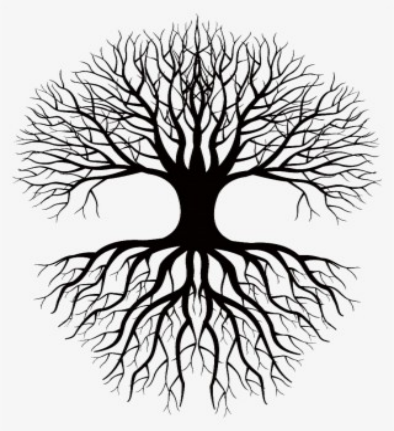
ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Βαθμονόμηση Αδρονικών Πιδάκων από τη Διάσπαση του Τόπ-Κουάρκ με χρήση Μηχανικής Μάθησης



ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΘΕΟΛΟΓΗ ΝΙΚΟΛΑΟΥ

ΕΠΙΒΛΕΠΩΝ:

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΟΥΣΟΥΡΗΣ
Αναπληρωτής Καθηγητής ΣΕΜΦΕ

ΑΘΗΝΑ, ΙΟΥΛΙΟΣ 2022

Abstract

The subject of this thesis is the study of the supervised machine learning regression algorithms used in the field of elementary particle physics. In particular, the algorithms which were studied and applied were the linear discriminant, the multilayer perceptron neural networks and the boosted descision trees. The data that was used to both train and test the algorithms is related to high-energy physics, a sector which requires big data analysis. More specifically, the data came from Monte Carlo simulations of proton-proton collisions inside the detectors of the European Organization for Nuclear Research (CERN). The analysis was made both theoretically and practically which is the way the chapters of this thesis are structured. Theoretically by studying the underlying mathematics of the regression models that were used and practically by programming these methods with the TMVA software of ROOT. The code written for the study of the methodes in the pyROOT format is given in the annexes at the end.

Περίληψη

Στην παρούσα διπλωματική εργασία γίνεται μελέτη των αλγορίθμων παλινδρόμησης μηχανικής μάθησης με επίβλεψη στον τομέα της φυσικής των στοιχειωδών σωματιδίων. Συγκεκριμένα οι αλγόριθμοι που μελετήθηκαν και εφαρμόστηκαν είναι: ο απλός γραμμικός ταξινομητής, τα τεχνητά νευρωνικά δίκτυα και τα ενδυναμωμένα δέντρα απόφασης. Τα δεδομένα που χρησιμοποιήθηκαν τόσο για την εκπαίδευση, όσο και για τον έλεγχο των μεθόδων σχετίζονται με την φυσική υψηλών ενεργειών, αντικείμενο το οποίο απαιτεί ανάλυση μεγάλου όγκου δεδομένων. Ειδικότερα τα δεδομένα προήλθαν από Monte Carlo προσομοίωση συγκρούσεων πρωτονίου-πρωτονίου εντός των ανιχνευτών του Ευρωπαικού Κέντρου Πυρηνικών Ερευνών (CERN). Η ανάλυση έγινε τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο. Θεωρητικά μελετώντας το μαθηματικό υπόβαθρο των μοντέλων παλινδρόμησης και πρακτικά προγραμματίζοντας τις μεθόδους μέσα από το λογισμικό TMVA του ROOT. Ο κώδικας που γράφτηκε για την μελέτη των μεθόδων στην γλώσσα pyROOT δίνεται στα παραρτήματα στο τέλος.

© (2022) Εθνικό Μετσόβιο Πολυτεχνείο. All rights Reserved. Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς το συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σ’αυτό το έγγραφο εκφράζουν το συγγραφέα και δεν πρέπει να ερμηνευτεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Ευχαριστίες

Η διπλωματική αυτή εργασία πραγματοποιήθηκε το εαρινό εξάμηνο του ακαδημαικού έτους 2021-2022 στη Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω θερμά τον επιβλέπων καθηγητή Κωνσταντίνο Κουσουρή για την συνεχή βοήθεια και καθοδήγηση που μου παρείχε καθόλη την διάρκεια του εξαμήνου. Επίσης ένα μεγάλο ευχαριστώ χρωστάω στην ομάδα των διδακτορικών του φοιτητών, πάντα έτοιμοι να με κατατοπίσουν για το πως εξαφανίζονται τα πολλά Error του κώδικα. Τέλος δεν θα μπορούσα να μην ευχαριστήσω τους φίλους και τους ανθρώπους γύρω μου για τη στήριξη και τη δύναμη που μου δίνουν. Η εργασία αυτή είναι αφιερωμένη στον πατέρα, την μητέρα και τις δύο αδελφές μου που όλα αυτά τα χρόνια εξακολουθούν να με αντέχουν.

Σκοπός μας πρέπει να είναι οι σωστές ερωτήσεις και όχι οι απαντήσεις

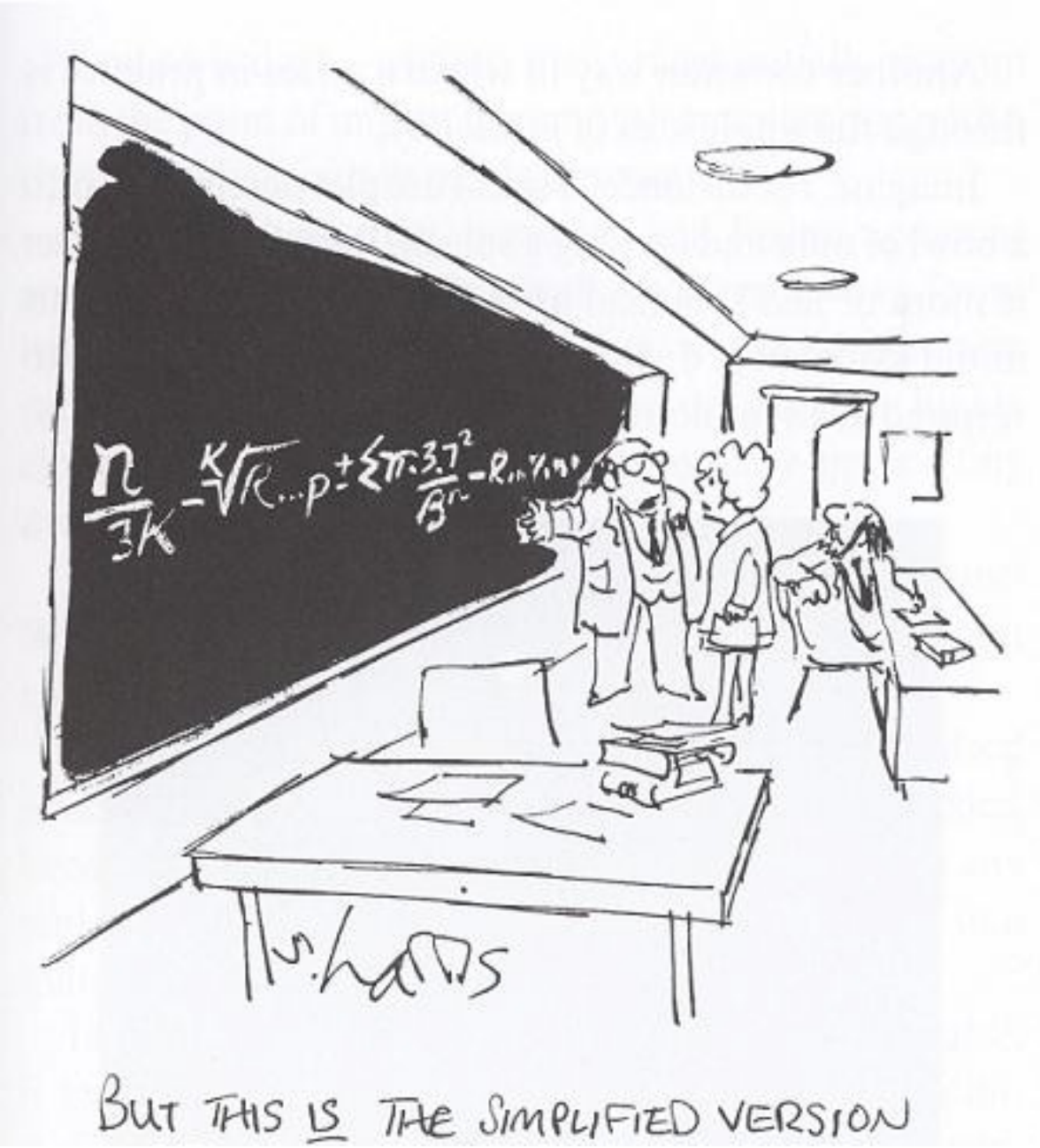
-Douglas Adams

ΠΕΡΙΕΧΟΜΕΝΑ

ΘΕΩΡΙΑ.....2
1.1 Σωματιδιακή Φυσική.....3
1.1.1 Το Καθιερωμένο Πρότυπο.....3
1.1.2 Αδρονικοί Πίδακες (Hadronic Jets).....4
1.1.3 Ο Μεγάλος Αδρονικός Επιταχυντής (Large Hadron Collider)….5
1.1.4 Βαθμονοποίηση Αδρονικών Πηδάκων (Calibration of Hadronic Jets).....6
1.1.5 Μεταβλητές Χαρακτηρισμού.....6
1.2 Μηχανική Μάθηση.....9
1.2.1 Η Έννοια της Μηχανικής Μάθησης.....9
1.2.2 Μέθοδοι Μηχανικής Μάθησης.....11
1.2.3 Το Φαινόμενο της Υπερεκπαίδευσης (Over-Training).....19
ΕΦΑΡΜΟΓΕΣ.....21
2.1 Εργαλία Ανάλυσης Δεδομένων.....22
2.2 Από τα Δεδομένα της Monte Carlo Προσομοίωσης στα Δεδομένα για την Παλινδρόμηση.....22
2.3 Υπολογισμός της Απόκρισης (Response).....24
2.4 Γραμμικός Ταξινομητής (LD).....25
2.5 Ενδυναμωμένα Δέντρα Απόφασης (BDT’s).....27
2.6 Τεχνητά Νευρωνικά Δίκτυα (NN’s).....30
2.7 Σύγκριση και των 3ων Μεθόδων (NN’s vs BDT’s vs LD).....39
2.8 Απόδοση στους Υπόχωρους (Performance in Subspace).....41
Επίλογος.....45
Παραρτήματα.....47
Παράρτημα 1.....48
Παράρτημα 2.....50
Παράρτημα 3.....51
Παράρτημα 4.....52
Παράρτημα 5.....56
Παράρτημα 6.....58
Παράρτημα 7.....64
Παράρτημα 8.....66
Παράρτημα 9.....73
Παράρτημα 10.....75
Παράρτημα 11……79
ΒΙΒΛΙΟΓΡΑΦΕΙΑ.....84

Κεφάλαιο 1

ΘΕΩΡΙΑ



1.1 Σωματιδιακή Φυσική

Η Σωματιδιακή Φυσική ή Φυσική των Στοιχειωδών Σωματιδίων είναι κλάδος της φυσικής που μελετά τα στοιχειώδη σωματίδια από τα οποία συγκροτείται η ύλη καθώς και την συμπεριφορά και τις αλληλεπιδράσεις μεταξύ τους. Λέγεται επίσης και Φυσική Υψηλών Ενεργειών (High Energy Physics), επειδή πολλά στοιχειώδη σωμάτια δεν υφίστανται υπό τις συμβατικές συνθήκες που συναντάμε στη φύση όπως τη γνωρίζουμε στον πλανήτη μας, αλλά μπορούν να δημιουργηθούν και να ανιχνευθούν μέσω ενεργειακών κρούσεων με άλλα σωματίδια, όπως γίνεται στους επιταχυντές σωματιδίων (LHC, RCS, SLAC).

1.1.1 Το Καθιερωμένο Πρότυπο

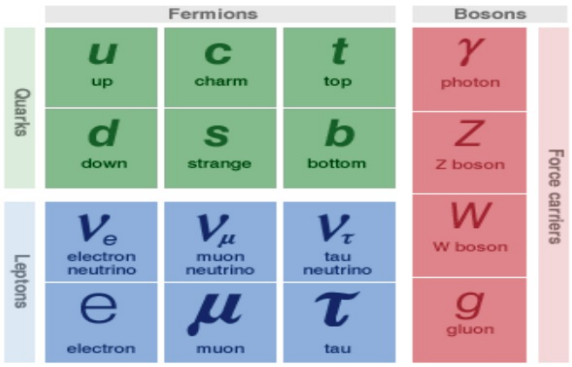
Το Καθιερωμένο Πρότυπο (Standard Model) είναι η φυσική θεωρία που περιγράφει τα δομικά συστατικά της ύλης και τις μεταξύ τους ισχυρές, ασθενείς και ηλεκτρομαγνητικές αλληλεπιδράσεις. Πρόκειται για μια πολύ καλά θεμελιωμένη θεωρία που έχει προβλέψει πολλά πειραματικά αποτελέσματα, όπως την ύπαρξη πολλών σωματιδίων και έχει ανταπεξέλθει σε πάρα πολλούς πειραματικούς ελέγχους ανά τα χρόνια. Τα βασικά κομμάτια που λείπουν από αυτη την θεωρία είναι η περιγραφή των βαρυτικών αλληλεπιδράσεων, η εξήγηση της ύπαρξης σκοτεινής ύλης-ενέργειας και η εξήγηση του φαινομένου της ταλάντωσης των νετρίνων.

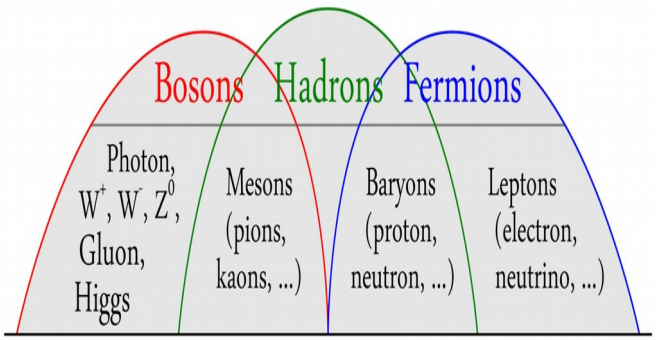
Τα στοιχειώδη σωματίδια χωρίζονται σε 2 κατηγορίες. Τα φερμιόνια που είναι οι δομικοί λίθοι της ύλης και τα μποζόνια που είναι οι φορείς των αλληλεπιδράσεων (Force Cariers) μεταξύ των στοιχειωδών σωματιδίων. Σύμφωνα με το Καθιερωμένο Πρότυπο όσον αφορά τα φερμιόνια υπάρχουν 6 κουάρκ, 6 λεπτόνια καθώς και τα αντισωματίδιά τους. Όσον αφορά τα μποζόνια υπάρχουν 3 μποζόνια (W^+, W^-, Z^0) φορείς της ασθενής αλληλεπίδρασης, το γκλουόνιο (g) ως φορέας της ισχυρής αλληλεπίδρασης και το φωτόνιο (γ) ως φορέας της ηλεκτρομαγνητικής αλληλεπίδρασης. [5]

Επιπλέον υπάρχει η κατηγορία των Αδρονίων στην οποία υπάρχουν 2 οικογένειες:

- Τα Βαρυόνια, που αποτελούνται από περιττό αριθμό κουάρκ (συνήθως τρία κουάρκ) όπως το πρωτόνιο (uud) και το νετρόνιο (udd).
- Τα Μεσόνια, που αποτελούνται από ζυγό αριθμό κουάρκ (συνήθως δύο κουάρκ ή ένα κουάρκ και ένα αντικουάρκ).

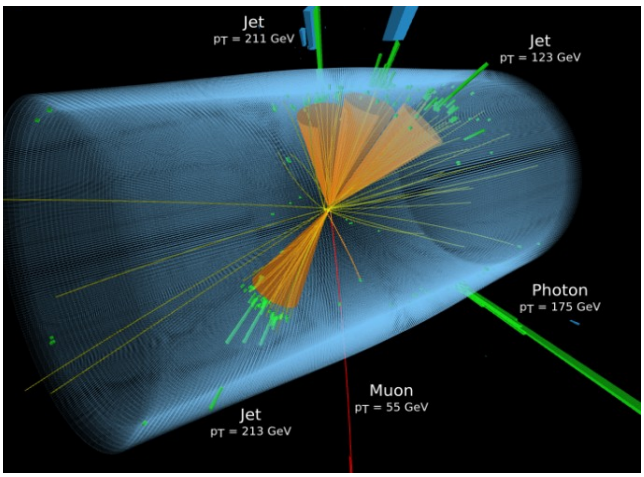
Αναλυτικά όλα τα στοιχειώδες σωματίδια μαζί με την κατηγοριοποίησή τους βρίσκονται στα παρακάτω διαγράμματα. Σχεδόν όλα τα ελεύθερα Αδρόνια και Αντιαδρόνια πιστεύεται ότι είναι ασταθή και τελικά διασπώνται σε άλλα σωματίδια. Η μόνη γνωστή πιθανή εξαίρεση είναι τα ελεύθερα πρωτόνια, τα οποία φαίνεται να είναι σταθερά, ή τουλάχιστον, χρειάζονται τεράστιο χρόνο για να διασπαστούν. [10]

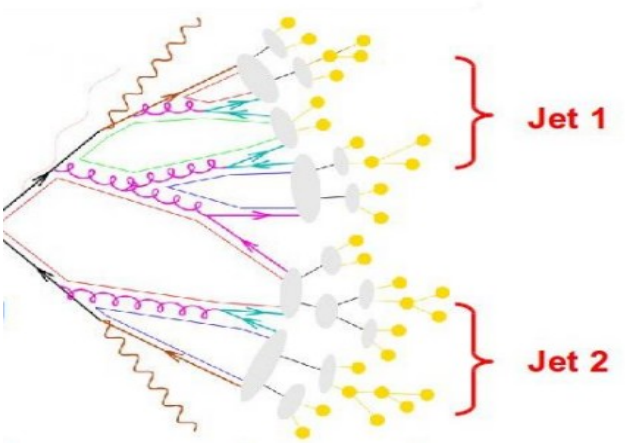




1.1.2 Αδρονικοί Πίδακες (Hadronic Jets)

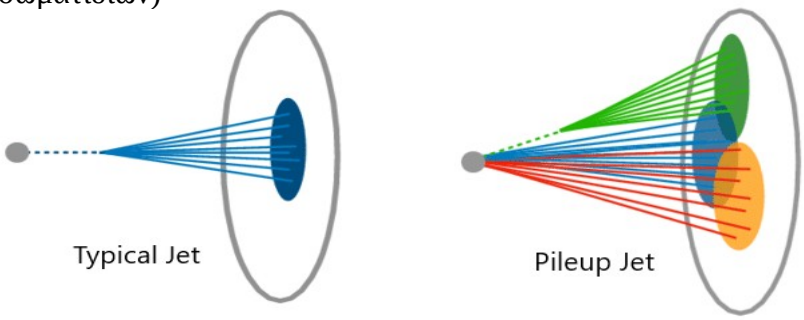
Ένας πίδακας αδρονίων είναι ένας στενός κώνος αδρονίων και άλλων σωματιδίων που παράγεται από την αδρονοποίηση ενός κουάρκ ή γκλουονίου. Τα σωματίδια που φέρουν χρωματικό φορτίο, όπως τα κουάρκ, δεν μπορούν να υπάρχουν σε ελεύθερη μορφή λόγω του περιορισμού της κβαντικής χρωμοδυναμικής (QCD) που επιτρέπει μόνο άχρωμες καταστάσεις. Όταν ένα αντικείμενο περιέχει θραύσματα χρωματικού φορτίου, κάθε θραύσμα "κουβαλάει" μαζί του μέρος του χρωματικού του φορτίου. Προκειμένου να υπακούσουν στον περιορισμό της χρωμοδυναμικής (QCD), αυτά τα θραύσματα δημιουργούν άλλα αντικείμενα με χρώμα γύρω τους για να σχηματίσουν συνολικά ένα άχρωμα αντικείμενα (αδρόνια). Το σύνολο αυτών των αντικειμένων ονομάζεται πίδακας (jet). Οι πίδακες αυτοί μετρώνται σε ανιχνευτές σωματιδίων και μελετώνται προκειμένου να προσδιοριστούν οι ιδιότητες των αρχικών κουάρκ.





Οι αδρονικοί πίδακες μπορούν να θεωρηθούν και ως τοπικά συμπλέγματα ενέργειας τα οποία συνδέονται με τροχιές στοιχειωδών σωματιδίων. Για τον λόγο αυτό μπορούμε να τα χωρίσουμε σε 2 κατηγορίες: [6]

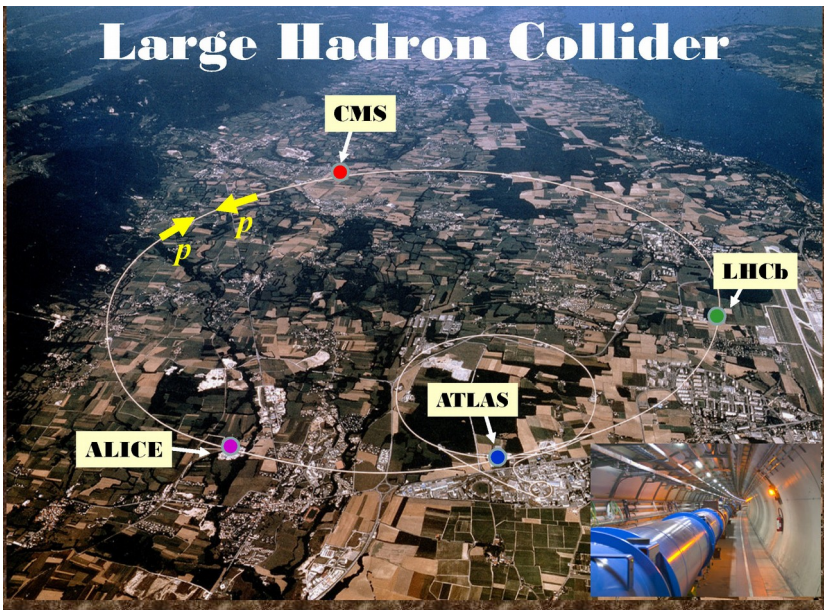
- Prompt Jets (Αδρονικοί πίδακες που προκύπτουν από τις κύριες συγκρούσεις σε επιταχυντές σωματιδίων)
- Pileup Jets (Αδρονικοί πίδακες που προκύπτουν από τις δευτερεύουσες συγκρούσεις σε επιταχυντές σωματιδίων)



Για να οριστεί ένας αδρονικός πίδακας χρειάζεται ένας αλγόριθμος και ένα μοντέλο ανακατασκευής. Το πρώτο ορίζει πώς ορισμένα δεδομένα (σωματίδια ή στοιχεία των ανιχνευτών) ομαδοποιούνται σε πίδακες, ενώ το δεύτερο καθορίζει πώς αποδίδεται μια τιμή ορμής σε έναν πίδακα. Σε πειράματα σωματιδιακής φυσικής, οι πίδακες συνήθως κατασκευάζονται από συστάδες εναποθέσεων ενέργειας στα θερμιδόμετρα του ανιχνευτή. Κατά τη μελέτη προσομοιωμένων γεγονότων, οι πίδακες των θερμιδόμετρων μπορούν να ανακατασκευαστούν με βάση μια προσομοιωμένη απόκριση του ανιχνευτή. Τυπικοί αλγόριθμοι ανακατασκευής αδρονικών πιδάκων είναι ο αλγόριθμος k_T (k_T algorithm) και ο αλγόριθμος κώνου (cone algorithm).

1.1.3 Ο Μεγάλος Αδρονικός Επιταχυντής (Large Hadron Collider)

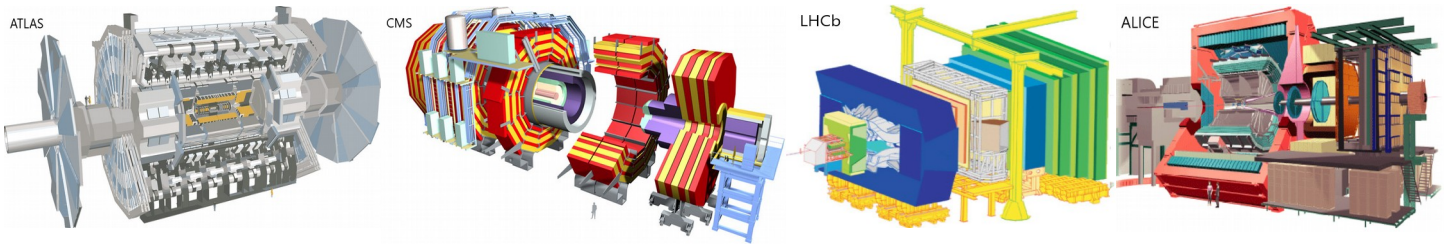
Ο Μεγάλος Αδρονικός Επιταχυντής (LHC) είναι ο μεγαλύτερος και ισχυρότερος επιταχυντής σωματιδίων στον κόσμο. Ξεκίνησε να λειτουργεί για πρώτη φορά στις 10 Σεπτεμβρίου 2008 και παραμένει μέχρι και σήμερα η τελευταία προσθήκη στο συγκρότημα επιταχυντών του Ευρωπαϊκού Κέντρου Πυρηνικών Ερευνών (CERN). Ο LHC αποτελείται από έναν δακτύλιο διαμέτρου 27 χιλιομέτρων και βρίσκεται στα Γάλλο-Ελβετικά σύνορα μεταξύ 50 και 175 μέτρων κάτω από την γή.



Μέσα στον επιταχυντή, δύο δέσμες πρωτονίων υψηλής ενέργειας ταξιδεύουν σε αντίθετες κατευθύνσεις εντός δύο σωλήνων εξαιρετικά υψηλού κενού, με ταχύτητα κοντά στην ταχύτητα του φωτός προτού συγκρουστούν. Οι δέσμες καθοδηγούνται κυκλικά γύρω από τον δακτύλιο του LHC με την βοήθεια ενός πολύ ισχυρού μαγνητικού πεδίου (8.3 Tesla) που διατηρείται από υπεραγώγιμους ηλεκτρομαγνήτες. Οι ηλεκτρομαγνήτες κατασκευάζονται από πηνία ειδικού ηλεκτρικού καλωδίου που λειτουργεί σε υπεραγώγιμη κατάσταση, διοχετεύοντας αποτελεσματικά την ηλεκτρομαγνητική ενέργεια στις δέσμες πρωτονίων χωρίς αντιστάσεις ή απώλειες. Αυτό απαιτεί ψύξη των μαγνητών στους 1.85 Kelvin, θερμοκρασία ψυχρότερη από το κενό του διαστήματος. Για το λόγο αυτό, μεγάλο μέρος του επιταχυντή συνδέεται με ένα σύστημα διανομής υγρού ηλίου, το οποίο συνεχώς ψύχει τους μαγνήτες. Ο επιταχυντής έχει δυνατότητα να επιτύχει συγρούσεις με ενέργεια της τάξης των 13TeV στο σύστημα κέντρου μάζας, το οποίο τον καθιστά και τον ισχυρότερο επιταχυντή μέχρι σήμερα. [13] [14]

Στον LHC οι δέσμες πρωτονίων μπορούν να συγκρουστούν σε 4 προκαθορισμένα σημεία, στα οποία έχουν τοποθετηθεί 4 ανιχνευτές σωματιδίων:

- ATLAS (A Toroidal LHC ApparatuS)
- CMS (Compact Muon Solenoid)
- LHCb (LHC Beauty)
- ALICE (A Large Ion Collider Experiement)



1.1.4 Βαθμονοποίηση Αδρονικών Πηδάκων (Calibration of Hadronic Jets)

Μεγάλο μέρος της επιτυχίας του LHC βασίζεται στην ικανότητα ανακατασκευής αδρονικών πιδάκων και μέτρησης της ενέργειάς τους με ακρίβεια (βαθμονόμηση). Παρά τη δύσκολία του έργου και τα δύο πειράματα που ασχολούνται με αυτό (CMS, ATLAS) έχουν επιτύχει ένα άνευ προηγουμένου επίπεδο ακρίβειας ως προς τη βαθμονόμηση (calibration) αδρονικών πιδάκων. Δύο βασικές ιδέες είναι που επηρέασαν σημαντικά αυτά τα αποτελέσματα: Η πρώιμη υιοθέτηση του αλγορίθμου k_T (k_T algorithm) και η χρήση της μεθόδου αφαίρεσης συσσωρευμένων γεγονότων συμβάν-προς-συμβάν (event-by-event pileup subtraction) για τη μείωση της επίδρασης των αποτελεσμάτων από συσσωρευμένες διακυμάνσεις.

Ο σκοπός της βαθμονόμησης ενός αδρονικού πίδακα είναι συγκεκριμένος και σημαντικός. Η ενεργειακή κλίμακα των ανακατασκευασμένων πιδάκων δεν ανταποκρίνεται πληρως στην κλίμακα ενέργειας των αδρονικών πιδάκων που κατασκευάζονται από την Μόντε Κάρλο προσομοίωση, η οποία συμπεριλαμβάνει και τις δραστηριότητες των υποκείμενων γεγονότων (Underlying Events). Μια στοχευμένη λοιπόν βαθμονόμηση ενέργειας απαιτείται για την, κατά μέσο όρο, βαθμονόμηση του ανακατασκευασμένου πίδακα ενέργειας ως προς τον αδρονικό πίδακα που κατασκευάστηκε από την Μόντε Κάρλο προσομοίωση. Η βαθμονόμηση με αυτό τον τρόπο φέρνει την ενεργειακή κλίμακα των αδρονικών πιδάκων που προκύπτουν από τα δεδομένα (data) του πειράματος στην ίδια βάση με τα δεδομένα από την προσομοίωση (simulation).

Ένα βασικό στοιχείο για τη μέτρηση και τον χαρακτηρισμό ενός αδρονικού πίδακα είναι η τιμή της απόκρισης (Response), που ορίζεται ως η αναλογία μεταξύ της ανακατασκευασμένης ενέργειας πίδακα και της αντίστοιχης ενέργειας ως προς την προσομοίωση:

$$R(E,\eta) = \left\langle \frac{E^{reco}}{E^{truth}} \right\rangle$$

Σε αυτή την διπλωματική η εγκάρσια ορμή p_T (transverse momentum) θα χρησιμοποιηθεί για τον καθορισμό της απόκρισης. [7]

1.1.5 Μεταβλητές Χαρακτηρισμού

Η κατάλληλη επιλογή των χαρακτηριστικών μεταβλητών (predictors) είναι ιδιαίτερα σημαντική καθώς βάση αυτών θα γίνουν οι διακρίσεις και η ανάλυση των δεδομένων. Κάθε μεταβλητή για ένα συγκεκριμένο σύνολο δεδομένων έχει μια διακριτική ικανότητα, η οποία όταν είναι μεγαλύτερη από τις υπόλοιπες, καθιστά την μεταβλητή αυτη ιδιαίτερα σημαντική. Οι μεταβλητές των δεδομένων που χρησιμοποιήθηκαν για αυτή την διπλωματική, οι οποίες έχουν προκύψει από Monte Carlo προσομοίωση, είναι 12 και παρακάτω αναλύεται για την κάθε μια η φυσική της σημασία.

- Μεταβλητή “**jetRawPt**”

Η μεταβλητή jetRawPt ορίζεται ως η εγκάρσια ορμή του αδρονικού πίδακα πριν εφαρμοστούν οι διορθώσεις ενέργειας (energy corrections)

- Μεταβλητή “**jetPt**”

Η μεταβλητή jetPt ορίζεται ως η εγκάρσια ορμή του αδρονικού πίδακα αφού εφαρμοστούν οι διορθώσεις ενέργειας (θα συμβολιστεί στα αποτελέσματα του πρακτικού μέρους ως Corrected)

- Μεταβλητή “**jetEta**”

Η μεταβλητή jetEta ορίζεται ως η ψευδωκύτητα (pseudorapidity) του αδρονικού πίδακα και είναι μια χωρική συντεταγμένη που περιγράφει τη γωνία ενός πίδακα σε σχέση με τον άξονα της δέσμης. Ορίζεται από την σχέση:

$$\eta \equiv -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$$

όπου θ είναι η πολική γωνία, δηλαδή αυτή μεταξύ της ορμής **p** και του θετικού ημιάξονα της δέσμης. Επιλύνοντας την παραπάνω σχέση ως προς θ έχουμε:

$$\theta = 2\arctan\left(e^{-\eta}\right)$$

Η ψευδωκύτητα επίσης μπορεί να δοθεί σε συνάρτηση ως προς την ορμή από την σχέση:

$$\eta = \frac{1}{2}\ln\left(\frac{|\mathbf{p}| + p_{\mathrm{L}}}{|\mathbf{p}| - p_{\mathrm{L}}}\right) = \operatorname{arctanh}\left(\frac{p_{\mathrm{L}}}{|\mathbf{p}|}\right)$$

Όπου |**p**| το μέτρο της τρισδιάστατης ορμής και p_L η συνιστώσα της ορμής κατά μήκος του άξονα της δέσμης (longitudinal momentum).

Η φυσική σημασία της ψευδωκύτητας (η) είναι πως βάση κάποιων προσεγγίσεων μπορεί να μας βοηθήσει στον υπολογισμό της ωκύτητας (rapidity). Η ωκύτητα (y) χρησιμοποιείται συνήθως ως μέτρο για τη σχετικιστική ταχύτητα και δίνεται από την σχέση:

$$y \equiv \frac{1}{2}\ln\left(\frac{E + p_{\mathrm{L}}}{E - p_{\mathrm{L}}}\right)$$

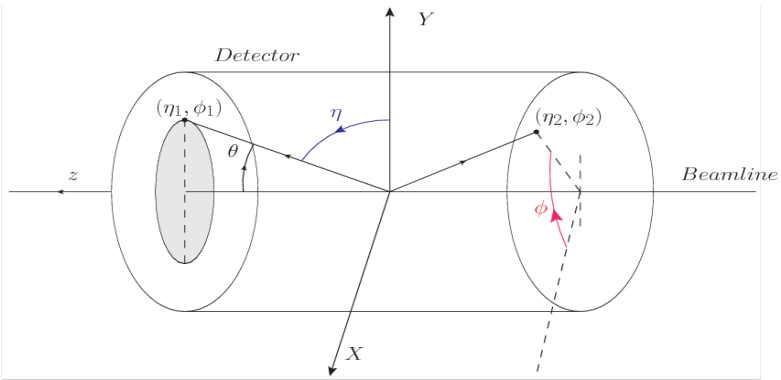
Όταν ένα σωματίδιο έχει ταχύτητα κοντά στην ταχύτητα του φωτός ή έχει αμελητέα μάζα, τότε μπορούμε να κάνουμε την προσέγγιση: $m \ll p \Rightarrow E \approx p \Rightarrow \eta \approx y$

Αν θέλουμε να είμαστε πιο ακριβής τότε η ωκύτητα ως συνάρτηση της ψευδωκύτητας δίνεται από:

$$y = \ln\left(\frac{\sqrt{m^2 + p_{\mathrm{T}}^2\cosh^2\eta} + p_{\mathrm{T}}\sinh\eta}{\sqrt{m^2 + p_{\mathrm{T}}^2}}\right)$$

- Μεταβλητή “**jetPhi**”

Η μεταβλητή jetPhi ορίζεται ως η αζιμουθιακή γωνία (Azimuthal angle) του αδρονικού πίδακα, μετριέται σε rad και παίρνει τιμές μεταξύ [- π , π].



Μαζί οι μεταβλητές jetEta (η) και jetPhi (φ) ορίζουν τις χωρικές συντεταγμένες και συμβάλουν στον προσδιορισμό της θέσης ενός αδρονικού πίδακα εντός του ανιχνευτή.

- Μεταβλητή “**jetMass**”

Η μεταβλητή jetMass ορίζεται ως το αναλλοίωτο της μάζας, της τετρα-ορμής του αδρονικού πίδακα

- Μεταβλητή “**jetChf**”

Η μεταβλητή jetChf ορίζεται ως το ποσοστό της ενέργειας του αδρονικού πίδακα που κουβαλάνε τα φορτισμένα Αδρόνια (Charged Hadron Fraction). Παίρνει τιμές μεταξύ [0 , 1].

- Μεταβλητή “**jetNhf**”

Η μεταβλητή jetNhf ορίζεται ως το ποσοστό της ενέργειας του αδρονικού πίδακα που κουβαλάνε τα αφόρτιστα Αδρόνια (Neutral Hadron Fraction). Παίρνει τιμές μεταξύ [0 , 1].

- Μεταβλητή “**jetPhf**”

Η μεταβλητή jetPhf ορίζεται ως το ποσοστό της ενέργειας του αδρονικού πίδακα που προέρχεται απο Φωτόνια (Photon Fraction). Παίρνει τιμές μεταξύ [0 , 1].

- Μεταβλητή “**jetMuf**”

Η μεταβλητή jetMuf ορίζεται ως το ποσοστό της ενέργειας του αδρονικού πίδακα που προέρχεται απο Μιόνια (Muon Fraction). Παίρνει τιμές μεταξύ [0 , 1].

- Μεταβλητή “**jetElf**”

Η μεταβλητή jetElf ορίζεται ως το ποσοστό της ενέργειας του αδρονικού πίδακα που προέρχεται απο Ηλεκτρόνια (Electron Fraction). Παίρνει τιμές μεταξύ [0 , 1].

Μια επιπλέον μεταβλητή που θα μας φανεί χρήσιμη από τα δεδομένα του δέντρου το οποίο θα παράξει η προσομοίωση Monte Carlo είναι η **“jetIsBtag”**. Μέσω αυτής της μεταβλητής, η οποία παίρνει μπουλιανές τιμές True ή False, θα καταφέρουμε να επιλέξουμε τους αδρονικούς πίδακες που θα έχουν προκύψει από την διάσπαση του χαμηλού (bottom) κουάρκ. Για τον λόγο αυτό και τα δεδομένα που θα χρησιμοποιήσουμε τελικά για την εκπαίδευση των ταξινομητών μας θα έχουν τα ονόματα:

- bjetRawPt
- bjetPt
- bjetEta
- bjetPhi
- bjetMass
- bjetChf
- bjetNhf
- bjetMuf
- bjetElf

[9] [11] [12]

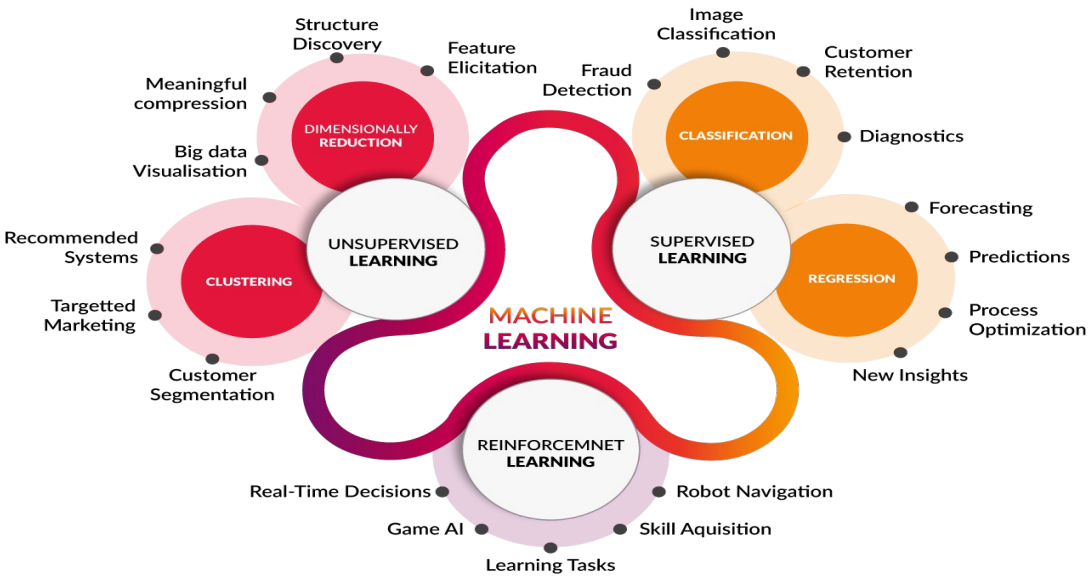
1.2 Μηχανική Μάθηση

1.2.1 Η Έννοια της Μηχανικής Μάθησης

Η μηχανική μάθηση ως έννοια υπάρχει εδώ και αρκετές δεκαετίες. Ο όρος «μηχανική μάθηση» επινοήθηκε από τον Άρθουρ Σάμιουελ, επιστήμονα υπολογιστών στην IBM και πρωτοπόρο στην τεχνητή νοημοσύνη και τα παιχνίδια υπολογιστών. Ο Σάμιουελ σχεδίασε ένα πρόγραμμα υπολογιστή για να παίζει πούλια. Όσο περισσότερο το πρόγραμμα έπαιζε το παιχνίδι, τόσο περισσότερα μάθαινε από την εμπειρία, χρησιμοποιώντας αλγόριθμους για να κάνει προβλέψεις. Αυτή η ιδιότητα των αλγορίθμων μηχανικής μάθησης είναι που τους διαφοροποιεί ριζικά από την απλή εκτέλεση πάγιων βηματικών προγραμματιστικών εντολών [15]. Ως κλάδος λοιπόν η μηχανική μάθηση διερευνά την ανάλυση και την κατασκευή αλγορίθμων που μπορούν να μάθουν από δεδομένα και να κάνουν προβλέψεις για δεδομένα. [1]

Η μηχανική μάθηση συνήθως ταξινομείται σε τρεις μεγάλες κατηγορίες, βάση του τρόπου εκμάθησης του αλγορίθμου. Αυτές οι κατηγορίες είναι:

- Μάθηση με Επίβλεψη (Supervised Learning): Το υπολογιστικό πρόγραμμα δέχεται τις παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα από έναν «δάσκαλο», και ο στόχος του είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα.
- Μάθηση χωρίς Επίβλεψη (Unsupervised Learning): Το υπολογιστικό πρόγραμμα χωρίς να του παρέχεται κάποια εμπειρία από τον αλγόριθμο μάθησης, πρέπει να βρεί την δομή των δεδομένων εισόδου.
- Ενισχυτική Μάθηση (Reinforcement Learning): Το υπολογιστικό πρόγραμμα αλληλεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτευχθεί ένας συγκεκριμένος στόχος (όπως η οδήγηση ενός οχήματος), χωρίς κάποιος δάσκαλος να του λέει ρητά αν έχει φτάσει κοντά στο στόχο του. Ένα άλλο παράδειγμα είναι να μάθει να παίζει ένα παιχνίδι εναντίον κάποιου αντιπάλου, όπως για παράδειγμα το σκάκι, χωρίς ρητές οδηγίες στόχου.



Μια άλλη κατηγοριοποίηση των προβλημάτων μηχανικής μάθησης προκύπτει από το επιθυμητό αποτέλεσμα του σύστηματος μηχανικής μάθησης. Αυτές οι κατηγορίες είναι:

- Η ταξινόμηση (Classification), τα δεδομένα εισόδου χωρίζονται σε δύο ή περισσότερες κλάσεις, και ο αλγόριθμος θα πρέπει να κατασκευάσει ένα μοντέλο, το οποίο θα αντιστοιχίζει τα δεδομένα εισόδου στις αντίστοιχες κλάσεις (multi-label ταξινόμηση). Αυτό συνήθως εμπίπτει στην μάθηση με επίβλεψη.
- Η παλινδρόμηση (Regression), τα αποτελέσματα εξώδου είναι συνεχή και όχι διακριτά, επίσης πρόβλημα που εμπίπτει στην μάθηση με επίβλεψη.
- Η συσταδοποίηση (Clustering), ένα σύνολο εισόδων χωρίζεται σε ομάδες. Σε αντίθεση με την ταξινόμηση, οι ομάδες δεν είναι γνωστές εκ των προτέρων, καθιστώντας αυτόν τον διαχωρισμό τυπική εργασία μάθησης χωρίς επίβλεψη.
- Η εκτίμηση πυκνότητας (Density Estimator), βρίσκει την κατανομή των δεδομένων εισόδου σε κάποιο χώρο.
- Η μείωσης διάστασης (Dimensionality Reduction), τα δεδομένα απλοποιούνται και αντιστοιχίζονται σε ένα χώρο λιγότερων διαστάσεων.

Σε αυτή την διπλωματική εργασία έγινε χρήση της μηχανικής μάθηση για παλινδρόμηση. Η παλινδρόμηση είναι μια τεχνική για τη διερεύνηση της σχέσης μεταξύ ανεξάρτητων μεταβλητών ή χαρακτηριστικών και μιας εξαρτημένης μεταβλητής ή αποτελέσματος. Χρησιμοποιείται ως μέθοδος για την προγνωστική μοντελοποίηση στη μηχανική μάθηση, στην οποία χρησιμοποιείται ένας αλγόριθμος για την πρόβλεψη συνεχών αποτελεσμάτων. Η παλινδρόμηση μπορεί να βρεθεί σε πολλές διαφορετικές εφαρμογές της μηχανικής μάθησης. Είτε για την ενίσχυση οικονομικών προβλέψεων, είτε για την προβλεψη τάσεων της υγειονομικής περίθαλψης, η ανάλυση παλινδρόμησης μπορεί να προσφέρει σε πολλούς κλάδους βασικές γνώσεις για τη λήψη αποφάσεων. Χρησιμοποιείται ήδη για την πρόβλεψη τιμών κατοικιών, τιμών μετοχών ή στην περίπτωση μας την πρόβλεψη των συνεχών τιμών της τετραορμής από τον αδρονικό πίδακα του χαμηλού κουάρκ.

Μαζί με την ταξινόμηση (Classification), η παλινδρόμηση (Regression) είναι μία από τις κύριες εφαρμογές της μηχανικής μάθησης. Η ταξινόμηση, όπως αναφέραμε προηγουμένος, είναι η κατηγοριοποίηση αντικειμένων σε διακριτά αποτελέσματα όπως για παράδειγμα αν ένα εισερχόμενο email ανήκει στην κλάση "spam" ή "όχι spam". Από την άλλη η παλινδρόμηση είναι η πρόβλεψη συνεχών αποτελεσμάτων όπως για παράδειγμα η τιμή μιας μετοχής ή η ενέργεια-ορμή ενός στοιχειώδους σωματιδίου.

Η μηχανική μάθηση με επίβλεψη είναι αναπόσπαστη προσέγγιση και στις δύο περιπτώσεις. Όπως συμβαίνει σε όλες τις περιπτώσεις μηχανικής μάθησης με επίβλεψη, θα πρέπει να λαμβάνεται ιδιαίτερη μέριμνα ώστε να διασφαλίζεται ότι τα δεδομένα εκπαίδευσης έχουν σωστή ετικέτα (label) και είναι αντιπροσωπευτικά του συνολικού πληθυσμού των δεδομένων. Εάν τα δεδομένα εκπαίδευσης δεν είναι αντιπροσωπευτικά, το μοντέλο πρόβλεψης θα είναι υπερβολικά προσαρμοσμένο σε δεδομένα που δεν αντιπροσωπεύουν νέα και μη ορατά δεδομένα. Αυτό θα έχει ως αποτέλεσμα ανακριβείς προβλέψεις μόλις αναπτυχθεί το μοντέλο και γενικευτεί (Overfitting). Παρακάτω όπως και στο πρακτικό κομμάτι αυτής της εργασίας θα δούμε πιο αναλυτικά αυτό το φαινόμενο όπως και πως μπορούμε να το αντιμετοπίσουμε. [3] [4]

1.2.2 Μέθοδοι Μηχανικής Μάθησης

■ Γραμμική Διακριτική Ανάλυση (Linear Discrimination Analysis)

Στη μέθοδο της γραμμικής διακριτικής ανάλυσης, η επιλογή γεγονότων γίνεται σε έναν μετασχηματισμένο χώρο μεταβλητών με μηδενικές γραμμικές συσχετίσεις, διακρίνοντας τις μέσες τιμές των κατανομών του σήματος (signal) και του υποβάθρου (background). Η γραμμική διάκριτική ανάλυση προσδιορίζει έναν άξονα στον (συσχετιζόμενο) υπερχώρο των μεταβλητών εισόδου έτσι ώστε, κατά την προβολή των κλάσεων εξόδου (σήμα-υπόβαθρο) επάνω σε αυτόν τον άξονα να απωθούνται όσο το δυνατόν πιο μακριά το σήμα από το υπόβαθρο, ενώ γεγονότα της ίδιας κλάσης να έρχονται σε κοντινή απόσταση. Η ιδιότητα γραμμικότητας αυτού του ταξινομητή αντικατοπτρίζεται στη μετρική με την οποία προσδιορίζεται το "μακριά" και το "κοντά" στον πίνακα συνδιακύμανσης του διαχωριστικού μεταβλητού χώρου.

Η Γραμμική Διακριτική Ανάλυση παρέχει ταξινόμηση δεδομένων χρησιμοποιώντας ένα γραμμικό μοντέλο, όπου το γραμμικό αναφέρεται στην γραμμική συνάρτηση y(x) ως προς τις παραμέτρους β:

$$y(\mathbf{x}) = \mathbf{x}^\top \beta + \beta_0$$

όπου το β_0 δηλώνει την προκατάληψη (bias) και ρυθμίζεται έτσι ώστε $y(x) \geq 0$ για το σήμα (signal) και $y(x) < 0$ για το υπόβαθρο (background).

Υποθέτοντας ότι υπάρχουν m+1 παράμετροι $\beta_0, \ldots, \beta_m$ που πρέπει να εκτιμηθούν χρησιμοποιώντας ένα σύνολο εκπαίδευσης που αποτελείται των n γεγονότων, η καθοριστική εξίσωση για το β είναι :

$$Y = X\beta$$

όπου έχουμε απορροφήσει το β_0 στο διάνυσμα β και έχουμε εισάγει τους πίνακες :

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad , \quad X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1m} \\ 1 & x_{21} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

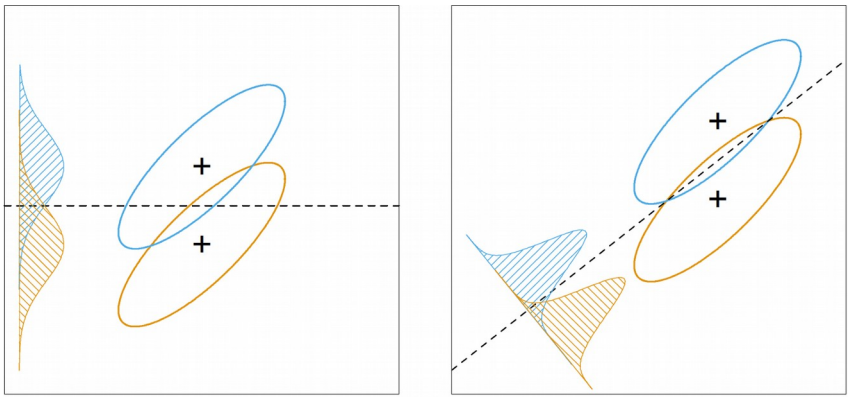
όπου η σταθερή στήλη στο X αντιπροσωπεύει τη μεροληψία (bias) β_0 και το Y αποτελείται από τις τιμές-στόχους με $y_i = 1$ αν το i-ωστό συμβάν ανήκει στην κλάση σήματος (signal) και $y_i = 0$ εάν το i-ωστό συμβάν ανήκει στην τάξη υποβάθρου (background). Εφαρμόζοντας τη μέθοδο των ελαχίστων τετραγώνων, λαμβάνουμε τώρα τις κανονικές εξισώσεις για το πρόβλημα ταξινόμησης, που δίνονται από :

$$X^T X\beta = X^T Y \iff \beta = (X^T X)^{-1} X^T Y$$

Ο μετασχηματισμός $(X^T X)^{-1} X^T$ είναι γνωστός ως ο Moore-Penrose ψευδοαντίστροφος του X και μπορεί να θεωρηθεί ως η γενίκευση του αντίστροφου πίνακα για μη τετραγωνικούς πίνακες. Απαιτεί βέβαια ότι ο πίνακας X έχει πλήρη τάξη. Εάν χρησιμοποιούνται γεγονότα με βάρος, αυτό λαμβάνεται απλώς υπόψη με την εισαγωγή ενός διαγώνιου πίνακα βάρους W και τροποποιώντας τις κανονικές εξισώσεις ως εξής:

$$\beta = (X^T W X)^{-1} X^T W Y$$

Θεωρώντας δύο γεγονότα x_1 και x_2 στο όριο απόφασης, έχουμε $y(x_1) = y(x_2) = 0$ και ως εκ τούτου $(x_1 - x_2)^T\beta = 0$. Έτσι βλέπουμε ότι ο Γραμμικός Ταξινομητής (LD) μπορεί να ερμηνευθεί γεωμετρικά : Καθορίζει το όριο απόφασης με το να βρίσκει ένα ορθογώνιο διάνυσμα β.

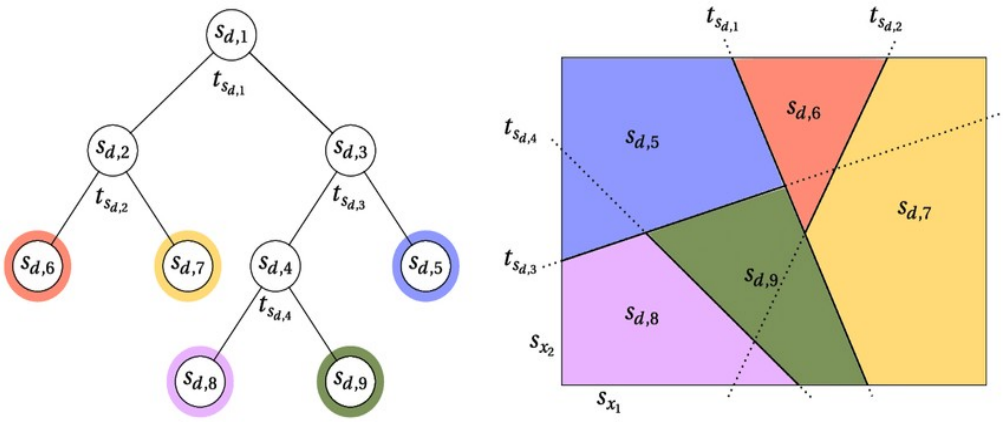


Το παραπάνω μοντέλο βέβαια είναι σχεδιασμένο για ταξινόμηση μεταξύ σήματος (signal) $y_i=1$ και υποβάθρου (background) $y_i=0$. Είναι απλό να εφαρμοστεί ο αλγόριθμος LD για παλινδρόμηση αντικαθιστώντας τους δυαδικούς στόχους $y_i \in 0,1$ στα δεδομένα εκπαίδευσης με τις μετρούμενες τιμές της συνάρτησης που πρόκειται να εκτιμηθεί. Η συνάρτηση y(x) που προκύπτει είναι τότε η καλύτερη εκτίμηση για τα δεδομένα που λαμβάνονται με παλινδρόμηση ελαχίστων τετραγώνων.

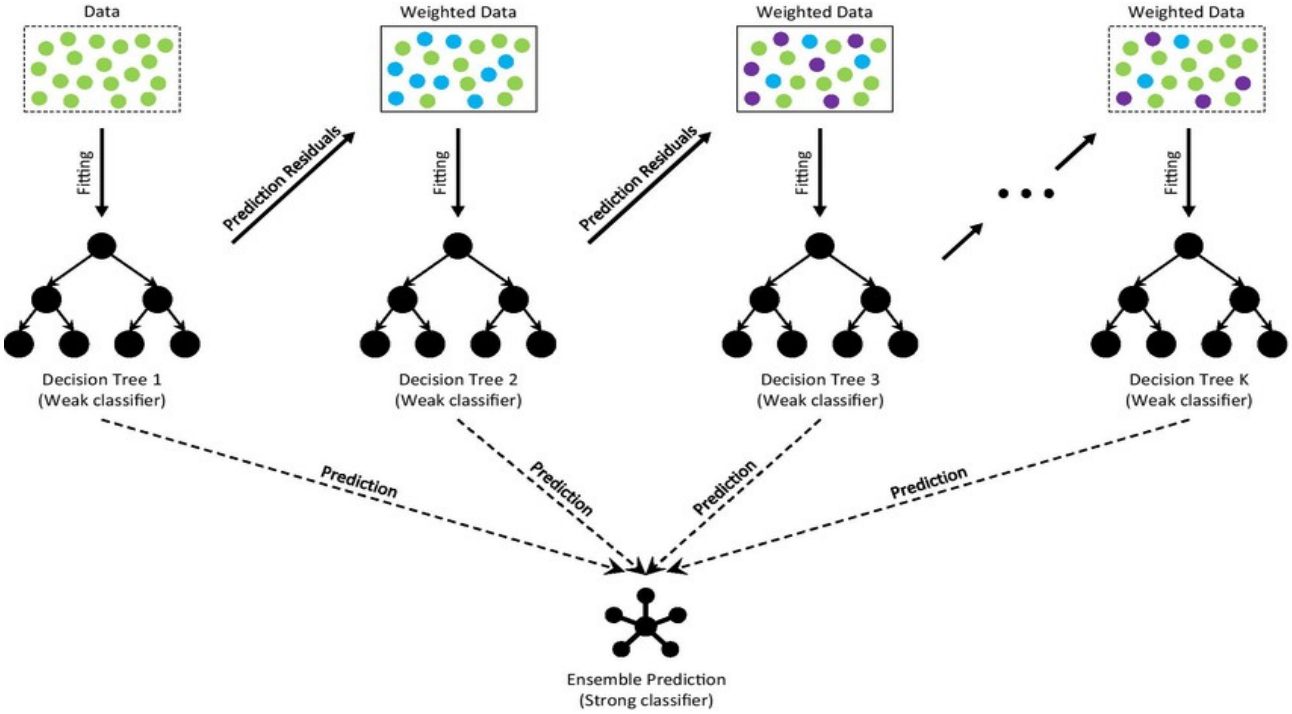
Ο Γραμμικός Ταξινομητής (LD) είναι βέλτιστος για Gaussian κατανεμημένες μεταβλητές με γραμμικές συσχετίσεις. Μπορεί να είναι ανταγωνιστικός με τους μη γραμμικούς ταξινομητές σε ορισμένες περιπτώσεις και για αυτό σε κάθε ανάλυση δεδομένων χρησιμοποιείτε ως σημείο αναφοράς. [1] [11]

■ Ενισχυμένα Δέντρα Απόφασης (Boosted Descision Trees)

Ένα δέντρο απόφασης-παλινδρόμησης είναι ένας ταξινομητής-παλινδρομητής δυαδικής δομής δέντρου. Επαναλαμβανόμενες αποφάσεις ναι/όχι (αριστερά/δεξιά) λαμβάνονται σε μία μεμονωμένη μεταβλητή κάθε φορά έως ότου εκπληρωθεί ένα κριτήριο διακοπής. Ο χώρος φάσης χωρίζεται με αυτόν τον τρόπο σε πολλές περιοχές και τελικά τα γεγονότα ταξινομούνται ως σήμα ή υπόβαθρο επάνω στον τελευταίο κόμβο του δέντρου (φύλλο). Σε περίπτωση δέντρων παλινδρόμησης ο διαχωρισμός στους κόμβους εκτελείται ως προς την μεταβλητή που δίνει τη μέγιστη μείωση στη μέση τιμή του τετραγώνου του σφάλματος ως προς μια σταθερή τιμή της μεταβλητής στόχου. Στο τέλος της διαδικασίας κάθε φύλλο αντιπροσωπεύει μια συγκεκριμένη τιμή αυτης της μεταβλητής στόχου.



Η ενίσχυση (Boosting) ενός δέντρου απόφασης-παλινδρόμησης επεκτείνει αυτήν την έννοια από ένα δέντρο σε πολλά δέντρα που σχηματίζουν ένα δάσος. Τα δέντρα του δάσους προέρχονται από την διαδοχική εφαρμογή του αλγορίθμου σε επανασταθμισμένες (boosted) εκδόσεις των δεδομένων εκπαίδευσης και στη συνέχεια λαμβάνεται ένας σταθμισμένος μέσος όρος από την ακολουθία των αλγορίθμων που έχουν προκύψει. Σαν μέθοδος έχει εισαχθεί στις τεχνικές ταξινόμησης από τις αρχές της δεκαετίας του '90 και σε πολλές περιπτώσεις αυτή η απλή στρατηγική έχει ως αποτέλεσμα δραματικές αυξήσεις στην απόδοση. Η ενίσχυση σταθεροποιεί την απόκριση των δέντρων απόφασης σε σχέση με τις διακυμάνσεις στο δείγμα εκπαίδευσης και είναι σε θέση να μειώσει τα φαινόμενα της υπερεκπαίδευσης. Στη συνέχεια, θα χρησιμοποιείται ο όρος δέντρο απόφασης και για τα δέντρα απόφασης (Classification) και για τα δέντρα παλινδρόμησης (Regression).



Τα δέντρα απόφασης είναι αρκετά διαδεδομένοι και γνωστοί ταξινομητές διότι επιτρέπουν μια ξεκάθαρη ερμηνεία του μοντέλου μέσω μίας δισδιάστατης εικόνας δέντρου. Από αυτή την άποψη η δομή τους είναι παρόμοια με τις ορθογώνιες περικοπές. Ωστόσο, ενώ μια ανάλυση που βασίζεται σε ορθογώνιες περικοπές είναι σε θέση να επιλέξει μόνο έναν υπερκύβο ως περιοχή του χώρου φάσης, το δέντρο απόφασης μπορεί να χωρίσει το χώρο φάσης σε έναν μεγάλο αριθμό υπερκύβων, ο καθένας εκ των οποίων προσδιορίζεται είτε ως "σήμα" είτε ως "υπόβαθρο" ή του αποδίδεται μια σταθερή τιμή στόχου (target value) σε περίπτωση δέντρου παλινδρόμησης.

Ένα μειονέκτημα των δέντρων απόφασης είναι η αστάθειά τους σε σχέση με τις στατιστικές διακυμάνσεις στο δείγμα εκπαίδευσης από το οποίο προκύπτει και η δομή τους. Για παράδειγμα, αν έχουμε δύο μεταβλητές εισοδύναμης ισχύς διαχωρισμού, μια διακύμανση στο δείγμα εκπαίδευσης μπορεί να προκαλέσει τον αλγόριθμο ανάπτυξης του δέντρου να χωρίσει τον χώρο φάσης ως προς την μια μεταβλητή, ενώ χωρίς αυτή την διακύμανση η άλλη μεταβλητή θα μπορούσε να έχει επιλεγεί. Σε αυτήν περίπτωση ολόκληρη η δομή του δέντρου αλλοιώνεται κάτω από αυτόν τον κόμβο απόφασης, το οποίο πιθανώς να προκαλέσει μια ουσιαστικά διαφορετική απόκριση στον ταξινομητή. Αυτό το μειονέκτημα λύνεται μέσω της διαδικασίας ενίσχυσης (boosting), ωστόσο το πλεονέκτημα της απλής ερμηνείας του ενός δέντρου απόφασης έχει χαθεί από το δάσος των δέντρων απόφασης που θα έχουν δημιουργηθεί.

Η εκπαίδευση-κατασκευή-ανάπτυξη ενός δέντρου απόφασης είναι η διαδικασία κατά την οποία καθορίζονται τα κριτήρια διαχωρισμού για κάθε κόμβο του δέντρου. Η εκπαίδευση ξεκινά με τον πρώτο κόμβο (ρίζα) του δέντρου, όπου ένα αρχικό κριτήριο διαχωρισμού χωρίζει το συνολικό δείγμα εκπαίδευσης. Αυτός ο διαχωρισμός έχει ως αποτέλεσμα να δημιουργηθούν δύο υποσύνολα εκπαιδευτικών γεγονότων με το καθένα, μέσω του ίδιου αλγορίθμου, να προσδιορίζει επαναλαμβανόμενα τις επόμενες διαίρεσης. Αυτή η διαδικασία συνεχίζεται εως ότου χτιστεί ολόκληρο το δέντρο. Σε κάθε κόμβο, η διαίρεση του συνόλου προσδιορίζεται με την εύρεση της βέλτιστης μεταβλητής και αντίστοιχης τιμής κοπής που παρέχουν τον καλύτερο διαχωρισμό μεταξύ σήματος και υποβάθρου. Ο διαχωρισμός των κόμβων σταματά μόλις το δέντρο φτάσει στο επιθημιτό όριο που έχει καθορίσει ο χρήστης κατά την διαμόρφωση του δέντρου από τις παραμέτρους: Μέγιστο Βάθος (Max Depth) και Αριθμός Δέντρων (NTrees).

Μπορούν να διαμορφωθούν αρκετά κριτήρια διαχωρισμού για την αξιολόγηση της απόδοσης μιας μεταβλητής και μιας συγκεκριμένης κοπής ενός κόμβου. Όλα τα κριτήρια διαχωρισμού έχουν ένα μέγιστο όταν τα δείγματα είναι πλήρως “μπερδεμένα”, δηλ. για καθαρότητα $p = 0,5$, και πέφτουν στο μηδέν όταν το δείγμα αποτελείται μόνο από μία κλάση. Οι δοκιμές δεν έχουν αποκαλύψει σημαντική διαφορά απόδοσης μεταξύ των ακόλουθων κριτηρίων διαχωρισμού:

- Δείκτης Gini (Gini Index): $l = p \cdot (1 - p)$
- Διασταυρούμενη Εντροπία (Cross Entropy): $l = -p \cdot \ln(p) - (1 - p) \cdot \ln(1 - p)$
- Μέσω Τετραγωνικό Σφάλμα (Average squared error): $l = 1/N \cdot \sum(y - y´)^2$

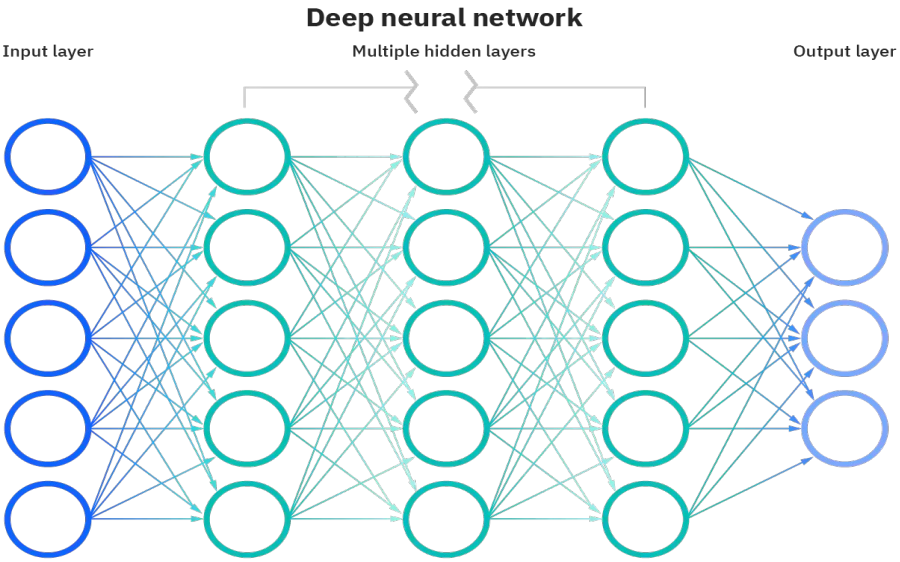
Το Μέσω Τετραγωνικό Σφάλμα χρησιμοποιείται για τα δέντρα παλινδρόμησης όπου y είναι ο στόχος παλινδρόμησης για το κάθε γεγονός στον κόμβο και y´ είναι η μέση τιμή από όλα τα γεγονότα στον κόμβο.

Στη βιβλιογραφία τα Ενισχυμένα Δέντρα Απόφασης μερικές φορές αναφέρονται ως οι καλύτεροι ταξινομητές για χρήση "out of the box". Αυτό συμβαίνει διότι απαιτείται μικρό “κούρδισμα” της μεθόδου τους για να επιτευχθούν ικανοποιητικά καλά αποτελέσματα. Αυτό οφείλεται στην απλότητα που έχουν εγγενώς τα δέντρα απόφασης μιας και κάθε βήμα εκπαίδευσης (κόμβος διάσπασης) περιλαμβάνει μόνο μια μονοδιάστατη βελτιστοποίηση κοπής. Τα δέντρα απόφασης επιπλέον δεν είναι ευαίσθητα στο να συμπεριλαμβάνονται κατά την εκπαίδευσή τους μεταβλητές χωρίς καλή διακρίτική ικανότητα. Σε αντίθεση με τα νευρωνικά δίκτυα που είναι συνήθως πιο δύσκολο να αντιμετωπίσουν τέτοιες πρόσθετες μεταβλητές, ο αλγόριθμος εκπαίδευσης των δέντρων απόφασης θα αγνοήσει τις μεταβλητές χωρίς καλή διακρίτική ικανότητα αφού ο διαχωρισμός για κάθε κόμβο θα χρησιμοποιήσει μόνο την καλύτερη διαθέσιμη διακριτική μεταβλητή.

Παρόλα αυτά η απλότητα των Δέτρων Απόφασης έχει το μειονέκτημα ότι η θεωρητικά καλύτερη απόδοσή τους σε ένα δεδομένο πρόβλημα είναι γενικά κατώτερη από άλλες τεχνικές όπως τα νευρωνικά δίκτυα. Βέβαια, σε αρκετά ακαδημαϊκά παραδείγματα με πιο σύνθετους συσχετισμούς των μεταβλητών τα ενισχυμένα δέντρα απόφασης συχνά υπερτερούν των άλλων τεχνικών. Αυτό συμβαίνει διότι είτε δεν υπάρχει ικανοποιητικός όγκος δεδομένα εκπαίδευσης, τον οποίο θα χρειάζονταν οι άλλοι ταξινομητές όπως τα νευρωνικά δίκτυα, είτε δεν έχει προσδιοριστεί η βέλτιστη διαμόρφωση (αριθμός κρυφών στρωμάτων, αριθμός μεταβλητών-νευρώνων κτλ.) του ανταγωνιστικού νευρωνικού δικτύου. [1] [11]

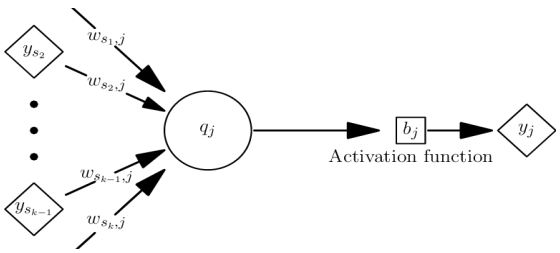
▪ Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks)

Ένα Τεχνητό Νευρωνικό Δίκτυο (ANN) είναι γενικά οποιαδήποτε προσομοιωμένη συλλογή διασυνδεδεμένων νευρώνων, με κάθε νευρώνα να παράγει μια συγκεκριμένη απόκριση από ένα δεδομένο σύνολο σημάτων εισόδου. Εφαρμόζοντας ένα εξωτερικό σήμα στους νευρώνες εισόδου το δίκτυο τίθεται σε μια καθορισμένη κατάσταση που μπορεί να μετρηθεί από την απόκριση των νευρώνων εξόδου. Μπορεί λοιπόν να σκεφτεί κανείς ένα νευρωνικό δίκτυο ως μια αντιστοίχιση από ένα χώρο μεταβλητών εισόδου $x_1, \ldots, x_n$ σε ένα χώρο μεταβλητών εξόδου $y_1, \ldots, y_m$. Η αντιστοίχιση μπορεί να είναι μη γραμμική εάν τουλάχιστον ένας νευρώνας έχει μη γραμμική ανταπόκριση στη συμβολή του.



Ενώ θεωρητικά ένα νευρωνικό δίκτυο με n νευρώνες μπορεί να έχει n^2 συνδέσεις, η πολυπλοκότητα μπορεί να μειωθεί οργανώνοντας τους νευρώνες σε στρώματα και επιτρέποντας μόνο άμεσες συνδέσεις από ένα δεδομένο στρώμα νευρώνων στο επόμενο στρώμα. Το πρώτο στρώμα ενός νευρωνικού δικτύου είναι το στρώμα εισόδου (Input layer), το τελευταίο το στρώμα εξόδου (Output layer) και όλα τα ενδιάμεσα είναι τα κρυφά στρώματα (Hidden layers). Αυτό το είδος νευρωνικού δικτύου ονομάζεται πολυεπίπεδος perceptron (Multilayer Perseptron) και όλες οι υλοποιήσεις νευρωνικού δικτύου που έγιναν σε αυτην την διπλωματική είναι αυτού του τύπου. [2] [8]

Για ένα πρόβλημα ταξινόμησης (Classification) σήματος-υποβάθρου με μεταβλητές εισόδου n_{var} το στρώμα εισόδου αποτελείται από n_{var} νευρώνες που έχουν τις τιμές εισόδου, $x_1, \ldots, x_n$ και έναν νευρώνα στο επίπεδο εξόδου που έχει την μεταβλητή εξόδου. Για ένα πρόβλημα παλινδρόμησης (Regression) η δομή του δικτύου είναι παρόμοια, εκτός από το ότι για την παλινδρόμηση πολλών στόχων κάθε ένας από τους στόχους αντιπροσωπεύεται και από έναν νευρώνα εξόδου. Ένα βάρος σχετίζεται με κάθε σύνδεση μεταξύ της εξόδου ενός νευρώνα και της εισόδου ενός άλλου νευρώνα. Κατά τον υπολογισμό της τιμής εισόδου για την συνάρτηση απόκρισης ενός νευρώνα, οι τιμές εξόδου όλων των νευρώνων που συνδέονται με τον δεδομένο νευρώνα πολλαπλασιάζεται με αυτό το βάρος (w_{ij}).



Συνάρτηση Απόκρησης Νευρώνα

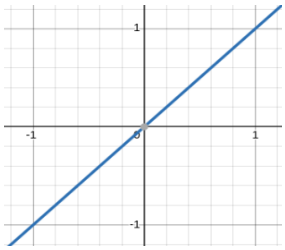
Η συνάρτηση απόκρισης ενός νευρώνα (ρ) οδηγεί την είσοδο του νευρώνα $i_1, \ldots, i_n$ στην έξοδο αυτού του νευρώνα. Συχνά μπορεί να διαχωριστεί σε μια συνάρτηση σύναψης (κ) και μια συνάρτηση ενεργοποίησης (α) έτσι ώστε: $\rho = \kappa \circ \alpha$

Οι πιο συνηθισμένες συναρτήσεις σύναψης είναι :

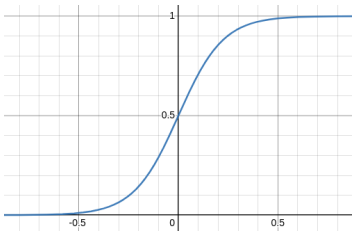
$$\kappa : \ (y_1^{(\ell)}, .., y_n^{(\ell)} | w_{0j}^{(\ell)}, .., w_{nj}^{(\ell)}) \rightarrow \begin{cases} w_{0j}^{(\ell)} + \sum_{i=1}^{n} y_i^{(\ell)} w_{ij}^{(\ell)} & \text{Συνάρτηση Αθροίσματος,} \\ w_{0j}^{(\ell)} + \sum_{i=1}^{n} \left(y_i^{(\ell)} w_{ij}^{(\ell)} \right)^2 & \text{Άθροισμα Τετραγώνων,} \\ w_{0j}^{(\ell)} + \sum_{i=1}^{n} |y_i^{(\ell)} w_{ij}^{(\ell)}| & \text{Άθροισμα Απόλυτων Τιμών,} \end{cases}$$

Οι πιο συνηθισμένες συναρτήσεις ενεργοποίησης είναι :

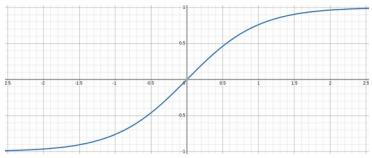
- Γραμμική (Linear): $a(x) = x$



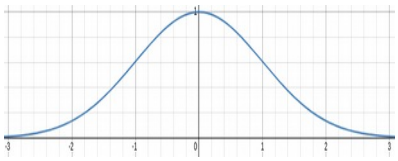
- Σιγμοειδής (Sigmoid) $a(x) = \frac{1}{1+e^{-kx}}$



- Υπερβολική Εφαπτομένη (Hyperbolic Tangent) $a(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



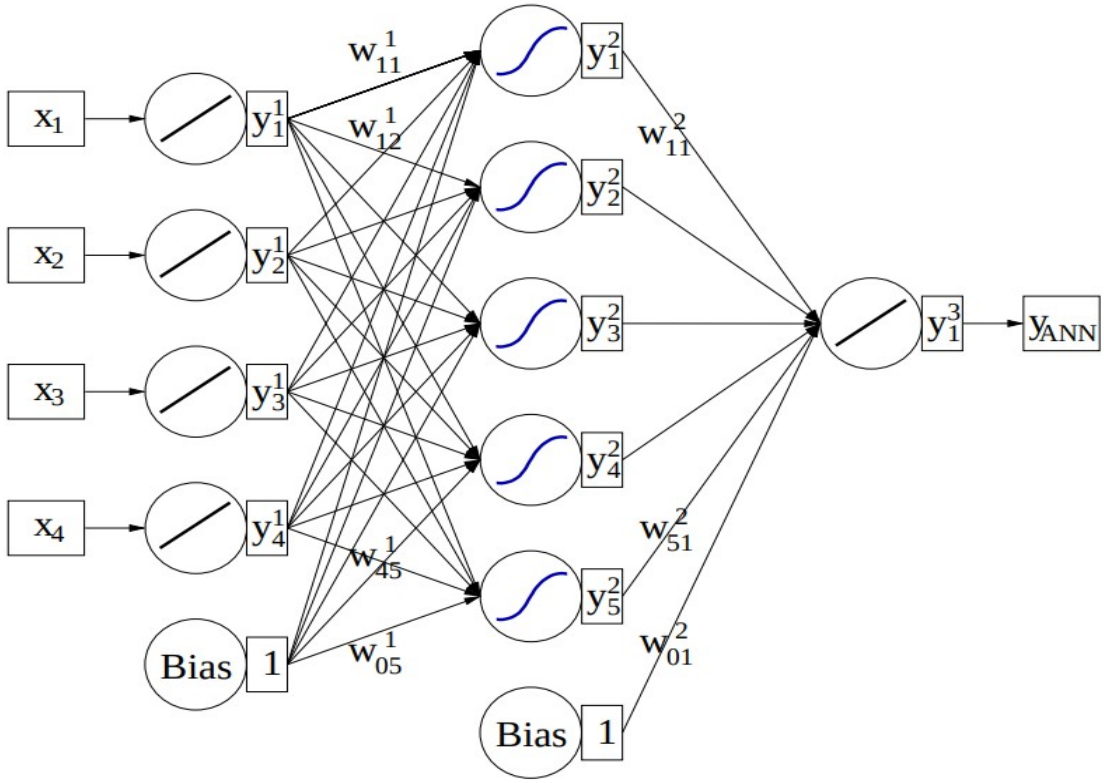
- Ακτινική (Radial) $a(x) = e^{-\frac{x^2}{2}}$



Μέθοδοι Εκπαίδευσης Νευρωνικού Δικτύου

- Οπισθοδρόμηση (Back-propagation)

Ο πιο συνηθισμένος αλγόριθμος για τον υπολογισμό των βαρών για την βελτιστοποίηση της απόδοση ενός νευρωνικού δικτύου είναι η οπισθοδρόμηση (Back-Propagation). Ανήκει στην οικογένεια των μεθόδων μάθησης υπό επίβλεψη, όπου είναι γνωστή η επιθυμητή έξοδος για κάθε συμβάν εισόδου. Για ένα νευρωνικό δίκτυο για παράδειγμα με ένα ενιαίο κρυφό στρώμα, συνάρτηση ενεργοποίησης την Tanh και γραμμική συνάρτηση ενεργοποίησης στο στρώμα εξόδου:



Η έξοδος του δικτύου y_{ANN} δίνεται από:

$$y_{\mathrm{ANN}} = \sum_{j=1}^{n_{\mathrm{h}}} y_j^{(2)} w_{j1}^{(2)} = \sum_{j=1}^{n_{\mathrm{h}}} \tanh\left(\sum_{i=1}^{n_{\mathrm{var}}} x_i w_{ij}^{(1)}\right) \cdot w_{j1}^{(2)}$$

Όπου n_{var} και n_h είναι ο αριθμός των νευρώνων στο στρώμα εισόδου και στο κρυφό στρώμα, αντίστοιχα, $w^{(1)}_{ij}$ είναι το βάρος μεταξύ του νευρώνα του επιπέδου εισόδου i και του νευρώνα από το κρυφό επίπεδο j και $w^{(2)}_{j1}$ είναι το βάρος μεταξύ του κρυφού στρώματος του νευρώνα j και του νευρώνα εξόδου. Για την συνάρτηση σύναψης (κ) χρησιμοποιήθηκε η απλή αθροιστική.

Κατά τη διαδικασία εκμάθησης το δίκτυο τροφοδοτείται με N εκπαιδευτικά συμβάντα $x_a = (x_1, \ldots, x_{nvar})_a$, a = 1, . . . , N. Για κάθε εκπαιδευτικό συμβάν υπολογίζεται και συγκρίνεται η έξοδος του νευρωνικού δικτύου $y_{ANN,a}$ με την επιθυμητή έξοδο π.χ. για ταξινόμηση $y'_{ANN,\alpha} \in \{1, 0\}$ (1 για το σήμα και 0 για το υπόβαθρο). Μια συνάρτηση σφάλματος E, μετρά τη συμφωνία της απόκρισης δικτύου με την επιθυμητή (Supervised Learning), και ορίζεται από την σχέση:

$$E(\mathbf{x}_1, \ldots, \mathbf{x}_N|\mathbf{w}) = \sum_{a=1}^{N} E_a(\mathbf{x}_a|\mathbf{w}) = \sum_{a=1}^{N} \frac{1}{2} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right)^2$$

Όπου **w** δηλώνει το σύνολο των ρυθμιζόμενων βαρών του δικτύου. Το βέλτιστο σύνολο των βαρών για την ελαχιστοποίηση την συνάρτηση σφάλματος μπορεί να βρεθεί μέσα από την μεθόδο της απότομης κατάβασης (steepest or gradiant descent). Ξεκινώντας από ένα τυχαίο σύνολο $\mathbf{w}^{(\rho)}$ τα βάρη ενημερώνονται με μικρά βήματα ως προς τον χώρο των βαρών προς την κατεύθυνση $-\nabla_{\mathbf{w}}E$, όπου η συνάρτηση σφάλματος E μειώνεται πιο γρήγορα. Ο θετικός αριθμός η μπροστά από το ανάδελτα ονομάζεται ρυθμός μάθησης (Learning Rate) και η μεταβολή του δίνει το πόσο μεγάλα/μικρά βήματα βελτίωσης θα κάνει ο αλγόριθμος.

$$\mathbf{w}^{(\rho+1)} = \mathbf{w}^{(\rho)} - \eta \nabla_{\mathbf{w}} E$$

Τα βάρη που συνδέονται με το στρώμα εξόδου ενημερώνονται μέσω της σχέσης:

$$\Delta w_{j1}^{(2)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{j1}^{(2)}} = -\eta \sum_{a=1}^{N} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right) y_{j,a}^{(2)}$$

και τα βάρη που συνδέονται με τα κρυφά στρώματα ενημερώνονται από την σχέση:

$$\Delta w_{ij}^{(1)} = -\eta \sum_{a=1}^{N} \frac{\partial E_a}{\partial w_{ij}^{(1)}} = -\eta \sum_{a=1}^{N} \left(y_{\mathrm{ANN},a} - \hat{y}_a\right) y_{j,a}^{(2)} (1 - y_{j,a}^{(2)}) w_{j1}^{(2)} x_{i,a}$$

Αυτή η μέθοδος εκπαίδευσης του δικτύου ονομάζεται μαζική εκμάθηση (Bulk Learning), καθώς το άθροισμα των σφαλμάτων όλων των δεδομένων εκπαίδευσης χρησιμοποιείται για την ενημέρωση των βαρών. Μια εναλλακτική επιλογή είναι η λεγόμενη διαδικτυακή εκμάθηση (Online Learning), όπου η ενημέρωση των βαρών γίνεται σε κάθε ένα γεγονός ξεχωριστά. Οι ενημερώσεις των βαρών σε αυτή την περίπτωση λαμβάνονται από τις παραπάνω εξισώσεις αφαιρώντας τα αθροίσματα. Είναι πολύ σημαντικό για την διαδικτυακή εκμάθηση να χρησιμοποιήσετε δείγμα εκπαίδευσης με αρκετά καλή τυχαιότητα. Η διαδικτυακή εκμάθηση είναι η μέθοδος που χρησιμοποιήθηκε για τα νευρωνικά δίκτυα αυτης της διπλωματικής.

- Μέθοδος BFGS

Η μέθοδος BFGS (Broyden-Fletcher-Goldfarb-Shannon) διαφέρει από την μέθοδο οπισθοδρόμησης (BP) στην χρήση της δεύτερης παραγώγου για τη συνάρτηση σφάλματος ως προς την προσαρμογή των βαρών. Ο αλγόριθμος BFGS που αποτελείται από τέσσερα κύρια βήματα.

I. Υπολογισμός των διανυσμάτων D και Y. Το διάνυσμα των μεταβολών βάρους D αντιπροσωπεύει την εξέλιξη μεταξύ μιας επανάληψης του αλγορίθμου από το βήμα (k−1) στην επόμενη βήμα (k). Κάθε βάρος συνάρτησης σύναψης αντιστοιχεί σε ένα στοιχείο αυτού του διανύσματος. Το διάνυσμα Y είναι το διάνυσμα των σφαλμάτων κλίσης.

$$\begin{aligned} D_i^{(k)} &= w_i^{(k)} - w_i^{(k-1)} \\ Y_i^{(k)} &= g_i^{(k)} - g_i^{(k-1)} \end{aligned}$$

Όπου i είναι ο δείκτης των συνάψεων, g_i είναι η i-ωστή κλίση της i-ωστής συναψης, w_i είναι το βάρος της i-ωστής σύναψη και το k δηλώνει τον μετρητή επανάληψης.

II. Προσέγγιση του αντιστρόφου Hessian πίνακα, H^{-1}, στην k επανάληψη από την σχέση:

$$H^{-1(k)} = \frac{D \cdot D^T \cdot (1 + Y^T \cdot H^{-1(k-1)} \cdot Y)}{Y^T \cdot D} - D \cdot Y^T \cdot H + H \cdot Y \cdot D^T + H^{-1(k-1)}$$

III. Υπολογισμός του διανύσματος των μεταβολών βάρους D από την σχέση:

$$D^{(k)} = -H^{-1(k)} \cdot Y^{(k)}$$

IV. Υπολογισμός ενός νέου διανύσματος βαρών D εφαρμόζοντας τον αλγόριθμο αναζήτησης γραμμής (line search alogorithm). Στη αναζήτηση γραμμής η συνάρτηση σφάλματος (E) προσεγγίζεται τοπικά με μια παραβολή. Ο αλγόριθμος υπολογίζει την δεύτερη παράγωγο και καθορίζει το σημείο όπου αναμένεται το ελάχιστο αυτης της παραβολής. Το συνολικό σφάλμα αξιολογείται για αυτό ακριβώς το σημείο. Στη συνέχεια, ο αλγόριθμος αξιολογεί σημεία κατά μήκος της γραμμής που ορίζεται από την κατεύθυνση της κλίσης στο χώρο των βαρών για να βρεί το απόλυτο ελάχιστο. Τα βάρη στο ελάχιστο χρησιμοποιούνται για την επόμενη επανάληψη. Η παράμετρος εκμάθησης, η οποία ορίζει το πόσο αλλάζουν τα βάρη ανά εποχή κατά μήκος της γραμμής όπου βρίσκετε το ελάχιστο, πολλαπλασιάζεται με τον ρυθμό μάθησης όσο το σφάλμα εκπαίδευσης του νευρικού διχτυού με τα αλλαγμένα βάρη είναι κάτω από αυτό με τα αμετάβλητα βάρη. Εάν το σφάλμα εκπαίδευσης του νέου νευρικού δικτύου είναι μεγαλύτερο ως προς την αρχική παράμετρο εκμάθησης, διαιρείται με το ρυθμό εκμάθησης μέχρις ότου το σφάλμα εκπαίδευσης να γίνει μικρότερο. Ο βηματικός και κατά προσέγγιση υπολογισμός του $H^{-1(k)}$ γίνεται λιγότερο ακριβής όσο αυξανεται ο αριθμός των επαναλήψεων.

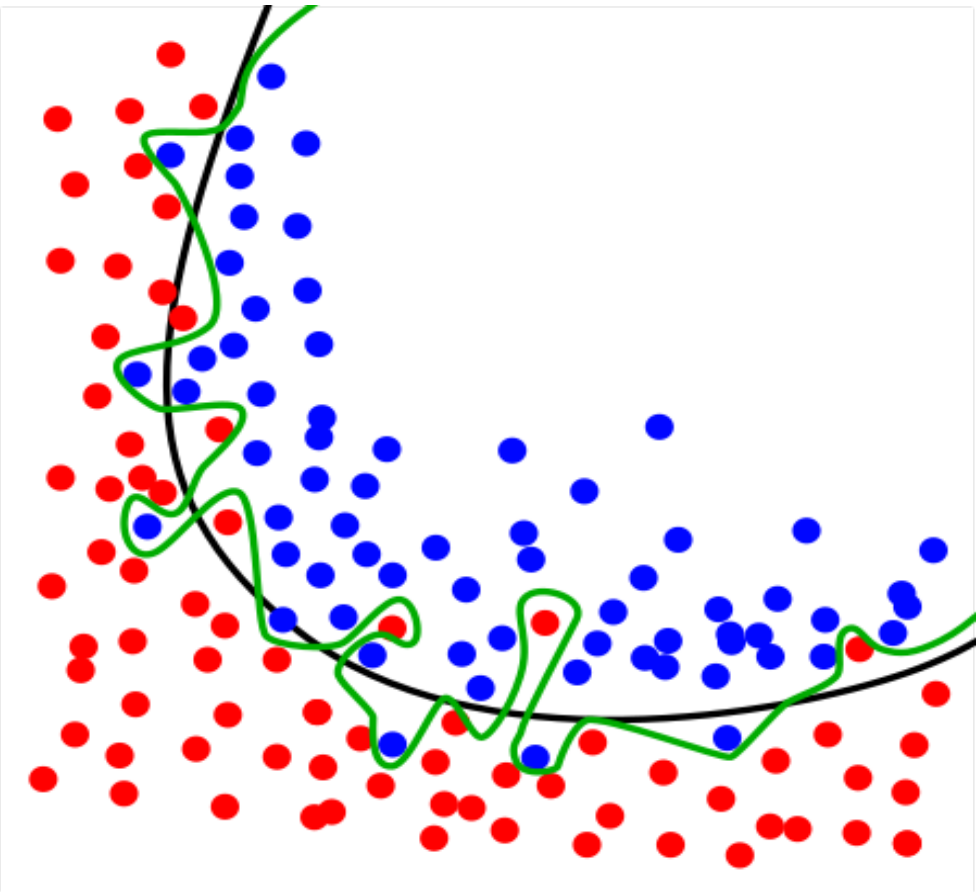
Το πλεονέκτημα της μεθόδου BFGS σε σύγκριση με την μέθοδο οπισθοδρόμησης (BP) είναι ο μικρότερος αριθμός επαναλήψεων που χρειάζεται. Ωστόσο, επειδή ο υπολογιστικός χρόνος για μια επανάληψη είναι ανάλογος του τετραγώνου του αριθμού των συνάψεων, τα μεγάλα δίκτυα τιμωρούνται ιδιαίτερα. [2] [11]

1.2.3 Το Φαινόμενο της Υπερεκπαίδευσης (Over-Training)

Στη μαθηματική μοντελοποίηση, το φαινόμενο της υπερεκπαίδευσης ορίζεται ως "η παραγωγή μιας ανάλυσης που αντιστοιχεί πολύ στενά ή επακριβώς σε ένα συγκεκριμένο σύνολο εκπαίδευσης, και ως εκ τούτου μπορεί να μην ταιριάζει σε επιπλέον καινούρια δεδομένα ή να μην είναι σε θέση να προβλέψει αξιόπιστα μελλοντικές παρατηρήσεις". Ένα υπερεκπαιδευμένο μοντέλο είναι ένα μαθηματικό μοντέλο που περιέχει περισσότερες παραμέτρους από αυτές που μπορούν να δικαιολογηθούν από τα δεδομένα. Η ουσία της υπερεκπαίδευσης είναι να έχουν εξαχθεί κάποια από τα δεδομένα του υποβάθρου (δηλαδή του θορύβου) από το σύνολο εκπαίδευσης σαν μια παραλλαγή η οποία να αντιπροσώπευε την υποκείμενη δομή του μοντέλου.

Η πιθανότητα υπερεκπαίδευσης ή αλλίως στα αγγλικά “Over-Fitting” υπάρχει επειδή το κριτήριο για την επιλογή του μοντέλου δεν είναι το ίδιο με το κριτήριο που χρησιμοποιείται για να κριθεί η

καταλληλότητά του. Για παράδειγμα, ένα μοντέλο μπορεί να επιλεγεί μεγιστοποιώντας την απόδοσή του σε κάποιο σύνολο εκπαίδευσης, και ωστόσο η καταλληλότητά του μπορεί να καθοριστεί από την ικανότητά του να αποδίδει καλά σε "αόρατα" δεδομένα εντός του συνόλου εκπαίδευσης. Τότε υπερεκπαίδευση εμφανίζεται όταν το μοντέλο αρχίζει να «απομνημονεύει» δεδομένα εκπαίδευσης αντί να «μαθαίνει» και να γενικεύει από μια τάση. Για παράδειγμα, εάν ο αριθμός των παραμέτρων είναι ίδιος ή μεγαλύτερος από τον αριθμό των παρατηρήσεων, τότε ένα μοντέλο μπορεί να προβλέψει τέλεια τα δεδομένα εκπαίδευσης απλώς απομνημονεύοντας τα δεδομένα στο σύνολό τους. Ωστόσο, ένα τέτοιο μοντέλο τυπικά θα αποτύχει σοβαρά σε οποιαδήποτε νέα προβλεψη διότι θα έχει χάσει την ικανότητα γενίκευσης. Παρακάτω είναι μια κλασική εικόνα για την κατανόηση ενός μοντέλου που έχει υπερεκπαιδευτεί, με πράσινο χρώμα, σε αντίθεση με ένα μοντέλο που ακόμα έχει την ικανότητα γενίκευσης, μαύρο χρώμα.



Η πιθανότητα υπερεκπαίδευσης δεν εξαρτάται μόνο από τον αριθμό των παραμέτρων και των δεδομένων, αλλά και από τη συμβατότητα της δομής του μοντέλου (Ενδυναμομένο Δέντρο Απόφασης, Νευρωνικό Δίκτυο, Γραμμικός Ταξινομητής) με το σχήμα των δεδομένων και το αναμενόμενο επίπεδο θορύβου ή υποβάθρου στα δεδομένα. Ακόμη και όταν το προσαρμοσμένο μοντέλο δεν έχει υπερβολικό αριθμό παραμέτρων, είναι αναμενόμενο ότι η προσαρμοσμένη σχέση που προέκυψε από τα δεδομένα εκπαίδευσης θα φαίνεται να αποδίδει λιγότερο καλά σε ένα νέο σύνολο δεδομένων από ό,τι στο σύνολο εκπαίδευσης.

Η βάση ορισμένων τεχνικών για τη μείωση της πιθανότητας ή του ποσοστού της υπερεκπαίδευσης είναι είτε η ρητή τιμωρία υπερβολικά πολύπλοκων μοντέλων ή η δοκιμή της ικανότητας του μοντέλου να γενικεύει αξιολογώντας την απόδοσή του σε ένα σύνολο δεδομένων που δεν χρησιμοποιούνται για εκπαίδευση (Testing Set). Την δεύτερη μέθοδο θα χρησιμοποιήσουμε σε αυτη την διπλωματική εργασία. [8]

Κεφάλαιο 2

ΕΦΑΡΜΟΓΕΣ

"THAT'S NOT THE WAY IT WORKS. DIFFERENT PEOPLE CRUNCHING THE NUMBERS SHOULD NOT GET DIFFERENT RESULTS."

2.1 Εργαλία Ανάλυσης Δεδομένων

Ξεκινώντας το 2ο μέρος της εργασίας θα ήταν καλό να γίνει μια αναφορά στα ηλεκτρονικά εργαλεία που χρησιμοποιήθηκαν για να γίνει η ανάλυση δεδομένων την οποία και θα αναλύσουμε διεξοδικά. Αρχικά έγινε χρήση του ROOT το οποίο είναι ένα αντικειμενοστραφές πρόγραμμα όπως και βιβλιοθήκη δεδομένων που αναπτύχθηκε από το CERN. Είναι σχεδιασμένο για ανάλυση δεδομένων σωματιδιακής φυσικής και περιέχει πολλά χαρακτηριστικά ειδικά για αυτό το πεδίο. Η διεπαφή που μπορεί να έχει ο χρήστης με το ROOT είναι είτε μέσω της γλώσσας C++ είτε μέσω της γλώσσας Python από το περιβάλλον pyROOT το οποίο και επιλέχθηκε.

Η ανάλυση όπως και τα μοντέλα παλινδρόμησης υλοποιήθηκαν μέσω του TMVA (Toolkit for Multivariate Analysis) το οποίο παρέχει ένα ενσωματωμένο περιβάλλον ROOT για επεξεργασία, παράλληλη αξιολόγηση και τεχνικές παλινδρόμησης πολλών μεταβλητών. Όλες οι τεχνικές πολλαπλών μεταβλητών στο TMVA ανήκουν στην οικογένεια των αλγορίθμων μάθησης υπό επίβλεψη. Γίνεται χρήση δεδομένων εκπαίδευσης, για τα οποία γνωρίζουμε την επιθυμητή έξοδο (Label), έτσι ώστε να προσδιοριστεί η συνάρτηση αντιστοίχισης που περιγράφει μια προσέγγιση της υποκείμενης συναρτησιακής συμπεριφοράς η οποία με την σειρά της ορίζει την τιμή στόχου. Η συνάρτηση αντιστοίχισης μπορεί να περιέχει διάφορους βαθμούς προσεγγίσεων και μπορεί να είναι μια μεμονωμένη καθολική συνάρτηση ή ένα σύνολο τοπικών μοντέλων. Το TMVA έχει σχεδιαστεί ειδικά για τις ανάγκες των εφαρμογών της φυσικής υψηλών ενεργειών (High Energy Physics), αλλά δεν περιορίζεται σε αυτές.

Το πακέτο χρησιμοποιήθηκε για να Μοντελοποιήσει:

- Γραμμική Διακριτική Ανάλυση (Linear Discrimination Analysis)
- Ενισχυμένα Δέντρα Απόφασης (Boosted Descision Trees)
- Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks)

2.2 Από τα Δεδομένα της Monte Carlo Προσομοίωσης στα Δεδομένα για την Παλινδρόμηση

Όλη η ανάλυσή μας ξεκινάει από τα δεδομένα της Monte Carlo προσομοίωσης τα οποία είναι αποθηκευμένα στο αρχείο: “jetsCalibs_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root”. Μέσα σε αυτό το αρχείο είναι αποθηκευμένα σε μορφή δέντρου (TTree) τα δεδομένα που θέλουμε να αναλύσουμε.

➔ Τι είναι τα Δέντρα (TTree) και γιατί τα χρειαζόμαστε?

Για την περίπτωση που θέλουμε να αποθηκεύσουμε μεγάλες ποσότητες αντικειμένων ίδιας κατηγορίας, το ROOT έχει σχεδιάσει τις κλάσεις TTree ειδικά για αυτόν τον σκοπό. Η κλάση TTree

είναι φτιαγμένη για να μειώνει το χώρο που καταλαμβάνουν τα δεδομένα και επιπλέον βελτιώνει την ταχύτητα πρόσβασής μας σε αυτά.

Ένα TTree είναι σε θέση να αποθηκεύσει όλα τα είδη δεδομένων, όπως αντικείμενα, πίνακες ή ιστογράμματα. Όταν χρησιμοποιούμε ένα TTree, γεμίζουμε τις διακλαδώσεις (Branches) και τα φύλλα (Leafs) με δεδομένα τα οποία εγγράφονται στο δίσκο όταν το δέντρο είναι γεμάτο. Είναι σημαντικό να συνειδητοποιήσουμε ότι κάθε αντικείμενο δεν γράφεται μεμονωμένα, αντιθέτως τα αντικείμενα συλλέγονται και γράφονται ομαδικά. Σε αυτό το σημείο το TTree εκμεταλλεύεται τη συμπίεση και παράγει ένα πολύ μικρότερο αρχείο από ό,τι αν τα αντικείμενα ήταν γραμμένα μεμονωμένα.

Το TTree χρησιμοποιείται επίσης για τη βελτιστοποίηση της πρόσβασης μας στα δεδομένα. Ένα δέντρο χρησιμοποιεί μια ιεραρχία διακλαδώσεων (Branches) και κάθε κλάδος μπορεί να διαβαστεί ανεξάρτητα από οποιονδήποτε άλλο κλάδο. Τώρα, ας υποθέσουμε ότι τα Px και Py είναι δεδομένα και θα θέλαμε να υπολογίσουμε το Px^2+Py^2 για κάθε γεγονός όπως και να κάνουμε ένα ιστόγραμμα του αποτελέσματος.

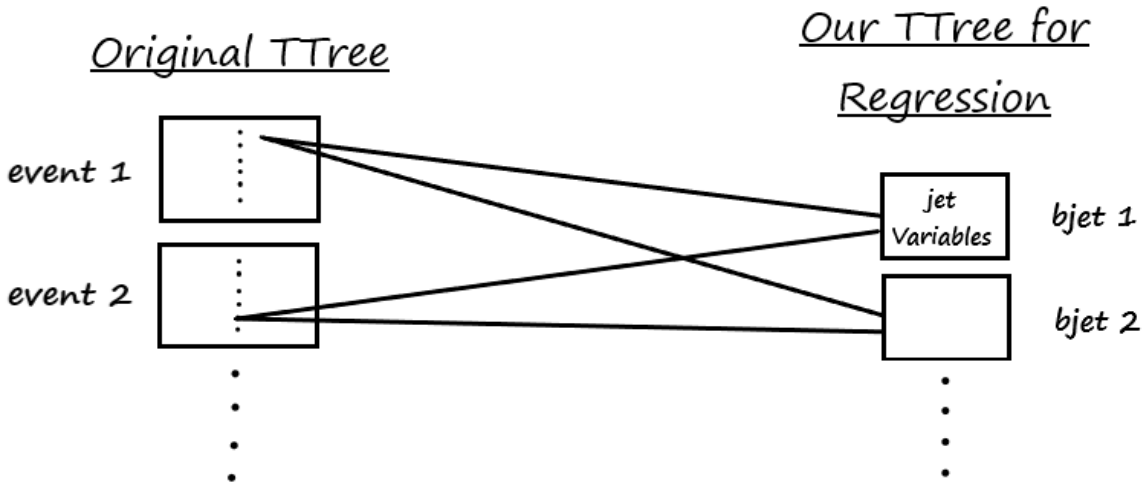
Αν είχαμε αποθηκεύσει ένα εκατομμύρια γεγονότα (events) χωρίς ένα TTree θα έπρεπε να:

- Διαβάσουμε κάθε γεγονός στο σύνολό του στη μνήμη
- Εξάγουμε τα Px και Py από το συμβάν
- Υπολογίσουμε το άθροισμα των τετραγώνων
- Συμπληρώσουμε το ιστόγραμμα

Θα έπρεπε να το κάνουμε αυτό ένα εκατομμύριο φορές! Αυτό είναι πολύ χρονοβόρο και πραγματικά δεν χρειάζεται να διαβάζουμε ολόκληρο το κάθε γεγονός ένα εκατομμύριο φορές. Το μόνο που χρειαζόμαστε είναι δύο μικρά μέλη δεδομένων (το Px και το Py). Από την άλλη πλευρά, εάν χρησιμοποιήσουμε ένα δέντρο με έναν κλάδο που να περιέχει Px και έναν άλλο κλάδο που να περιέχει Py, μπορούμε να διαβάσουμε όλες τις τιμές των Px και Py διαβάζοντας μόνο τους κλάδους Px και Py. Αυτό είναι που κάνει τη χρήση του TTree πολύ ελκυστική.

➔ Τι θα κρατήσουμε από τα Δεδομένα της Προσομοίωσης?

Σε αυτό το σημείο χρειάζεται να φτιάξουμε ένα νέο δέντρο (TTree) από το αρχικό δέντρο της προσομοίωσης στο οποίο οι αδρονικοί πίδακες (b-jets) με τις παραμέτρους που θέλουμε να μελετήσουμε να είναι οι είσοδοι και όχι τα γεγονότα (events). Το νέο αρχείο ονομάστηκε file.root και το νέο δέντρο (TTree) που θα χρησιμοποιήσουμε για την παλινδρόμηση ονομάστηκε my_tree.

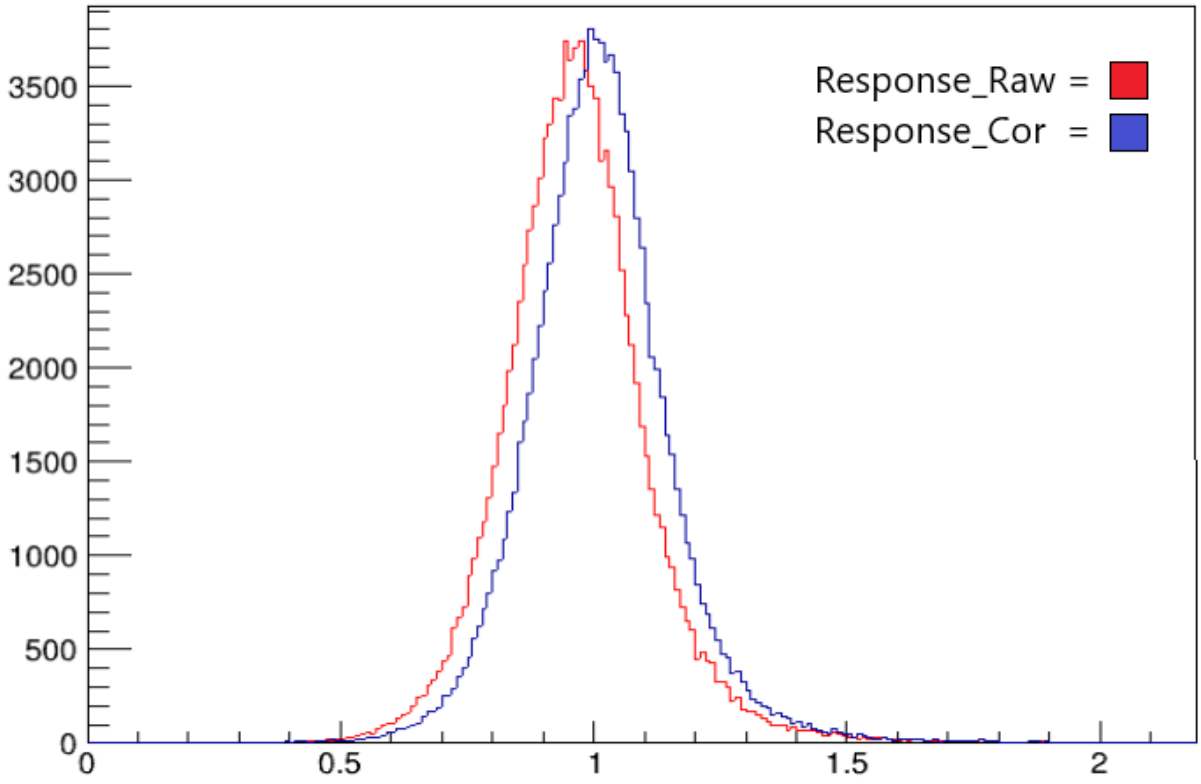


Οι μεταβλητές που θα χρησιμοποιήσουμε και θα κρατήσουμε από το αρχικό στο τελικό μας δέντρο πέραν αυτών που αναφέραμε στο κεφάλαιο 1.1.5 θα είναι επιπλέον η bgenJetPt και η bgenJetEta οι οποίες είναι οι generated τιμές για το Pt και το Eta. Τέλος θα γίνει μέσω των τιμών του Eta και του Phi ο υπολογισμός του ΔR_{min} και θα αποθηκευτεί σε ένα ξεχωριστό Branch με όνομα DR_min. Αναλυτικά ο κώδικας βρίσκεται στο Παράρτημα 1.

$$\Delta R_{min} = [(\Delta\eta)^2 + (\Delta\varphi)^2]^{1/2}$$

2.3 Υπολογισμός της Απόκρισης (Response)

Τώρα είμαστε σε θέση να κάνουμε τον πρώτο υπολογισμό της απόκρισης (Response) για το bjetPt και το bjetRawPt. Όπως αναφέραμε και στην παράγραφο 1.1.4 η απόκριση υπολογίζεται διαιρώντας τις τιμές από το Branch του “bjetPt” ή του “bjetRawPt” με τις παραγώμενες (generated) τιμές που βρίσκονται στο Branch “bgenJetPt”. Αν τα αποτελέσματα ήταν “τέλεια” το ιστόγραμμα των τιμών που θα προέκυπτε θα ήταν μια συνάρτηση δέλτα στο 1. Δυστυχώς όμως δεν είναι έτσι η κατάσταση. Όπως φαίνεται και παρακάτω η εικόνα των Response και για τις δύο τιμές (bjetPt/bgenjetPt bjetRawPt/bgenjetPt) ακολουθεί μια γκαουσιανού τύπου κατανομή κοντά στο 1 και ειδικά για το “bjetRawPt” η κατανομή φαίνεται να είναι μετατοπισμένη λίγο προς τα αριστερά. Αναλυτικά ο κώδικας για το πως φτιάχτηκαν τα Response βρίσκεται στο Παράρτημα 2.



➔ Πως θα χρησιμοποιούμε τα Ιστογράμματα των Αποκρίσεων (Response)?

Αυτές τις γκαουσιανές κατανομές των Response είναι που θα χρησιμοποιούμε συνεχώς στα παρακάτω βήματα για να αξιολογήσουμε το πόσο "καλός” ή όχι είναι ένας παλινδρομητής (Γραμμικός, Δέντρο Απόφασης ή Νευρωνικό Δίκτυο). Αυτό θα το κάνουμε μέσα από ένα

γκαουσιανό fit που θα εφαρμόζουμε πάνω σε κάθε ιστόγραμμα απόκρισης. Έπειτα μέσα από αυτό το fit θα υπολογίζουμε τα μέτρα διασποράς (πόσο κοντά ή μακρία είμαστε σε μια συνάρτηση δέλτα) μέσω 3ων παραμέτρων τις οποίες, μαζί με τα σφάλματά τους, θα τοποθετούμε σε ένα τελικό διάγραμμα (Results). Αναλυτικά οι παράμετροι που θα χρησιμοποιήσουμε είναι:

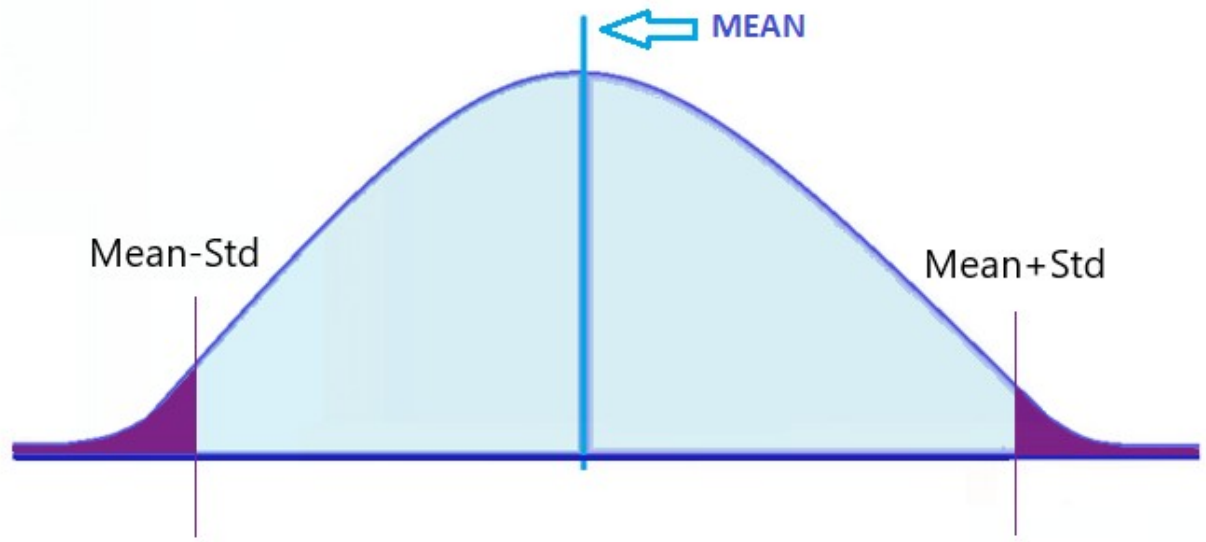
- Η Μέση Τιμή (Mean)

Η Μέση τιμή αποτελεί την πιο κοινή τιμή σε μια συλλογή δεδομένων. Είναι ένα μέτρο της κεντρικής τάσης μιας κατανομής πιθανότητας και αναφέρεται επίσης και ως αναμενόμενη τιμή (expected value).

- Η Τυπική Απόκλιση (Standard Deviation ή Std) πάνω σε όλη την κατανομή. Δηλαδή συνυπολογίζει και τις “ουρές” της κατανομής

Η τυπική απόκλιση είναι ένα μέτρο της ποσότητας διακύμανσης ή διασποράς ενός συνόλου τιμών. Μια χαμηλή τυπική απόκλιση υποδηλώνει ότι οι τιμές τείνουν να είναι κοντά στη μέση τιμή (ή αλλιώς αναμενόμενη τιμή) του συνόλου, ενώ μια υψηλή τυπική απόκλιση υποδηλώνει ότι οι τιμές κατανέμονται σε ένα ευρύτερο εύρος.

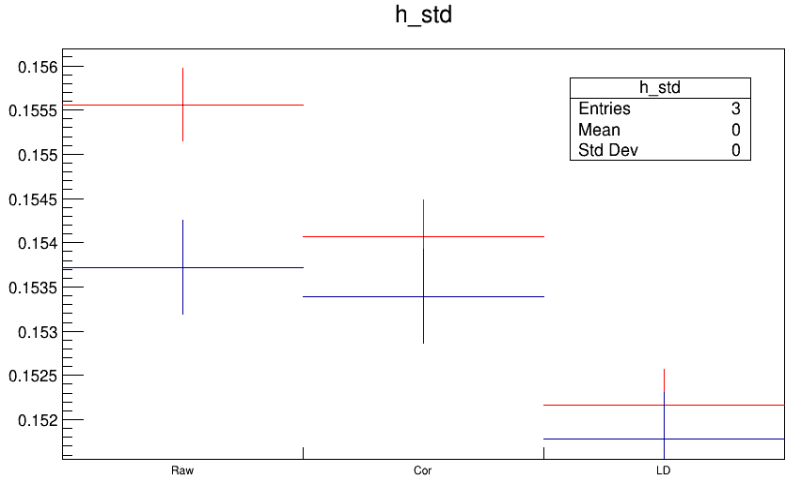
- Η Παράμετρος σίγμα (sigma) η οποία είναι η τυπική απόκλιση σε πεπερασμένο εύρος της κατανομής (truncated fit) μεταξύ των τιμών: **mean ± std**

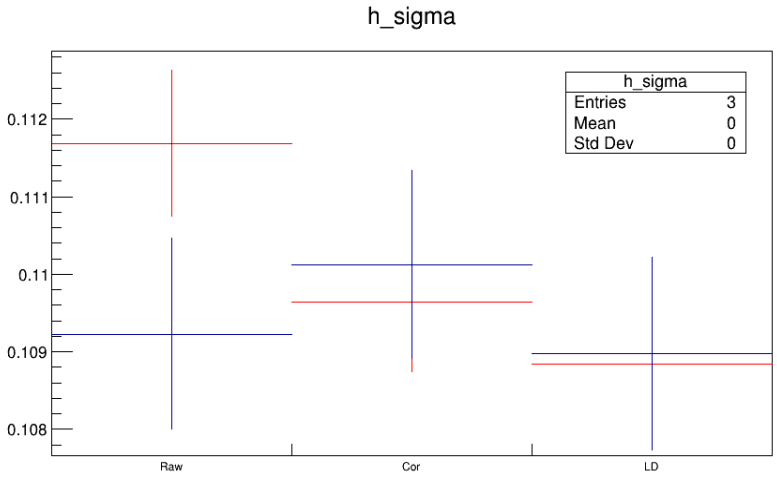


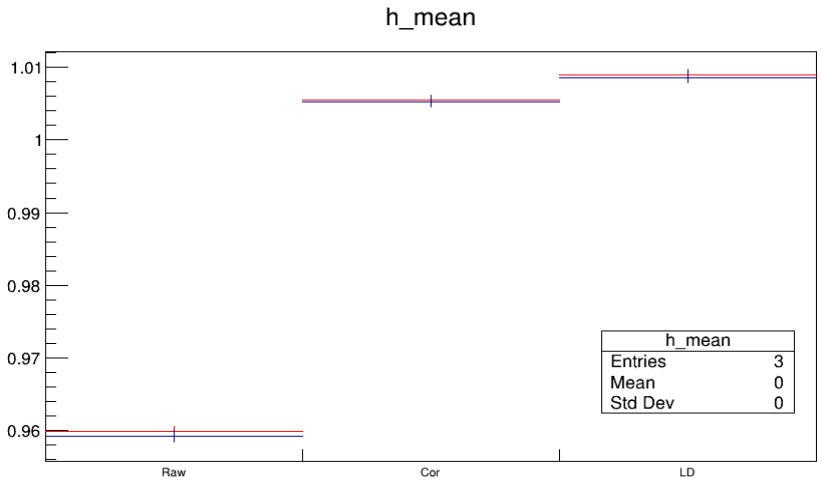
2.4 Γραμμικός Ταξινομητής (LD)

Ας εφαρμόσουμε λοιπόν όλα τα παραπάνω για την πρώτη και πιο απλή από τις μεθόδους μας, τον Γραμμικό Ταξινομητή (LD). Όλα ξεκινάνε από το TMVA όπου μέσω της κλάσης factory ορίζουμε πως θα γίνει χρήση του LD για παλινδρόμηση και εισάγουμε τις μεταβλητές (AddVariable) που θα χρησιμοποιήσουμε για την παλινδρόμηση από το δέντρο (TTree) my_tree. Επιπλέον ορίζουμε τον στόχο (Target) της παλινδρόμησης να είναι η μεταβλητή “bgenJetPt” βάση της οποίας θα γίνει η διαδικασία της μάθησης υπό επίβλεψη. Έπειτα λέμε στην μέθοδο την αναλογία βάση της οποίας τα δεδομένα θα χωριστούν σε δείγματα εκπαίδευσης (Training) και Ελέγχου (Testing) που στην περίπτωσή μας είναι 70% Training – 30% Testing. Αναλυτικά ο κώδικας του TMVA για τον Γραμμικό Ταξινομητή (LD) βρίσκεται στο Παράρτημα 3.

Το TMVA σαν αποτέλεσμα δημιουργεί ένα νέο αρχείο με το όνομα TMVA_LD.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στο Branch με το όνομα LD. Στόχος μας τώρα είναι να συγκρίνουμε το Response από το Branch LD με τα Response από το Raw και το Corrected. Επιπλέον όμως πρέπει να ελέγξουμε αν υπάρχει διαφορά στα αποτελέσματα (Results) μεταξύ συνόλου Εκπαίδευσης (Training) και συνόλου Δοκιμών (Testing). Τα διαγράμματα των αποτελεσμάτων θα είναι ως προς την Μέση Τιμή (Mean), την Τυπική Απόκλιση (std) και το Σίγμα (sigma). Ο κώδικας για την δημιουργεία των Response ιστογραμμάτων, του fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και την δημιουργεία των παρακάτω διαγραμμάτων με τα αποτελέσματα (Results) βρίσκεται στο Παράρτημα 4.







Μια πρώτη ματιά στα αποτελέσματα του γραμμικού Ταξινομητή μας λέει τα εξής. Αρχικά όπως φαίνεται σε κάθε περίπτωση (Raw-Cor-LD) είτε για το mean, είτε για το std, είτε για το sigma υπάρχει μια μικρή διαφορά στα αποτελέσματα μεταξύ TestTree (Μπλέ) και TrainTree (Κόκκινο). Αυτό είναι λογικό και αναμενόμενο μιας και οι στατιστικές διακυμάνσεις σε κάθε περίπτωση μπορεί να αλλάξουν λίγο το αποτέλεσμα. Εδώ οι διαφορές δεν είναι έντονες το οποίο είναι και λογικό αφού ο Γραμμικός Ταξινομητής πέρα των στατιστικών διακυμάνσεων δεν μπορεί να έχει φαινόμενα υπερεκπαίδευσης.

Η βασική παρατήρηση που πρέπει να γίνει στα παραπάνω αποτελέσματα είναι πως ως προς το std αλλά και ως προς το sigma ο γραμμικός ταξινομητής βελτιώνει τα αποτελέσματα μειώνοντάς τα, δηλαδή “στενεύοντας” την κατανομή της απόκρισης. Αυτό που φαίνεται πως δεν βελτιώνεται είναι το mean όπου ο LD απομακρίνει το αποτέλεσμα από την επιθυμητιτή τιμή του 1 σε σχέση με την Corrected απόκριση (Response), η οποία είναι πιο κοντά στο 1.

2.5 Ενδυναμωμένα Δέντρα Απόφασης (BDT's)

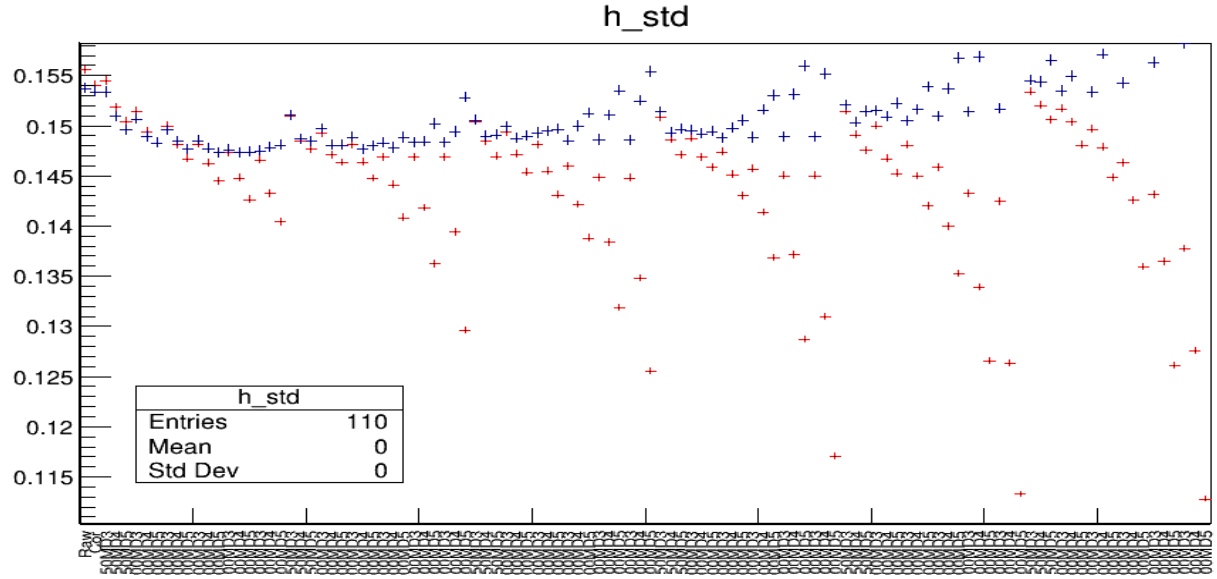
Ας δούμε τώρα τα αποτελέσματα για την δεύτερη από τις μεθόδους μας, τα ενδυναμωμένα δέντρα απόφασης (Boosted Decision Trees). Όλα ξεκινάνε πάλι από το TMVA όπου μέσω της κλάσης factory ορίζουμε πως θα γίνει χρήση BDT's για παλινδρόμηση και εισάγουμε τις μεταβλητές (AddVariable) που θα χρησιμοποιήσουμε για την παλινδρόμηση από το δέντρο (TTree) my_tree. Επιπλέον ορίζουμε τον στόχο (Target) της παλινδρόμησης να είναι η μεταβλητή "bgenJetPt" βάση της οποίας θα γίνει η διαδικασία της μάθησης υπό επίβλεψη. Έπειτα ορίζουμε την αναλογία δείγματος Εκπαίδευσης (Training) και Ελέγχου (Testing) που και στην περίπτωσή των BDT's είναι 70% Training – 30% Testing.

Η βασική διαφορά τώρα που υπάρχει για τα Ενδυναμωμένα Δέντρα Απόφασης, σε σχέση με τον Γραμμικό Ταξινομητή, είναι πως μέσω του TMVA είμαστε σε θέση να αλλάξουμε τις υπερπαραμέτρους του κάθε ενός BDT το οποίο θέλουμε να φτιάξουμε. Αυτές οι υπερπαράμετροι είναι το Shrinkage (Ρυθμός Μάθησης), το NTrees (Αριθμός των Δέντρων) και το MaxDepth (Μέγιστο Βάθος Δέντρου). Επειδή τώρα έχουμε αυτη την δυνατότητα δεν θα φτιάξουμε μόνο ένα BDT ενός τύπου, αλλά θα "σκανάρουμε" όλες αυτές τις υπερπαραμέτρους και θα φτιάξουμε συνολικά 108 Ενδυναμομένα Δέντρα Απόφασης έτσι ώστε να βρούμε το βέλτιστο. Οι τιμές που θα έχουν οι παράμετροι μας θα είναι:

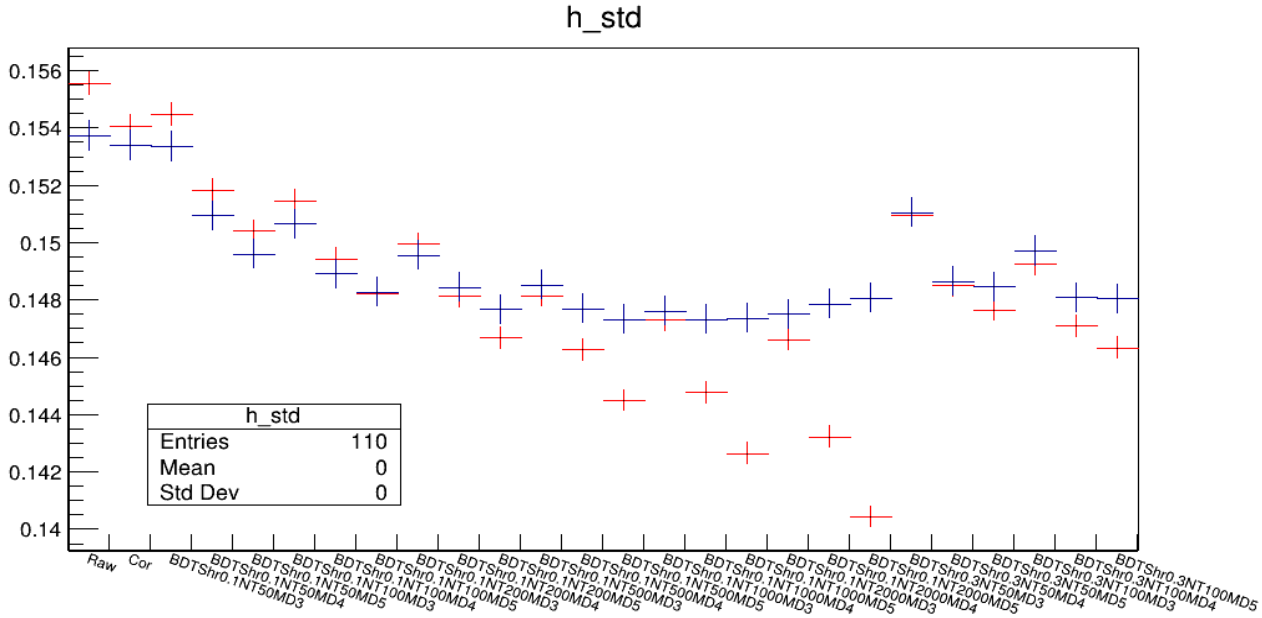
- Shrinkage : [0.1 , 0.3 , 0.5 , 0.7 , 1.0 , 1.5]
- NTrees : [50 , 100 , 200 , 500 , 1000 , 2000]
- MaxDepth : [3 , 4 , 5]

Το TMVA σαν αποτέλεσμα δημιουργεί ένα νέο αρχείο με το όνομα TMVA_BDT.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 108 Branches με ονόματα BDTShrxNTyMDz , όπου x, y, z οι παραπάνω τιμές για το Shrinkage, το NTrees και το MaxDepth. Αναλυτικά ο κώδικας του TMVA για τα Ενδυναμωμένα Δέντρα Απόφασης (BDTs) βρίσκεται στο Παράρτημα 5.

Στόχος μας τώρα είναι να συγκρίνουμε τα Response από τα Branches: BDTShrxNTyMDz μεταξύ τους και ως προς τα Response από το Raw και το Corrected. Επιπλέον όμως πρέπει να ελέγξουμε αν υπάρχει διαφορά στα αποτελέσματα (Results) μεταξύ συνόλου Εκπαίδευσης (Training) και συνόλου Δοκιμών (Testing). Ας μελετήσουμε για αρχή το διάγραμμα του std που προκύπτει:



Όπως και πρίν με μπλέ χρώμα είναι τα αποτελέσματα από το TestTree και με κόκκινο χρώμα τα αποτελέσματα από το TrainTree. Πρίν όμως σχολιάσουμε τα συμπεράσματα που προκύπτουν είναι σημαντικό να μην μας φοβίσει ο άξονας x από το παραπάνω διάγραμμα. Όπως και πριν στις δύο πρώτες θέσεις του άξονα x έχει τοποθετηθεί το αποτέλεσμα από την απόκριση του Raw και του Corrected ιστογράμματος και στις επόμενες 108 θέσεις έχουν τοποθετηθεί όλοι οι συνδυασμοί των υπερπαραμέτρων για τα Ενδυναμομένα Δέντρα Απόφασης. Είναι λογικό να μην μπορούμε να διακρίνουμε ξεκάθαρα σε ποιο BDT απευθύνεται κάθε σημείο όταν έχουμε μαζί και τα 108 δέντρα αλλά με ένα απλό ζούμ στο πρώτο κομμάτι των 30 τιμών του διαγράμματος θα ξεδιαλυνθεί αρκετά η κατάσταση.

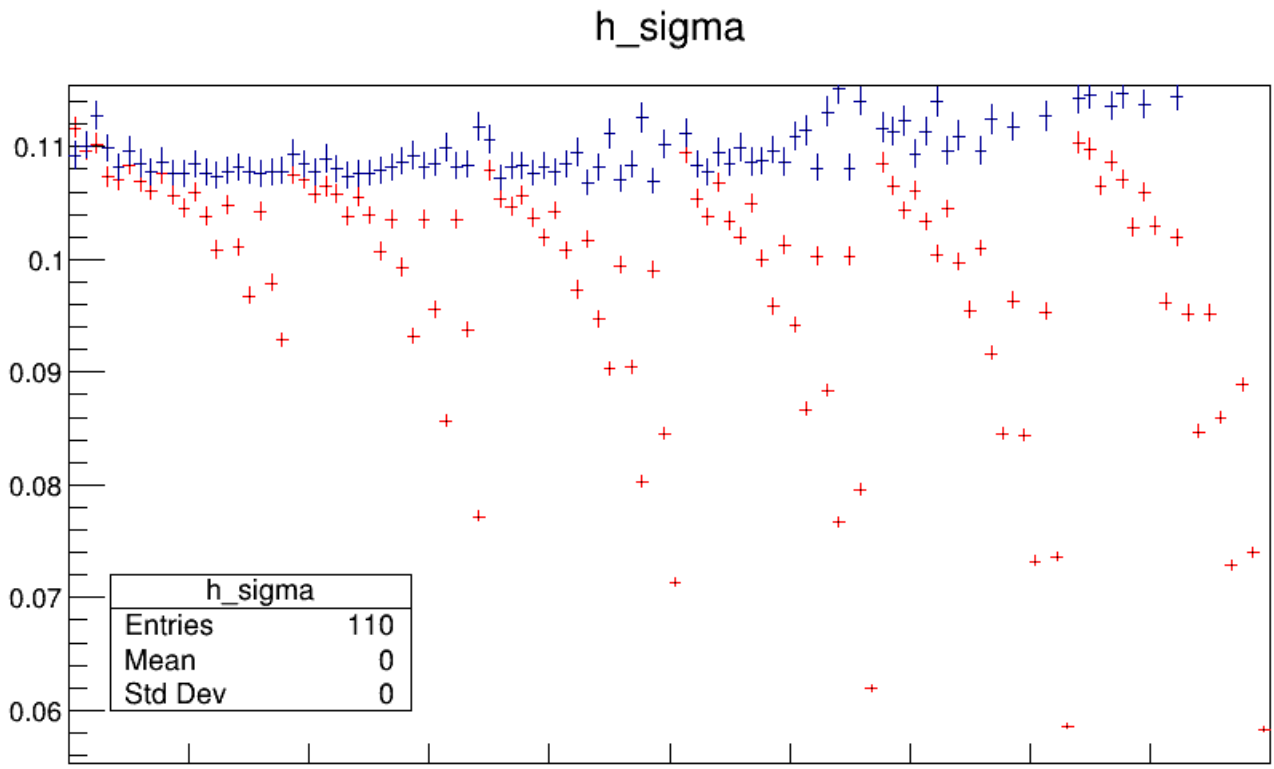


Τώρα μπορεί να φανεί καθαρά πως μετά τις δύο πρώτες τιμές του Raw και του Corrected έχουμε για αρχή με Shrinkage = 0.1 , NTrees = 50 και Max Depth = 3 τα αποτελέσματα του πρώτου BDT. Με βάση την σειρά τώρα του τριπλού for loop με το οποίο και τα δημιουργήσαμε εμφανίζονται όλοι οι άλλοι συνδιασμοί. Είναι βασικό να θυμόμαστε πως ανά τρία διπλανά Results αλλάζει η τιμή του Max Depth μεταξύ των τιμών 3, 4 και 5. Έπειτα αλλάζει η τιμή του NTrees 6 φορές από 50 μέχρι 2000. Και τέλος αλλάζει ανά 18 Results η τιμή του Shrinkage από 0.1 σε 1.5. Για αυτό αν κοιτάξουμε πολύ προσεκτικά το 1ο διάγραμμα με τα 108 δέντρα μπορούμε να δούμε αυτή την περιοδική κίνηση παραβολοειδούς πτώσης 6 φορές, μια για κάθε διαφορετική τιμή του Shrinkage.

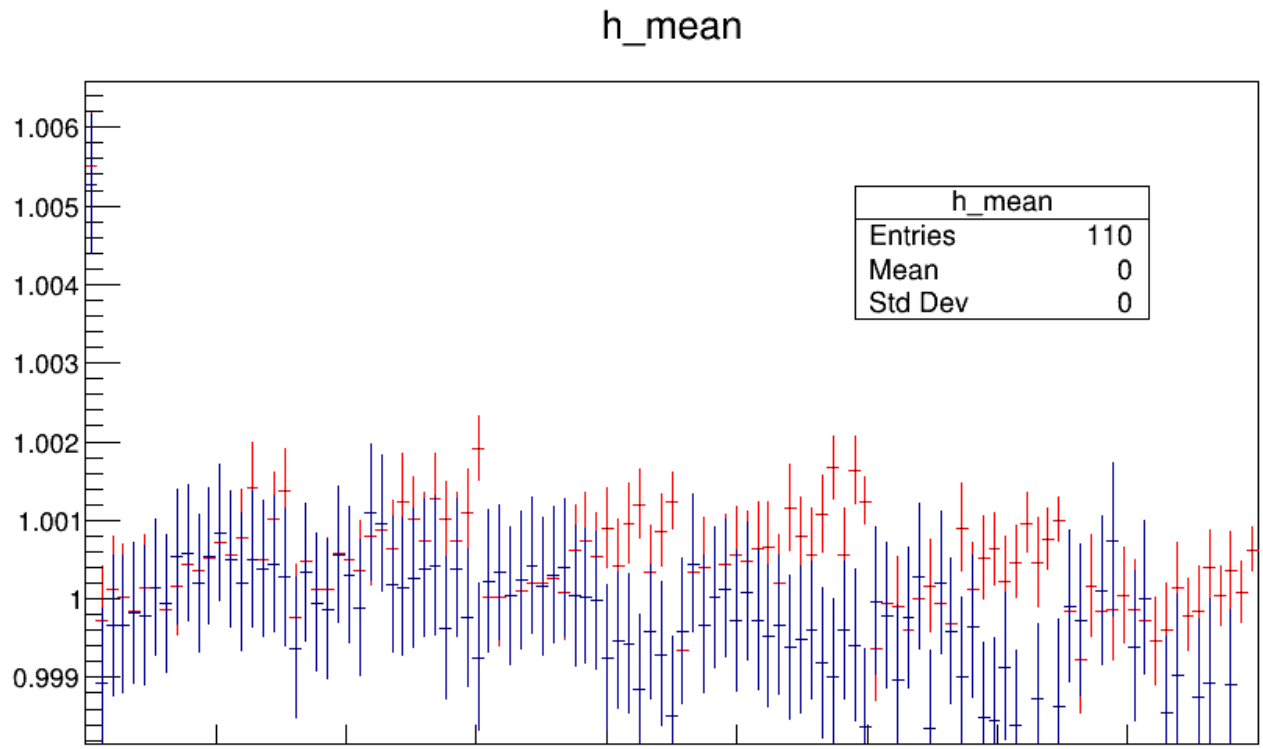
Ας σχολιάσουμε τώρα την έντονη διαφορά που φαίνεται να έχουν τα αποτελέσματα από το TestTree (Μπλέ) με τα αποτελέσματα από το TrainTree (Κόκκινο). Αρχικά για τις πρώτες χαμηλές τιμές του NT και του MD στην αρχή κάθε παραβολής και για τις 6 τιμές του Shrinkage φαίνεται τα αποτελέσματα του TestTree (Μπλέ) και του TrainTree (Κόκκινο) να είναι αρκετά κοντά. Η έντονη απομάκρυνση μεταξύ των δύο αρχίζει να συμβαίνει όταν αυξηθεί πολύ η τιμή του Number of Trees στο 500, 1000 και 2000. Για αυτές τις τιμές σε συνδιασμό με ένα Max Depth 4 ή 5 το Result που προκύπτει από το TrainTree (Κόκκινο) είναι πάρα πολύ καλύτερο, ενώ το Result του TestTree (Μπλέ) όχι μόνο δεν βελτιώνεται αλλά και χειροτερεύει.

Αυτό όπως είδαμε και στην θεωρία είναι ένα κλασσικό παράδειγμα υπερεκπαίδευσης που δείχνει ότι η μέθοδος έχει “παραμάθει” καλά το σύνολο εκπαίδευσης με αποτέλεσμα να χάνει την ικανότητα γενίκευσης στο Σύνολο Δοκιμής (Test Set). Από αυτή την εικόνα λοιπόν εμείς πρέπει να διαλέξουμε το δέντρο με τις υπερπαραμέτρους x, y ,z για το οποίο όχι μόνο τα αποτελέσματα είναι βέλτιστα αλλα και δεν έχει μεγάλη απόκλιση το TestTree (Μπλέ) με το αποτέλεσμα του TrainTree (Κόκκινο).

Όμοια εικόνα αποτελεσμάτων προκύπτει και για τις τιμές του σίγμα τονίζοντας πάλι το φαινόμενο της υπερκπαίδευσης για μεγάλες τιμές στις παραμέτρους NT και MD:



Στα αποτελέσματα της μέσης τιμής τώρα δεν φαίνεται να υπάρχει μεγάλη διαφορά μεταξύ Train-Test συνόλων με τις τιμές να είναι όλες πολύ κοντά στο 1 και σε απόσταση μεταξύ τους εντός του στατιστικού σφλαματος. Μόνο η τιμή του Raw, όπως αναμέναμε κιόλας, είναι σχετικά μακριά:



Ο κώδικας για την δημιουργεία των Response ιστογραμμάτων για τα 108 δέντρα απόφασης, του truncated fit και την δημιουργεία των παραπάνω διαγραμμάτων με τα αποτελέσματα (Results) του mean, του std και του sigma βρίσκεται στο Παράρτημα 6.

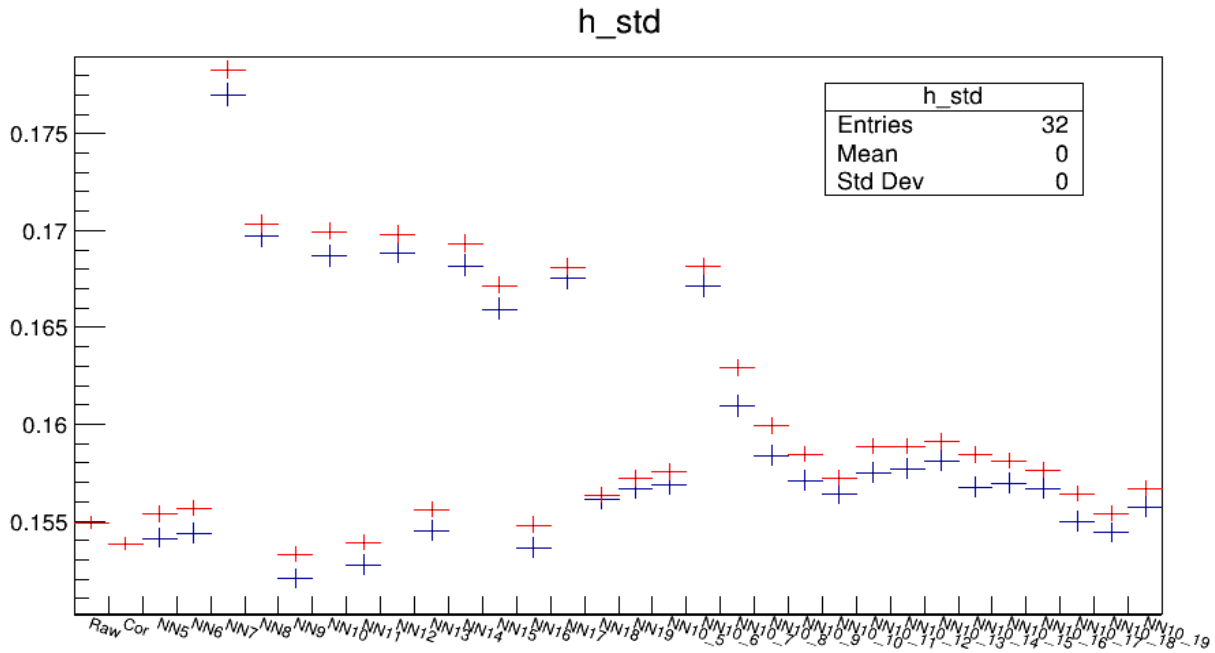
2.6 Τεχνητά Νευρωνικά Δίκτυα (NN's)

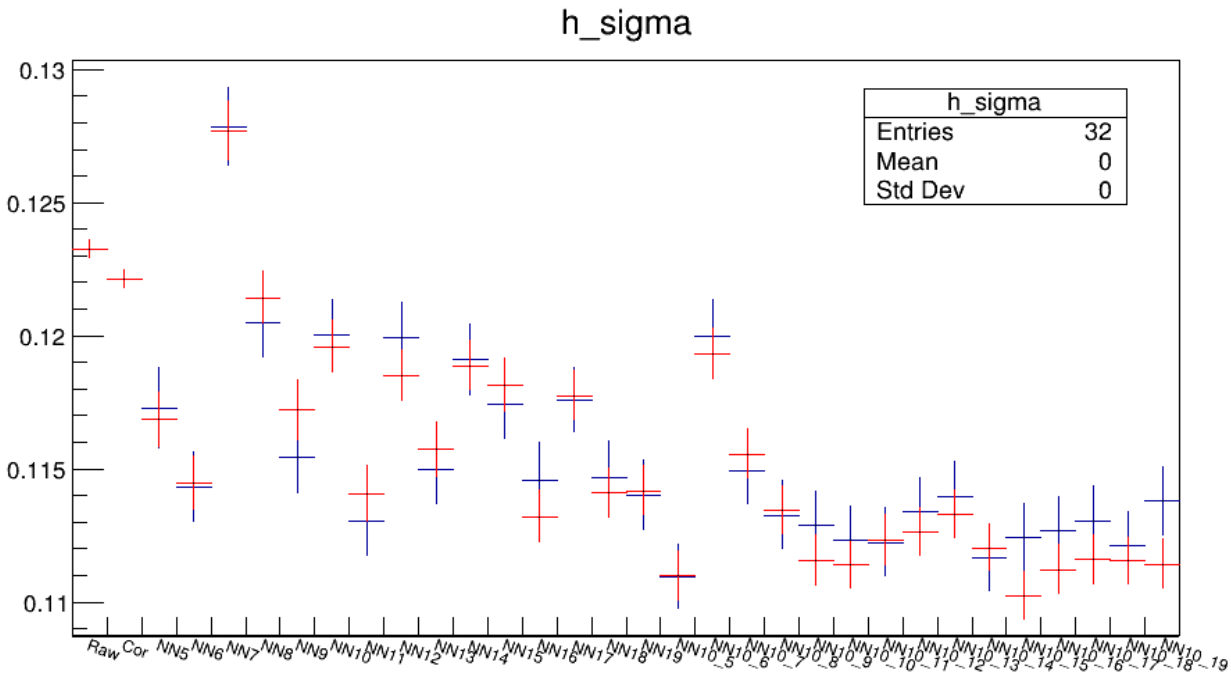
Ας δούμε τώρα τα αποτελέσματα για την τρίτη από τις μεθόδους μας, τα τεχνητά νευρωνικά δίκτυα (Artificial Neural Networks). Όλα ξεκινάνε πάλι από το TMVA όπου μέσω της κλάσης factory ορίζουμε πως θα γίνει χρήση NN's για παλινδρόμηση και εισάγουμε τις μεταβλητές (AddVariable) που θα χρησιμοποιήσουμε για την παλινδρόμηση από το δέντρο (TTree) my_tree. Επιπλέον ορίζουμε τον στόχο (Target) της παλινδρόμησης να είναι η μεταβλητή "bgenJetPt" βάση της οποίας θα γίνει η διαδικασία της μάθησης υπό επίβλεψη. Έπειτα ορίζουμε την αναλογία δείγματος Εκπαίδευσης (Training) και Ελέγχου (Testing) που και στην περίπτωσή των NN's είναι 70% Training – 30% Testing.

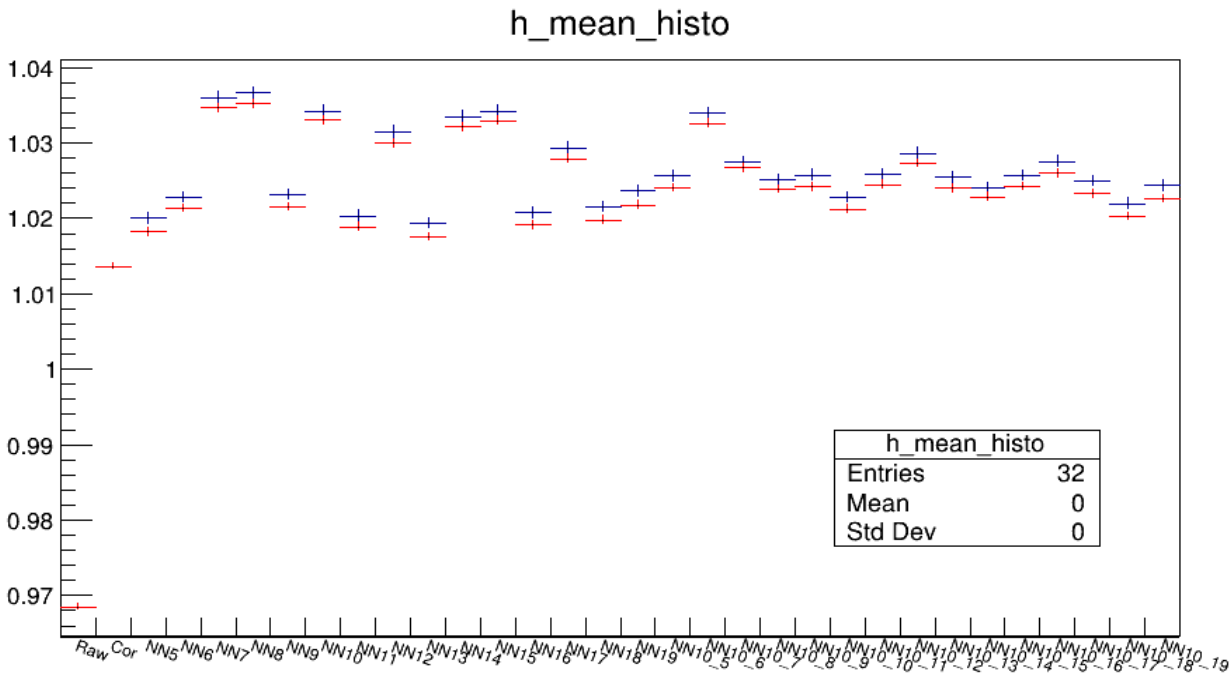
Η διαφορά των Νευρωνικών Δικτύων τώρα από τις άλλες μεθόδους είναι πως μέσω του TMVA είμαστε σε θέση να ορίσουμε την αρχιτεκτονική του MLP Νευρωνικού Δικτύου, τον αριθμό των κύκλων εκπαίδευσης, την συνάρτηση ενεργοποίησης όπως και την μέθοδο εκπαίδευσης. Αυτές οι παράμετροι είναι: HiddenLayers (Κρυφά Στρώματα), TrainingMethod (Μέθοδος Εκπαίδευσης), NeuronType (Συνάρτηση Ενεργοποίησης) και NCycles (Κύκλοι Εκπαίδευσης ή Epochs). Επειδή τώρα έχουμε αυτη την δυνατότητα δεν θα φτιάξουμε μόνο ένα Νευρωνικό Δίκτυο ενός τύπου, αλλά θα ελένξουμε όλους τους πιθανούς συνδιασμούς έτσι ώστε να βρούμε το βέλτιστο. Οι τιμές που θα έχουν οι παράμετροί μας θα είναι:

- HiddenLayers : 1 HL με αριθμό Νευρώνων [5 , 6 , 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , . . . , 19]
  2 HL με αριθμό Νευρώνων [10_5 , 10_6 , 10_7 , 10_8 , 10_9 , . . . , 10_19]
  3 HL με αριθμό Νευρώνων [10_5_2 , 10_5_3 , 10_5_4 , 10_6_2 , 10_6_3 , 10_6_4 , 10_7_2 , 10_7_3 , 10_7_4]
- NeuronType : [tanh , sigmoid]
- TrainingMethod : [BP , BFGS]

➔ Για αρχή ας δούμε τα αποτελέσματα από Νευρωνικά Δίκτυα μέχρι 2 κρυφών στρωμάτων με μέθοδο εκπαίδευσης την μέθοδο **Οπισθοδρόμησης (BP)** και συνάρτηση ενεργοποίησης την **σιγμοειδή**:







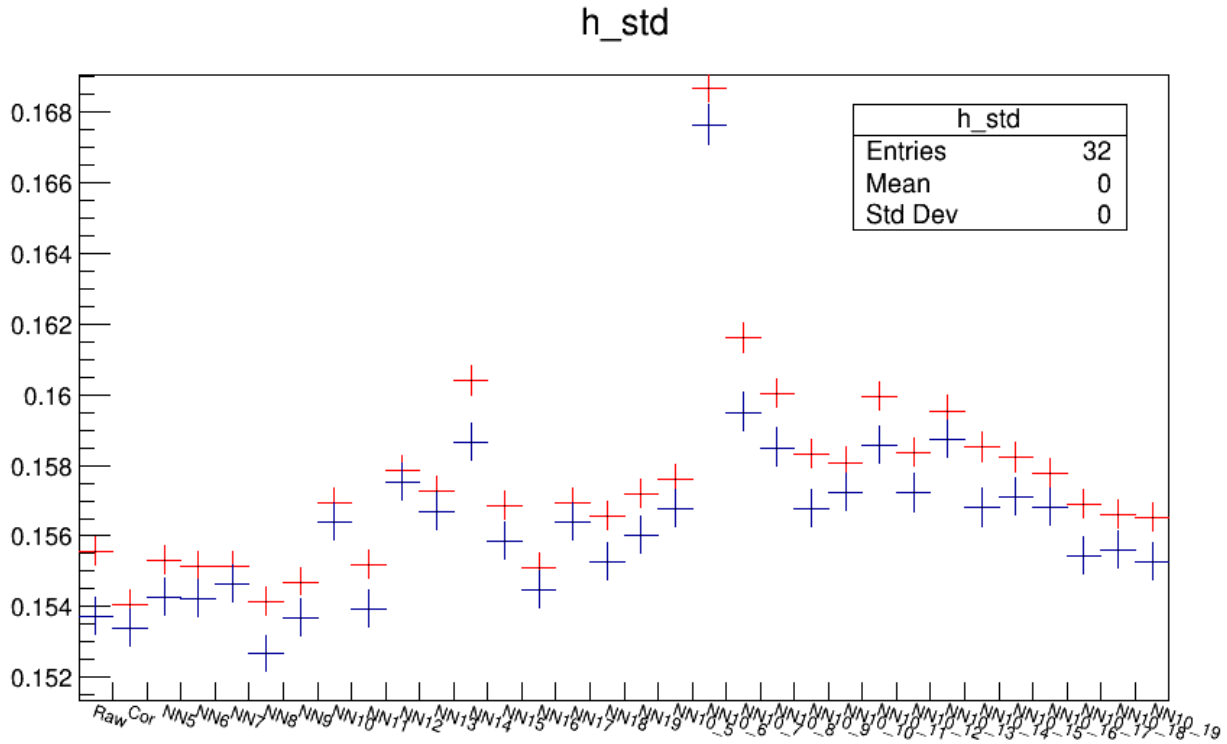
Τα αποτέλεσματα αυτά δημιουργήθηκαν μέσω του TMVA το οποίο έφτιαξε ένα νέο αρχείο με το όνομα TMVA_NN_BP.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 30 Branches με ονόματα MLPX_Y , όπου X, Y οι παραπάνω τιμές για την αρχιτεκτονική του Νευρωνικού Δικτύου. Αναλυτικά ο κώδικας του TMVA για τα Νευρωνικά Δίκτυα με Μέθοδο Οπισθοδρόμησης (NNs with Back-Propegation) βρίσκεται στο Παράρτημα 7.

Στόχος μας όπως και πρίν είναι να συγκρίνουμε τα Response από τα Branches: MLPX_Y μεταξύ τους και ως προς τα Response από το Raw και το Corrected. Επιπλέον όμως πρέπει να ελέγξουμε αν υπάρχει διαφορά στα αποτελέσματα (Results) μεταξύ συνόλου Εκπαίδευσης (Training) και συνόλου Δοκιμών (Testing).

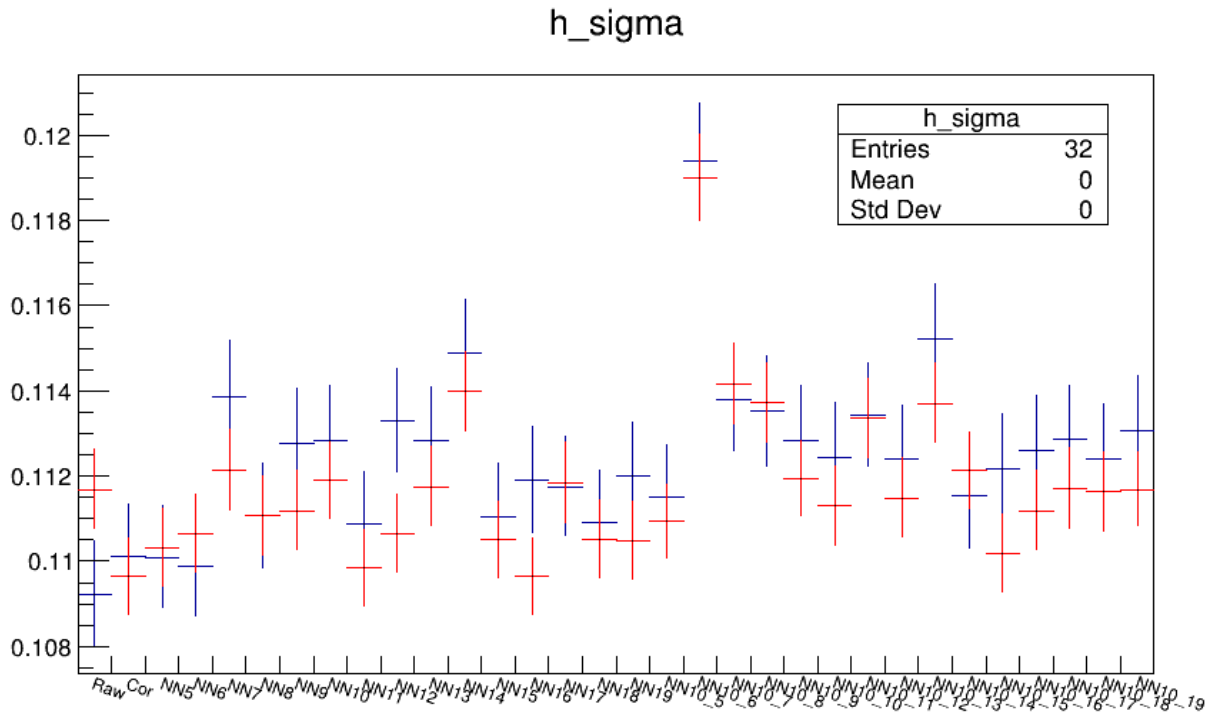
Όπως και στα προηγούμενα διαγράμματα αποτελεσμάτων με μπλέ χρώμα είναι τα αποτελέσματα από το TestTree και με κόκκινο χρώμα από το TrainTree. Όπως φαίνεται σε κάθε περίπτωση (Raw-Cor-MLPX_Y) είτε για το mean, είτε για το std, είτε για το sigma δεν υπάρχει μεγάλη διαφορά στα αποτελέσματα μεταξύ TestTree (Μπλέ) και TrainTree (Κόκκινο) όπως αυτή που είχαμε δεί προηγουμένος στα Ενδυναμωμένα Δέντρα Απόφασης. Οι διαφορές είναι πιο πολύ σταθερής στατιστικής φύσης που μας δείχνει ότι δεν έχουμε φαινόμενα υπερεκπαίδευσης.

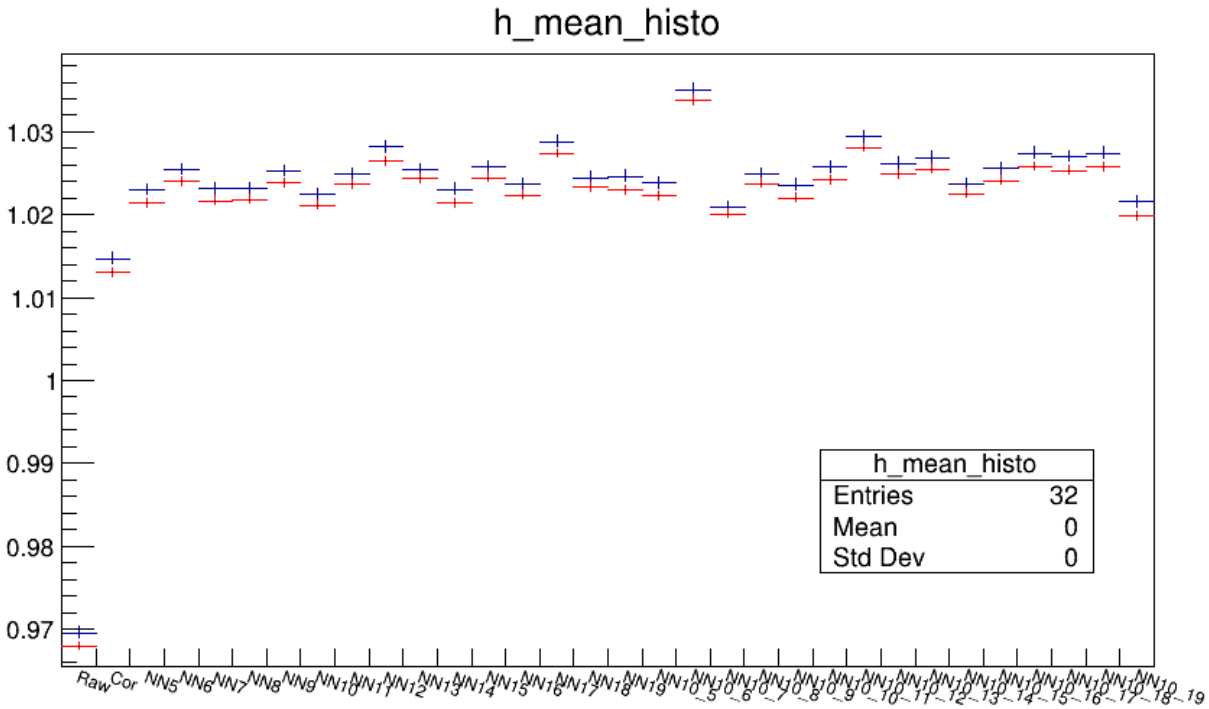
Επιπλέον κοιτάζοντας τα αποτελέσματα ως προς τις αρχιτεκτονικές φαίνεται πως τα Νευρωνικά Δίκτυα με 2 κρυφά στρώματα έχουν μια πιο σταθερή συμπεριφορά, με τα αποτελέσματά τους να είναι πιο “κοντά” μεταξύ τους. Αντιθέτως τα αποτελέσμτα των Νευρωνικών Δικτύων με 1 κρυφό στρώμα έχουν πολύ μεγαλύτερη απόκλιση μεταξύ τους, ειδικά στο διάγραμμα των std. Συγκεκριμένα τώρα ως προς το διάγραμμα της μέσης τιμής πάλι φαίνεται μια πιο έντονη διαφοροποίηση στις τιμές των Νευρωνικών Δικτύων ενός κρυφού στρώματος και μια “σταθερότητα” για τα MLP με 2 κρυφά στρώματα αλλά η βασική παρατήρηση που πρέπει να γίνει είναι πως συγκρίνοντας με το αποτέλεσμα του Corrected καμία αρχιτεκτονική δεν φαίνεται να φέρνει πιο κοντά στο 1 την τιμή του mean. Ο κώδικας για την δημιουργεία των Response ιστογραμμάτων για τα 30 Νευρωνικά Δίκτυα, του fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και την δημιουργεία των παραπάνω διαγραμμάτων με τα αποτελέσματα (Results) του mean, του std και του sigma βρίσκεται στο Παράρτημα 8.

➔ Ας δούμε τώρα τα αποτελέσματα από Νευρωνικά Δίκτυα μέχρι 2 κρυφών στρωμάτων με μέθοδο εκπαίδευσης την μέθοδο **BFGS** και συνάρτηση ενεργοποίησης πάλι την **σιγμοειδή**:



Τα αποτέλεσματα αυτά δημιουργήθηκαν μέσω του TMVA το οποίο έφτιαξε ένα νέο αρχείο με το όνομα TMVA_NN.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 30 Branches με ονόματα MLPX_Y , όπου X, Y οι παραπάνω τιμές για την αρχιτεκτονική του Νευρωνικού Δικτύου. Ο κώδικας του TMVA για τα Νευρωνικά Δίκτυα με Μέθοδο BFGS (NNs with BFGS) είναι παρόμοιος με αυτόν στο Παράρτημα 7.

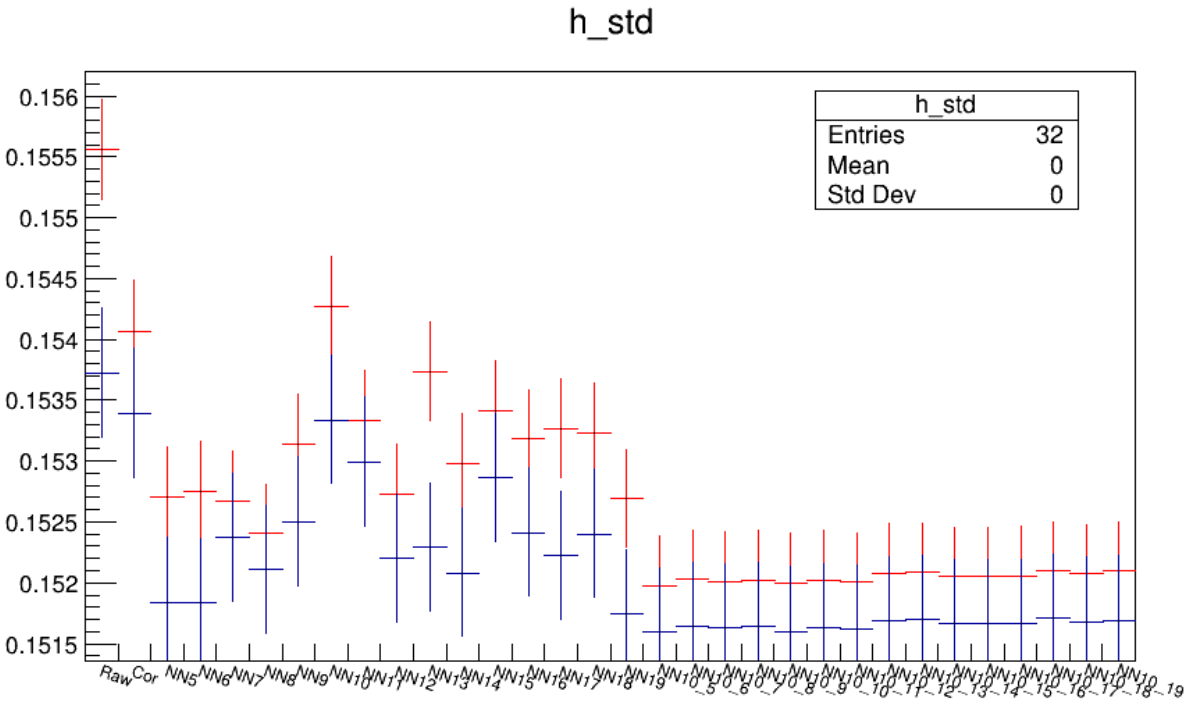


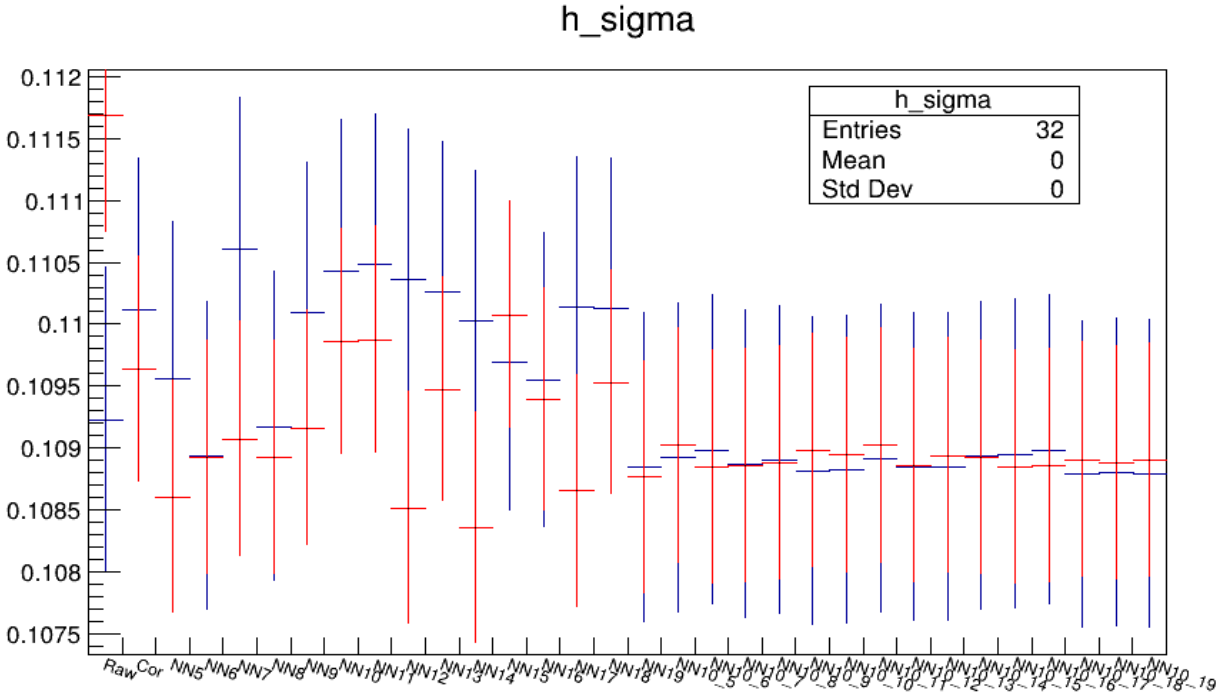


Όπως και στα προηγούμενα διαγράμματα αποτελεσμάτων με μπλέ χρώμα είναι τα αποτελέσματα από το TestTree και με κόκκινο χρώμα από το TrainTree. Όπως φαίνεται σε κάθε περίπτωση (Raw-Cor-MLPX_Y) είτε για το mean, είτε για το std, είτε για το sigma δεν υπάρχει μεγάλη διαφορά στα αποτελέσματα μεταξύ TestTree (Μπλέ) και TrainTree (Κόκκινο) όπως αυτή που είχαμε δεί στα Ενδυναμωμένα Δέντρα Απόφασης. Οι διαφορές είναι πιο πολύ σταθερής στατιστικής φύσης που μας δείχνει ότι ούτε σε αυτή την περίπτωση δεν έχουμε φαινόμενα υπερεκπαίδευσης.

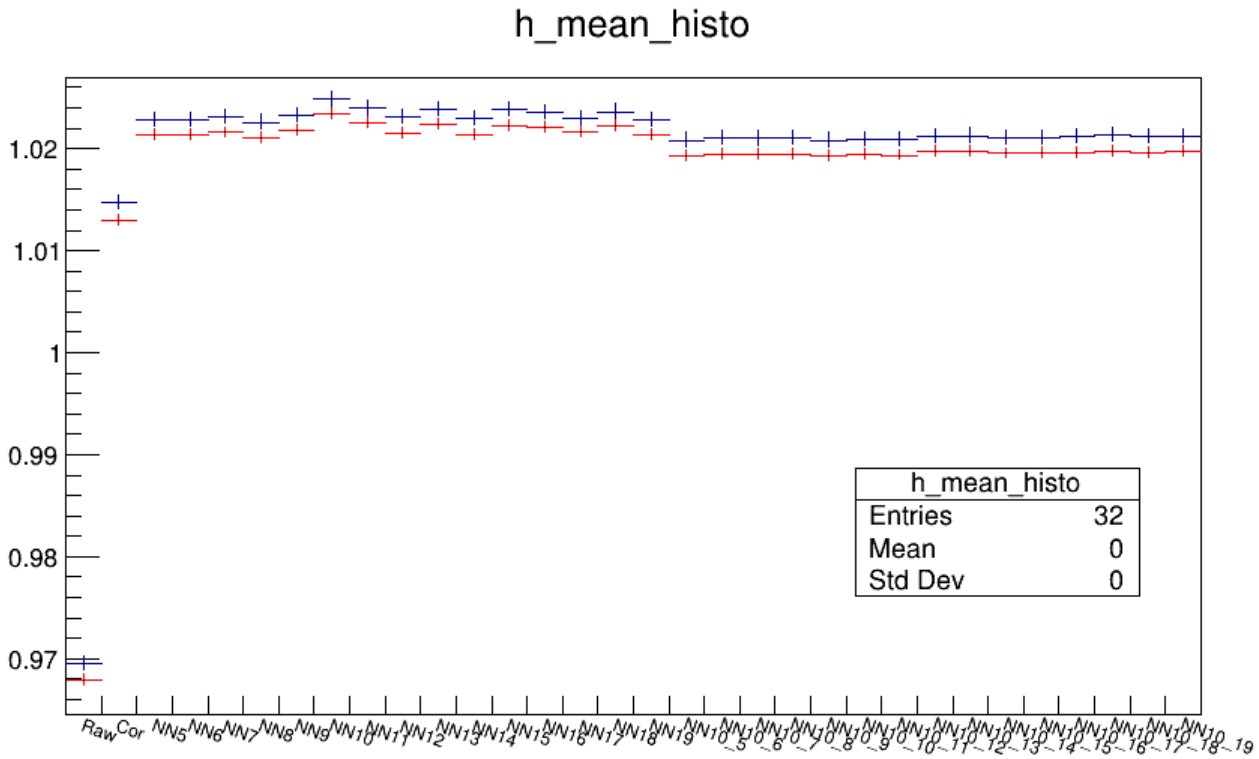
Επιπλέον κοιτάζοντας τα αποτελέσματα ως προς τις αρχιτεκτονικές φαίνεται πως τα Νευρωνικά Δίκτυα με 1 κρυφό στρώμα είναι λίγο καλύτερα, όχι όμως με πολύ έντονη διαφορά. Η Βασική παρατήρηση είναι πως με την μέθοδο BFGS φαίνεται να έχουμε καλύτερα αποτελέσματα από την BP ως προς το mean, το sigma αλλά και το std.

➔ Ας δούμε τώρα τα αποτελέσματα από Νευρωνικά Δίκτυα μέχρι 2 κρυφών στρωμάτων με μέθοδο εκπαίδευσης την μέθοδο **BFGS** και συνάρτηση ενεργοποίησης την **tanh**:





Τα αποτέλεσματα δημιουργήθηκαν πάλι μέσω του TMVA το οποίο έφτιαξε ένα νέο αρχείο με το όνομα TMVA_NN_Tanh.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 30 Branches με ονόματα MLPX_Y , όπου X, Y οι τιμές για την αρχιτεκτονική του Νευρωνικού Δικτύου. Ο κώδικας του TMVA για τα Νευρωνικά Δίκτυα με Μέθοδο BFGS (NNs with BFGS) με συνάρτηση ενεργοποίησης την tanh είναι παρόμοιος με αυτόν στο Παράρτημα 8.

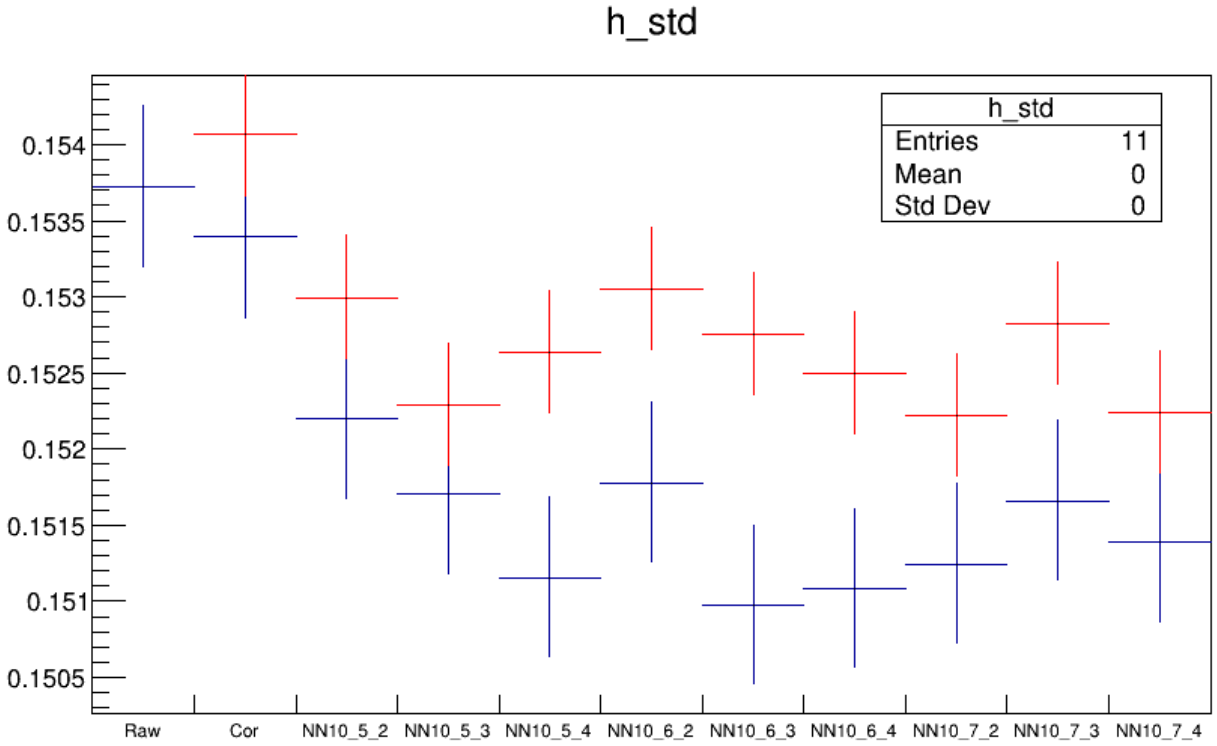


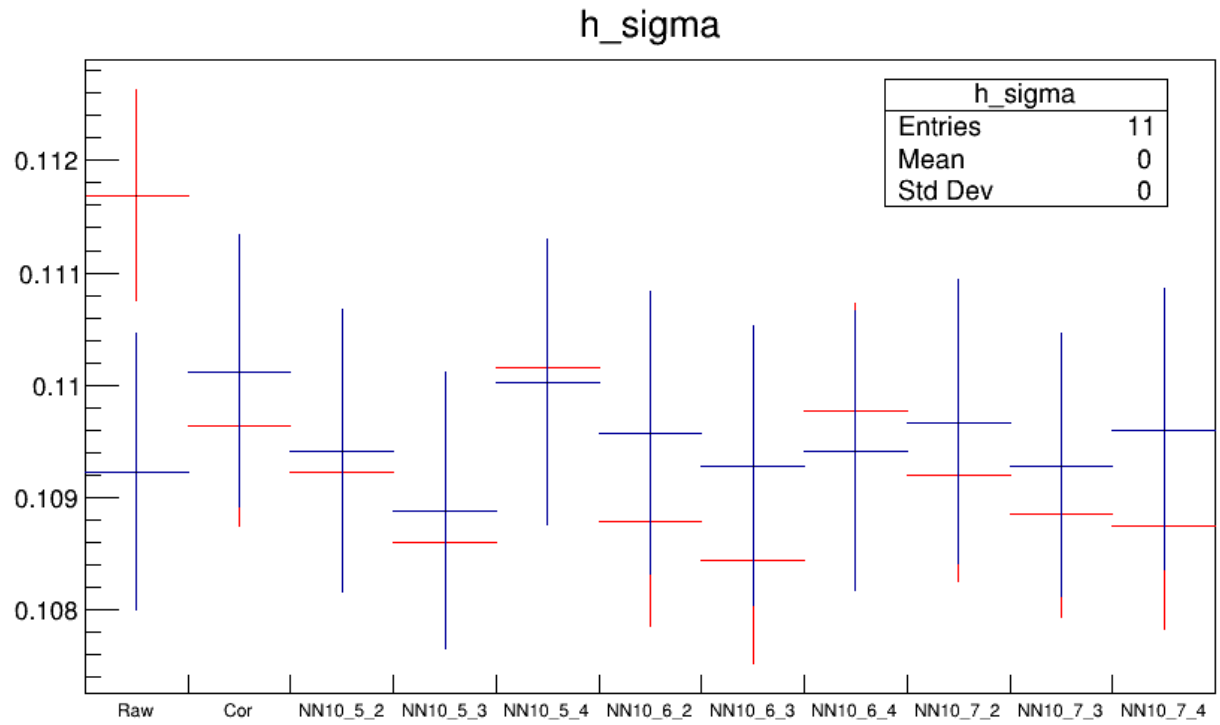
Όπως και στα προηγούμενα διαγράμματα αποτελεσμάτων με μπλέ χρώμα είναι τα αποτελέσματα από το TestTree και με κόκκινο χρώμα από το TrainTree. Όπως φαίνεται σε κάθε περίπτωση (Raw-Cor-MLPX_Y) είτε για το mean, είτε για το std, είτε για το sigma δεν υπάρχει μεγάλη διαφορά στα αποτελέσματα μεταξύ TestTree (Μπλέ) και TrainTree (Κόκκινο) όπως αυτή που είχαμε δεί στα Ενδυναμωμένα Δέντρα Απόφασης. Οι διαφορές είναι πιο πολύ σταθερής στατιστικής φύσης που μας δείχνει πως ούτε σε αυτή την περίπτωση έχουμε φαινόμενα υπερεκπαίδευσης.

Κοιτάζοντας τα αποτελέσματα ως προς τις αρχιτεκτονικές φαίνεται πως τα Νευρωνικά Δίκτυα με 2 κρυφά στρώματα έχουν μια πιο σταθερή συμπεριφορά, με τα αποτελέσματά τους να είναι πιο “κοντά” μεταξύ τους. Αντιθέτως τα αποτελέσμτα των Νευρωνικών Δικτύων με 1 κρυφό στρώμα έχουν πολύ μεγαλύτερη απόκλιση μεταξύ τους και μάλιστα τις περισσότερες φορές τα αποτελέσματά τους είναι αρκετά χειρότερα. Συγκεκριμένα τώρα ως προς το διάγραμμα του std φαίνεται μια πιο έντονη διαφοροποίηση στις τιμές των Νευρωνικών Δικτύων ενός κρυφού στρώματος και μια “σταθερότητα” για τα MLP με 2 κρυφά στρώματα αλλά η βασική παρατήρηση που πρέπει να γίνει είναι πως συγκρίνοντας με το αποτέλεσμα του Corrected και του Raw μόνο τα Νευρωνικά Δίκτυα με 2 κρυφά στρώματα κάνουν κάποια σοβαρή βελτίωση. Βέβαια είναι σημαντικό να σχολιάσουμε πως για αυτη την περίπτωση τα αποτελέσματα του sigma και του std είναι πολύ κοντά μεταξύ τους και αυτό φαίνεται από το πόσο έντονες φαίνονται τώρα οι κατακόρυφες τιμές του σφάλματος. Συγκρίνοντας τώρα αυτά τα αποτελέσματα με τους προηγούμενες δύο συνδιασμούς μεθόδων εκπαίδευσης και συναρτήσεων ενεργοποίησης φαίνεται πως τα Νευρωνικά Δίκτυα 2 κρυφών στρωμάτων με μέθοδο εκπαίδευσης την BFGS και συνάρτηση ενεργοποίησης την tanh δίνουν τα καλύτερα αποτελέσματα. Ας ελέγξουμε όμως πρίν δούμε τα διαγράμματα σύγκρισης για τα αποτελέσματα όλων των παραπάνω και την περίπτωση των Νευρωνικών Δικτύων με 3 κρυφά στρωμάτων

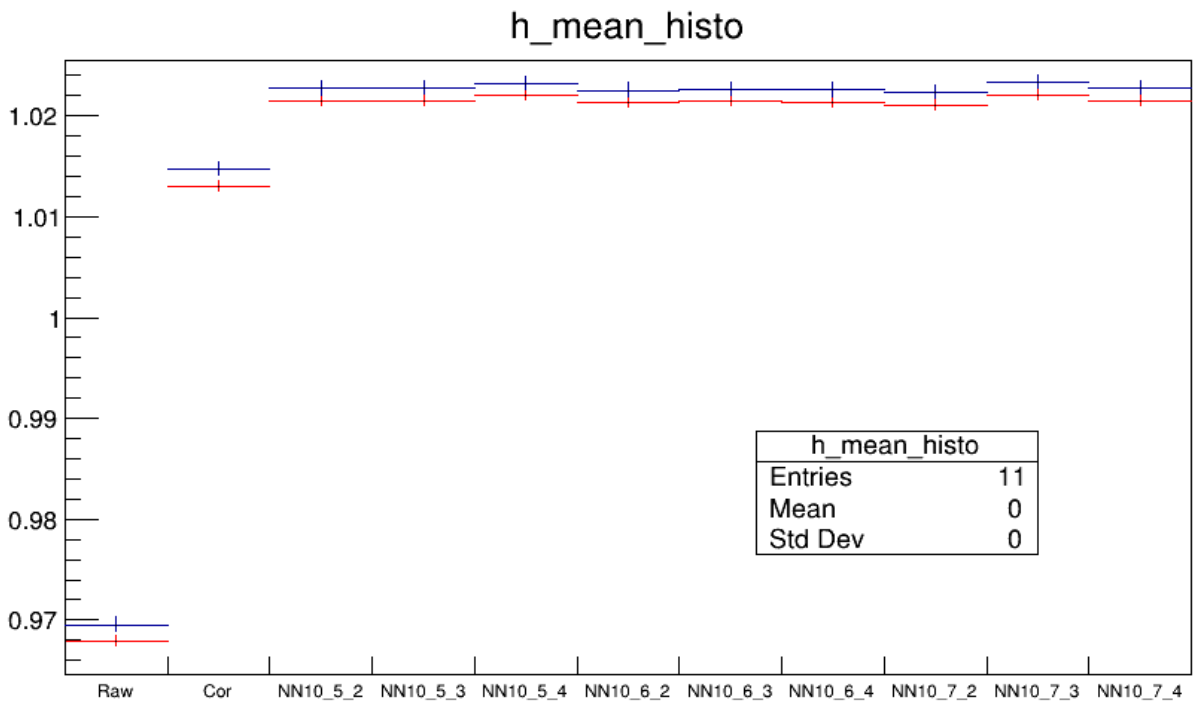
Ο κώδικας για την δημιουργεία των Response ιστογραμμάτων για τα 30 Νευρωνικά Δίκτυα με την μέθοδο BFGS και συνάρτηση ενεργοποίησης την tanh, του fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και την δημιουργεία των παραπάνω διαγραμμάτων με τα αποτελέσματα (Results) του mean, του std και του sigma είναι παρόμοιος με αυτόν στο Παράρτημα 8.

➔ Ας δούμε τώρα τα αποτελέσματα από Νευρωνικά Δίκτυα με 3 κρυφά στρωμάτων με μέθοδο εκπαίδευσης την μέθοδο **BFGS** και συνάρτηση ενεργοποίησης την **tanh**:





Τα αποτέλεσματα δημιουργήθηκαν πάλι μέσω του TMVA το οποίο έφτιαξε ένα νέο αρχείο με το όνομα TMVA_NN_Tanh_3HL.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 9 Branches με ονόματα MLPX_Y_Z , όπου X, Y, Z οι τιμές για την αρχιτεκτονική του Νευρωνικού Δικτύου. Αναλυτικά ο κώδικας του TMVA για τα Νευρωνικά Δίκτυα με 3 κρυφά στρώματα με Μέθοδο BFGS και με συνάρτηση ενεργοποίησης την tanh βρίσκεται στο Παράρτημα 9.



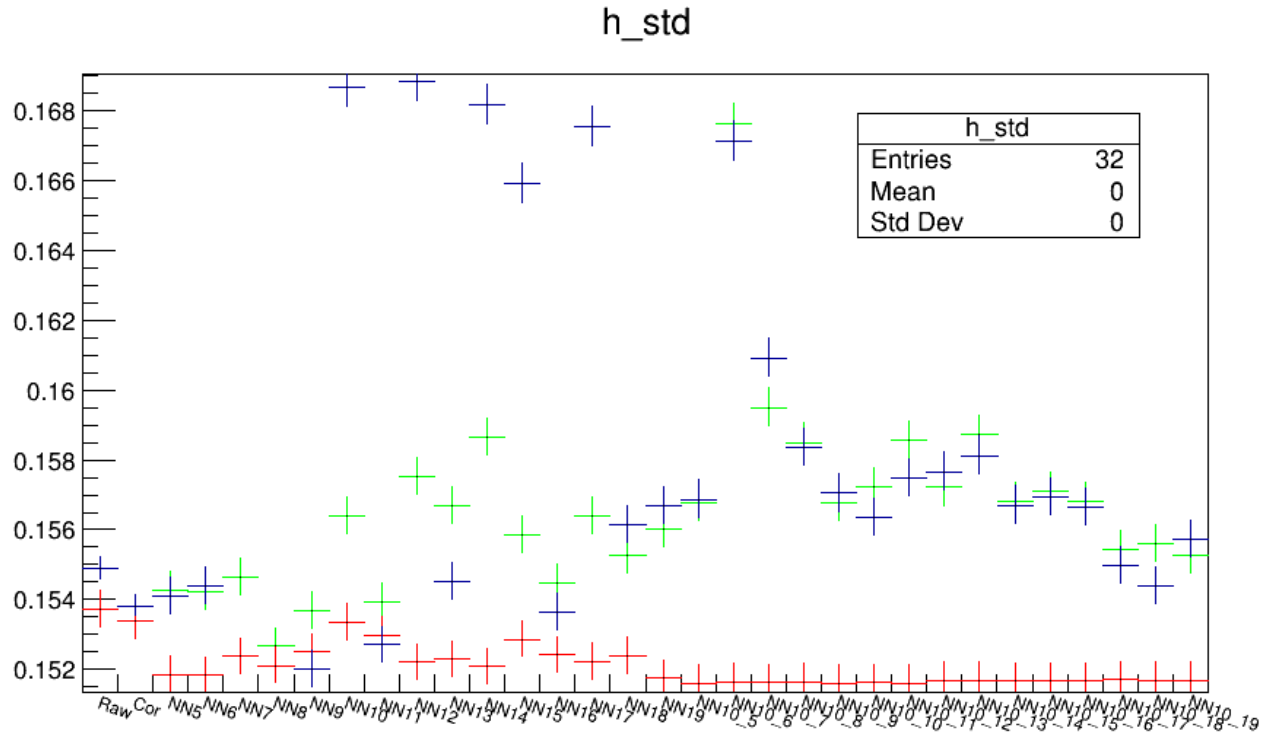
Όπως και στα προηγούμενα διαγράμματα αποτελεσμάτων με μπλέ χρώμα είναι τα αποτελέσματα από το TestTree και με κόκκινο χρώμα από το TrainTree. Όπως φαίνεται σε κάθε περίπτωση (Raw-Cor-MLPX_Y_Z) είτε για το mean, είτε για το std, είτε για το sigma δεν υπάρχει μεγάλη διαφορά στα αποτελέσματα μεταξύ TestTree (Μπλέ) και TrainTree (Κόκκινο) όπως αυτή που είχαμε δεί στα Ενδυναμωμένα Δέντρα Απόφασης. Οι διαφορές είναι πιο πολύ σταθερής στατιστικής φύσης που μας δείχνει ότι δεν έχουμε φαινόμενα υπερεκπαίδευσης.

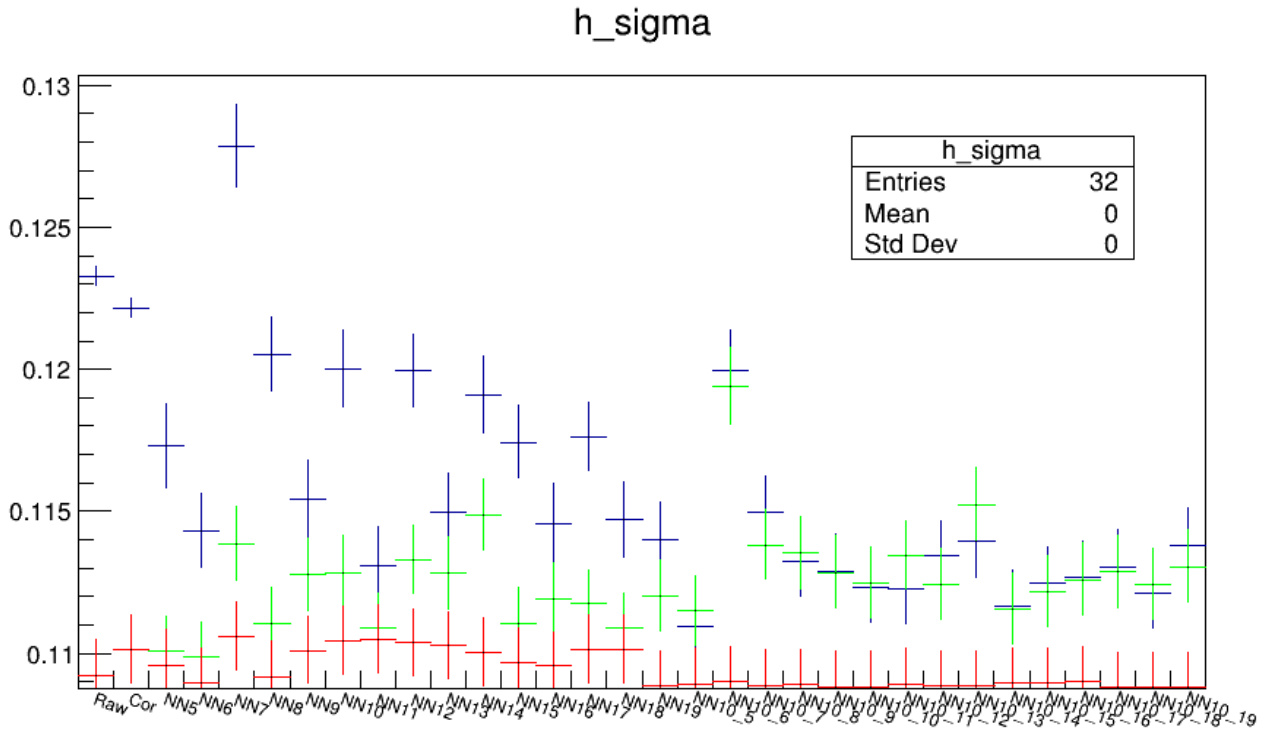
Συγκρίνοντας τώρα αυτά τα αποτελέσματα με τους προηγούμενους τρείς συνδιασμούς μεθόδων εκπαίδευσης και συναρτήσεων ενεργοποίησης για Νευρωνικά Δίκτυα 2 κρυφών στρωμάτων φαίνεται πως τα Νευρωνικά Δίκτυα με 3 κρυφά στρώματα δεν προσφέρουν έντονες βελτιώσεις στα αποτελέσματα. Μια μικρή βελίωση υπάρχει μόνο ως προς το std για τις αρχιτεκτονικές NN10_5_4 , NN10_6_3 , NN10_6_4 σε σύγκριση με τα Νευρωνικά Δίκτυα 2 κρυφών στρωμάτων με μέθοδο εκπαίδευσης την μέθοδο BFGS και συνάρτηση ενεργοποίησης την tanh. Αυτό μας δείχνει πως περαιτέρω έρευνα για πιο πολύπλοκες και “βαθιές” δομές Νευρωνικών Δικτύων (Deep Neural Networks) σε δομές τύπου MLP τουλάχιστον δεν χρειάζεται.

Ο κώδικας για την δημιουργεία των Response ιστογραμμάτων για τα 9 Νευρωνικά Δίκτυα 3ων κρυφών στρωμάτων με την μέθοδο BFGS και συνάρτηση ενεργοποίησης την tanh, του fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και την δημιουργεία των παραπάνω διαγραμμάτων με τα αποτελέσματα (Results) του mean, του std και του sigma είναι παρόμοιος με αυτόν στο Παράρτημα 8.

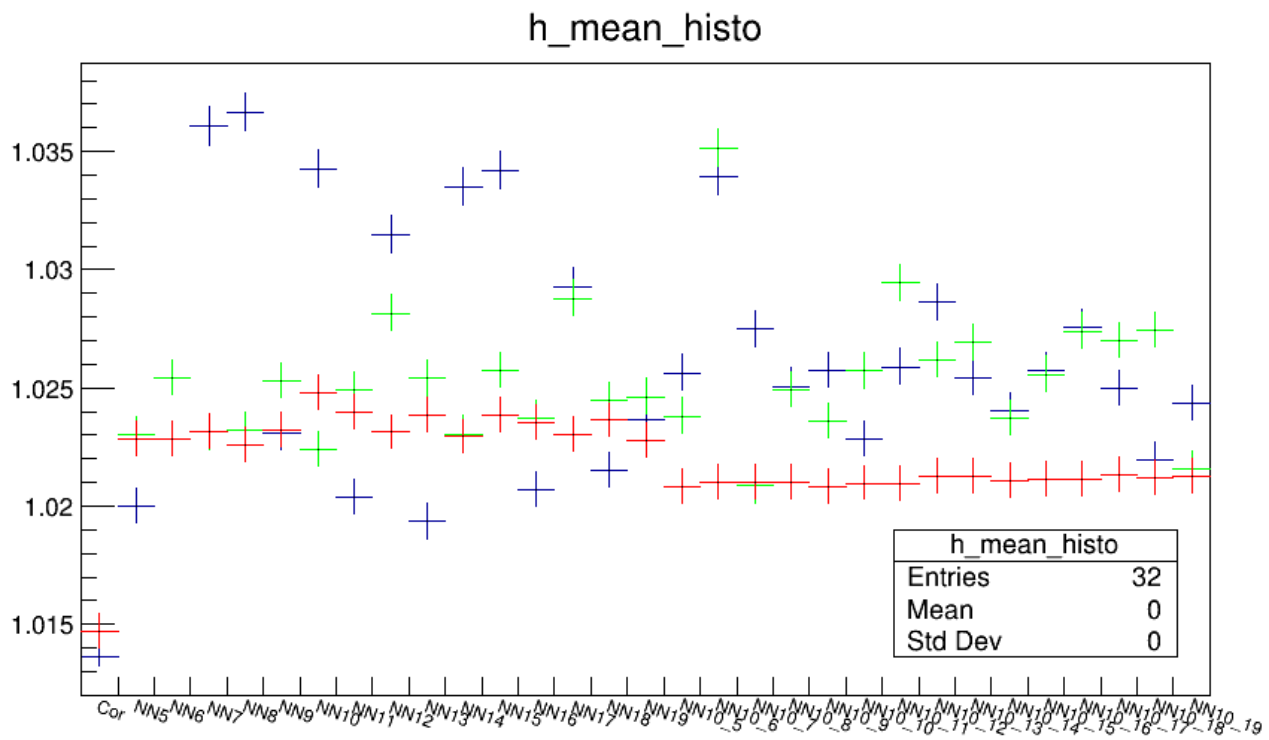
Ας κοιτάξουμε τώρα σε ένα διάγραμμα std, ένα διάγραμμα sigma και ένα διάγραμμα mean τα αποτελέσματα και των τριών τύπων Νευρωνικών Δικτύων έως 2 κρυφών στρωμάτων που φτιάξαμε παραπάνω. Στα διαγράμματα αυτά θα βάλουμε μόνο τα αποτελέσμτα που προκύπτουν από το TestTree του κάθε αρχείου TMVA_NN.root , TMVA_NN_BP.root , TMVA_NN_Tanh.root που έχουμε υλοποιήσει. Ο χρωματικός κώδικας που θα χρησιμοποιήσουμε είναι: **Μπλέ** για μέθοδο εκπαίδευσης την BP και συνάρτηση ενεργοποίησης την sigmoid, **Πράσινο** για μέθοδο εκπαίδευσης την BFGS και συνάρτηση ενεργοποίησης την sigmoid και **Κόκκινο** για μέθοδο εκπαίδευσης την BFGS και συνάρτηση ενεργοποίησης την tanh.

➔ Τα αποτελέσματα και των τριών τύπων Νευρωνικών Δικτύων έως 2 κρυφών στρωμάτων με διαφορετικές μεθόδους εκπαίδευσης και διαφορετικές συναρτήσεις ενεργοποίησης:



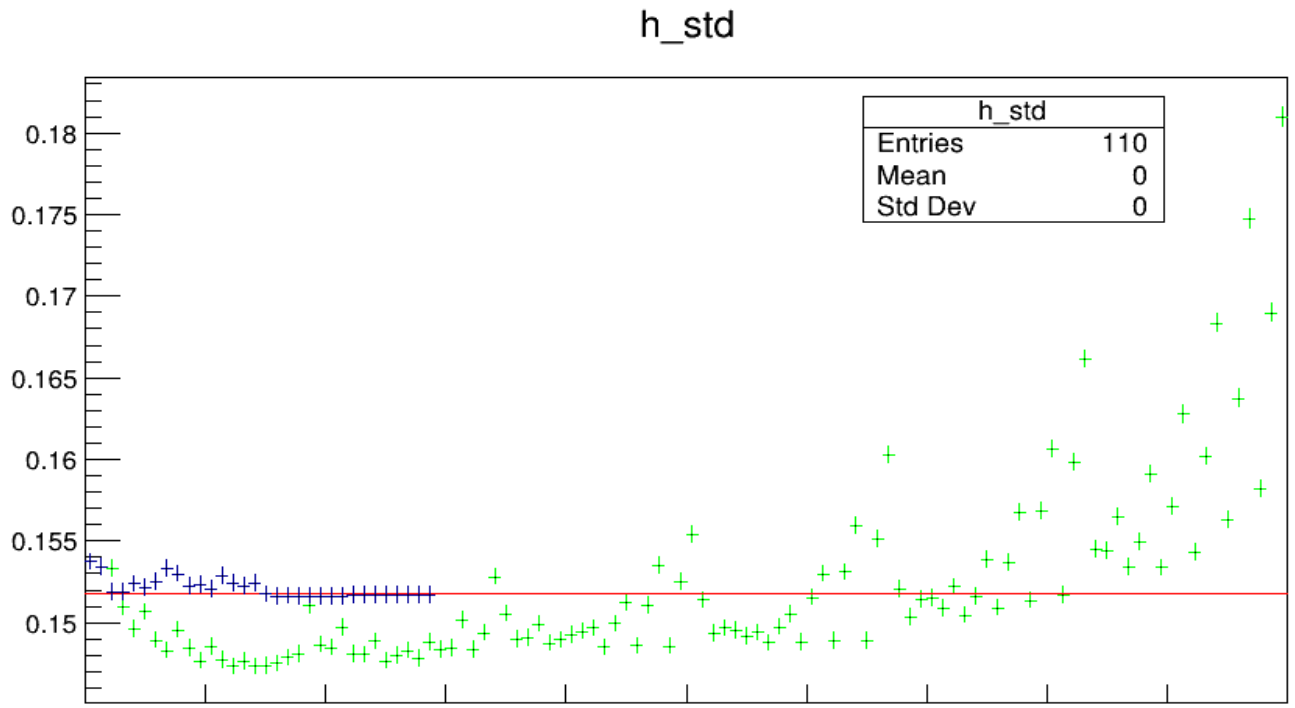


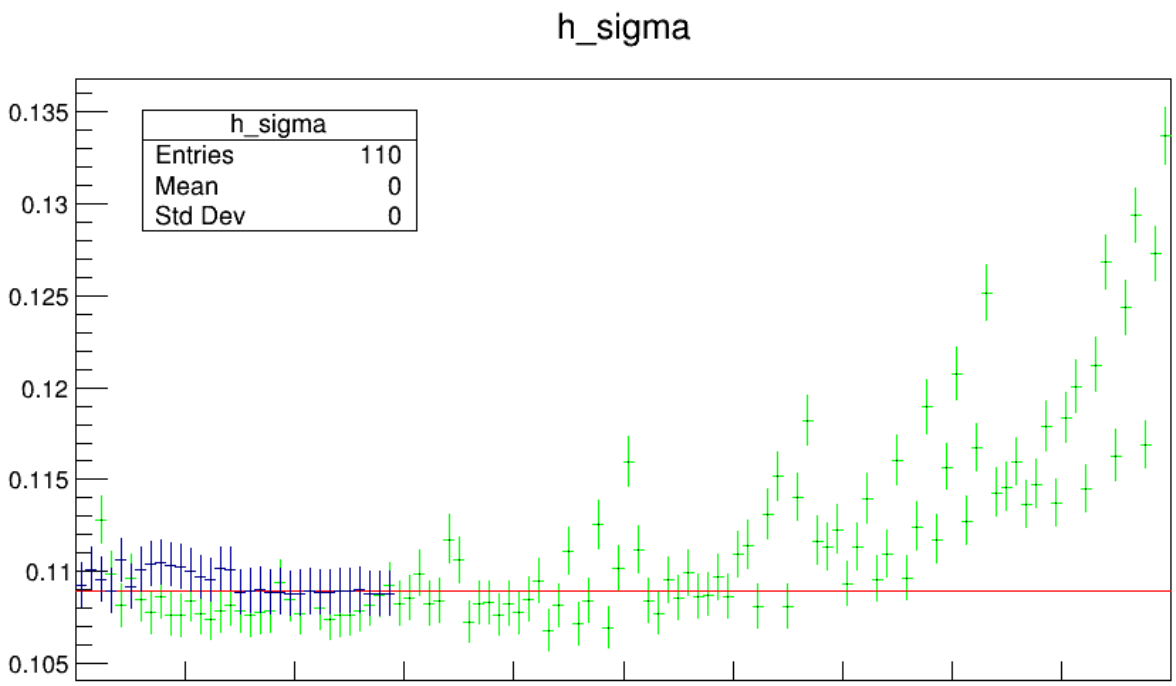
Κοιτάζοντας και τα 3 διαγράμματα των αποτελεσμάτων μαζί φαίνεται καθαρά πως ο βέλτιστος συνδιασμός συνάρτησης ενεργοποίησης και μεθόδου εκπαίδευσης είναι ο Κόκκινος, δηλαδή μέθοδος εκπαίδευσης **BFGS** και συνάρτηση ενεργοποίησης **tanh**. Συγκεκριμένα αυτός ο συνδιασμός φαίνεται να έχει τα βέλτιστα αποτελέσμτα για Νευρωνικά Δίκτυα δύο κρυφών στρωμάτων όπου και στην περίπτωση του std και στην περίπτωση του sigma αλλά και στην περίπτωση του mean τα αποτελέσματα έχουν μια σταθερότητα.

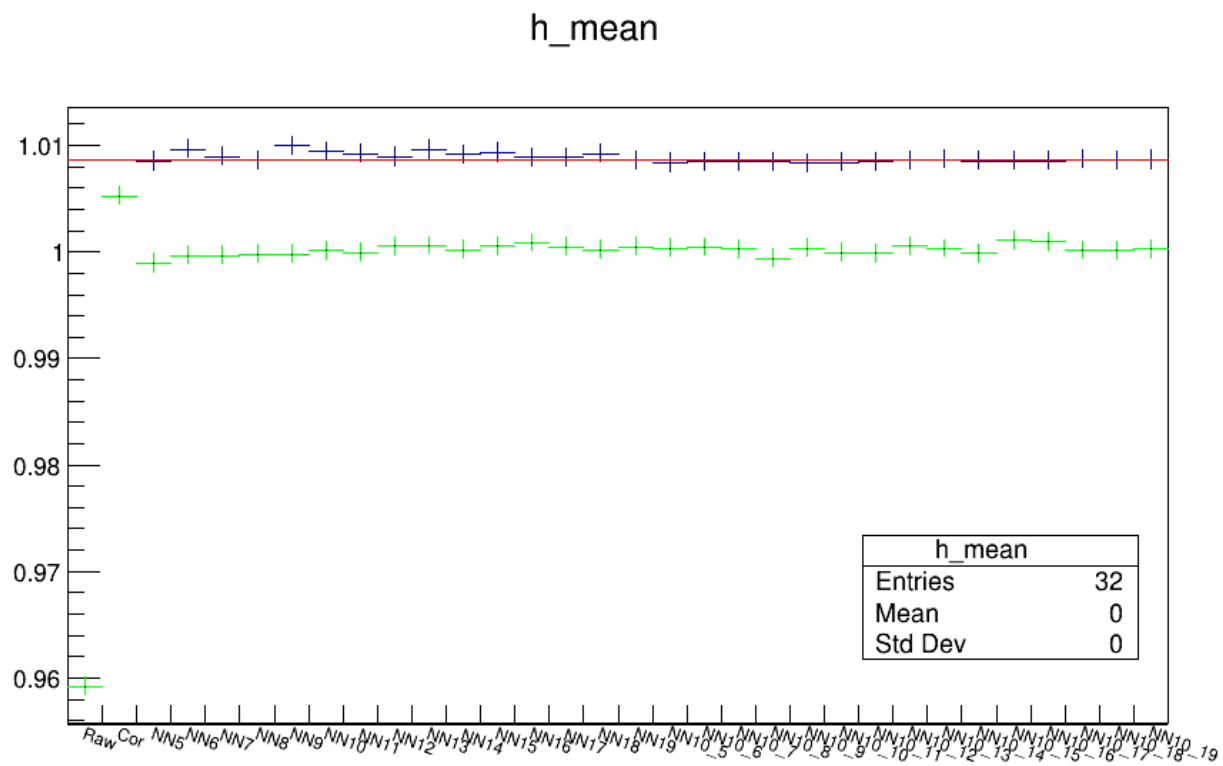


2.7 Σύγκριση και των 3ων Μεθόδων (NN's vs BDT's vs LD)

Είμαστε λοιπόν τώρα σε θέση να συγκρίνουμε τα αποτελέσματα και των 3ων μεθόδων Παλινδρόμησης που φτιάξαμε. Για να το κάνουμε αυτό θα βάλουμε σε ένα διάγραμμα std, ένα διάγραμμα sigma και ένα διάγραμμα mean τα αποτελέσματα από τα 108 δέντρα απόφασης, τα 30 καλύτερα Νευρωνικά Δίκτυα όπως και τα αποτελέσματα που προέκυψαν για τον LD. Ο χρωματικός κώδικας που θα χρησιμοποιήσουμε είναι: **Μπλέ** για τα Νευρωνικά Δίκτυα (NN), **Πράσινο** για τα Ενδυναμωμένα Δέντρα Απόφασης (BDT) και **Κόκκινο** για τον Γραμμικό Ταξινομητή (LD).

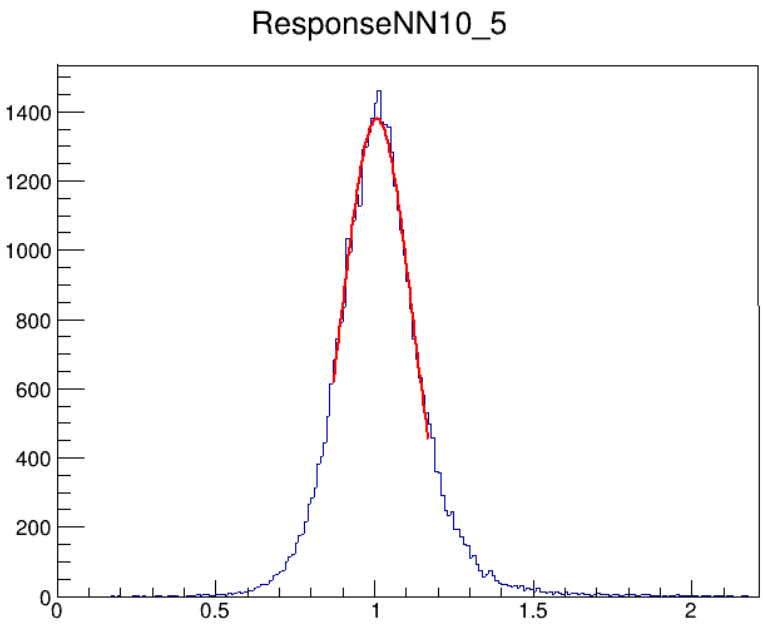


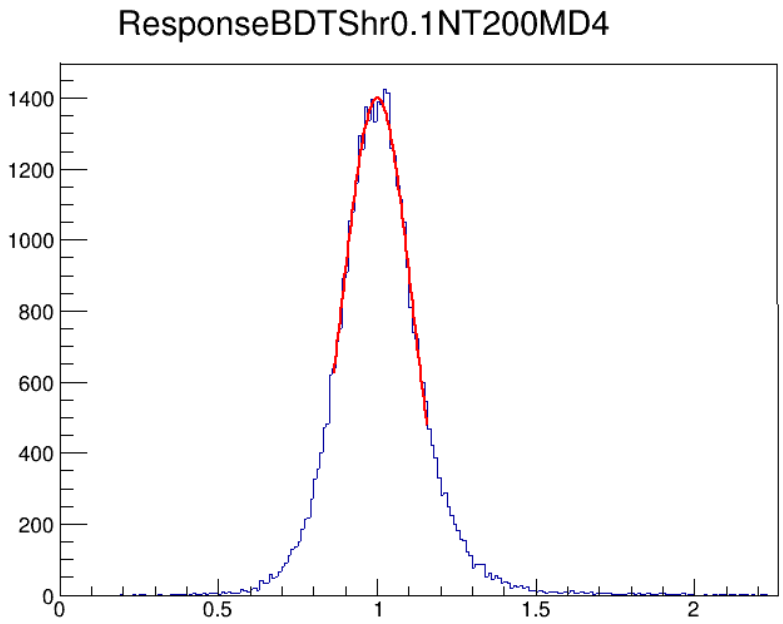


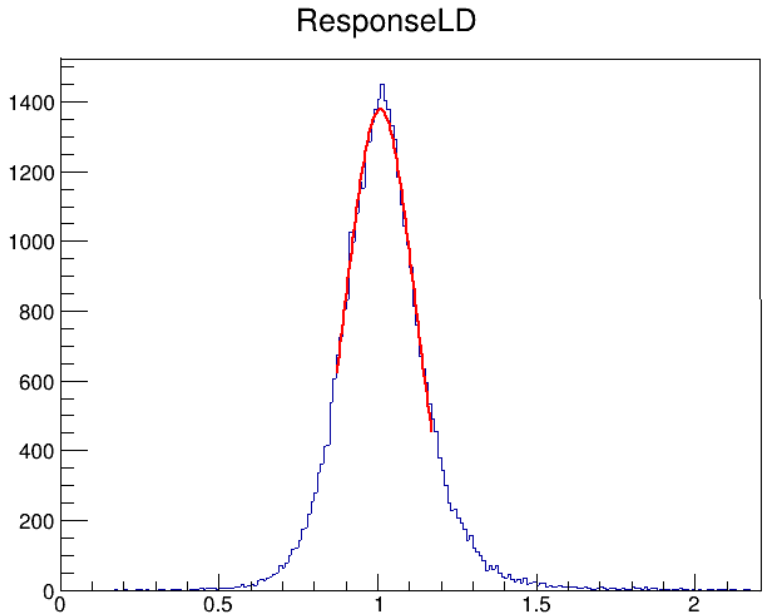


Από τις παραπάνω τρείς εικόνες των αποτελεσμάτων είναι ξεκάθαρο πως τα Ενδυναμωμένα Δέντρα Απόφασης και ως προς τις mean και ως προς το sigma αλλά και ως προς το std δίνουν τα καλύτερα αποτελέσματα. Τα Νευρωνικά Δίκτυα για ορισμένες δομές με δύο κρυφά στρώματα είναι ελάχιστα καλύτερα από τον Γραμμικό Ταξινομητή ως προς το sigma και το std ενώ για το mean φαίνεται να μην δίνουν την τόσο έντονη βελτίωση των BDT's τα οποία είναι ακριβώς πάνω στο 1. Από αυτά τα διαγράμματα είμαστε σε θέση να βρούμε την καλύτερη δομή και έτσι να επιλέξουμε το καλύτερο Νευρωνικό Δίκτυο όπως και το καλύτερο Δέντρο Απόφασης. Αυτά θα είναι: Το Νευρωνικό Δίκτυο με δύο κρυφά στρώματα με αριθμό Νευρώνων **NN_10_5** και το Δέντρο Απόφασης (που δεν έχει δεχτεί Υπερεκπαίδευση) με τις υπερπαραμέτρους: **BDTShr0.1NT200MD4**

➔ Τα διαγράμματα των Αποκρίσεων (Response) μαζί με το fit της Γκαουσιανής για το καλύτερο Νευρωνικό Δίκτυο, το καλύτερο Δέντρο Απόφασης όπως και του Γραμμικού Ταξινομητή βρίσκονται παρακάτω:







2.8 Απόδοση στους Υπόχωρους (Performance in Subspace)

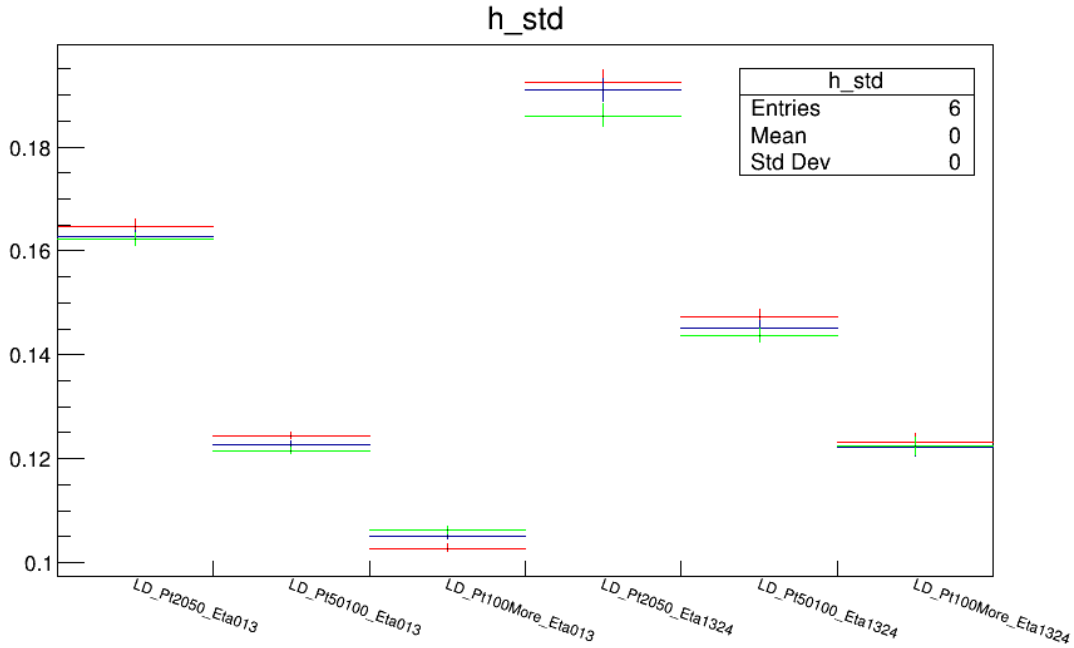
Ως τελευταίο προγραμματιστικό βήμα αυτή της διπλωματικής, ή αλλιώς “Task” όπως τα αποκαλούμε με τον επιβλέποντα καθηγητή, διαλέξαμε να γίνει ένας έλεγχος στο κατά πόσο υπάρχει κάποιο bias στους υπόχωρους του Pt και του Eta για τις μετρήσεις μας. Για να το κάνουμε αυτό θα χρειαστεί να υπολογίσουμε την απόκριση (Response) για συγκεκριμένες τιμές του Pt και του Eta και έπειτα για αυτά τα Response να δούμε την εικόνα των αποτελεσμάτων (Results) που θα προκύψει. Ας ξεκινήσουμε λοιπόν πρώτα ορίζοντας μέσω των παραμέτρων “genJetPt” και “genEta” ποίοι θα είναι αυτοί οι υπόχωροι:

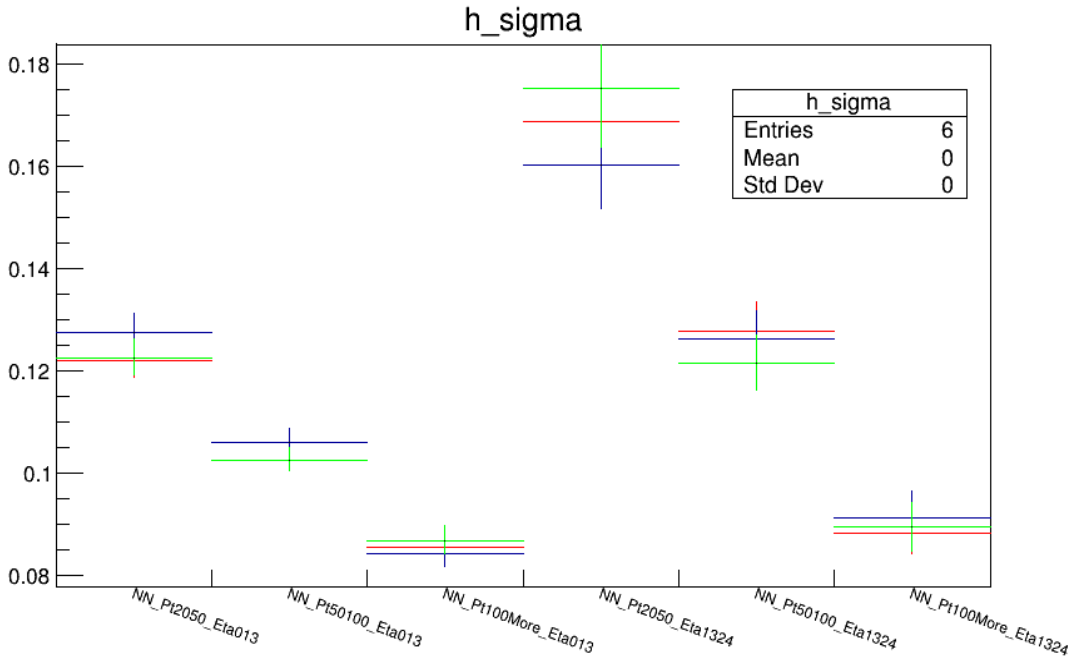
- Για την Παράμετρο “genJetPt” : [20 , 50] , [50 , 100] , [100 , 100+]

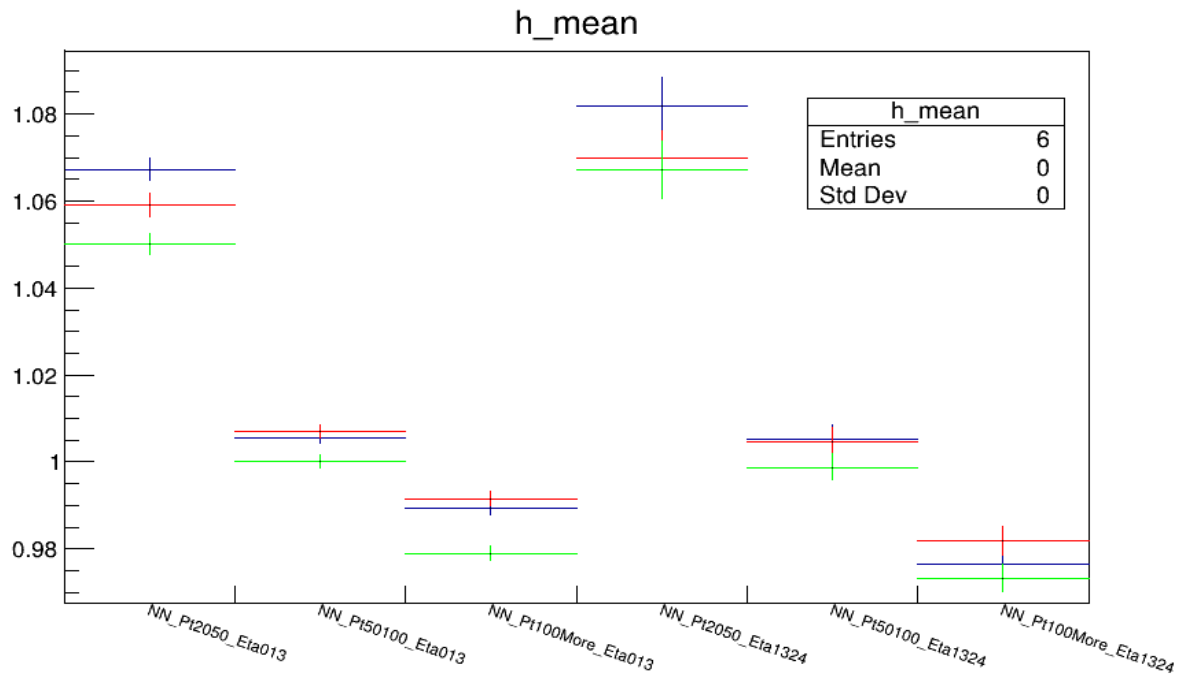
- Για την Παράμετρο “genEta” : [-1.3 , 1.3] , [-2.4 , -1.3] U [1.3 , 2.4]

Θα διαλέξουμε δηλαδή ως προς το Pt μικρές, μεσαιες και αρκετά υψηλές τιμές της ορμής και για το η (Eta) θα σκανάρουμε το “βαρέλι” του ανιχνευτή στο κοντινό και στο μακρινό του χώρο. Αυτή η ανάλυση των υπόχωρων θα γίνει για τον Γραμμικό Ταξινομητή (LD), το καλύτερο Δέντρο Απόφασης (BDTShr0.1NT200MD4), το καλύτερο Νευρωνικό Δίκτυο (NN10_5) όπως και για τις Raw – Corrected τιμές από το TTree (my_tree) του αρχικού αρχείου αρχείου (file.root). Αναλυτικά ο κώδικας για το πως φτιάχτηκαν τα νέα ιστογράμματα του Response και οι νέοι πίνακες που θα δούμε παρακάτω με τα Results του mean, του sigma και του std, βρίσκονται στο Παράρτημα 10.

➔ Ας δούμε για αρχή τα αποτελέσματα που προκύπτουν για το std, το sigma και το mean ως προς το καλύτερο Δέντρο Απόφασης (Πράσινο), το καλύτερο Νευρωνικό Δίκτυο (Μπλέ) και τον Γραμμικό Ταξινομητή (Κοκκινο).

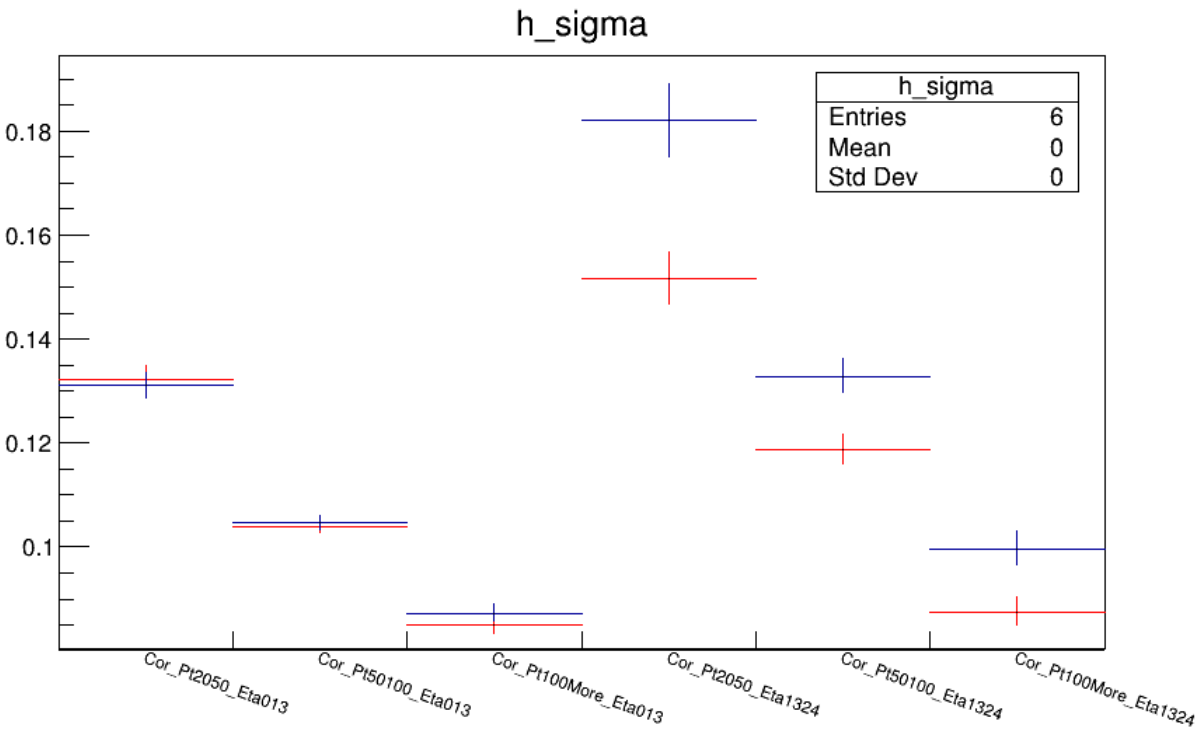


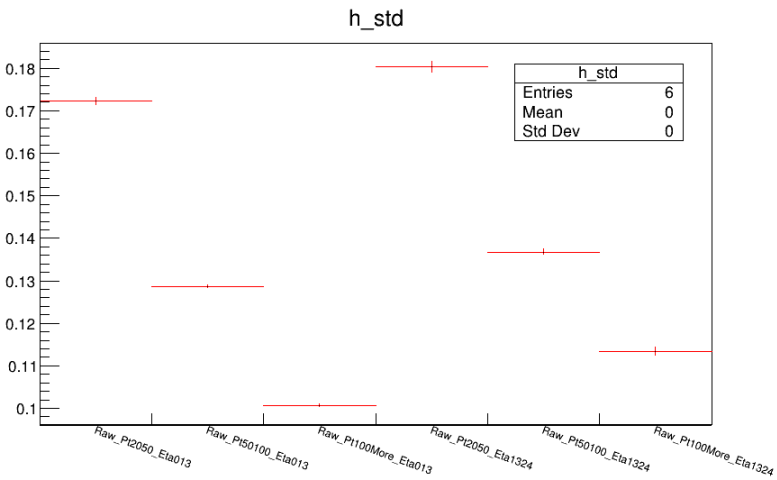


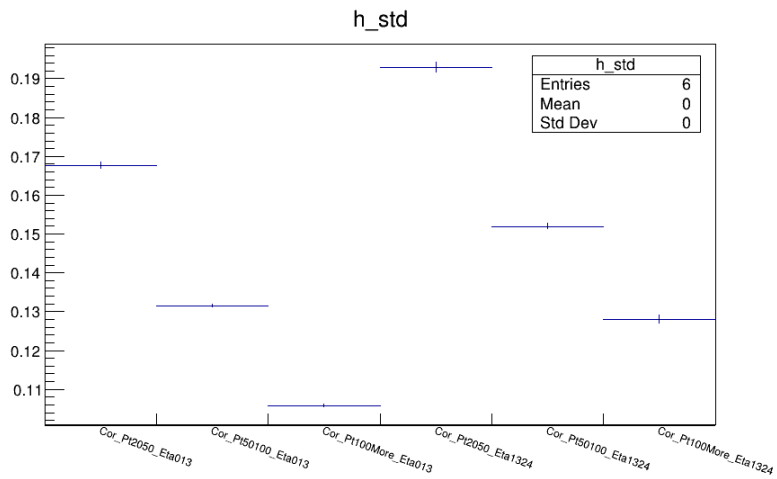


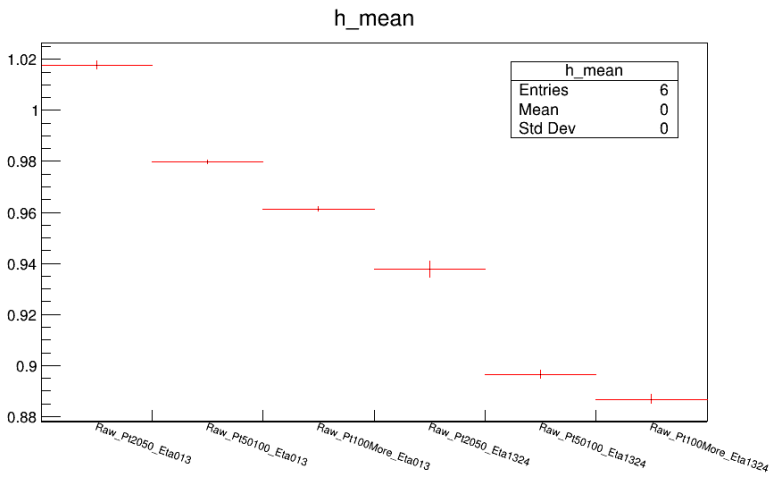
Τα παραπάνω διαγράμματα των αποτελεσμάτων μας δείχνουν καθαρά την συμπεριφορά που αναμένα για τις αποκρίσεις στους αντίστοιχους υπόχωρους. Για συγκεκριμένη τιμή του η (Eta) έστω μεταξύ | 0 – 1.3 | όσο αυξάνεται η τιμή του Pt βλέπουμε ότι μειώνεται το std και το sigma στο Response ενώ το mean πλησιάζει όλο και περισσότερο το 1. Δηλαδή για φιξαρισμένη περιοχή στο βαρέλι του ανιχνευτή η αύξηση της ορμής "στενεύει" το Response και άρα κάνει καλύτερα τα αποτελέσματά μας. Αυτό το βλέπουμε και για την περιοχή του η (Eta) μεταξύ | 0 – 1.3 | και για την περιοχή | 1.3 – 2.4 |. Δεν πρέπει να μας ανησυχεί το ότι για τις πιο υψηλές τιμές της ορμής Pt (100+) το mean δείχνει λίγο να απομακρύνεται από την τιμή του 1 μιας και το mean πάντα είναι πιο εύκολο να διορθωθεί απότι το std ή το sigma.

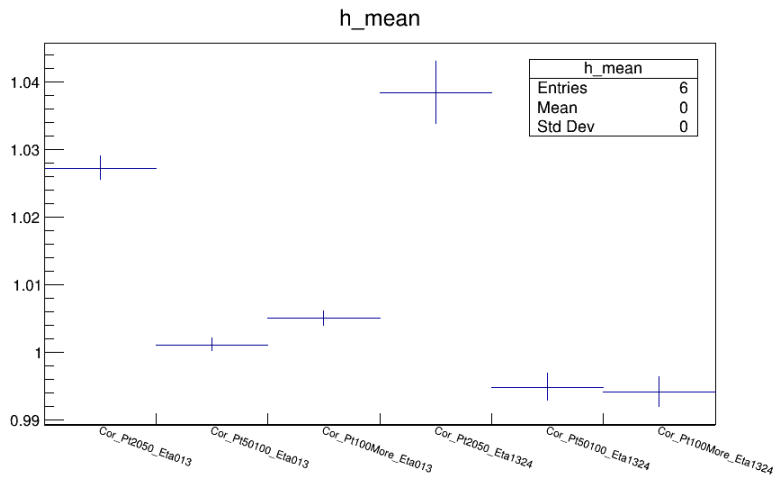
➔ Ας δούμε τώρα τα αποτελέσματα που προκύπτουν για το std, το sigma και το mean ως προς το Raw (Κόκκινο) και το Corrected (Μπλέ) Response.











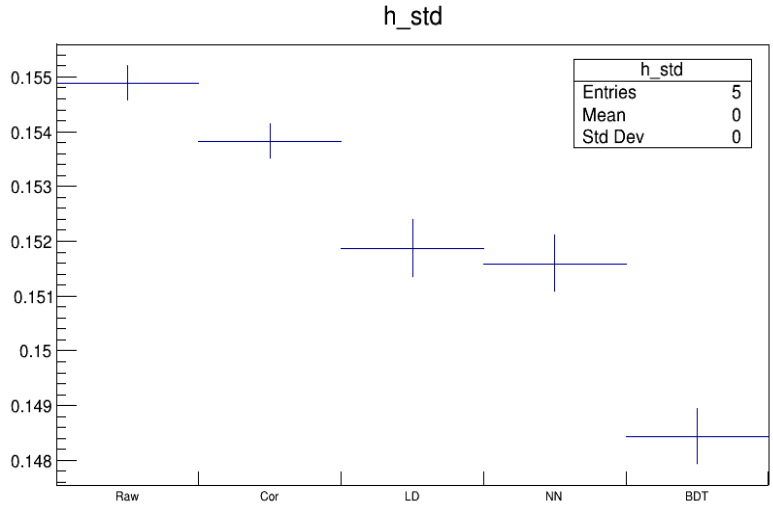
Για τις περιπτώσεις του std και του sigma για το Raw και το Corrected φαίνεται να έχουμε την συμπεριφορά που αναμέναμε και είδαμε και προηγουμένος στο BDT, το NN, και τον LD. Αυτό που βλέπουμε δηλαδή είναι πως η αύξηση της ορμής Pt για συγκεκριμένη τιμή του η (Eta) “στενεύει” κάθε φορά το ιστόγραμμα του Response, μέσω της μείωσης των τιμών του σίγμα και του std. Η διαφορά εδώ βρίσκετε στα αποτελέσματα του Mean τόσο για το Corrected και ειδικά για το Raw. Όπως φαίνεται στο Corrected (Μπλέ) για την περίπτωση με η (Eta) | 0 – 1.3 | και ορμής Pt πάνω από 100 (100More) το αποτέλεσμα που προκύπτει δεν είναι χαμηλότερο από το διπλανό του που έχει η (Eta) | 0 – 1.3 | και ορμή Pt μεταξύ 50 και 100. Την ιδιαίτερα ανώμαλη συμπεριφορά όμως την έχει το Raw το οποίο δεν ακολουθεί καθόλου το μοτίβο που είδαμε πριν στο BDT, το NN, και τον LD. Αυτα τα αποτελέσματα όμως δεν πρέπει να μας ανησυχούν διότι εκτός του ότι πάντα είναι πιο εύκολο να διορθωθεί το mean σε σχέση με το std ή το sigma, επιπλέον πρέπει να θυμηθούμε πως ειδικά για το Raw δεν περιμένουμε να υπάρχει φυσιολογική συμπεριφορά ως προς το mean το οποίο κιόλας δεν είναι καθόλου βαθμονομημένο.

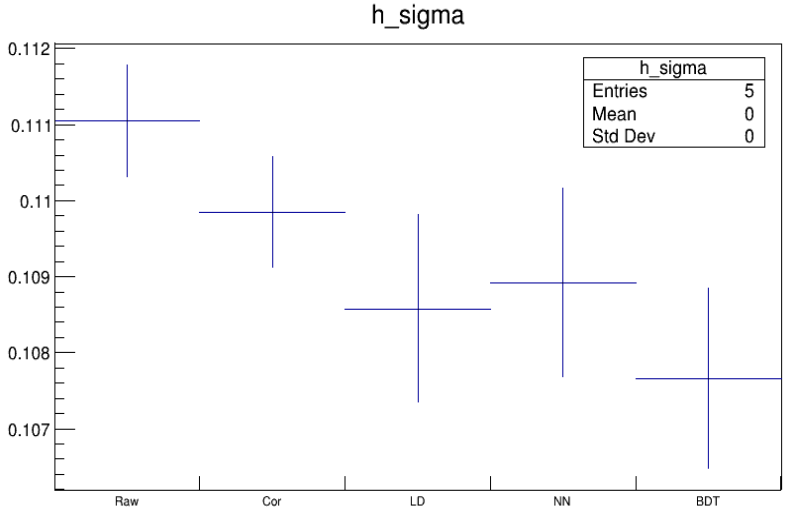
Όλος ο παραπάνω έλεγχος λοιπόν δεν αποκάλυψε κάποιο bias στους υπόχωρους του Pt και του Eta για την παλινδρόμηση που κάναμε. Απλός έδειξε και επιβεβαίωσε την συμπεριφορά που αναμέναμε να έχουν τα αποτελέσματα για την αύξηση της ορμής Pt ως προς συγκεκριμένες τιμές του η (Eta).

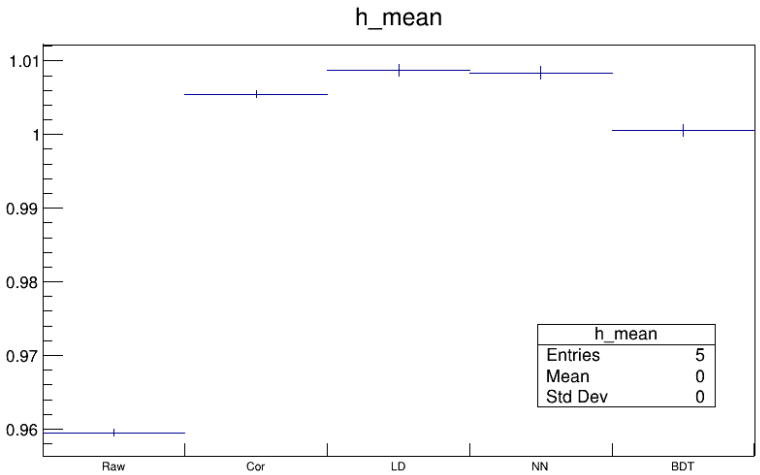
Επίλογος

"I'm feeling a sense of conclusion here, so let's draw things to a close."

Κλείνοντας είναι πολύ σημαντικό να αναφερθούν συνοπτικά και στοχευμένα τα συμπεράσματα που προκύπτουν από την ανάλυση που έγινε στο 2ο κεφάλαιο. Συμπίπτουν τα αποτελέσματά που προέκυψαν με την θεωρία του 1ου κεφαλαίου? Για να μπορέσουμε να το απαντήσουμε αυτό ας δούμε όλα τα αποτελέσμα (Results) των τριών μεθόδων παλινδρόμησης ως προς τα αποτελέσματα του Raw και του Corrected μιας και δεν πρέπει να ξεχνάμε ότι ως προς αυτά τα Response είναι που θέλουμε να κάνουμε την βελτιστοποίηση. Στα παρακάτω 3 τελευταία διαγράμματα έχουν τοποθετηθεί τα αποτελέσματα του καλύτερου Νευρωνικού Δικτύου (NN10_5) του καλύτερου Ενδυναμωμένου Δέντρου Απόφασης (BDTShr0.1NT200MD4) και αντίστοιχα του LD, του Raw και του Corrected. Αναλυτικά ο κώδικας για το πως φτιάχτηκαν βρίσκεται στο Παράρτημα 11.







Όπως φαίνεται καθαρά και όπως ξαν�ναφέραμε τα καλύτερα αποτελέσματα τα δίνει το Δέντρο Απόφασης και για τις τρείς παραμέτρους (mean, std, sigma). Συμφωνεί δηλαδή με την θεωρία ως η καλύτερη “out of the box” μέθοδος μιας και το μόνο που χρειάστηκε ήταν ένα απλό “σκανάρισμα” στις υπερπαραμέτρους. Σε αντίθεση με τα MLP Νευρωνικά Δίκτυα για τα οποία δοκιμάσαμε ένα μεγάλο αριθμό από αρχιτεκτονικές, διαφορετικές μεθόδους εκπαίδευσης και συναρτήσεις ενεργοποίησης για να καταφέρουν να είναι σε ένα μικρό βαθμό καλύτερο από τον γραμμικό ταξινομητή. Σίγουρα πάντως και οι τρείς μέθοδοι έδωσαν αρκετά ικανοποιητικές βελτιώσεις ως προς το Raw και το Corrected το οποία ήταν και ο αρχικός μας στόχος.

Μια περαιτέρω έρευνα για πιο βαθειές δομές MLP Νευρωνικών Δικτύων (Deep Learning) ή διαφορετικές δομές όπως τα convolutional neural networks ή τα recursive neural netwokrs είναι πιθανό να βελτίωναν σε ένα βαθμό τα τελικά αποτελέσματα για τα NN, αλλά θα αφήσουμε αυτή την έρευνα για μελλοντικές εργασίες.

Παραρτήματα

Παράρτημα 1

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου διαβάσαμε από το αρχείο της προσομοίωσης Monte Carlo: jetsCalibs_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root τις παραμέτρους που θα χρειαστούμε για την Παλινδρόμηση και τα αποθηκεύσαμε σε ένα νέο αρχείο: file.root μέσω στο TTree: my_tree

```
import ROOT
from ROOT import TFile,TCanvas,TTree,TH1F,gStyle,gPad,TVector2
from array import array

#open ROOT file
m_file = ROOT.TFile.Open( 'jetsCalibs_TT_TuneCUETP8M2T4_13TeV-powheg-pythia8.root', 'read' )

#Get the TTree
tree = m_file.Get("vj/events")

#Create an Output file and re-write it if it exists already
#Create a Tree
myfile = TFile( 'file.root', 'recreate'  )
mytree = TTree( 'my_tree'  , 'first_tree' )

#Define the Branches Variables
bjetRawPt  = array("f",[0.])
bjetPt     = array("f",[0.])
bjetEta    = array("f",[0.])
bjetPhi    = array("f",[0.])
bjetMass   = array("f",[0.])
bjetNhf    = array("f",[0.])
bjetPhf    = array("f",[0.])
bjetMuf    = array("f",[0.])
bjetElf    = array("f",[0.])
bjetChf    = array("f",[0.])
DR_min     = array("f",[0.])
bgenJetPt  = array("f",[0.])
bgenJetEta = array("f",[0.])

#Creat the Branches
mytree.Branch('bjetRawPt' , bjetRawPt , 'bjetRawPt/F' )
mytree.Branch('bjetPt'    , bjetPt    , 'bjetPt/F'    )
mytree.Branch('bjetEta'   , bjetEta   , 'bjetEta/F'   )
mytree.Branch('bjetPhi'   , bjetPhi   , 'bjetPhi/F'   )
mytree.Branch('bjetMass'  , bjetMass  , 'bjetMass/F'  )
mytree.Branch('bjetChf'   , bjetChf   , 'bjetChf/F'   )
mytree.Branch('bjetNhf'   , bjetNhf   , 'bjetNhf/F'   )
mytree.Branch('bjetPhf'   , bjetPhf   , 'bjetPhf/F'   )
```

```
mytree.Branch('bjetMuf'   , bjetMuf   , 'bjetMuf'     )
mytree.Branch('bjetElf'   , bjetElf   , 'bjetElf/F'   )
mytree.Branch('DR_min'    , DR_min    , 'DR_min/F'    )
mytree.Branch('bgenJetPt' , bgenJetPt , 'bgenJetPt/F' )
mytree.Branch('bgenJetEta', bgenJetEta, 'bgenJetEta/F')

#Creat Histogram for DR with more bins

h_DR_min   = TH1F("h_DR_min","h_DR_min",300,0.0,1.0)

#Loop over the events and then loop over the jets in each event
#And then loop over the Genjets
for i in range(tree.GetEntries()): #loop over the events
  tree.GetEntry(i)
  if tree.nJets>0:   #Remove the events without a jet(tree.nJets==0)
    for iJet in range(tree.nJets):  #loop over the jets in each event
      if tree.jetIsBtag[iJet] == True:
        bjetRawPt[0]  = float(tree.jetRawPt[iJet])
        bjetPt[0]     = float(tree.jetPt[iJet])
        bjetEta[0]    = float(tree.jetEta[iJet])
        bjetPhi[0]    = float(tree.jetPhi[iJet])
        bjetMass[0]   = float(tree.jetMass[iJet])
        bjetChf[0]    = float(tree.jetChf[iJet])
        bjetNhf[0]    = float(tree.jetNhf[iJet])
        bjetPhf[0]    = float(tree.jetPhf[iJet])
        bjetMuf[0]    = float(tree.jetMuf[iJet])
        bjetElf[0]    = float(tree.jetElf[iJet])

        #Loop over the GenJets
        dR_min = 10000.0
        for iGenJet in range(tree.genJetPt.size()):
                        dR_Jet_GenJet   =   ((tree.jetEta[iJet]-tree.genJetEta[iGenJet])**2.0          +
(TVector2.Phi_mpi_pi(tree.jetPhi[iJet]-tree.genJetPhi[iGenJet]))**2.0)**0.5
          if dR_Jet_GenJet < dR_min:
            dR_min = dR_Jet_GenJet
            bgenJetPt[0]  = float(tree.genJetPt[iGenJet])
            bgenJetEta[0] = float(tree.genJetEta[iGenJet])
        DR_min[0]         = float(dR_min)
        h_DR_min.Fill(dR_min)
        #Fill the Tree
        mytree.Fill()

#Write all created objects to the file
myfile.Write()
#Close the file
myfile.Close()
```

Παράρτημα 2

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου διαβάζουμε από το αρχείο που φτιάχτηκε στο Παράρτημα 1: file.root τις παραμέτρους (bjetPt, bjetRawPt, bgenJetPt) που θα χρειαστούμε για να υπολογίσουμε το Response και αφού γίνει ο υπολογισμός αποθηκεύουμε το αποτέλεσμα σε ένα Ιστόγραμμα (Response_Raw, Response_Cor).

```
import ROOT
from ROOT import TFile,TCanvas,TH1F,gStyle,gPad

gStyle.SetOptStat(0)

#Open files only to be read
newfile = TFile.Open("file.root", "read")

#Calls the script Tree we made before
newtree  = newfile.Get("my_tree")

#Opens a file and re-write it
outf=TFile("Response_Raw_Cor.root","recreate")

#Create the Histogram
Response_Raw = TH1F("Response_Raw","Response_Raw",300,0.0,3.0)
Response_Cor = TH1F("Response_Cor","Response_Cor",300,0.0,3.0)

#Read all entries and fill histogram with the ratio
for i in range(newtree.GetEntries()):
  newtree.GetEntry(i)
  Response_Raw.Fill(newtree.bjetRawPt/newtree.bgenJetPt)
  Response_Cor.Fill(newtree.bjetPt/newtree.bgenJetPt)

#Change file
outf.cd()

#Writes the histograms in my file
Response_Raw.Write()
Response_Cor.Write()

#Close the files
outf.Close()
newfile.Close()
```

Παράρτημα 3

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου κάνοντας χρήση του TMVA φτιάξαμε το αρχείο με όνομα TMVA_LD.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στο Branch με το όνομα LD.

```
from ROOT import TMVA, TFile, TTree, TCut
from subprocess import call
from os.path import isfile

# Setup TMVA
TMVA.Tools.Instance()

#Create an Output file and re-write it if it exists already
output = TFile.Open('TMVA_LD.root', 'RECREATE')
factory = TMVA.Factory('TMVARegression', output,
        '!V:!Silent:Color:DrawProgressBar:Transformations=D,G:AnalysisType=Regression')

# Load data
#Get the Data from the TFile and TTree we made before
data = TFile.Open('file.root')
tree = data.Get('my_tree')
dataloader = TMVA.DataLoader('dataset')
dataloader.AddVariable('bjetRawPt')
dataloader.AddVariable('bjetPt')
dataloader.AddVariable('bjetEta')
dataloader.AddVariable('bjetPhi')
dataloader.AddVariable('bjetMass')
dataloader.AddVariable('bjetChf')
dataloader.AddVariable('bjetNhf')
dataloader.AddVariable('bjetPhf')
dataloader.AddVariable('bjetMuf')
dataloader.AddVariable('bjetElf')
dataloader.AddVariable('bgenJetEta')
#Target ekpedeusis ta GenJet

dataloader.AddTarget('bgenJetPt')
dataloader.AddRegressionTree(tree, 1.0)
dataloader.PrepareTrainingAndTestTree(TCut(''),
        'nTrain_Regression=70000:SplitMode=Random:NormMode=NumEvents:!V')
# Book methods
factory.BookMethod(dataloader, TMVA.Types.kLD, "LD", '!H:!V')
# Run TMVA
factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```

Παράρτημα 4

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου φτιάξαμε τα Response ιστογράμματα από το αρχείο TMVA_LD.root, κάναμε το fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και επιπλέον φτιάξαμε τα ιστογράμματα με τα αποτελέσματα (Results) για το mean, sigma και std.

```
import ROOT
from ROOT import TFile,TCanvas,TH1F,gStyle,gPad

gStyle.SetOptStat(0)

#Opens a file only to be read
newfile = TFile.Open("TMVA_LD.root", "read")

#Calls the script Tree that was made before
newtree= newfile.Get("dataset/TestTree")              # /TrainTree for Training Set

#Opens a file and re-write it
outf=TFile("histos_ResponseLD_Test.root","recreate") # "histos_ResponseLD_Train.root"
                                                      # for Training Set

#==Make the Response Histograms==#

#Create the histogram
ResponseLD  = TH1F("ResponseLD","ResponseLD",300,0.0,3.0)

ResponseRaw = TH1F("ResponseRaw","ResponseRaw",300,0.0,3.0)
ResponseCor = TH1F("ResponseCor","ResponseCor",300,0.0,3.0)

#Read all entries and fill histograms
for i in range(newtree.GetEntries()):
  newtree.GetEntry(i)
  ResponseLD.Fill(newtree.LD/newtree.bgenJetPt)
  ResponseRaw.Fill(newtree.bjetRawPt/newtree.bgenJetPt)
  ResponseCor.Fill(newtree.bjetPt/newtree.bgenJetPt)

#Change file
outf.cd()

#Writes the histograms in my file
ResponseLD.Write()
ResponseRaw.Write()
ResponseCor.Write()

#Close the files
outf.Close()
```

```
newfile.Close()

from ROOT import TH1F, TF1, TFile
from ROOT import gROOT

#Opens a file only to be read

LDfile = TFile.Open("histos_ResponseLD_Test.root","read")  #
histos_ResponseLD_Train.root for
# Training Set

#Read the Histograme from the TFiles
hCor = LDfile.Get("ResponseCor")    # The Correncted Response Histograme
hRaw = LDfile.Get("ResponseRaw")    # The Raw        Response Histograme
hLD  = LDfile.Get("ResponseLD")     # The LD         Response Histograme

#Read the std, stdErr, mean, meanErr from the Histos
stdCor     = hCor.GetStdDev()
stdErrCor  = hCor.GetStdDevError()
meanCor    = hCor.GetMean()
meanErrCor = hCor.GetMeanError()

stdRaw     = hRaw.GetStdDev()
stdErrRaw  = hRaw.GetStdDevError()
meanRaw    = hRaw.GetMean()
meanErrRaw = hRaw.GetMeanError()

stdLD     = hLD.GetStdDev()
stdErrLD  = hLD.GetStdDevError()
meanLD    = hLD.GetMean()
meanErrLD = hLD.GetMeanError()

#=Do the truncated Fits=#

#For the Corrected Histograme           #== +/- 1 std's ==#
x1Cor  = meanCor - stdCor
x2Cor  = meanCor + stdCor
FitFunctionCor = TF1( 'Cor', 'gaus', x1Cor, x2Cor)
hCor.Fit(FitFunctionCor, 'R+')

#For the Raw Histograme
x1Raw  = meanRaw - stdRaw
x2Raw  = meanRaw + stdRaw
FitFunctionRaw = TF1( 'Raw', 'gaus', x1Raw, x2Raw)
hRaw.Fit(FitFunctionRaw, 'R+')

#For the LD Histograme                  #== +/- 1 std's ==#
x1LD  = meanLD - stdLD
x2LD  = meanLD + stdLD
FitFunctionLD = TF1( 'LD', 'gaus', x1LD, x2LD)
hLD.Fit(FitFunctionLD, 'R+')
```

```
#Save the Mean and Sigma values from the Fits

mCor_fit    = FitFunctionCor.GetParameter(1)
mCorErr_fit = FitFunctionCor.GetParError(1)
sCor_fit    = FitFunctionCor.GetParameter(2)
sCorErr_fit = FitFunctionCor.GetParError(2)

mRaw_fit    = FitFunctionRaw.GetParameter(1)
mRawErr_fit = FitFunctionRaw.GetParError(1)
sRaw_fit    = FitFunctionRaw.GetParameter(2)
sRawErr_fit = FitFunctionRaw.GetParError(2)

mLD_fit    = FitFunctionLD.GetParameter(1)
mLDErr_fit = FitFunctionLD.GetParError(1)
sLD_fit    = FitFunctionLD.GetParameter(2)
sLDErr_fit = FitFunctionLD.GetParError(2)

#===Do the Evolving Plot===#

#Create an Output file to store the Histos
outf = TFile('histos_ResultsLD_Test.root', 'recreate')  # histos_ResultsLD_Train.root for

# Training Set

#Creat the mean, sigma and std Histos
h_mean  = TH1F("h_mean" ,"h_mean" ,3,0,3)
h_sigma = TH1F("h_sigma","h_sigma",3,0,3)
h_std   = TH1F("h_std"  ,"h_std"  ,3,0,3)

#Creat a Histo only for the LD Results (with 1 bin)
h_LDMean   = TH1F("h_LDMean"  ,"h_LDMean"  ,1,0,110)
h_LDSigma  = TH1F("h_LDSigma" ,"h_LDSigma" ,1,0,110)
h_LDStd    = TH1F("h_LDStd"   ,"h_LDStd"   ,1,0,110)

#Fill the Histos

h_mean.SetBinContent(1,mRaw_fit)
h_mean.SetBinError(1,mRawErr_fit)
h_mean.SetBinContent(2,mCor_fit)
h_mean.SetBinError(2,mCorErr_fit)
h_mean.SetBinContent(3,mLD_fit)
h_mean.SetBinError(3,mLDErr_fit)

h_sigma.SetBinContent(1,sRaw_fit)
h_sigma.SetBinError(1,sRawErr_fit)
h_sigma.SetBinContent(2,sCor_fit)
h_sigma.SetBinError(2,sCorErr_fit)
h_sigma.SetBinContent(3,sLD_fit)
h_sigma.SetBinError(3,sLDErr_fit)
```

```
h_std.SetBinContent(1,stdRaw)
h_std.SetBinError(1,stdErrRaw)
h_std.SetBinContent(2,stdCor)
h_std.SetBinError(2,stdErrCor)
h_std.SetBinContent(3,stdLD)
h_std.SetBinError(3,stdErrLD)

#For the 1 Bin Histos => Fill 1 Value

h_LDMean.SetBinContent(1,mLD_fit)
h_LDSigma.SetBinContent(1,sLD_fit)
h_LDStd.SetBinContent(1,stdLD)

#Name the Bins of the Histos

h_mean.GetXaxis().SetBinLabel(1,"Raw")
h_mean.GetXaxis().SetBinLabel(2,"Cor")
h_mean.GetXaxis().SetBinLabel(3,"LD")

h_sigma.GetXaxis().SetBinLabel(1,"Raw")
h_sigma.GetXaxis().SetBinLabel(2,"Cor")
h_sigma.GetXaxis().SetBinLabel(3,"LD")

h_std.GetXaxis().SetBinLabel(1,"Raw")
h_std.GetXaxis().SetBinLabel(2,"Cor")
h_std.GetXaxis().SetBinLabel(3,"LD")

#Writes the histograms in my file

h_mean.Write()
h_sigma.Write()
h_std.Write()
hLD.Write()
hCor.Write()                    #Save the Fit+Histo
hRaw.Write()                    #Save the Fit+Histo
h_LDMean.Write()
h_LDSigma.Write()
h_LDStd.Write()

#Close the files
outf.Close()
```

Παράρτημα 5

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου κάνοντας χρήση του TMVA φτιάξαμε το αρχείο με όνομα TMVA_BDT.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 108 Branches με ονόματα BDTShrxNTyMDz , όπου x, y, z οι τιμές για το Shrinkage, το NTrees και το MaxDepth.

```
from ROOT import TMVA, TFile, TTree, TCut
from subprocess import call
from os.path import isfile

# Setup TMVA

TMVA.Tools.Instance()

#Create an Output file and re-write it if it exists already

output = TFile.Open('TMVA_BDT.root', 'RECREATE')
factory = TMVA.Factory('TMVARegression', output,
        '!V:!Silent:Color:DrawProgressBar:Transformations=D,G:AnalysisType=Regression')

# Load data

#Get the Data from the TFile and TTree we made before

data = TFile.Open('file.root')
tree = data.Get('my_tree')
dataloader = TMVA.DataLoader('dataset')

#Add the Variables

dataloader.AddVariable('bjetRawPt')
dataloader.AddVariable('bjetPt')
dataloader.AddVariable('bjetEta')
dataloader.AddVariable('bjetPhi')
dataloader.AddVariable('bjetMass')
dataloader.AddVariable('bjetChf')
dataloader.AddVariable('bjetNhf')
dataloader.AddVariable('bjetPhf')
dataloader.AddVariable('bjetMuf')
dataloader.AddVariable('bjetElf')
dataloader.AddVariable('bgenJetEta')

#Target ekpedeusis ta GenJet

dataloader.AddTarget('bgenJetPt')
```

```
dataloader.AddRegressionTree(tree, 1.0)
dataloader.PrepareTrainingAndTestTree(TCut(''),
        'nTrain_Regression=70000:SplitMode=Random:NormMode=NumEvents:!V')

#===Book Methods===#

Shrinkage = ['0.1','0.3','0.5','0.7','1.0' ,'1.5' ]
NTrees    = ['50' ,'100','200','500','1000','2000']
MaxDepth  = ['3'  ,'4'  ,'5'                      ]

#Scan over BDT HyperParameters

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      factory.BookMethod(dataloader, TMVA.Types.kBDT, 'BDTShr'+S+'NT'+NT+'MD'+MD,
'!H:!V:NTrees='+NT+':BoostType=Grad:Shrinkage='+S+':nCuts=20:MaxDepth='+MD)

# Run TMVA

factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```

Παράρτημα 6

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου φτιάξαμε τα Response ιστογράμματα από το αρχείο TMVA_BDT.root, κάναμε το fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και επιπλέον φτιάξαμε τα ιστογράμματα με τα αποτελέσματα (Results) για το mean, sigma και std.

```
import ROOT
from ROOT import TFile,TCanvas,TH1F,gStyle,gPad
from array import array

gStyle.SetOptStat(0)

#Opens a file only to be read
newfile = TFile.Open("TMVA_BDT.root", "read")

#Calls the script Tree that was made before
newtree= newfile.Get("dataset/TestTree")                # /TrainTree for Training Set

#Opens a file and re-write it
outf=TFile("histos_BDToutput_Test.root","recreate")    # "histos_ResponseLD_Train.root"
for                                                                    #   Training Set

#Create the histograms
Shrinkage   = ['0.1','0.3','0.5','0.7','1.0' ,'1.5' ]
NTrees      = ['50' ,'100','200','500','1000','2000']
MaxDepth    = ['3'  ,'4'  ,'5'                      ]
name_holder = {}
BDT_holder  = {}

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      name_holder['ResponseBDTShr'+S+'NT'+NT+'MD'+MD] =
TH1F('ResponseBDTShr'+S+'NT'+NT+'MD'+MD,'ResponseBDTShr'+S+'NT'+NT+'MD'+M
D,300,0.0,3.0)
      BDT_holder['BDTShr'+S+'NT'+NT+'MD'+MD] = array('f',[0.])

newtree.SetBranchAddress('BDTShr'+S+'NT'+NT+'MD'+MD,BDT_holder['BDTShr'+S+'NT'
+NT+'MD'+MD])

ResponseRaw = TH1F("ResponseRaw","ResponseRaw",300,0.0,3.0)
ResponseCor = TH1F("ResponseCor","ResponseCor",300,0.0,3.0)

#Read all entries and fill histograms
for i in range(newtree.GetEntries()):
  newtree.GetEntry(i)
```

```
  ResponseRaw.Fill(newtree.bjetRawPt/newtree.bgenJetPt)
  ResponseCor.Fill(newtree.bjetPt/newtree.bgenJetPt)
  for S in Shrinkage:
    for NT in NTrees:
      for MD in MaxDepth:

name_holder['ResponseBDTShr'+S+'NT'+NT+'MD'+MD].Fill(BDT_holder['BDTShr'+S+'NT'
+NT+'MD'+MD][0]/newtree.bgenJetPt)

#Change file
outf.cd()
#Writes the histograms in my file
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      name_holder['ResponseBDTShr'+S+'NT'+NT+'MD'+MD].Write()
ResponseRaw.Write()
ResponseCor.Write()

#Close the files
outf.Close()
newfile.Close()

from ROOT import TH1F, TF1, TFile
from ROOT import gROOT

#Opens a the TFiles only to be read
BDTfile = Tfile.Open("histos_BDToutput_Test.root","read")  # histos_BDToutput_Train.root
for                                                                                  # Training Set

#Read the Histogrames from the TFiles
hCor = BDTfile.Get("ResponseCor")    # The Correncted Response Histograme
hRaw = BDTfile.Get("ResponseRaw")    # The Raw        Response Histograme

Shrinkage   = ['0.1','0.3','0.5','0.7','1.0' ,'1.5' ]
NTrees      = ['50' ,'100','200','500','1000','2000']
MaxDepth    = ['3'  ,'4'  ,'5'                      ]
h_holder    = {}

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD] =
BDTfile.Get('ResponseBDTShr'+S+'NT'+NT+'MD'+MD) # The BDT's Histogrames

#Read the std, stdErr, mean, meanErr from the Histos
stdCor     = hCor.GetStdDev()
stdErrCor  = hCor.GetStdDevError()
meanCor    = hCor.GetMean()
meanErrCor = hCor.GetMeanError()
```

```
stdRaw     = hRaw.GetStdDev()
stdErrRaw  = hRaw.GetStdDevError()
meanRaw    = hRaw.GetMean()
meanErrRaw = hRaw.GetMeanError()

std_holder     = {}
stdErr_holder  = {}
mean_holder    = {}
meanErr_holder = {}

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      std_holder['stdBDTShr'+S+'NT'+NT+'MD'+MD]         =
h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].GetStdDev()
      stdErr_holder['stdErrBDTShr'+S+'NT'+NT+'MD'+MD]   =
h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].GetStdDevError()
      mean_holder['meanBDTShr'+S+'NT'+NT+'MD'+MD]       =
h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].GetMean()
      meanErr_holder['meanErrBDTShr'+S+'NT'+NT+'MD'+MD] =
h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].GetMeanError()

#==Do the truncated Fits==#

#For the Corrected Histograme
x1Cor  = meanCor - stdCor
x2Cor  = meanCor + stdCor
FitFunctionCor = TF1( 'Cor', 'gaus', x1Cor, x2Cor)
hCor.Fit(FitFunctionCor, 'R+')

#For the Raw Histograme
x1Raw  = meanRaw - stdRaw
x2Raw  = meanRaw + stdRaw
FitFunctionRaw = TF1( 'Raw', 'gaus', x1Raw, x2Raw)
hRaw.Fit(FitFunctionRaw, 'R+')

#For the BDT Histogrames

x1BDT_holder       = {}
x2BDT_holder       = {}
FitFunction_holder = {}

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      x1BDT_holder['x1BDT'+S+'NT'+NT+'MD'+MD]  =
mean_holder['meanBDTShr'+S+'NT'+NT+'MD'+MD] -
std_holder['stdBDTShr'+S+'NT'+NT+'MD'+MD]
      x2BDT_holder['x2BDT'+S+'NT'+NT+'MD'+MD]  =
mean_holder['meanBDTShr'+S+'NT'+NT+'MD'+MD] +
std_holder['stdBDTShr'+S+'NT'+NT+'MD'+MD]
```

```
      FitFunction_holder['FitFunctionBDTShr'+S+'NT'+NT+'MD'+MD] =
TF1('BDT'+S+'NT'+NT+'MD'+MD,'gaus', x1BDT_holder['x1BDT'+S+'NT'+NT+'MD'+MD],
x2BDT_holder['x2BDT'+S+'NT'+NT+'MD'+MD])

h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].Fit(FitFunction_holder['FitFunctionBDTShr'+S+
'NT'+NT+'MD'+MD], 'R+')

#Save the Mean and Sigma values from the Fits
mCor_fit    = FitFunctionCor.GetParameter(1)
mCorErr_fit = FitFunctionCor.GetParError(1)
sCor_fit    = FitFunctionCor.GetParameter(2)
sCorErr_fit = FitFunctionCor.GetParError(2)

mRaw_fit    = FitFunctionRaw.GetParameter(1)
mRawErr_fit = FitFunctionRaw.GetParError(1)
sRaw_fit    = FitFunctionRaw.GetParameter(2)
sRawErr_fit = FitFunctionRaw.GetParError(2)

mBDTfit_holder    = {}
mBDTErrfit_holder = {}
sBDTfit_holder    = {}
sBDTErrfit_holder = {}

for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      mBDTfit_holder['mBDT'+S+'NT'+NT+'MD'+MD]       =
FitFunction_holder['FitFunctionBDTShr'+S+'NT'+NT+'MD'+MD].GetParameter(1)
      mBDTErrfit_holder['mBDTErr'+S+'NT'+NT+'MD'+MD] =
FitFunction_holder['FitFunctionBDTShr'+S+'NT'+NT+'MD'+MD].GetParError(1)
      sBDTfit_holder['sBDT'+S+'NT'+NT+'MD'+MD]       =
FitFunction_holder['FitFunctionBDTShr'+S+'NT'+NT+'MD'+MD].GetParameter(2)
      sBDTErrfit_holder['sBDTErr'+S+'NT'+NT+'MD'+MD] =
FitFunction_holder['FitFunctionBDTShr'+S+'NT'+NT+'MD'+MD].GetParError(2)

#===Do the Results Plot===#

#Create an Output file to store the Histos
outf = TFile('histos_ResultsBDTs_Test.root', 'recreate')

#Create the mean, sigma and std Histos => Raw,Cor + 108 BDT's = 110 bins
h_mean  = TH1F("h_mean" ,"h_mean" ,110,0,110)
h_sigma = TH1F("h_sigma","h_sigma",110,0,110)
h_std   = TH1F("h_std"  ,"h_std"  ,110,0,110)

#New for Task10 => Better Visually
h_RawMean   = TH1F("h_RawMean"  ,"h_RawMean"  ,1,0,110)
h_RawSigma  = TH1F("h_RawSigma" ,"h_RawSigma" ,1,0,110)
h_RawStd    = TH1F("h_RawStd"   ,"h_RawStd"   ,1,0,110)
h_CorMean   = TH1F("h_CorMean"  ,"h_CorMean"  ,1,0,110)
h_CorSigma  = TH1F("h_CorSigma" ,"h_CorSigma" ,1,0,110)
```

```
h_CorStd    = TH1F("h_CorStd"   ,"h_CorStd"   ,1,0,110)

#Fill the Histos
h_mean.SetBinContent(1,mRaw_fit)
h_mean.SetBinError(1,mRawErr_fit)
h_mean.SetBinContent(2,mCor_fit)
h_mean.SetBinError(2,mCorErr_fit)
#Here for the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_mean.SetBinContent(i,mBDTfit_holder['mBDT'+S+'NT'+NT+'MD'+MD])
      h_mean.SetBinError(i,mBDTErrfit_holder['mBDTErr'+S+'NT'+NT+'MD'+MD])
      i = i + 1

h_sigma.SetBinContent(1,sRaw_fit)
h_sigma.SetBinError(1,sRawErr_fit)
h_sigma.SetBinContent(2,sCor_fit)
h_sigma.SetBinError(2,sCorErr_fit)
#Here for the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_sigma.SetBinContent(i,sBDTfit_holder['sBDT'+S+'NT'+NT+'MD'+MD])
      h_sigma.SetBinError(i,sBDTErrfit_holder['sBDTErr'+S+'NT'+NT+'MD'+MD])
      i = i + 1

h_std.SetBinContent(1,stdRaw)
h_std.SetBinError(1,stdErrRaw)
h_std.SetBinContent(2,stdCor)
h_std.SetBinError(2,stdErrCor)
#Here for the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_std.SetBinContent(i,std_holder['stdBDTShr'+S+'NT'+NT+'MD'+MD])
      h_std.SetBinError(i,stdErr_holder['stdErrBDTShr'+S+'NT'+NT+'MD'+MD])
      i = i + 1

#Here for the Cor and Raw Histos => Task10
h_RawMean.SetBinContent(1,mRaw_fit)
h_RawSigma.SetBinContent(1,sRaw_fit)
h_RawStd.SetBinContent(1,stdRaw)
h_CorMean.SetBinContent(1,mCor_fit)
h_CorSigma.SetBinContent(1,sCor_fit)
h_CorStd.SetBinContent(1,stdCor)

#Name the Bins of the Histos
```

```
h_mean.GetXaxis().SetBinLabel(1,"Raw")
h_mean.GetXaxis().SetBinLabel(2,"Cor")
#Name the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_mean.GetXaxis().SetBinLabel(i,'BDTShr'+S+'NT'+NT+'MD'+MD)
      i = i + 1

h_sigma.GetXaxis().SetBinLabel(1,"Raw")
h_sigma.GetXaxis().SetBinLabel(2,"Cor")
#Name the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_sigma.GetXaxis().SetBinLabel(i,'BDTShr'+S+'NT'+NT+'MD'+MD)
      i = i + 1

h_std.GetXaxis().SetBinLabel(1,"Raw")
h_std.GetXaxis().SetBinLabel(2,"Cor")
#Name the 108 BDT's
i = 3
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_std.GetXaxis().SetBinLabel(i,'BDTShr'+S+'NT'+NT+'MD'+MD)
      i = i + 1

#Writes the histograms in my file
h_mean.Write()
h_sigma.Write()
h_std.Write()
h_RawMean.Write()
h_RawSigma.Write()
h_RawStd.Write()
h_CorMean.Write()
h_CorSigma.Write()
h_CorStd.Write()

#Plus Write the Fitted Histos
for S in Shrinkage:
  for NT in NTrees:
    for MD in MaxDepth:
      h_holder['hBDTShr'+S+'NT'+NT+'MD'+MD].Write()
hCor.Write()
hRaw.Write()

#Close the files
outf.Close()
```

Παράρτημα 7

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου κάνοντας χρήση του TMVA φτιάξαμε το αρχείο με όνομα TMVA_NN_BP.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 30 Branches με ονόματα MLP_X_Y , όπου X, Y ο αριθμός των Νευρώνων σε κάθε κρυφό στρώμα του Νετρωνικού Δικτύου.

```
from ROOT import TMVA, TFile, TTree, TCut
from subprocess import call
from os.path import isfile

# Setup TMVA

TMVA.Tools.Instance()

#Create an Output file and re-write it if it exists already

output = TFile.Open('TMVA_NN_BP.root', 'RECREATE')
factory = TMVA.Factory('TMVARegression', output,
        '!V:!Silent:Color:DrawProgressBar:Transformations=D,G:AnalysisType=Regression')

# Load data

#Get the Data from the TFile and TTree we made before
data = TFile.Open('file.root')
tree = data.Get('my_tree')
dataloader = TMVA.DataLoader('dataset')

dataloader.AddVariable('bjetRawPt')
dataloader.AddVariable('bjetPt')
dataloader.AddVariable('bjetEta')
dataloader.AddVariable('bjetPhi')
dataloader.AddVariable('bjetMass')
dataloader.AddVariable('bjetChf')
dataloader.AddVariable('bjetNhf')
dataloader.AddVariable('bjetPhf')
dataloader.AddVariable('bjetMuf')
dataloader.AddVariable('bjetElf')

#Target ekpedeusis ta GenJet

dataloader.AddTarget('bgenJetPt')

dataloader.AddRegressionTree(tree, 1.0)
dataloader.PrepareTrainingAndTestTree(TCut(''),
```

```
        'nTrain_Regression=70000:SplitMode=Random:NormMode=NumEvents:!V')

#======Book methods======#
#Scan over NN Arcitectures

#==IMPORTANT==# For Changes in the input Variables only change the NVars Value!

#Make a 1d matrix (N) with strings : NVars/2 , NVars/2+1 , ... , 2NVars => Here ['5' , '6' ,
'7' , ... , '20']
NVars = 10                  #Number of Variables => TMVA knows it as N
N  = []                     #Use only integers

for n in range(NVars/2,2*NVars):
  N.append(str(n))

# 1 Hidden Layer

for HL in N:
  factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP"+HL,
     '!H:!V:NCycles=350:VarTransform=N:HiddenLayers='+HL+':TrainingMethod=BP')

# 2 Hidden Layers

for HL in N:
  factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP"+str(NVars)+"_"+HL,
                                  '!H:!V:NCycles=350:VarTransform=N:HiddenLayers='+str(NVars)
+','+HL+'TrainingMethod=BP')

# Run TMVA

factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```

Παράρτημα 8

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου φτιάξαμε τα Response ιστογράμματα από το αρχείο TMVA_NN_BP.root, κάναμε το fit τους σε πεπερασμένο εύρος της κατανομής (truncated fit) και επιπλέον φτιάξαμε τα ιστογράμματα με τα αποτελέσματα (Results) για το mean, sigma και std.

```
import ROOT
from ROOT import TFile,TCanvas,TH1F,gStyle,gPad
from array import array

gStyle.SetOptStat(0)

#Opens a file only to be read
newfile = TFile.Open("TMVA_NN_BP.root", "read")

#Calls the script Tree that was made before
newtree= newfile.Get("dataset/TestTree")          #/TrainTree for Training Set

#Opens a file and re-write it
outf=TFile("histos_NNBPoutput_Test.root","recreate") # "histos_ResponseLD_Train.root"
for Training Set

#Create the histograms
NVars = 10

N  = []
for n in range(NVars/2,2*NVars):
  N.append(str(n))
name_holder = {}
NN_holder   = {}

for HL in N:
     name_holder['ResponseNN'+HL] =
TH1F('ResponseNN'+HL,'ResponseNN'+HL,300,0.0,3.0) # 1 Hidden Layer
     name_holder['ResponseNN'+str(NVars)+"_"+HL] = TH1F('ResponseNN'+str(NVars)
+"_"+HL,'ResponseNN'+str(NVars)+"_"+HL,300,0.0,3.0) #2 HL
     NN_holder['NN'+str(NVars)+"_"+HL] = array('f',[0.])  # 2 HL
     NN_holder['NN'+HL]                = array('f',[0.])  # 1 HL
     newtree.SetBranchAddress('MLP'+str(NVars)+"_"+HL,NN_holder['NN'+str(NVars)
+"_"+HL])
     newtree.SetBranchAddress('MLP'+HL,NN_holder['NN'+HL])

#Read all entries and fill histograms
for i in range(newtree.GetEntries()):
  newtree.GetEntry(i)
  for HL in N:
```

```
    name_holder['ResponseNN'+HL].Fill(NN_holder['NN'+HL][0]/newtree.bgenJetPt) #Eirini
#1 HL
    name_holder['ResponseNN'+str(NVars)+"_"+HL].Fill(NN_holder['NN'+str(NVars)
+"_"+HL][0]/newtree.bgenJetPt) #2 HL
#Change file
outf.cd()

#Writes the histograms in my file
for HL in N:
  name_holder['ResponseNN'+HL].Write()              #1 HL
  name_holder['ResponseNN'+str(NVars)+"_"+HL].Write() #2 HL

#Close the files
outf.Close()
newfile.Close()


from ROOT import TH1F, TF1, TFile
from ROOT import gROOT


#Opens a the TFiles only to be read
newfile = TFile.Open("RawCorResponse.root", "read")
NNfile = TFile.Open("histos_NNBPoutput_Test.root","read")  #
"histos_NNBPoutput_Train.root" for Train Set

#Read the Histogrames from the TFiles
hCor = newfile.Get("h_Pt_over_GenPt")    # The Correncted Histograme
hRaw = newfile.Get("h_RawPt_over_GenPt") # The Raw        Histograme

NVars = 10

N  = []
for n in range(NVars/2,2*NVars):
  N.append(str(n))
h_holder    = {}

for HL in N:
  h_holder['hNN'+HL] = NNfile.Get('ResponseNN'+HL)                          # The 1 HL NN's
Histogrames
  h_holder['hNN'+str(NVars)+"_"+HL] = NNfile.Get('ResponseNN'+str(NVars)+"_"+HL) #
The 2 HL NN's Histogrames

#Read the std, stdErr, mean, meanErr from the Histos
stdCor     = hCor.GetStdDev()
stdErrCor  = hCor.GetStdDevError()
meanCor    = hCor.GetMean()
meanErrCor = hCor.GetMeanError()

stdRaw     = hRaw.GetStdDev()
stdErrRaw  = hRaw.GetStdDevError()
```

```
meanRaw    = hRaw.GetMean()
meanErrRaw = hRaw.GetMeanError()

std_holder     = {}
stdErr_holder  = {}
mean_holder    = {}
meanErr_holder = {}

for HL in N:
  std_holder['stdNN'+HL]         = h_holder['hNN'+HL].GetStdDev()                    # 1 HL
  stdErr_holder['stdErrNN'+HL]   = h_holder['hNN'+HL].GetStdDevError()
  mean_holder['meanNN'+HL]       = h_holder['hNN'+HL].GetMean()
  meanErr_holder['meanErrNN'+HL] = h_holder['hNN'+HL].GetMeanError()

  std_holder['stdNN'+str(NVars)+"_"+HL]         = h_holder['hNN'+str(NVars)
+"_"+HL].GetStdDev()     # 2 HL
  stdErr_holder['stdErrNN'+str(NVars)+"_"+HL]   = h_holder['hNN'+str(NVars)
+"_"+HL].GetStdDevError()
  mean_holder['meanNN'+str(NVars)+"_"+HL]       = h_holder['hNN'+str(NVars)
+"_"+HL].GetMean()
  meanErr_holder['meanErrNN'+str(NVars)+"_"+HL] = h_holder['hNN'+str(NVars)
+"_"+HL].GetMeanError()

#==Do the truncated Fits==#

#For the Corrected Histograme
x1Cor  = meanCor - 2*stdCor
x2Cor  = meanCor + 2*stdCor
FitFunctionCor = TF1( 'Cor', 'gaus', x1Cor, x2Cor)
hCor.Fit(FitFunctionCor, 'R+')

#For the Raw Histograme
x1Raw  = meanRaw - 2*stdRaw
x2Raw  = meanRaw + 2*stdRaw
FitFunctionRaw = TF1( 'Raw', 'gaus', x1Raw, x2Raw)
hRaw.Fit(FitFunctionRaw, 'R+')

#For the NN Histogrames

x1NN_holder        = {}
x2NN_holder        = {}
FitFunction_holder = {}

for HL in N:
  x1NN_holder['x1NN'+HL]  = mean_holder['meanNN'+HL] - std_holder['stdNN'+HL]
  x2NN_holder['x2NN'+HL]  = mean_holder['meanNN'+HL] + std_holder['stdNN'+HL]
  FitFunction_holder['FitFunctionNN'+HL] = TF1('NN'+HL,'gaus', x1NN_holder['x1NN'+HL],
x2NN_holder['x2NN'+HL])
  h_holder['hNN'+HL].Fit(FitFunction_holder['FitFunctionNN'+HL], 'R+')
```

```
  x1NN_holder['x1NN'+str(NVars)+"_"+HL]  = mean_holder['meanNN'+str(NVars)+"_"+HL] -
std_holder['stdNN'+str(NVars)+"_"+HL]
  x2NN_holder['x2NN'+str(NVars)+"_"+HL]  = mean_holder['meanNN'+str(NVars)+"_"+HL]
+ std_holder['stdNN'+str(NVars)+"_"+HL]
  FitFunction_holder['FitFunctionNN'+str(NVars)+"_"+HL] = TF1('NN'+str(NVars)
+"_"+HL,'gaus', x1NN_holder['x1NN'+str(NVars)+"_"+HL], x2NN_holder['x2NN'+str(NVars)
+"_"+HL])
  h_holder['hNN'+str(NVars)+"_"+HL].Fit(FitFunction_holder['FitFunctionNN'+str(NVars)
+"_"+HL], 'R+')

#Save the Mean and Sigma values from the Fits
mCor_fit    = FitFunctionCor.GetParameter(1)
mCorErr_fit = FitFunctionCor.GetParError(1)
sCor_fit    = FitFunctionCor.GetParameter(2)
sCorErr_fit = FitFunctionCor.GetParError(2)

mRaw_fit    = FitFunctionRaw.GetParameter(1)
mRawErr_fit = FitFunctionRaw.GetParError(1)
sRaw_fit    = FitFunctionRaw.GetParameter(2)
sRawErr_fit = FitFunctionRaw.GetParError(2)

mNNfit_holder    = {}
mNNErrfit_holder = {}
sNNfit_holder    = {}
sNNErrfit_holder = {}

for HL in N:
  mNNfit_holder['mNN'+HL]       = FitFunction_holder['FitFunctionNN'+HL].GetParameter(1)
# 1 HL
  mNNErrfit_holder['mNNErr'+HL] =
FitFunction_holder['FitFunctionNN'+HL].GetParError(1)
  sNNfit_holder['sNN'+HL]       = FitFunction_holder['FitFunctionNN'+HL].GetParameter(2)
  sNNErrfit_holder['sNNErr'+HL] = FitFunction_holder['FitFunctionNN'+HL].GetParError(2)

  mNNfit_holder['mNN'+str(NVars)+"_"+HL]       =
FitFunction_holder['FitFunctionNN'+str(NVars)+"_"+HL].GetParameter(1) # 2 HL
  mNNErrfit_holder['mNNErr'+str(NVars)+"_"+HL] =
FitFunction_holder['FitFunctionNN'+str(NVars)+"_"+HL].GetParError(1)
  sNNfit_holder['sNN'+str(NVars)+"_"+HL]       =
FitFunction_holder['FitFunctionNN'+str(NVars)+"_"+HL].GetParameter(2)
  sNNErrfit_holder['sNNErr'+str(NVars)+"_"+HL] =
FitFunction_holder['FitFunctionNN'+str(NVars)+"_"+HL].GetParError(2)

#===Do the Results Plot===#

#Create an Output file to store the Histos
outf = TFile('histos_NNBPResults_Test.root', 'recreate')  #
'histos_NNBPResults_Train.root' for Train Test

#Create the mean, sigma and std Histos => Raw,Cor + 15 NN's with 1 HidLayr + 15 NN
with 2 HL = 32 bins
```

```
h_mean_fit   = TH1F("h_mean_fit"   ,"h_mean_fit"   ,32,0,32)
h_mean_histo = TH1F("h_mean_histo" ,"h_mean_histo" ,32,0,32)
h_sigma      = TH1F("h_sigma"      ,"h_sigma"      ,32,0,32)
h_std        = TH1F("h_std"        ,"h_std"        ,32,0,32)

h_RawMean   = TH1F("h_RawMean"  ,"h_RawMean"  ,1,0,32)
h_RawSigma  = TH1F("h_RawSigma" ,"h_RawSigma" ,1,0,32)
h_RawStd    = TH1F("h_RawStd"   ,"h_RawStd"   ,1,0,32)
h_CorMean   = TH1F("h_CorMean"  ,"h_CorMean"  ,1,0,32)
h_CorSigma  = TH1F("h_CorSigma" ,"h_CorSigma" ,1,0,32)
h_CorStd    = TH1F("h_CorStd"   ,"h_CorStd"   ,1,0,32)

#Fill the Histos
h_mean_fit.SetBinContent(1,mRaw_fit)
h_mean_fit.SetBinError(1,mRawErr_fit)
h_mean_fit.SetBinContent(2,mCor_fit)
h_mean_fit.SetBinError(2,mCorErr_fit)

#Here for the 30 NN's
i = 3
for HL in N:
  h_mean_fit.SetBinContent(i,mNNfit_holder['mNN'+HL])                    # 1 HL => i=3,..,17
  h_mean_fit.SetBinError(i,mNNErrfit_holder['mNNErr'+HL])
  h_mean_fit.SetBinContent(i+15,mNNfit_holder['mNN'+str(NVars)+"_"+HL])      # 2 HL =>
i=18,..,32
  h_mean_fit.SetBinError(i+15,mNNErrfit_holder['mNNErr'+str(NVars)+"_"+HL])
  i = i + 1

#Fill the h_mean_histo => New for Task12
h_mean_histo.SetBinContent(1,meanRaw)
h_mean_histo.SetBinError(1,meanErrRaw)
h_mean_histo.SetBinContent(2,meanCor)
h_mean_histo.SetBinError(2,meanErrCor)

#Here for the 30 NN's
i = 3
for HL in N:
  h_mean_histo.SetBinContent(i,mean_holder['meanNN'+HL])                    # 1 HL =>
i=3,..,17
  h_mean_histo.SetBinError(i,meanErr_holder['meanErrNN'+HL])
  h_mean_histo.SetBinContent(i+15,mean_holder['meanNN'+str(NVars)+"_"+HL])   # 2 HL
=> i=18,..,32
  h_mean_histo.SetBinError(i+15,meanErr_holder['meanErrNN'+str(NVars)+"_"+HL])
  i = i + 1

h_sigma.SetBinContent(1,sRaw_fit)
h_sigma.SetBinError(1,sRawErr_fit)
h_sigma.SetBinContent(2,sCor_fit)
h_sigma.SetBinError(2,sCorErr_fit)
```

```
#Here for the 30 NN's
i = 3
for HL in N:
  h_sigma.SetBinContent(i,sNNfit_holder['sNN'+HL])
  h_sigma.SetBinError(i,sNNErrfit_holder['sNNErr'+HL])
  h_sigma.SetBinContent(i+15,sNNfit_holder['sNN'+str(NVars)+"_"+HL])
  h_sigma.SetBinError(i+15,sNNErrfit_holder['sNNErr'+str(NVars)+"_"+HL])
  i = i + 1

h_std.SetBinContent(1,stdRaw)
h_std.SetBinError(1,stdErrRaw)
h_std.SetBinContent(2,stdCor)
h_std.SetBinError(2,stdErrCor)

#Here for the 30 NN's
i = 3
for HL in N:
  h_std.SetBinContent(i,std_holder['stdNN'+HL])
  h_std.SetBinError(i,stdErr_holder['stdErrNN'+HL])
  h_std.SetBinContent(i+15,std_holder['stdNN'+str(NVars)+"_"+HL])
  h_std.SetBinError(i+15,stdErr_holder['stdErrNN'+str(NVars)+"_"+HL])
  i = i + 1

#Here for the Cor and Raw Histos => Task10
h_RawMean.SetBinContent(1,mRaw_fit)
h_RawSigma.SetBinContent(1,sRaw_fit)
h_RawStd.SetBinContent(1,stdRaw)
h_CorMean.SetBinContent(1,mCor_fit)
h_CorSigma.SetBinContent(1,sCor_fit)
h_CorStd.SetBinContent(1,stdCor)

#Name the Bins of the Histos
h_mean_fit.GetXaxis().SetBinLabel(1,"Raw")
h_mean_fit.GetXaxis().SetBinLabel(2,"Cor")

#Name the 30 NN's
i = 3
for HL in N:
  h_mean_fit.GetXaxis().SetBinLabel(i   ,'NN'+HL)
  h_mean_fit.GetXaxis().SetBinLabel(i+15,'NN'+str(NVars)+"_"+HL)
  i = i + 1

#Name the h_mean_histo => New for Task12
h_mean_histo.GetXaxis().SetBinLabel(1,"Raw")
h_mean_histo.GetXaxis().SetBinLabel(2,"Cor")

#Name the 30 NN's
i = 3
for HL in N:
  h_mean_histo.GetXaxis().SetBinLabel(i   ,'NN'+HL)
  h_mean_histo.GetXaxis().SetBinLabel(i+15,'NN'+str(NVars)+"_"+HL)
```

```
  i = i + 1

h_sigma.GetXaxis().SetBinLabel(1,"Raw")
h_sigma.GetXaxis().SetBinLabel(2,"Cor")

#Name the 30 BDT's
i = 3
for HL in N:
  h_sigma.GetXaxis().SetBinLabel(i   ,'NN'+HL)
  h_sigma.GetXaxis().SetBinLabel(i+15,'NN'+str(NVars)+"_"+HL)
  i = i + 1

h_std.GetXaxis().SetBinLabel(1,"Raw")
h_std.GetXaxis().SetBinLabel(2,"Cor")

#Name the 30 NN's
i = 3
for HL in N:
  h_std.GetXaxis().SetBinLabel(i   ,'NN'+HL)
  h_std.GetXaxis().SetBinLabel(i+15,'NN'+str(NVars)+"_"+HL)
  i = i + 1

#Writes the histograms in my file
h_mean_fit.Write()
h_mean_histo.Write()
h_sigma.Write()
h_std.Write()
h_RawMean.Write()
h_RawSigma.Write()
h_RawStd.Write()
h_CorMean.Write()
h_CorSigma.Write()
h_CorStd.Write()

#Plus Write the Fitted Histos
for HL in N:
  h_holder['hNN'+HL].Write()
  h_holder['hNN'+str(NVars)+"_"+HL].Write()

#Close the files
outf.Close()
```

Παράρτημα 9

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου κάνοντας χρήση του TMVA φτιάξαμε το αρχείο με όνομα TMVA_NN_Tanh_3HL.root μέσα στο οποίο υπάρχουν 2 TTrees. Ένα TestTree και ένα TrainTree με το κάθε ένα να έχει ως Branches τις μεταβλητές εκπαίδευσης όπως και τον στόχο εκπαίδευσης (στην περίπτωσή μας ως προς το Pt) στα 9 Branches με ονόματα MLP_X_Y_Z , όπου X, Y, Z ο αριθμός των Νευρώνων σε κάθε κρυφό στρώμα του Νετρωνικού Δικτύου.

```
from ROOT import TMVA, TFile, TTree, TCut
from subprocess import call
from os.path import isfile

# Setup TMVA
TMVA.Tools.Instance()

#Create an Output file and re-write it if it exists already
output = TFile.Open('TMVA_NN_Tanh_3HL.root', 'RECREATE')
factory = TMVA.Factory('TMVARegression', output,
        '!V:!Silent:Color:DrawProgressBar:Transformations=D,G:AnalysisType=Regression')

# Load data
#Get the Data from the TFile and TTree we made in Task 5
data = TFile.Open('file.root')
tree = data.Get('my_tree')
dataloader = TMVA.DataLoader('dataset')

dataloader.AddVariable('bjetRawPt')
dataloader.AddVariable('bjetPt')
dataloader.AddVariable('bjetEta')
dataloader.AddVariable('bjetPhi')
dataloader.AddVariable('bjetMass')
dataloader.AddVariable('bjetChf')
dataloader.AddVariable('bjetNhf')
dataloader.AddVariable('bjetPhf')
dataloader.AddVariable('bjetMuf')
dataloader.AddVariable('bjetElf')
dataloader.AddVariable('bgenJetEta')

#Target ekpedeusis ta GenJet
dataloader.AddTarget('bgenJetPt')
dataloader.AddRegressionTree(tree, 1.0)
dataloader.PrepareTrainingAndTestTree(TCut(''),
        'nTrain_Regression=70000:SplitMode=Random:NormMode=NumEvents:!V')

#nTrain_Regression IMPORTANT : 70000 : peripou 70% of the entries we have (111074)
```

```
#======Book methods======#
#Scan over NN Arcitectures
# 3 Hidden Layers
factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_5_2",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,5,2:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_5_3",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,5,3:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_5_4",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,5,4:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_6_2",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,6,2:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_6_3",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,6,3:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_6_4",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,6,4:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_7_2",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,7,2:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_7_3",
    '!H:!
V:NCycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,7,3:TrainingMetho
d=BFGS')

factory.BookMethod(dataloader, TMVA.Types.kMLP, "MLP10_7_4",
    '!H:!
V:Ncycles=1500:NeuronType=tanh:VarTransform=N:HiddenLayers=10,7,4:TrainingMethod
=BFGS')
# Run TMVA
factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```

Παράρτημα 10

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου υπολογίστηκε το Response για τους υπόχωρους του Pt και του Eta όπως και του πως φτιάχτηκαν τα τελικά διαγράμματα με τα αποτελέσματα (Results)

```
import ROOT
from ROOT import TFile,TCanvas,TH1F,gStyle,gPad

gStyle.SetOptStat(0)

#Opens a file only to be read
newfile = TFile.Open("TMVA_BDT_Best.root", "read")  # BDT => Or LD, NN ,Raw ,Cor

#Calls the script Tree that was made before
newtree = newfile.Get("dataset/TestTree")

#Opens a file and re-write it
outf=TFile("histos_ResponseBDT_Divide_Test.root","recreate")

#Create the histogram

genPt  = ['2050','50100','100More']
genEta = ['013' ,'1324']
name_holder = {}

for Pt in genPt:
  for Eta in genEta:
    name_holder['ResponseBDT_Pt'+Pt+'_Eta'+Eta] =
TH1F('ResponseBDT_Pt'+Pt+'_Eta'+Eta,'ResponseBDT_Pt'+Pt+'_Eta'+Eta,300,0.0,3.0)

#Read all entries and fill histograms
for i in range(newtree.GetEntries()):
  newtree.GetEntry(i)
  if newtree.bgenJetEta > -1.3 and newtree.bgenJetEta < 1.3:
    if newtree.bgenJetPt > 20  and newtree.bgenJetPt  < 50:
      name_holder['ResponseBDT_Pt2050_Eta013'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)
    if newtree.bgenJetPt > 50  and newtree.bgenJetPt  < 100:
      name_holder['ResponseBDT_Pt50100_Eta013'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)
    if newtree.bgenJetPt > 100:
      name_holder['ResponseBDT_Pt100More_Eta013'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)
  if (newtree.bgenJetEta > -2.4 and newtree.bgenJetEta < -1.3) or (newtree.bgenJetEta >
1.3 and newtree.bgenJetEta < 2.4):
    if newtree.bgenJetPt > 20  and newtree.bgenJetPt  < 50:
```

```
      name_holder['ResponseBDT_Pt2050_Eta1324'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)
    if newtree.bgenJetPt > 50  and newtree.bgenJetPt  < 100:
      name_holder['ResponseBDT_Pt50100_Eta1324'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)
    if newtree.bgenJetPt > 100:

name_holder['ResponseBDT_Pt100More_Eta1324'].Fill(newtree.BDTShr01NT200MD4/
newtree.bgenJetPt)

#Change file
outf.cd()

#Writes the histograms in my file
for Pt in genPt:
  for Eta in genEta:
    name_holder['ResponseBDT_Pt'+Pt+'_Eta'+Eta].Write()

#Close the files
outf.Close()
newfile.Close()

from ROOT import TH1F, TF1, TFile
from ROOT import gROOT

#Opens a file only to be read

BDTfile = TFile.Open("histos_ResponseBDT_Divide_Test.root","read")

genPt    = ['2050','50100','100More']
genEta   = ['013' ,'1324']
h_holder = {}

for Pt in genPt:
  for Eta in genEta:
    h_holder['hBDT_Pt'+Pt+'_Eta'+Eta] = BDTfile.Get('ResponseBDT_Pt'+Pt+'_Eta'+Eta)

std_holder     = {}
stdErr_holder  = {}
mean_holder    = {}
meanErr_holder = {}

for Pt in genPt:
  for Eta in genEta:
    std_holder['stdBDT_Pt'+Pt+'_Eta'+Eta]         =
h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].GetStdDev()
    stdErr_holder['stdErrBDT_Pt'+Pt+'_Eta'+Eta]   =
h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].GetStdDevError()
    mean_holder['meanBDT_Pt'+Pt+'_Eta'+Eta]       =
h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].GetMean()
```

```
    meanErr_holder['meanErrBDT_Pt'+Pt+'_Eta'+Eta] =
h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].GetMeanError()

#=Do the truncated Fits=#

#For the BDT Histograme                    #== +/- 1 std's ==#

x1BDT_holder       = {}
x2BDT_holder       = {}
FitFunction_holder = {}

for Pt in genPt:
  for Eta in genEta:
    x1BDT_holder['x1BDT_Pt'+Pt+'_Eta'+Eta]  =
mean_holder['meanBDT_Pt'+Pt+'_Eta'+Eta] - std_holder['stdBDT_Pt'+Pt+'_Eta'+Eta]
    x2BDT_holder['x2BDT_Pt'+Pt+'_Eta'+Eta]  =
mean_holder['meanBDT_Pt'+Pt+'_Eta'+Eta] + std_holder['stdBDT_Pt'+Pt+'_Eta'+Eta]
    FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+Eta] =
TF1('BDT_Pt'+Pt+'_Eta'+Eta,'gaus', x1BDT_holder['x1BDT_Pt'+Pt+'_Eta'+Eta],
x2BDT_holder['x2BDT_Pt'+Pt+'_Eta'+Eta])

h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].Fit(FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+
Eta], 'R+')

mBDTfit_holder    = {}
mBDTErrfit_holder = {}
sBDTfit_holder    = {}
sBDTErrfit_holder = {}

for Pt in genPt:
  for Eta in genEta:
    mBDTfit_holder['mBDT_Pt'+Pt+'_Eta'+Eta]       =
FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+Eta].GetParameter(1)
    mBDTErrfit_holder['mErrBDT_Pt'+Pt+'_Eta'+Eta] =
FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+Eta].GetParError(1)
    sBDTfit_holder['sBDT_Pt'+Pt+'_Eta'+Eta]       =
FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+Eta].GetParameter(2)
    sBDTErrfit_holder['sErrBDT_Pt'+Pt+'_Eta'+Eta] =
FitFunction_holder['FitFunctionBDT_Pt'+Pt+'_Eta'+Eta].GetParError(2)

#===Do the Results Plot===#

#Create an Output file to store the Histos
outf = TFile('histos_ResultsBDT_Divide_Test.root', 'recreate')

#Creat the mean, sigma and std Histos => 6Divided for 1BDT = 6
h_mean  = TH1F("h_mean" ,"h_mean" ,6,0,6)
h_sigma = TH1F("h_sigma","h_sigma",6,0,6)
h_std   = TH1F("h_std"  ,"h_std"  ,6,0,6)

i = 1
```

```
for Eta in genEta:
  for Pt in genPt:
    h_mean.SetBinContent(i,mBDTfit_holder['mBDT_Pt'+Pt+'_Eta'+Eta])
    h_mean.SetBinError(i,mBDTErrfit_holder['mErrBDT_Pt'+Pt+'_Eta'+Eta])
    i = i + 1

i = 1
for Eta in genEta:
  for Pt in genPt:
    h_sigma.SetBinContent(i,sBDTfit_holder['sBDT_Pt'+Pt+'_Eta'+Eta])
    h_sigma.SetBinError(i,sBDTErrfit_holder['sErrBDT_Pt'+Pt+'_Eta'+Eta])
    i = i + 1

i = 1
for Eta in genEta:
  for Pt in genPt:
    h_std.SetBinContent(i,std_holder['stdBDT_Pt'+Pt+'_Eta'+Eta])
    h_std.SetBinError(i,stdErr_holder['stdErrBDT_Pt'+Pt+'_Eta'+Eta])
    i = i + 1

i = 1
for Eta in genEta:
  for Pt in genPt:
    h_mean.GetXaxis().SetBinLabel(i,'BDT_Pt'+Pt+'_Eta'+Eta)
    i = i + 1

#Name the Divided BDT Histos
i = 1
for Eta in genEta:
  for Pt in genPt:
    h_sigma.GetXaxis().SetBinLabel(i,'BDT_Pt'+Pt+'_Eta'+Eta)
    i = i + 1

i = 1
for Eta in genEta:
  for Pt in genPt:
    h_std.GetXaxis().SetBinLabel(i,'BDT_Pt'+Pt+'_Eta'+Eta)
    i = i + 1

#Writes the histograms in my file

h_mean.Write()
h_sigma.Write()
h_std.Write()

for Eta in genEta:
  for Pt in genPt:
    h_holder['hBDT_Pt'+Pt+'_Eta'+Eta].Write()

#Close the files
outf.Close()
```

Παράρτημα 11

Στο παράρτημα αυτό δίνεται ο κώδικας (σε pyROOT) βάση του οποίου φτιάχτηκαν τα τελικά διαγράμματα των αποτελεσμάων (Results) διαλέγοντας το καλύτερο BDT, το καλύτερο NN, τον LD και τα Raw-Corrected.

```
from ROOT import TH1F, TF1, TFile
from ROOT import gROOT

#Open files only to be read
RawCorfile = TFile.Open("Response_Raw_Cor.root","read")
NNfile     = TFile.Open("histos_NNTanhoutput_Test.root","read")
BDTfile    = TFile.Open("histos_BDToutput_Test.root","read")
LDfile     = TFile.Open("histos_ResponseLD_Test.root","read")

#Read the Histogrames from the TFiles
hCor       = RawCorfile.Get("Response_Cor")
hRaw       = RawCorfile.Get("Response_Raw")
hNN        = NNfile.Get("ResponseNN10_5")
hBDT       = BDTfile.Get("ResponseBDTShr0.1NT200MD4")
hLD        = LDfile.Get("ResponseLD")

#Read the std, stdErr, mean, meanErr from the Histos
stdCor     = hCor.GetStdDev()
stdErrCor  = hCor.GetStdDevError()
meanCor    = hCor.GetMean()
meanErrCor = hCor.GetMeanError()

stdRaw     = hRaw.GetStdDev()
stdErrRaw  = hRaw.GetStdDevError()
meanRaw    = hRaw.GetMean()
meanErrRaw = hRaw.GetMeanError()
```

```
stdNN      = hNN.GetStdDev()
stdErrNN   = hNN.GetStdDevError()
meanNN     = hNN.GetMean()
meanErrNN  = hNN.GetMeanError()

stdBDT     = hBDT.GetStdDev()
stdErrBDT  = hBDT.GetStdDevError()
meanBDT    = hBDT.GetMean()
meanErrBDT = hBDT.GetMeanError()

stdLD      = hLD.GetStdDev()
stdErrLD   = hLD.GetStdDevError()
meanLD     = hLD.GetMean()
meanErrLD  = hLD.GetMeanError()

#=Do the truncated Fits=#
#For the Corrected Histograme
x1Cor  = meanCor - stdCor
x2Cor  = meanCor + stdCor
FitFunctionCor = TF1( 'Cor', 'gaus', x1Cor, x2Cor)
hCor.Fit(FitFunctionCor, 'R+')

#For the Raw Histograme
x1Raw  = meanRaw - stdRaw
x2Raw  = meanRaw + stdRaw
FitFunctionRaw = TF1( 'Raw', 'gaus', x1Raw, x2Raw)
hRaw.Fit(FitFunctionRaw, 'R+')

#For the NN Histograme
x1NN  = meanNN - stdNN
x2NN  = meanNN + stdNN
FitFunctionNN = TF1( 'NN', 'gaus', x1NN, x2NN)
hNN.Fit(FitFunctionNN, 'R+')
```

```
#For the BDT Histograme
x1BDT  = meanBDT - stdBDT
x2BDT  = meanBDT + stdBDT
FitFunctionBDT = TF1( 'BDT', 'gaus', x1BDT, x2BDT)
hBDT.Fit(FitFunctionBDT, 'R+')

#For the LD Histograme
x1LD  = meanLD - stdLD
x2LD  = meanLD + stdLD
FitFunctionLD = TF1( 'LD', 'gaus', x1LD, x2LD)
hLD.Fit(FitFunctionLD, 'R+')

#Save the Mean and Sigma values from the Fits

mCor_fit    = FitFunctionCor.GetParameter(1)
mCorErr_fit = FitFunctionCor.GetParError(1)
sCor_fit    = FitFunctionCor.GetParameter(2)
sCorErr_fit = FitFunctionCor.GetParError(2)
mRaw_fit    = FitFunctionRaw.GetParameter(1)
mRawErr_fit = FitFunctionRaw.GetParError(1)
sRaw_fit    = FitFunctionRaw.GetParameter(2)
sRawErr_fit = FitFunctionRaw.GetParError(2)
mNN_fit    = FitFunctionNN.GetParameter(1)
mNNErr_fit = FitFunctionNN.GetParError(1)
sNN_fit    = FitFunctionNN.GetParameter(2)
sNNErr_fit = FitFunctionNN.GetParError(2)
mBDT_fit    = FitFunctionBDT.GetParameter(1)
mBDTErr_fit = FitFunctionBDT.GetParError(1)
sBDT_fit    = FitFunctionBDT.GetParameter(2)
sBDTErr_fit = FitFunctionBDT.GetParError(2)
mLD_fit    = FitFunctionLD.GetParameter(1)
mLDErr_fit = FitFunctionLD.GetParError(1)
sLD_fit    = FitFunctionLD.GetParameter(2)
sLDErr_fit = FitFunctionLD.GetParError(2)
```

```
#===Do the Results Plot===#

#Create the Output file to store the Final Result Histos
outf = TFile('histos_FinalResults.root', 'recreate')

#Creat the mean, sigma and std Histos => Raw+Cor+NN+BDT+LD = 5
h_mean  = TH1F("h_mean" ,"h_mean" ,5,0,5)
h_sigma = TH1F("h_sigma","h_sigma",5,0,5)
h_std   = TH1F("h_std"  ,"h_std"  ,5,0,5)

#Fill the Histos
h_mean.SetBinContent(1,mRaw_fit)
h_mean.SetBinError(1,mRawErr_fit)
h_mean.SetBinContent(2,mCor_fit)
h_mean.SetBinError(2,mCorErr_fit)
h_mean.SetBinContent(3,mLD_fit)
h_mean.SetBinError(3,mLDErr_fit)
h_mean.SetBinContent(4,mNN_fit)
h_mean.SetBinError(4,mNNErr_fit)
h_mean.SetBinContent(5,mBDT_fit)
h_mean.SetBinError(5,mBDTErr_fit)
h_sigma.SetBinContent(1,sRaw_fit)
h_sigma.SetBinError(1,sRawErr_fit)
h_sigma.SetBinContent(2,sCor_fit)
h_sigma.SetBinError(2,sCorErr_fit)
h_sigma.SetBinContent(3,sLD_fit)
h_sigma.SetBinError(3,sLDErr_fit)
h_sigma.SetBinContent(4,sNN_fit)
h_sigma.SetBinError(4,sNNErr_fit)
h_sigma.SetBinContent(5,sBDT_fit)
h_sigma.SetBinError(5,sBDTErr_fit)
h_std.SetBinContent(1,stdRaw)
h_std.SetBinError(1,stdErrRaw)
h_std.SetBinContent(2,stdCor)
```

```
h_std.SetBinError(2,stdErrCor)
h_std.SetBinContent(3,stdLD)
h_std.SetBinError(3,stdErrLD)
h_std.SetBinContent(4,stdNN)
h_std.SetBinError(4,stdErrNN)
h_std.SetBinContent(5,stdBDT)
h_std.SetBinError(5,stdErrBDT)

#Name the Bins of the Histos
h_mean.GetXaxis().SetBinLabel(1,"Raw")
h_mean.GetXaxis().SetBinLabel(2,"Cor")
h_mean.GetXaxis().SetBinLabel(3,"LD")
h_mean.GetXaxis().SetBinLabel(4,"NN")
h_mean.GetXaxis().SetBinLabel(5,"BDT")

h_sigma.GetXaxis().SetBinLabel(1,"Raw")
h_sigma.GetXaxis().SetBinLabel(2,"Cor")
h_sigma.GetXaxis().SetBinLabel(3,"LD")
h_sigma.GetXaxis().SetBinLabel(4,"NN")
h_sigma.GetXaxis().SetBinLabel(5,"BDT")

h_std.GetXaxis().SetBinLabel(1,"Raw")
h_std.GetXaxis().SetBinLabel(2,"Cor")
h_std.GetXaxis().SetBinLabel(3,"LD")
h_std.GetXaxis().SetBinLabel(4,"NN")
h_std.GetXaxis().SetBinLabel(5,"BDT")

#Writes the histograms in my file
h_mean.Write()
h_sigma.Write()
h_std.Write()
#Close the file
outf.Close()
```

ΒΙΒΛΙΟΓΡΑΦΕΙΑ

[1] S. Theodoridis, K. Koutroumbas “**Αναγνώριση Προτύπων**”

[2] Christopher M. Bishop “**Neural Networks fot Pattern Recognition**”

[3] Christopher M. Bishop “**Pattern Recognition and Machine Learning**”

[4] Tom M. Mitchell “**Machine Learning**”

[5] B. R. Martin, G. Shaw “**Φυσική των Στοιχειωδών Σωματιδίων**”

[6] Κ. Βαγιονάκης “**Σωματιδιακή Φυσική**”

[7] Ariel Schwartzman “**Jet Energy Calibration at the LHC**”

[8] Κ. Κουσουρής Σημειώσεις του Μαθήματος “**Αναγνώρηση Προτύπων και Νευρωνικά Δίκτυα**”

[9] Γ. Τσιπολίτης Σημειώσεις του Μαθήματος “**Τεχνολογία Ανιχνευτικών και Επιταχυντικών Διατάξεων**”

[10] Γ. Τσιπολίτης Σημειώσεις του Μαθήματος “**Στοιχειώδη Σωματίδια Ι**”

[11] **TMVA 4** (Toolkit for Multivariate Data Analysis with ROOT) **User’s Guide**

[12] **ROOT User’s Guide: 6 Release Cycle**

[13] https://home.**cern**/science/**experiments**

[14] https://**cms.cern/detector**

[15] Patric Henry Windston “**Artificial Inteligence**”