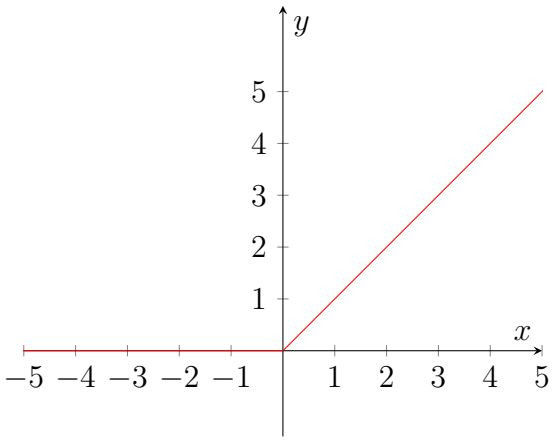
Σχήμα 22: derivative comparison

Από το παραπάνω σχήμα μπορούμε να δούμε ότι η παράγωγός της σιγμοειδούς έχει μέγιστη τιμή το 0.25. Αυτό σημαίνει ότι αν έχουμε πολλά κρυφά επίπεδα με αυτή την ενεργοποίηση, κάθε φορά που το σφάλμα θα διαδίδεται από το ένα επίπεδο στο άλλο θα υπο-τετραπλασιάζεται στην καλύτερη περίπτωση. Έτσι στο τέλος τα αρχικά επίπεδα θα κάνουν ελάχιστη διόρθωση στα βάρη τους. Αντιθέτως

η παράγωγος της υπερβολικής εφαπτομένης έχει μέγιστο το 1 άρα δίνει πολύ καλύτερα αποτελέσματα. Παρ΄ ότι μειώνεται το πρόβλημα όμως δεν εξαφανίζεται. Γι΄ αυτό οι επόμενες συναρτήσεις που θα δούμε αντιμετωπίζουν πλήρως το vanishing gradient problem.

Η επόμενη συνάρτηση που αυτή τη στιγμή είναι και η πιο διάσημη συνάρτηση ενεργοποίησης στη μηχανική μάθηση είναι η ReLU (rectified linear unit):

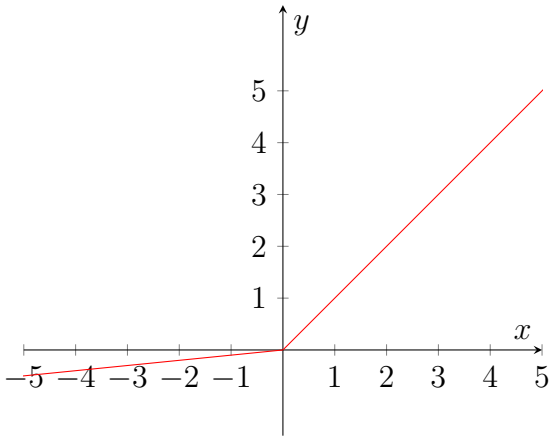
$$f(x) = \begin{cases} 0 & x < 0 \\ x & x >= 0 \end{cases}$$



Σχήμα 23: ReLU function

Η συνάρτηση αυτή έχει την ικανότητα να προσφέρει ταυτόχρονα τα πλεονεκτήματα της γραμμικότητας αλλά και της μη γραμμικότητας. Λόγω της γραμμικότητας δεν έχει το vanishing gradient problem και επίσης έχει πολύ απλή υλοποίηση. Αλλά η μη γραμμικότητα που έχει της επιτρέπει να εφαρμόζει την επιθυμητή μη γραμμικότητα που θέλουμε στα νευρωνικά δίκτυα. Ενώ η συνάρτηση είναι η πιο διάσημη δεν σημαίνει ότι δεν έχει και αυτή τα προβλήματά της. Συγκεκριμένα μιλάμε για το λεγόμενο dying ReLU problem. Συγκεκριμένα η παράγωγός της για αρνητικές τιμές είναι 0. Αυτό σημαίνει ότι όταν είναι αρνητική η είσοδος δεν θα μπορεί να διαδοθεί το σφάλμα από εκείνο το επίπεδο. Γι΄ αυτό και οι επόμενες συναρτήσεις που θα δούμε είναι παραλλαγές αυτής της συνάρτησης με σκοπό να αντιμετωπίσουν αυτό το πρόβλημα. Ξεκινάμε με την Leaky ReLU:

$$f(x) = \begin{cases} 0.01x & x < 0 \\ x & x >= 0 \end{cases}$$



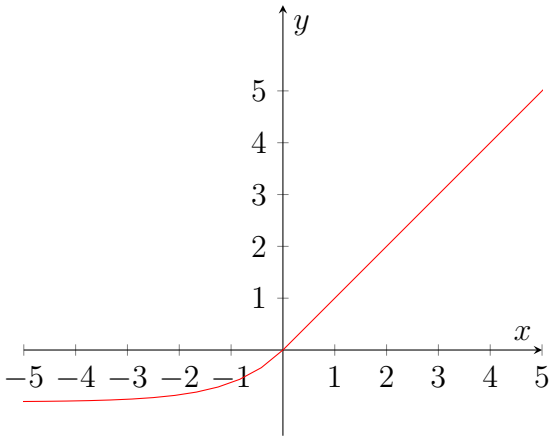
Σχήμα 24: ReLU function

Γενίκευση της αποτελει τη parametric ReLU με τύπο:

$$f(x) = \begin{cases} ax & x < 0 \\ x & x >= 0 \end{cases}$$

Αυτές αντιμετωπίζουν το πρόβλημα με έναν πολύ απλό τρόπο. Παρ΄ όλα αυτά έχουν βρεθεί και πιο πολύπλοκες συναρτήσεις που θα δούμε παρακάτω. Ακολουθεί η συνάρτηση ELU (exponential linear unit):

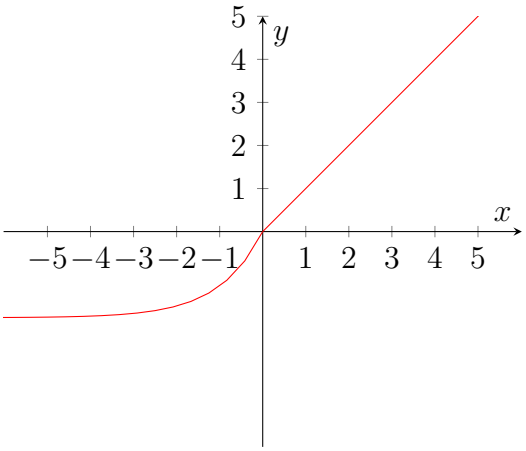
$$f(x) = \begin{cases} a(e^x - 1) & x < 0 \\ x & x >= 0 \end{cases}$$



Σχήμα 25: ELU function

Όπως και πριν μπορούμε να γενικεύσουμε αυτή τη συνάρτηση. Η γενικευμένη μορφή ονομάζεται SELU (scaled exponential linear unit):

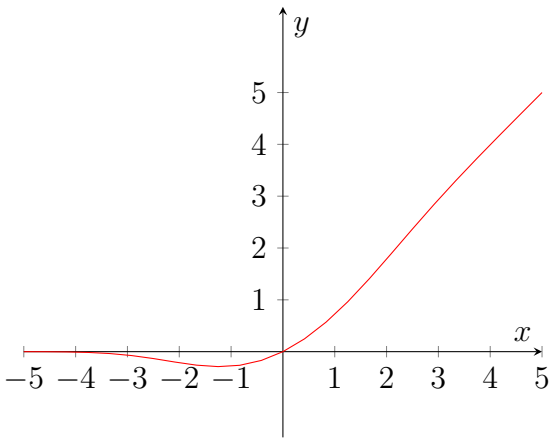
$$f(x) = \lambda \begin{cases} a(e^x - 1) & x < 0 \\ x & x >= 0 \end{cases}$$



Σχήμα 26: SELU function

Οι παραπάνω συναρτήσεις όπως έχουμε δει έχουν δύο κλάδους με σκοπό να πετύχουν τη γραμμικά και μη γραμμικά στοιχεία που θέλουμε. Οι παρακάτω συναρτήσεις καταφέρνουν να έχουν παρόμοια συμπεριφορά με έναν τύπο συνάρτησης. Η μία από αυτές είναι η GELU (Gaussian error linear unit):

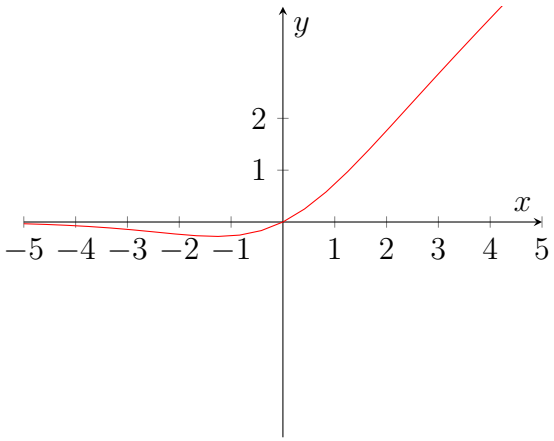
$$0.5x\left(1 + \tanh\left[\frac{\sqrt{2}}{\pi}(x + 0.044715x^3)\right]\right)$$



Σχήμα 27: GELU function

Η δεύτερη συνάρτηση είναι η συνάρτηση swish:

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



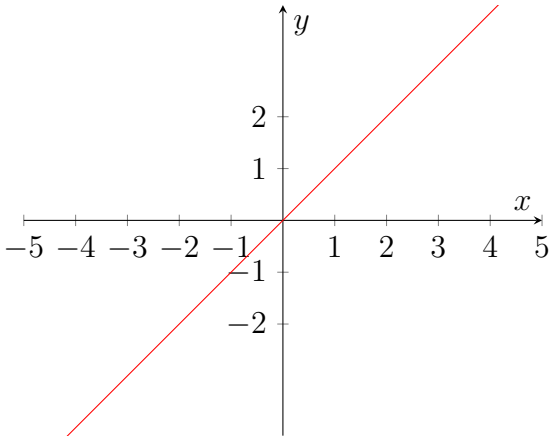
Σχήμα 28: swish function

Όλες αυτές οι συναρτήσεις έχουν παρόμοιες ιδιότητες. Πολλές από αυτές θεωρητικά μπορεί να είναι καλύτερες από τη ReLU αλλά στην πράξη αυτή εξακολουθεί να είναι η κυρίαρχη συνάρτηση για λόγους απλότητας.

Όλες οι συναρτήσεις που έχουμε δείξει μέχρι τώρα με εξαίρεση την σιγμοειδή χρησιμοποιούνται μόνο σε κρυφά επίπεδα. Παρ΄ όλα αυτά χρειαζόμαστε και κάποιες συναρτήσεις για τα επίπεδα εξόδου οι οποίες να ευνοούν το πρόβλημα που έχουμε να λύσουμε. Για παράδειγμα όπως έχουμε ήδη αναφέρει, για ένα πρόβλημα δυαδικής ταξινόμησης θα χρησιμοποιούσαμε τη σιγμοειδή συνάρτηση. Το ίδιο ισχύει και για ένα πρόβλημα ταξινόμησης πολλών ετικετών. Θα είχαμε δηλαδή πολλούς νευρώνες στην έξοδο και καθένας απο αυτους θα χρησιμοποιούσε τη σιγμοειδή σαν συνάρτηση ενεργοποίησης.

Τα προβλήματα τα οποία παραμένουν είναι η παλινδρόμηση και η ταξινόμηση πολλαπλών κλάσεων (αλλά μόνο μία ετικέτα τη φορά). Η συνάρτηση ενεργοποίησης που Χρησιμοποιείται στην παλινδρόμηση είναι η γραμμική συνάρτηση:

$$f(x) = \lambda x$$



Σχήμα 29: linear function

Αυτή η συνάρτηση βρίσκει χρήση μόνο σε επιπεδα εξόδου. Οι συναρτήσεις που έχουμε δείξει μέχρι στιγμής είναι όλες φραγμένες από μία ή και από δύο μεριές. Αυτό τις καθιστά ακατάλληλες για την εφαρμογή στην έξοδο ενός προβλήματος παλινδρόμησης στο οποίο η έξοδος θα πρέπει να μπορεί να πάρει οποιαδήποτε τιμή. Γι΄ αυτό και υπάρχει η γραμμική συνάρτηση για αυτές τις περιπτώσεις.

Τέλος μας μένει το πρόβλημα ταξινόμησης πολλαπλών κλάσεων. Θα μπορούσε να λυθεί με τον ίδιο τρόπο με το πρόβλημα πολλαπλών ετικετών. Όμως σε αυτήν την περίπτωση δεν μας ενδιαφέρει δύο νευρώνες να μπορούν να τείνουν στην τιμή 1 αλλά μόνο ένας τη φορά. Γι΄ αυτόν ακριβώς το λόγο υπάρχει μια καλύτερη συνάρτηση για αυτή τη δουλεια που λέγεται softmax:

$$f_i(x) = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}}$$

N είναι ο αριθμός των νευρώνων της εισόδου. Η συγκεκριμένη συνάρτηση όπως μπορεί να καταλάβει κανείς παράγει τιμές (όσοι και οι είσοδοι) που αθροίζονται σε 1. Ταυτόχρονα όμως επειδή χρησιμοποιούμε την εκθετική συνάρτηση οι μεγάλες τιμές γίνονται ακόμα πιο μεγάλες (δηλαδή παίρνουν μεγαλύτερο ποσοστό). Επισης η εκθετική συνάρτηση μας βοηθάει και σε άλλα προβλήματα όπως με τις αρνητικές τιμές και με μηδενικά αθροίσματα (διαίρεση με το 0).

5.5 Συναρτήσεις κόστους

Ένα ακόμα πολύ σημαντικό στοιχείο των νευρωνικών δικτύων είναι οι συναρτήσεις κόστους. Οι συναρτήσεις κόστους είναι αυτές που μας επιτρέπουν να αξιολογήσουμε την ποιότητα του δικτύου μας στο στάδιο της εκπαίδευσης αλλά και

της επαλήθευσης. Αυτό μπορεί να μας δώσει χρήσιμες πληροφορίες όπως για παράδειγμα αν υπάρχει over fitting (κάτι που θα σήμαινε ότι πρέπει να σταματήσουμε). Όπως είδαμε και με τις συναρτήσεις ενεργοποίησης υπάρχουν διαφορετικοί τύποι προβλήματος και για τον καθένα από αυτούς είχαμε και μια διαφορετική ενεργοποίηση εξόδου που ήταν κατάλληλη για το συγκεκριμένο πρόβλημα. Το ίδο ακριβώς συμβαίνει και με τις συναρτήσεις κόστους.

Ξεκινώντας με το πιο απλό πρόβλημα της παλινδρόμησης έχουμε δύο συναρτήσεις κόστους οι οποίες είναι πολύ παρόμοιες. Αυτές είναι οι mean absolute error (MAE) και mean square error (MSE):

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - p_i|$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - p_i)^2$$

Όπου:

y **:** Η έξοδος που παράχθηκε από το νευρωνικό δίκτυο

p **:** Η πραγματική έξοδος

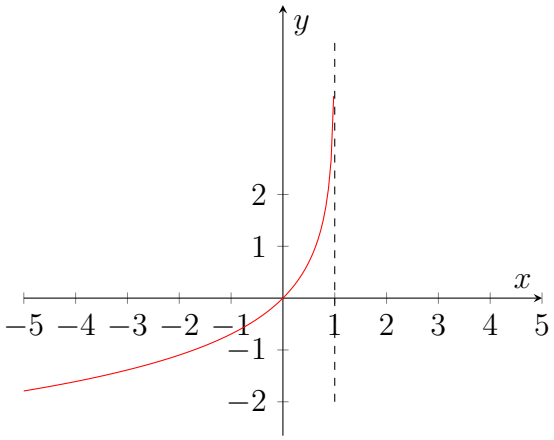
Ό λόγος που θέλουμε να έχουμε απόλυτη τιμή ή τετράγωνο είναι διότι μπορεί να υπάρχει και αρνητικό σφάλμα, κάτι το οποίο θα οδηγούσε τα σφάλματα στο να αναιρούν το ένα το άλλο (τα θετικά και αρνητικά) και στο τέλος θα παίρναμε ένα σφάλμα το οποίο δεν θα ήταν αντιπροσωπευτικό για το δίκτυο. Επίσης το MSE πολλές φορές προτιμάται λόγω του ότι είναι παραγωγίσιμη συνάρτηση.

Η επόμενη συνάρτηση κόστους που θα δούμε χρησιμοποιείται σε προβλήματα δυαδικής ταξινόμησης και λέγεται binary cross entropy:

$$C = -[p\log(y) + (1-p)\log(1-y)]$$

Για να καταλάβουμε πώς μπορεί να αντιπροσωπεύει το κόστος αυτή η συνάρτηση θα δείξουμε κάποια παραδειγματα. Αρχικά το p που είναι η πραγματική τιμή σε ένα πρόβλημα δυαδικής ταξινόμησης μπορεί να πάρει μόνο τις τιμές 0 ή 1. Αν λοιπόν είναι 0 τότε έχουμε:

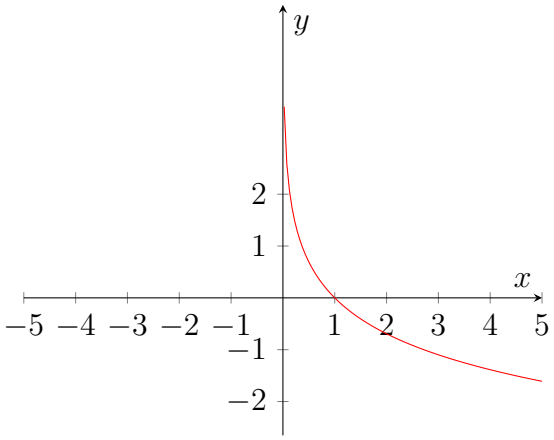
$$C = -\log(1-y)$$



Σχήμα 30: binary error for $p = 0$

Μπορούμε από τη γραφική παράσταση να δούμε ότι αν το x γίνει 0 (όσο και η πραγματική τιμή) τότε το σφάλμα γίνεται και αυτό 0 ενώ αν το x τείνει στο 1 τότε το σφάλμα τείνει στο άπειρο. Το αντιστοιχο συμβαίνει και για την περίπτωση που το p είναι 1:

$$C = -\log(y)$$



Σχήμα 31: binary error for $p = 1$

Με παρόμοιο τρόπο ορίζεται και η συνάρτηση κόστους για την ταξινόμηση πολλαπλών κλάσεων γνωστή και ως categorical crossentropy:

$$C = -\sum_{i=1}^{N} p_i \log(y_i)$$

Οι πραγματική έξοδος p σε αυτή την περίπτωση είναι ένα διάνυσμα που έχει παντου μηδενικά εκτός από μια τιμή που θα είναι 1. Αρα αν αυτή η τιμή είναι το p_k τότε έχουμε:

$$C = -\log(y_k)$$

Μπορούμε και εδώ να δούμε πως συμπεριφέρεται η συνάρτηση με τον ίδιο τρόπο που είδαμε στην δυαδική ταξινόμηση. Εδώ όμως αν παρατηρήσουμε η συνάρτηση έχει γίνει ανεξάρτητη όλων των άλλων εξόδων. Αν δηλαδή βγάζαμε μια έξοδο στην οποία όλες οι τιμές είχαν την τιμή 1, τότε το κόστος θα ήταν μηδενικό (κάτι που προφανώς δεν ισχύει). Πρέπει όμως να θυμηθούμε ότι στα προβλήματα αυτού του είδους η συνάρτηση ενεργοποίησης της εξόδου είναι η συνάρτηση softmax. Αυτό σημαίνει ότι μια τέτοια περίπτωση δεν μπορεί να συμβεί ποτέ. Βλέπουμε λοιπόν ότι μπορει να υπάρχει μια συνεργασία ανάμεσα στις ενεργοποιήσεις εξόδου και στις συναρτήσεις κόστους.

Τελευταίο είναι το πρόβλημα ταξινόμησης πολλαπλών ετικετών. Για αυτόν τον τύπο προβλήματος η συνάρτηση κόστους είναι ένας συνδυασμός των δύο προηγούμενων:

$$C = -\left[\sum_{i=1}^{N} p_i \log(y_i) + \sum_{i=1}^{N} (1 - p_i) \log(1 - y_i)\right]$$

Σε τέτοια προβλήματα χρησιμοποιούμε τη σιγμοειδή συνάρτηση σαν ενεργοποίηση εξόδου. Αυτό σημαίνει ότι δεν μπορούμε να έχουμε τις ίδιες ιδιότητες που είχε η συνάρτηση softmax οπότε πρέπει να έχουμε μια συνάρτηση κόστους που να λαμβάνει υπόψη της τις τιμές όλων των νευρώνων. Εδώ η πραγματική έξοδος μπορεί να έχει την τιμή 1 σε πάνω από μία θέσεις αλλά για λόγους απλότητας θα εξετάσουμε και πάλι το παράδειγμα που μόνο το p_k είναι 1:

$$C = -\left[\log(y_k) + \sum_{i=1, i \neq k}^{N} \log(1 - y_i)\right]$$

Έτσι μπορούμε να δούμε ότι αν κάποια τιμή που θα έπρεπε να είναι 0 τείνει στο 1 η το αντίστροφο το κόστος θα τείνει στο άπειρο ενώ αν οι τιμές είναι ίσες με τις πραγματικές τότε το κόστος είναι 0.

Το κόστος είναι επίσης ένας χρήσιμος παράγοντας για τις συνθήκες τερματισμού. Ως συνθήκη τερματισμού μπορούμε να ορίσουμε μια τιμή για το συνολικό σφάλμα, όπου άμα πέσει κάτω από αυτή την τιμή να τερματιστεί η εκπαίδευση. Μια άλλη συνθήκη θα μπορούσε να είναι ένας αριθμός εποχών ή θα μπορούσαμε να παρακολουθούμε το σφάλμα και σε περίπτωση που δεν βελτιωθεί για έναν ορισμένο αριθμό εποχών να τερματίσουμε πρόωρα την μάθηση.

6 Υλοποίηση

Όπως έχουμε ήδη αναφέρει, για την υλοποίηση αλγορίθμων θα χρησιμοποιήσουμε την γλώσσα C++ και σκοπός μας είναι τα μοντέλα που θα δημιουργήσουμε να είναι εύχρηστα και επαναχρησιμοποιήσιμα και από άλλους χρήστες. Θα χρειαστεί επομένως να δημιουργήσουμε κάποια εργαλεία με γενική χρήση. Στα πλαίσια αυτής της εργασίας υλοποιήθηκαν και εργαλεία τα οποία θα χρησιμοποιήσουμε αλλά δεν θα τα αναλύσουμε σε βάθος καθώς βγαίνουν εκτός του θέματος της επιστήμης των δεδομένων. Αναφορικά αυτά τα εργαλεία είναι:

CSV Parser Υλοποιήσαμε κάποιες συναρτήσεις για να μπορούμε να εισάγουμε στο πρόγραμμα μας αρχεία csv (comma seperated values) που είναι μια πολύ συχνή μορφή δεδομένων που μάλιστα μπορεί να παραχθεί και από το excel

Progress Bar Επειδή τα νευρωνικά δίκτυα και γενικότερα οι αλγόριθμοι μηχανικής μάθησης παίρνουν πολύ χρόνο υλοποιήσαμε μια γραμμή προόδου ώστε να μπορεί ο χρήστης ανά πάσα στιγμή να βλέπει σε ποιο στάδιο είναι το πρόγραμμα του και αν αυτό τρέχει σε φυσιολογικούς χρόνους.

Tensor Ίσως από τις πιο σημαντικές υλοποιήσεις μας και καρδία πολλών αλγορίθμων μηχανικής μάθησης. Ο τένσορας είναι μια δομή όπως ένας πίνακας με τη διαφορά ότι μπορεί πολύ περισσότερες διαστάσεις. Στην υλοποίηση αυτή δημιουργήθηκε η δομή του τένσορα, όλες οι βασικές πράξεις και συναρτήσεις για αυτόν και επιπλέον έχει προστεθεί κώδικας για την παραλληλοποίηση του με αποτέλεσμα να είναι πολύ πιο γρήγορες οι πράξεις. Παρόλο που είναι ένα πολύ σημαντικό κομμάτι για τη μηχανική μάθηση δεν θα αναλυθεί παραπάνω καθώς η δομή του είναι μεγάλη και πολύπλοκη.

Η τελευταία υλοποίησή μας η οποία είναι η πιο σημαντική και την οποία θα αναλύσουμε στη συνέχεια είναι αυτή του νευρωνικού δικτύου. Σε αυτή την υλοποίηση δημιουργήσαμε τα βασικά επίπεδα που χρησιμοποιούνται στα νευρωνικά δίκτυα και διάφορες συναρτήσεις υπολογισμού κόστους για διαφορετικούς τύπους προβλήματος. Επιπλέον δημιουργήσαμε τη δομή του δικτύου το οποίο μπορεί αποτελείται αποτελείται από πολλά επίπεδα και τη δομή του μοντέλου στην οποία μπορούμε να εισάγουμε ένα δίκτυο που θα φτιάξουμε, τα δεδομένα που θα εκπαιδεύσουμε και επιπλέον μπορούμε να δηλώσουμε και τις παραμέτρους εκπαίδευσης.

6.1 Επίπεδα

Τα δύο επίπεδα που επιλέξαμε να υλοποιήσουμε είναι το dense layer (fully connected) και το activation layer. Πριν όμως υλοποιήσουμε τα επίπεδα διεπαφή (interface). Η διεπαφή αυτή είναι το επίπεδο και ο κώδικας φαίνεται παρακάτω.

```
1 class Layer {
2 public:
3     virtual ~Layer () {}
4     virtual void forward (Tensor<double>* propagator) = 0;
5     virtual void backwards (Tensor<double>* propagator) = 0;
6     virtual void predict (Tensor<double>* propagator) = 0;
7     virtual void fix (double learning_rate) = 0;
8 };
```

Listing 1: Layer Interface

Από τον παραπάνω κώδικά πρέπει να καταλάβουμε κάποια βασικά πράγματα. Αυτή η κλάση θα κληρονομηθεί από όλα τα επίπεδα που θα φτιάξουμε. Έχουμε ορίσει συγκεκριμένες συναρτήσεις και τις έχουμε αναθέσει σε μηδέν. Αυτό σημαίνει ότι κάθε κλάση που θα την κληρονομήσει επιβάλλεται να υλοποιήσει αυτές τις συναρτήσεις (αλλιώς θα υπάρχει error). Αυτό θα μας βοηθήσει στην αντιμετώπιση προβλημάτων αλλά δίνει τη δυνατότητα σε έναν χρήστη να φτιάξει τα δικά του επίπεδα και να έχουν απευθείας πλήρη συμβατότητα με το υπόλοιπο πρόγραμμα. Σε αυτό συμβάλει σημαντικά η λέξη virtual που βρίσκεται στην αρχή κάθε συνάρτησης αλλά θα εξηγήσουμε τη σημασία της παρακάτω.

Οι συναρτήσεις των οποίων επιβάλλεται η υλοποίηση είναι:

forward Δέχεται έναν τένσορα σαν είσοδο και τον επεξεργάζεται (τον μετατρέπει ουσιαστικά στην έξοδο για να μπει σαν είσοδος στο επόμενο επίπεδο). Αυτή η συνάρτηση χρησιμοποιείται κατά την εκπαίδευση δώσουμε μια είσοδο στο δίκτυο μας και να υπολογίσει την έξοδο.

backwords Ουσιαστικά έχει ίδια χρήση με την forward αλλά ενώ η forward διαδίδει την είσοδο στην έξοδο η backwords διαδίδει το σφάλμα προς την αντίθετη κατεύθυνση (Back Propagation).

predict είναι ακριβώς ίδια στην υλοποίηση της με την forward, μόνο που παραλείπει κάποιες διεργασίες της που δεν είναι απαραίτητες. Χρησιμοποιείται στο στάδιο του validation.

fix Είναι ο τρόπος με τον οποίο μαθαίνει το επίπεδο (και κατα συνέπεια και το δίκτυο) διορθώνοντας κάποιες παραμέτρους (βάρη) του επιπέδου.

Έχοντας κατανοήσει τα παραπάνω μπορούμε να φτιάξουμε το πρώτο μας επίπεδο (dense):

```
class Dense : public Layer {
private:
    Tensor<double>* weights;
    Tensor<double>* biases;
    Tensor<double>* inputs;
    Tensor<double>* errors;
public:
    Dense (size_t input_size, size_t output_size);
    ~Dense ();
    void forward (Tensor<double>* propagator) override;
    void backwards (Tensor<double>* propagator) override;
    void predict (Tensor<double>* propagator) override;
    void fix (double learning_rate) override;
};
```

Listing 2: Dense layer in hpp file

Εδώ βλέπουμε τις επιπλέον παραμέτρους που χρειάζονται για αυτό το επίπεδο. Βλέπουμε τέσσερις τένσορες. Οι δύο από αυτους χρησιμοποιούνται για τον υπολογισμό της εξόδου (weigts και biases), ενώ οι άλλοι δύο χρησιμοποιούνται για την αποθήκευση των εισόδων και των σφαλμάτων. Μπορούμε να δούμε την υλοποίηση των συναρτήσεων παρακάτω:

```
Dense::Dense (size_t input_size, size_t output_size) {
    weights = new Tensor<double>(input_size, output_size);
    biases  = new Tensor<double>((size_t)1, output_size);
    inputs  = new Tensor<double>;
    errors  = new Tensor<double>;

    for(size_t i = 0; i < input_size; i++) {
            for( size_t j = 0; j < output_size; j++) {
                (*weights)(i, j) =
                    rand()/(double)(RAND_MAX) - 0.5;
            }
        }
        for(size_t i = 0; i < output_size; i++) {
            (*biases)((size_t)0, i) = 0;
        }
}
Dense::~Dense () {
    delete weights;
    delete biases;
```

```
20      delete inputs;
21      delete errors;
22  }
23  void Dense::forward (Tensor<double>* propagator) {
24      *inputs = *propagator;
25      *propagator = mul(*propagator, *weights) + *biases;
26  }
27  void Dense::backwards (Tensor<double>* propagator) {
28      *errors = *propagator;
29      *propagator = mul(*propagator, ~(*weights));
30  }
31  void Dense::predict (Tensor<double>* propagator) {
32      *propagator = mul(*propagator, *weights) + *biases;
33  }
34  void Dense::fix (double learning_rate) {
35      *weights = *weights - mul(~(*inputs), *errors)*
36          learning_rate/(double)errors->dims()[0];
37      *biases = *biases - (*errors)[0]*
38          learning_rate/(double)errors->dims()[0];
39  }
```

Listing 3: Dense layer in cpp file

Οι συναρτήσεις είναι οι εξής:

Desne (constructor) Δημιουργεί και αρχικοποιεί το επίπεδο. Συγκεκριμένα δέχεται το μέγεθος εισόδου και εξόδου και αρχικοποιεί με τα ανάλογα μεγέθη τους τους τένσορες για τα weights (αρχικοποιούνται τα στοιχεία του με τυχαίους αριθμους από -0.5 ως 0.5) και biases (αρχικοποιούνται σε μηδέν).

~ Desne (destructor) Απελευθερώνει τη μνήμη που δέσμευαν οι τένσορες.

forward Αρχικά σώσει την είσοδο (θα χρειαστεί σε επόμενο στάδιο). Έπειτα κάνει πολλαπλασιασμό πινάκων μεταξύ εισόδου και βαρών και μετά προσθέτει στο αποτέλεσμα τα biases.

backwards Αρχικά σώζει το σφάλμα και έπειτα υπολογίζει το σφάλμα του προηγούμενου επιπέδου. Για να το κάνει αυτό εκτελεί πολλαπλασιασμό πινάκων μεταξύ των σφαλμάτων του και τον **αντίστροφο** των βαρών.

predict Κάνει την ίδια δουλειά με το forward αλλά δεν χρειάζεται να αποθηκεύσει την είσοδο.

fix Διορθώνει τα βάρη και τα biases με τον τρόπο που έχουμε ήδη δει.

Τώρα θα υλοποιήσουμε το activation layer.

```
class Activation : public Layer {
private:
    Tensor<double>* inputs;
    std::function<void(Tensor<double>*)> func;
    std::function<void(Tensor<double>*)> der;
public:
    Activation (
        size_t input_size,
        std::function<void(Tensor<double>*)> func,
        std::function<void(Tensor<double>*)> der);

    ~Activation ();
    void forward (Tensor<double>* propagator) override;
    void backwards (Tensor<double>* propagator) override;
    void predict (Tensor<double>* propagator) override;
    void fix (double learning_rate) override;
};
```

Listing 4: Activation layer in hpp file

Παρατηρούμε ότι αυτό το επίπεδο δεν έχει βάρη αλλά έχει δύο συναρτήσεις τις οποίες μπορούμε να δηλώσουμε εμείς κατα τη δημιουργία αυτού του επιπέδου. Συγκεκριμένα αρχικοποιούμε ένα τέτοιο επίπεδο δίνοντας το μέγεθος εισόδου, μια συνάρτηση ενεργοποίησης και την παράγωγό της. Αυτή η υλοποίηση επιτρέπει σε οποιονδήποτε χρήστη να φτιάξει τις δικές του συναρτήσεις ενεργοποίησης και να μπορεί να τις χρησιμοποιήσει απευθείας σε συνδυασμό με τη βιβλιοθήκη μας. Η υλοποίηση των συναρτήσεων για αυτό το επίπεδο είναι πολύ πιο απλές.

```
Activation::Activation (
    size_t input_size,
    std::function<void(Tensor<double>*)> func,
    std::function<void(Tensor<double>*)> der) {

    inputs  = new Tensor<double>;
    this->func = func;
    this->der = der;
}
Activation::~Activation () {
    func = nullptr;
    der = nullptr;
    delete inputs;
}
void Activation::forward (Tensor<double>* propagator) {
```

```
16      *inputs = *propagator;
17      func(propagator);
18  }
19  void Activation::backwards (Tensor<double>* propagator) {
20      der(inputs);
21      *propagator = (*propagator) * (*inputs);
22  }
23  void Activation::predict (Tensor<double>* propagator) {
24      func(propagator);
25  }
26  void Activation::fix (double learning_rate) {
27
28  }
```

Listing 5: Activation layer in cpp file

Όπως φαίνεται και παραπάνω η συνάρτηση forward αποθηκεύει την είσοδο και εφαρμόζει τη συνάρτηση ενεργοποίησης στην είσοδο, ενώ η συνάρτηση backwards πολλαπλασιάζει το σφάλμα με την παράγωγο την αποθηκευμένης εισόδου. Τα επίπεδα ενεργοποίησης δεν μαθαίνουν κάτι αφού δεν έχουν βάρη άρα μπορούμε να αφήσουμε τη συνάρτηση κενή (αλλά πρέπει να την υλοποιήσουμε). Έχουμε επίσης υλοποιήσει τις πιο γνωστές συναρτήσεις ενεργοποίησης οι οποίες είναι:

1. sigmoid
2. tanh
3. relU
4. linear
5. softmax

```
1   void act::sigmoid (Tensor<double>* t) {
2       *t = 1/(1 + exp(-(*t)));
3   }
4   void act::tanh (Tensor<double>* t) {
5       *t = tanh(*t);
6   }
7   void act::relu (Tensor<double>* t) {
8       *t = max(*t, 0);
9   }
10  void act::softmax (Tensor<double>* t) {
11      Tensor<double> sums = exp(*t)[1];
12      *t = exp(*t)/sums;
```

```
13 }
14 void act::linear (Tensor<double>* t) {
15
16 }
17 void der::sigmoid (Tensor<double>* t) {
18     *t = exp(-(*t))/((1 + exp(-(*t)))*(1 + exp(-(*t))));
19 }
20 void der::tanh (Tensor<double>* t) {
21     *t = 1 - tanh(*t)*tanh(*t);
22 }
23 void der::relu (Tensor<double>* t) {
24     *t = (*t > 0);
25 }
26 void der::softmax (Tensor<double>* t) {
27
28 }
29 void der::linear (Tensor<double>* t) {
30
31 }
```

Listing 6: Activation functions

Εδώ παρατηρούμε ότι οι παράγωγοι για των linear και softmax δεν έχουν υλοποιηθεί αλλά αυτό δεν μας επηρεάζει διότι αυτές οι ενεργοποιήσεις βρίσκονται μόνο στο επίπεδο εξόδου και αν κάνουμε καποιες μαθηματικές απλοποιήσεις προκύπτει ότι δεν είναι απαραίτητη η οπισθοδρόμηση από αυτό το επίπεδο αλλά μπορούμε να περάσουμε απευθείας στο προηγούμενο.

6.2 Δίκτυο

Έχουμε λοιπόν υλοποιήσει τα δύο απαραίτητα επίπεδα για κάθε νευρωνικό δίκτυο μπορούμε να υλοποιήσουμε την κλάση του δικτύου.

```
1  class Network : public Layer {
2  private:
3      std::vector<Layer*> layers;
4  public:
5      Network ();
6      ~Network ();
7      void add (Layer* l);
8      void forward (Tensor<double>* propagator) override;
9      void backwards (Tensor<double>* propagator) override;
10     void predict (Tensor<double>* propagator) override;
11     void fix (double learning_rate) override;
```

```
12 };
```

Listing 7: Network layer in hpp file

Βλέπουμε ότι το δίκτυο περιέχει μία λίστα από επίπεδα. Επιπλέον έχει όλες τις συναρτήσεις που είχαν και τα επίπεδα. Στη συνάρτηση forward διατρέχουμε για κάθε επίπεδο με τη σειρά και εκτελούμε τη συνάρτηση forward ενώ στη συνάρτηση όπως κάνουμε και στις predict και fix ενώ στην backwards διατρέχουμε τα επίπεδα με την ανάποδη σειρά. Επιπλέον έχουμε και μια συνάρτηση για προσθέτουμε επίπεδα στο δίκτυο μας. Η υλοποίηση φαίνεται παρακάτω.

```
1  Network::Network () {
2      layers = std::vector<Layer*>{};
3  }
4  Network::~Network () {
5      for (auto layer : layers) delete layer;
6      }
7  void Network::add (Layer* l) {
8      layers.push_back(l);
9  }
10 void Network::forward (Tensor<double>* propagator) {
11     for (auto layer : layers) layer->forward(propagator);
12 }
13 void Network::backwards (Tensor<double>* propagator) {
14     for(size_t i = 1; i < layers.size(); i++)
15         layers[layers.size() - i - 1]->backwards(propagator);
16 }
17 void Network::predict (Tensor<double>* propagator) {
18     for (auto layer : layers) layer->predict(propagator);
19 }
20 void Network::fix (double learning_rate) {
21     for (auto layer : layers) layer->fix(learning_rate);
22 }
```

Listing 8: Network layer in cpp file

Έδω είναι σημαντικό να εξηγήσουμε τη λέξη virtual που χρησιμοποιήσαμε στη διεπαφή μας. Στην υλοποίηση του δικτύου μας έχουμε λίστα από επίπεδα. Επειδή όμως δεν η λίστα δεν περιέχει τα επίπεδα αλλά τους δείκτες σε αυτά μπορούμε να αρχικοποιήσουμε ένα επίπεδο (που είναι μια αδεια κλάση) με τον constructor της κλάσης Dense γράφοντας `Layer*l = new Dense(10, 5);`. Το ερώτημα είναι όταν καλέσουμε τη συνάρτηση `l.forward(some tensor);` θα καλέσουμε τη συνάρτηση forward της κλάσης Layer ή της Dense; Επειδή λοιπόν έχουμε προσθέσει τη λέξη virtual μπροστά από τις συναρτήσεις της διεπαφής θα κληθεί η συνάρτηση του Dense ενώ διαφορετικά θα γινόταν το ανάποδο. Αυτό μας δίνει μεγάλη

ευκολία στο να προσθέσουμε διαφορετικά επίπεδα στο δίκτυο με την ίδια συνάρτηση και να τα έχουμε όλα σε μία λίστα. Επιπλέον ένας χρήστης όπως αναφέραμε μπορεί να φτιάξει δικά του επίπεδα και να τα χρησιμοποιήσει με απόλυτη συμβατότητα αρκεί να κληρονομούν την κλάση Layer.

6.3 Μοντέλο

Η τελευταία κλάση που θα δούμε είναι η κλάση model. Αυτή η κλάση περιέχει ένα δίκτυο και έχει μια συνάρτηση για να το εκπαιδεύει. Επιπλέον αυτή η κλάση κρατάει και τις παραμέτρους εκπαίδευσης του δικτύου που είναι:

batch_size Πόσα δεδομένα στέλνουμε μαζί (batch) σε ένα iteration (δηλαδή πριν κάνουμε διόρθωση βαρών)

iterations Πόσα batches δεδομένων θα στείλουμε σε ένα epoch (δηλαδή πριν κάνουμε validation)

epochs Πόσα epochs θα διαρκέσει η εκπαίδευση

learning_rate Πόσο γρήγορα μαθαίνει το δίκτυο

validation_split Το ποσοστό των δεδομένων που θα χρησιμοποιηθούν για εκπαίδευση (Τα υπόλοιπα πάνε στο validation)

Επειδή η συνάρτηση που εκπαιδεύει το δίκτυο είναι πολύ μεγάλη δεν θα δείξουμε τον κώδικα αλλά οι παράμετροι εξηγούν τη λειτουργία αρκετά καλά και το μόνο που έχουμε να κάνουμε είναι να χρησιμοποιήσουμε επαναληπτικά, με βάση αυτές τις παραμέτρους, τις συναρτήσεις του δικτύου (forward, backwards, predict, fix).

Εδώ είναι επίσης το σημείο που πρέπει να φτιάξουμε τις συναρτήσεις που υπολογίζουν το σφάλμα για κάθε περίπτωση προβλήματος. Οι περιπτώσεις προβλήματος είναι:

1. Παλινδρόμηση (regression)
2. Δυαδική ταξινόμηση (binary classification)
3. Κατηγορηματική ταξινόμηση (categorical classification)
4. Πολυταξική ταξινόμηση (multy class classification)

```cpp
double err::regression (Tensor<double>* p, const Tensor<double>& e) {
    *p = *p - e;
    double cost = (abs(*p)[0][1]/(double)(p->dims()[0])).vec()[0];
    return cost;
}
double err::binary (Tensor<double>* p, const Tensor<double>& e) {
    Tensor<double> loss =
         -(e*log(*p) + ((double)1 - e)*log((double)1 - *p));
    *p = *p - e;
    double cost = (loss[0]/(double)(loss.dims()[0])).vec()[0];
    return cost;
}
double err::categorical (Tensor<double>* p, const Tensor<double>& e) {
    Tensor<double> loss = (-e*log(*p))[1];
    *p = *p - e;
    double cost = (loss[0]/(double)(loss.dims()[0])).vec()[0];
    return cost;
}
double err::multyclass (Tensor<double>* p, const Tensor<double>& e) {
    Tensor<double> loss =
        (-(e*log(*p) + ((double)1 - e)*log((double)1 - *p)))[1];
    *p = *p - e;
    double cost = (loss[0]/(double)(loss.dims()[0])).vec()[0];
    return cost;
}
```

Listing 9: Network layer in cpp file

Έτσι μπορούμε να υπολογίσουμε το σφάλμα για κάθε είδος προβλήματος. Αυτές ήταν οι πιο σημαντικές υλοποιήσεις που έγιναν στα πλαίσια αυτής της εργασίας.

7 Συμπέρασμα

Σε αυτή την εργασία είδαμε τα διαφορετικά προβλήματα που λύνει η επιστήμη των δεδομένων (ταξινόμηση, παλινδρόμηση και ομαδοποίηση). Επιπλέον είδαμε πολλές διαφορετικές μεθόδους που μπορούμε να προσεγγίσουμε κάθε ένα από αυτά τα προβλήματα. Είδαμε μεθόδους οι οποίες οποίες απαιτούσαν από τον χρήστη να ορίσει κάποιες παραμέτρους ή κάποιες που ήταν χρήσιμες μόνο σε συγκεκριμένες περιπτώσεις.

Τελικά καταλήξαμε ότι κυρίαρχος τρόπος να λυθούν αυτά τα προβλήματα είναι τα νευρωνικά δίκτυα. Τα νευρωνικά δίκτυα είναι μία πολύ γενικευμένη μέθοδος και υπάρχουν πολλά είδη για κάθε είδος προβλήματος που αναλύσαμε και πολύ περισσότερα ακόμα. Τα νευρωνικά δίκτυα έχουν την ικανότητα μάθησης το οποία τα καθιστά πολύ βολικά για κάθε πρόβλημα.

Αναφορές

[1] Data science. https://en.wikipedia.org/wiki/Data_science. Accessed: 2023-03-13.

[2] Rohit Garg. 7 types of classification algorithms. https://analyticsindiamag.com/7-types-classification-algorithms/. Accessed: 2023-03-15.

[3] Machine learning classification - 8 algorithms for data science aspirants. https://data-flair.training/blogs/machine-learning-classification-algorithms/. Accessed: 2023-03-15.

[4] Ekin Keserer. 8 types of machine learning classification algorithms. https://www.akkio.com/post/5-types-of-machine-learning-classification-algorithms. Accessed: 2023-03-15.

[5] Webb, Geoffrey I and Keogh, Eamonn and Miikkulainen, Risto. Naïve Bayes. *Encyclopedia of machine learning*, 15:713–714, 2010.

[6] Pavan Vadapalli. Naive Bayes explained: Function, Advantages and disadvantages applications in 2023. https://www.upgrad.com/blog/naive-bayes-explained/. Accessed: 2023-03-18.

[7] Rish, Irina and others. An empirical study of the naive Bayes classifier. 3(22):41–46, 2001.

[8] Jakkula, Vikramaditya. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006.

[9] Dhiraj K. Top 4 advantages and disadvantages of Support Vector Machine or SVM. https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107. Accessed: 2023-03-18.

[10] Real-Life Applications of SVM (Support Vector Machines). https://data-flair.training/blogs/applications-of-svm/. Accessed: 2023-03-22.

[11] What is the k-nearest neighbors algorithm? https://www.ibm.com/topics/knn. Accessed: 2023-03-24.

[12] Random Forest. https://en.wikipedia.org/wiki/Random_forest. Accessed: 2023-04-08.

[13] Advantages and disadvantages of decision tree in machine learning. https://www.analytixlabs.co.in/blog/decision-tree-algorithm. Accessed: 2023-04-08.

[14] Cluster analysis. https://en.wikipedia.org/wiki/Cluster_analysis. Accessed: 2023-04-10.

[15] k-Means Advantages and Disadvantages Machine Learning Google Developers. https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages. Accessed: 2023-04-16.

[16] How Density-based Clustering works. https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/how-density-based-clustering-works.htm. Accessed: 2023-04-19.

[17] DBSCAN. https://en.wikipedia.org/wiki/DBSCAN. Accessed: 2023-04-19.

[18] Density-Based Spatial Clustering of Applications with Noise (DBSCAN). https://ml-explained.com/blog/dbscan-explained. Accessed: 2023-04-19.

[19] Hierarchical clustering. https://en.wikipedia.org/wiki/Hierarchical_clustering, 2023. Accessed: 2023-04-22.

[20] Different types of Clustering in Machine Learning. https://vitalflux.com/different-types-clustering-algorithm-machine-learning/, 2022. Accessed: 2023-04-22.

[21] What Is a Neural Network? https://www.investopedia.com/terms/n/neuralnetwork.asp, 2023. Accessed: 2023-05-06.

[22] 25 Must Know Terms & concepts for Beginners in Deep Learning. https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/, 2023. Accessed: 2023-05-06.

[23] Using Activation Functions in Neural Networks. https://machinelearningmastery.com/using-activation-functions-in-neural-networks/, 2022. Accessed: 2023-05-07.

Α΄ Τύποι για QDA, LDA

Στον αλγόριθμο QDA ο παράγοντας $P(X|y)$ δίνεται από τον τύπο:

$$P(X|y=k) = \frac{1}{(2\pi)^{\frac{n}{2}} \left|\sum_k\right|^{\frac{1}{2}}} e^{-\frac{1}{2}(X-M_k)^T \sum_k^{-1} (X-M_k)}$$

όπου:

k η κλάση που εξετάζουμε

X το διάνυσμα χαρακτηριστικών του δείγματος

M_k το κέντρο των σημείων της κλάσης k

n η διάσταση του X

$\sum_k$ ο πίνακας συνδιακύμανσης

και

$$X = \langle x_1, x_2, \ldots, x_n \rangle$$

αν τα σημεία που ανήκουν στην κλάση k είναι $K_1, K_2, \ldots, K_m$ με:

$$K_i = \langle k_{i_1}, k_{i_2}, \ldots, k_{i_n} \rangle$$

τότε το M για αυτή την κλάση θα είναι:

$$M = \langle m_1, m_2, \ldots, m_n \rangle = \left\langle \frac{1}{m}\sum_{i=1}^{m} k_{i_1}, \frac{1}{m}\sum_{i=1}^{m} k_{i_2}, \ldots, \frac{1}{m}\sum_{i=1}^{m} k_{i_n} \right\rangle$$

Η διακύμανση είναι μας δείχνει τη διασπορά των σημείων της κλάσης από το κέντρο της κλάσης για μια συγκεκριμένη διάσταση και υπολογίζεται για κάθε διάσταση ξεχωριστά. Η διακύμανση για μια διάσταση u από τις n διαστάσεις θα είναι:

$$V_u = \frac{1}{m}\sum_{i=1}^{m} \left(k_{i_u} - m_u\right)^2$$

Πολύ παρόμοια είναι και η συνδιακύμανση η οποία όμως υπολογίζεται για δύο διαστάσεις u και z:

$$C_{uz} = \frac{1}{m}\sum_{i=1}^{m} \left(k_{i_u} - m_u\right)\left(k_{i_z} - m_z\right)$$

Ο πίνακας συνδιακύμανσης είναι ένας συμμετρικός πίνακας με διαστάσεις $n \times n$ και είναι:

$$\sum = \begin{bmatrix} V_1 & C_{12} & \dots & C_{1n} \\ C_{12} & V_2 & \dots & C_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1n} & C_{2n} & \dots & V_n \end{bmatrix}$$

Ο αλγόριθμος LDA είναι μια απλοποίηση του QDA όπου θεωρούμε ότι όλες οι κλάσεις έχουν τον ίδιο πίνακα συνδιακύμανσης το οποίο απλοποιεί πάρα πολύ τις πράξεις.