ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Κατηγοριοποίηση και ανάλυση φόρτου εργασίας εφαρμογών εγκατεστημένων σε περιβάλλοντα container με χρήση τεχνητής νοημοσύνης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΕΥΑΓΓΕΛΟΥ ΑΠΟΣΤΟΛΑΚΗ

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Νοέμβριος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Κατηγοριοποίηση και ανάλυση φόρτου εργασίας εφαρμογών εγκατεστημένων σε περιβάλλοντα container με χρήση τεχνητής νοημοσύνης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΕΥΑΓΓΕΛΟΥ ΑΠΟΣΤΟΛΑΚΗ

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Νοεμβρίου 2023.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Θ. Βαρβαρίγου Καθηγήτρια Ε.Μ.Π.	Ε. Βαρβαρίγος Καθηγητής Ε.Μ.Π.	Σ. Παπαβασιλείου Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2023

(Υπογραφή)

.....

ΕΥΑΓΓΕΛΟΣ ΑΠΟΣΤΟΛΑΚΗΣ

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

Περίληψη

Η ανάπτυξη εφαρμογών σε περιβάλλοντα Cloud, Fog και Edge Computing, αποτελεί πλέον το επικρατέστερο μοντέλο ανάπτυξης εφαρμογών. Σε αυτά τα ετερογενή περιβάλλοντα, υπάρχει μια πληθώρα από λύσεις που αφορούν την τοποθέτηση και την λειτουργία των εφαρμογών. Οι πάροχοι υπηρεσιών Infrastructure as a Service (IaaS) προσφέρουν πολλές λύσεις όσον αφορά την ενοικίαση πόρων. Η βελτιστοποίηση της κατανομής και χρήσης των πόρων σε αυτά τα κατανεμημένα περιβάλλοντα είναι καίριας σημασίας, για την ελαχιστοποίηση του κόστους αλλά και την κάλυψη των απαιτήσεων της εκάστοτε εφαρμογής. Επομένως, είναι απαραίτητο να υπάρχει ένα σύστημα, που να μπορεί να βοηθήσει στη λήψη πιο ενημερωμένων αποφάσεων, βασισμένων στις ανάγκες μιας συγκεκριμένης εφαρμογής.

Στην παρούσα διπλωματική εργασία, παρουσιάζουμε μια μέθοδο για βελτιστοποίηση της διαδικασίας δημιουργίας προφίλ εφαρμογών, και την κατηγοριοποίηση φόρτου εργασιών των εφαρμογών. Προτείνουμε τη βελτιστοποίηση πολυδιάστατων διανυσμάτων που αναπαριστούν τα προφίλ απαιτήσεων πόρων των εφαρμογών, τα οποία προφίλ είναι ανεξάρτητα του υποκείμενου υλικού. Υλοποιούμε ένα αλγόριθμο για την επιλογή των χαρακτηριστικών που μπορούν να περιγράψουν καλύτερα την χρήση πόρων μιας εφαρμογής, και να βοηθήσουν στην διάκριση μεταξύ διαφορετικών εφαρμογών, με βάση τη φύση τους και τις υπολογιστικές τους ανάγκες. Επιπλέον, αναπτύσσουμε ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) για την κατηγοριοποίηση των εφαρμογών σε γνωστά benchmark. Το ΤΝΔ χρησιμοποιείται επίσης για να βοηθήσει στη διαδικασία επιλογής χαρακτηριστικών, και κατά την αξιολόγηση των προτεινόμενων λύσεων.

Το μοντέλο ταξινόμησης αναπτύσσεται χρησιμοποιώντας τη βιβλιοθήκη Keras με το υπόβαθρο του TensorFlow, και η αρχιτεκτονική και η λειτουργία του εξηγούνται. Η τεχνική επιλογής χαρακτηριστικών αξιοποιεί έξι Filter Μεθόδους Επιλογής Χαρακτηριστικών για να κατατάξει τα χαρακτηριστικά, και να επιλέξει τα καλύτερα από αυτά, βάσει της αξίας τους για την διαδικασία της ταξινόμησης. Στη συνέχεια αναζητά τα υποσύνολα χαρακτηριστικών που μεγιστοποιούν την απόδοση του ταξινομητή. Τέλος, παρουσιάζεται μια σειρά πειραμάτων, με βάση τα οποία συγκρίνουμε διαφορετικά υποσύνολα χαρακτηριστικών, και αξιολογούμε την προτεινόμενη μεθοδολογία.

Λέξεις Κλειδιά: Τεχνητά Νευρωνικά Δίκτυα, Ταξινόμηση, Keras, TensorFlow, Επιλογή Χαρακτηριστικών

Abstract

Application deployment in the cloud-edge continuum has become the prevailing computing paradigm. In these heterogeneous environments there is a plethora of solutions regarding the implementation and operation of applications. A lot of hardware solutions are provided by Infrastructure as a Service (IaaS) providers. The demand for optimized resource usage and allocation is critical in these distributed architectures, in order to minimize the cost of operation and meet Quality of Service (QoS) requirements for each application. Thus, it is essential to have a system that can help make a more informed decision based on the needs of a specific application.

In this thesis we present a method for optimized application profiling and application workload categorization. We propose the optimization of multidimensional vectors that represent the hardware-agnostic profiles of applications' hardware requirements. We implement an algorithm for selecting the features that can best describe an application's resource usage and can help in the distinction between different applications in terms of nature and computational needs. Furthermore, we develop an Artificial Neural Network (ANN) to classify the applications to known benchmarks. The ANN is also used to help the feature selection process and during the evaluation of the suggested solutions.

The classification model is developed using the Keras library with TensorFlow backend and its architecture and functioning are explained. The feature selection technique uses six Filter Feature Selection methods to rank features, and select the best out of them, based on their importance for the classification procedure. Then it searches for the feature subsets that maximize the classifier's performance. Finally, a set of experiments are presented, on the basis of which, we compare different feature subsets, and evaluate the proposed methodology.

Keywords: Artificial Neural Networks, Classification, Keras, TensorFlow, Feature Selection

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω την καθηγήτρια κα Θεοδώρα Βαρβαρίγου για την δυνατότητα που μου έδωσε να ασχοληθώ με το σύγχρονο και ενδιαφέρον αντικείμενο της παρούσας διπλωματικής εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω τον διδάκτορα Αλέξανδρο Ψύχα για την βοήθεια, την παρακολούθηση, την υποστήριξη και τον χρόνο που αφιέρωσε κατά την εκπόνηση της εργασίας αυτής.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς, τα αδέρφια μου και τους ανθρώπους που ήταν κοντά μου, για την απεριόριστη στήριξη που μου παρείχαν σε καλές και δύσκολες στιγμές κατά την διάρκεια των σπουδών μου.

Ευάγγελος Αποστολάκης,

Αθήνα, 9η Νοεμβρίου 2023

Περιεχόμενα

Περίληψη 5
Abstract 6
Ευχαριστίες 7
Περιεχόμενα 9
Ευρετήριο Σχημάτων 11
Ευρετήριο πινάκων 12
Ευρετήριο Γραφημάτων 13
1 Εισαγωγή 14
1.1 Κίνητρο και σκοπός της διπλωματικής εργασίας 14
1.2 Διάρθρωση της διπλωματικής εργασίας 17
2 Επισκόπηση ερευνητικού τομέα 18
2.1 Δημιουργία προφίλ εφαρμογών και κατηγοριοποίηση φόρτου εργασιών με μηχανική μάθηση 18
2.2 Επιλογή Χαρακτηριστικών 21
3 Περιγραφή Συστήματος 24
3.1 Αρχική Προσέγγιση Συστήματος 24
3.2 Παρούσα προσέγγιση 31
3.2.1 Βελτιστοποίηση προφίλ με Επιλογή Χαρακτηριστικών 32
3.2.2 Τεχνητά Νευρωνικά Δίκτυα 35
4 Υλοποίηση Συστήματος 49
4.1 Τεχνολογίες υλοποίησης 49
4.2 Επιλογή Χαρακτηριστικών 50
4.3 Υλοποίηση ΤΝΔ 52
5 Πειραματική Αξιολόγηση 57
5.1 Μορφοποίηση συνόλου δεδομένων 57
5.2 Επιλογή Χαρακτηριστικών 71
5.3 Αξιολόγηση των επικρατέστερων υποσυνόλων 80

6 Συμπεράσματα και μελλοντικές κατευθύνσεις 98
Βιβλιογραφία 101

Ευρετήριο Σχημάτων

Σχήμα 1: Σχέση απόδοσης ταξινομητή - πλήθους χαρακτηριστικών [5] 16
Σχήμα 2: Μέθοδοι Dimensionality Reduction και Feature Selection 33
Σχήμα 3: Το perceptron [35] 37
Σχήμα 4: Το πολυεπίπεδο perceptron [36] 38
Σχήμα 5: Ο Αλγόριθμος Οπισθοδιάδοσης σε ένα ΤΝΔ 40
Σχήμα 6: Η συνάρτηση ReLU 41
Σχήμα 7: Η συνάρτηση Leaky ReLU 42
Σχήμα 8: Οπτικοποίηση της υπερεκπαίδευσης 44
Σχήμα 9: Η συνάρτηση κόστους στη διάρκεια της εκπαίδευσης 45
Σχήμα 10: Οπτικοποίηση του Dropout 46
Σχήμα 11: Scikit-learn 49
Σχήμα 12: TensorFlow 50
Σχήμα 13: Keras 50
Σχήμα 14: Αρχιτεκτονική ΤΝΔ 54
Σχήμα 15: Synthesized data creation with SMOTE 67

Ευρετήριο πινάκων

Πίνακας 1: Hardware Configuration A 26
Πίνακας 2: Hardware Configuration B..... 26
Πίνακας 3: Hardware Configuration C..... 26
Πίνακας 4: Hardware Configuration D 27
Πίνακας 5: Hardware Configuration E..... 27
Πίνακας 6: Training and Testing datasets 28
Πίνακας 7: Χαρακτηριστικά Προφίλ..... 30
Πίνακας 8: Σύνοψη ΤΝΔ 56
Πίνακας 9: Baseline Model Accuracy 58
Πίνακας 10: Baseline Model Classification Report..... 60
Πίνακας 11: HD Model Accuracy..... 62
Πίνακας 12: HD Model Classification Report 65
Πίνακας 13: HOD Model Accuracy 68
Πίνακας 14: HOD Model Classification Report..... 69
Πίνακας 15: Τα 10 αρχικά χαρακτηριστικά ανά πόρο 77
Πίνακας 16: Best Feature Subsets..... 81
Πίνακας 17: Subset_0 Classification Report..... 82
Πίνακας 18: Subset_1 Classification Report..... 85
Πίνακας 19: Subset_2 Classification Report..... 87
Πίνακας 20: Subset_3 Classification Report..... 90
Πίνακας 21: Subset_4 Classification Report..... 92
Πίνακας 22: Subset_5 Classification Report..... 95
Πίνακας 23: Συνολική απόδοση των επικρατέστερων υποσυνόλων 97

Ευρετήριο Γραφημάτων

Γράφημα 1: RSD Χαρακτηριστικών Προφίλ [6] 31
Γράφημα 2: Baseline Model Confusion Matrix 61
Γράφημα 3: HD Model Confusion Matrix 63
Γράφημα 4: Κατανομή δειγμάτων - κλάσεων στο ομογενοποιημένο dataset 66
Γράφημα 5: HOD Model Confusion Matrix 70
Γράφημα 6: Information Gain / Mutual Information Feature Importance 72
Γράφημα 7: Fisher's Score Feature Importance 73
Γράφημα 8: MAD Feature Importance 74
Γράφημα 9: Distance Correlation Feature Importance 75
Γράφημα 10: Permutation Feature Importance 76
Γράφημα 11: Kendall Correlation Coefficient 78
Γράφημα 12: Kendall Correlation Coefficient > 90% 79
Γράφημα 13: Subset_0 Confusion Matrix 83
Γράφημα 14: Subset_1 Confusion Matrix 86
Γράφημα 15: Subset_2 Confusion Matrix 88
Γράφημα 16: Subset_3 Confusion Matrix 91
Γράφημα 17: Subset_4 Confusion Matrix 93
Γράφημα 18: Subset_5 Confusion Matrix 96

1 Εισαγωγή

1.1 Κίνητρο και σκοπός της διπλωματικής εργασίας

Με τις τεχνολογίες Cloud, Fog Computing και Edge Computing, υπάρχει ένα επιπλέον επίπεδο πολυπλοκότητας, στην διαδικασία επιλογής της πιο κατάλληλης διαμόρφωσης υλικού για την εξυπηρέτηση των αναγκών της κάθε εφαρμογής [1]. Σε αυτά τα περιβάλλοντα τόσο οι υλικές υποδομές όσο και οι εφαρμογές είναι κατανεμημένες και παρουσιάζουν μεγάλη ετερογένεια. Ταυτόχρονα στα συστήματα αυτά, είναι πρωταρχικής σημασίας η αποδοτική αξιοποίηση των πόρων, ειδικά στις βαθμίδες fog και edge όπου αυτοί είναι περιορισμένοι [2]. Ο διαχειριστής ενός συστήματος καλείται να επιλέξει τις κατάλληλες υλικές υποδομές προς ενοικίαση από τους παρόχους [3], με γνώμονα τόσο τις επιδόσεις, όσο και το κόστος. Για να μπορέσει αυτή η επιλογή να γίνει με βέλτιστο τρόπο, χρειάζεται να μπορούμε να εξάγουμε πληροφορία για τις απαιτήσεις της εκάστοτε εφαρμογής σε πόρους, πριν την εκκίνηση της λειτουργίας της.

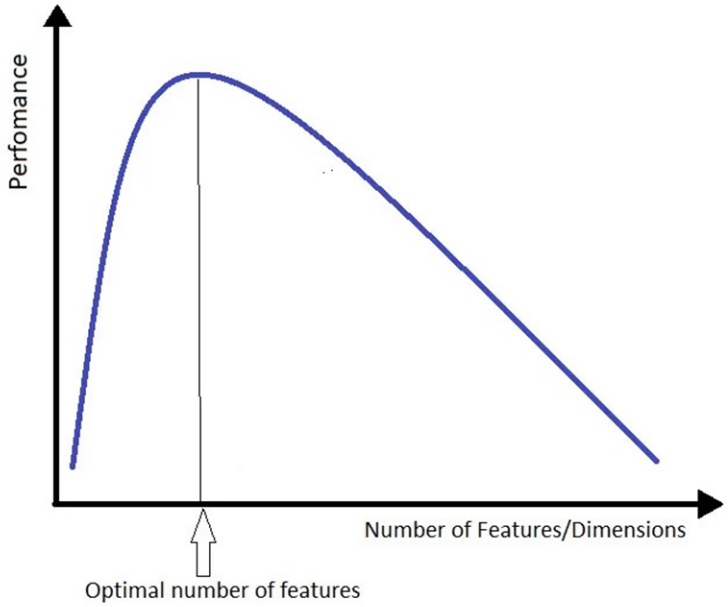
Η πιο διαδεδομένη μέθοδος χαρακτηρισμού φόρτου εργασιών εφαρμογών και ανάλυσης του αποτυπώματός τους στο υποκείμενο υλικό, είναι με χρήση benchmark. Τα benchmark είναι συνδυασμοί υπολογιστικών προγραμμάτων και διεργασιών που προσομοιώνουν συχνές υπολογιστικές διαδικασίες. Η συνήθης χρήση τους γίνεται για την περιγραφή και την σύγκριση επιδόσεων τόσο υλικού όσο και λογισμικού σε διαφορετικά υπολογιστικά συστήματα και περιβάλλοντα. Συνεπώς, η εκτέλεση benchmark που προσομοιώνουν λειτουργίες πραγματικών εφαρμογών, μπορεί να αποτελέσει σημείο αναφοράς για τον χαρακτηρισμό των ίδιων των εφαρμογών. Ο αριθμός των benchmark που υπάρχουν και είναι διαθέσιμα στους χρήστες είναι μεγάλος, και περιλαμβάνει benchmark που μπορούν να καλύψουν σχεδόν κάθε τύπο εφαρμογής. Παρόλα αυτά, δεν υπάρχει καμία επίσημη μέθοδος, για να μπορεί ο ιδιοκτήτης μιας εφαρμογής να γνωρίζει ποιο benchmark προσομοιώνει καλύτερα το

αποτύπωμα της εφαρμογής του στο υποκείμενο υλικό, ώστε να την χρησιμοποιήσει στην διαδικασία επιλογής υλικού.

Εξάγοντας προφίλ χρήσης πόρων από την εκτέλεση διαφορετικών benchmark και εφαρμογών, μπορεί να διεξαχθεί μια ταξινόμηση των εφαρμογών σε γνωστά benchmark. Η διαδικασία εξαγωγής προφίλ των εφαρμογών στοχεύει στην περιγραφή, την καταγραφή και την αξιολόγηση των απαιτήσεων της κάθε εφαρμογής σε υπολογιστικούς πόρους. Το προφίλ του κάθε benchmark και της κάθε εφαρμογής, είναι επί της ουσίας ένα πολυδιάστατο διάνυσμα, με κάθε χαρακτηριστικό (feature) αυτού του διανύσματος, να αντιπροσωπεύει μια μετρική χρήσης του υλικού. Τα χαρακτηριστικά των προφίλ αυτών, οφείλουν να ακολουθούν ορισμένους κανόνες, ώστε να έχουν κάποια αξία στην διαδικασία της ταξινόμησης. Το κάθε χαρακτηριστικό πρέπει να παρουσιάζει σταθερότητα της τιμής του για συγκεκριμένο benchmark ή εφαρμογή, αλλά και διαφοροποίηση της τιμής για εκτελέσεις διαφορετικών benchmark και εφαρμογών. Επίσης, τα χαρακτηριστικά των προφίλ θα πρέπει να είναι ανεξάρτητα της διαμόρφωσης του υλικού πάνω στο οποίο έγινε η εκάστοτε εκτέλεση.

Η εγκαθίδρυση των τεχνολογιών Cloud, Fog και Edge Computing, συνοδεύτηκε κι από την υιοθέτηση των containers, που αποτελούν μια τεχνική εικονοποίησης. Τα containers ενθυλακώνουν μια εφαρμογή μαζί με όλες τις εξαρτήσεις συστήματος, όπως βιβλιοθήκες, binaries, αρχεία παραμετροποίησης και άλλα εργαλεία, σε ένα μόνο εκτελέσιμο αρχείο. Τα containers, είναι μεταφέρσιμα μεταξύ διαφορετικών υπολογιστικών συστημάτων και μοιράζονται το λειτουργικό σύστημα του μηχανήματος που τα φιλοξενεί. Ταυτόχρονα όμως, απομονώνουν το λογισμικό που περιέχουν, εξασφαλίζοντας ότι η εκτέλεση του γίνεται πάντοτε με τον ίδιο τρόπο, ανεξαρτήτως του περιβάλλοντος στο οποίο εκτελείται το container. Επομένως, τα containers μας δίνουν τη δυνατότητα να μεταφέρουμε το ίδιο περιβάλλον εκτέλεσης σε διαφορετικά υπολογιστικά συστήματα, και να εκτελέσουμε τις ίδιες εφαρμογές με τρόπο τέτοιο ώστε η επίδοσή τους να εξαρτάται μόνο από τα υποκείμενα τεχνικά χαρακτηριστικά, είτε αυτά είναι πραγματικά, είτε εικονικά.

Οι εφαρμογές που εκτελούνται στα περιβάλλοντα που μελετάμε παρουσιάζουν μεγάλη ετερογένεια, επομένως έχουν και πολύ διαφορετικό αντίκτυπο στο υλικό. Το σύνολο των μετρικών που μπορούμε να χρησιμοποιήσουμε στην δημιουργία των προφίλ είναι μεγάλο, όμως δεν μπορούν να συνεισφέρουν όλα εξίσου στην διαδικασία της ταξινόμησης. Επιπλέον, όσο αυξάνονται τα χαρακτηριστικά που χρειάζονται για την περιγραφή του προβλήματος, το πλήθος των δεδομένων που απαιτούνται για αποδοτική ταξινόμηση αυξάνεται εκθετικά [4]. Επομένως, εάν το πλήθος των χαρακτηριστικών ξεπεράσει ένα συγκεκριμένο νούμερο, η απόδοση του ταξινομητή θα αρχίσει να μειώνεται. Για την αντιμετώπιση του προβλήματος της υψηλής διαστασιμότητας, χρησιμοποιούνται τεχνικές μείωσης διαστάσεων. Οι τεχνικές αυτές συχνά βασίζονται στην Επιλογή Χαρακτηριστικών (Feature Selection), όπου στόχος είναι η επιλογή ενός βέλτιστου υποσυνόλου των αρχικών χαρακτηριστικών.



Σχήμα 1: Σχέση απόδοσης ταξινομητή - πλήθους χαρακτηριστικών [5]

Το αντικείμενο της παρούσας διπλωματικής εργασίας, είναι η βελτιστοποίηση των προφίλ που εξάγονται από την εκτέλεση benchmark και εφαρμογών σε περιβάλλοντα container, μέσω επιλογής χαρακτηριστικών (Feature Selection). Παρουσιάζουμε μια μεθοδολογία επιλογής χαρακτηριστικών κατά την οποία χρησιμοποιούμε τόσο τις γνώσεις μας επάνω στον τομέα όσο και γνωστές μεθόδους επιλογής χαρακτηριστικών, για να βρούμε το υποσύνολο χαρακτηριστικών για το οποίο ο ταξινομητής μας πετυχαίνει τις καλύτερες επιδόσεις. Για την ταξινόμηση αναπτύσσουμε και εκπαιδεύουμε ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) χρησιμοποιώντας τη βιβλιοθήκη Keras με το υπόβαθρο του TensorFlow. Το ΤΝΔ επίσης χρησιμοποιείται στην διαδικασία επιλογής χαρακτηριστικών για την σύγκριση μεταξύ διάφορων υποσυνόλων χαρακτηριστικών, αλλά και στην τελική αξιολόγηση της μεθοδολογίας μας. Τα δεδομένα που χρησιμοποιούμε έχουν αντληθεί από την εκτέλεση γνωστών benchmark από τις πλατφόρμες sysbench και phoronix, τα οποία έχουν επιλεχθεί για να καλύπτουν ένα ευρύ φάσμα διαφορετικών τύπων εφαρμογών. Κάθε benchmark από αυτά που χρησιμοποιήθηκαν αποτελεί και μια από τις κλάσεις ταξινόμησης.

1.2 Διάρθρωση της διπλωματικής εργασίας

Η συνέχεια της διπλωματικής εργασίας, διαρθρώνεται ως εξής:

- Στο Κεφάλαιο 2 παρουσιάζονται σχετικές έρευνες και διαφορετικές προσεγγίσεις για την δημιουργία προφίλ εφαρμογών, την ταξινόμηση φόρτου εργασιών και την αντιμετώπιση της υψηλής διαστασιμότητας μέσω επιλογής χαρακτηριστικών.
- Στο Κεφάλαιο 3 εξετάζονται αφενός η αρχική προσέγγιση του συστήματος και τα δεδομένα πάνω στα οποία εργαστήκαμε, και αφετέρου η παρούσα προσέγγιση με στόχο την βελτιστοποίηση της διαδικασίας εξαγωγής προφίλ.
- Στο Κεφάλαιο 4 αναλύεται ο προτεινόμενος αλγόριθμος επιλογής χαρακτηριστικών, και εξηγούνται η αρχιτεκτονική και η λειτουργία του ΤΝΔ.
- Στο Κεφάλαιο 5 παρατίθενται η διαδικασία και τα αποτελέσματα της πειραματικής αξιολόγησης της υλοποίησής μας.
- Στο Κεφάλαιο 6 γίνεται η σύνοψη της εργασίας και εκθέτουμε παρατηρήσεις και σκέψεις σχετικά με μελλοντική πιθανή επέκταση πάνω στο αντικείμενο αυτής.

2 Επισκόπηση ερευνητικού τομέα

2.1 Δημιουργία προφίλ εφαρμογών και κατηγοριοποίηση φόρτου εργασιών με μηχανική μάθηση

Έχουν γίνει διάφορες προσπάθειες και έχουν υπάρξει διαφορετικές προσεγγίσεις σε ό,τι αφορά στο πρόβλημα του χαρακτηρισμού της συμπεριφοράς μιας εφαρμογής και ως εκ τούτου της πρόβλεψης και βελτιστοποίησης των επιδόσεών της. Μία από αυτές [6] αποτέλεσε και το εφαλτήριο της παρούσας διπλωματικής εργασίας. Στο [6] παρουσιάζεται ένα σύστημα εξαγωγής προφίλ χρήσης πόρων containerized εφαρμογών τα οποία είναι ανεξάρτητα από το υποκείμενο υλικό. Με χρήση των προφίλ που εξήχθησαν από εκτέλεση benchmark σε περιβάλλον container, και χρήση του αλγόριθμου μηχανικής μάθησης Random Forest, διεξάγεται κατηγοριοποίηση των εφαρμογών με βάση την φύση τους και τις υπολογιστικές τους ανάγκες. Το σύστημα αυτό μπορεί να βοηθήσει ώστε να γίνει η βέλτιστη επιλογή υλικών υποδομών προς ενοικίαση, και η βέλτιστη διαχείρισή τους, από τους παρόχους υπηρεσιών IaaS. Την συγκεκριμένη προσέγγιση την εξετάζουμε πιο αναλυτικά στο Κεφάλαιο 3.

Μια παρόμοια περίπτωση με το [6] είναι και το [7], το οποίο παρουσιάζει μια offline μεθοδολογία μηχανικής μάθησης για να προβλέψει την απόδοση HPC εφαρμογών, δηλαδή εφαρμογών με μεγάλες απαιτήσεις σε υπολογιστική ισχύ, οι οποίες εκτελούνται στο cloud. Συλλέγονται μετρικές ανεξάρτητες από τις υλικές υποδομές, με την χρήση ενός plugin του LLVM, οι οποίες αξιοποιούνται στην δημιουργία προφίλ εφαρμογών. Για να επιτευχθεί το παραπάνω, ο κώδικας των εφαρμογών πρέπει να ενορχηστρωθεί κατά τη μεταγλώττιση και συνεπώς είναι προαπαιτούμενο ο πηγαίος κώδικας να είναι διαθέσιμος.

Μία προσπάθεια δημιουργίας ενός συστήματος βελτιστοποίησης απόδοσης μέσω κατηγοριοποίησης των εισερχόμενων εργασιών έχει γίνει στο πεδίο των MapReduce εφαρμογών [8]. Η κατηγοριοποίηση γίνεται σε δύο φάσεις. Στην πρώτη φάση, αξιοποιούνται τεχνικές μηχανικής μάθησης για τον ορισμό κατηγοριών

εφαρμογών MapReduce για τις οποίες έχει προσδιοριστεί η βέλτιστη παραμετροποίηση συστήματος. Συγκεκριμένα χρησιμοποιείται μια παραλλαγή του αλγορίθμου k-means++, ενώ η είσοδος που δίνεται είναι ένα σύνολο εργασιών που είναι κοινώς αποδεκτά ως benchmarks. Στην δεύτερη φάση, δίνεται ως είσοδος μια άγνωστη διεργασία για να ταξινομηθεί σε μία από τις κατηγορίες που ορίστηκαν στην πρώτη φάση, ώστε να εφαρμοστεί η βέλτιστη παραμετροποίηση που έχει προσδιοριστεί για την κατηγορία αυτή. Τα διανύσματα που χαρακτηρίζουν τις εκτελέσεις προκύπτουν από την καταγραφή του χρόνου υπολογισμού που αφιερώνεται στον επεξεργαστή, τη μνήμη και το δίσκο. Τέλος, η εφαρμογή όλης της διαδικασίας σε ένα Amazon EC2 testbed φανέρωσε τα επίπεδα βελτιστοποίησης που προέκυψαν σε σχέση με την προεπιλεγμένη παραμετροποίηση. Παρόμοια είναι και η προσέγγιση του [9], όπου επιχειρήθηκε η δημιουργία προφίλ για τα ίδια τα benchmarks ώστε να αποκτηθεί ενδότερη κατανόηση της λειτουργίας και της φύσης τους. Χρησιμοποιούνται κάποια από τα πιο γνωστά benchmarks του Phoronix Test Suite και το εργαλείο SWAT prototype. Τα προφίλ εξάγονται από τα execution traces των benchmarks, τα οποία συλλέγονται με το εργαλείο LLTng και αναλύονται σε low-level μετρικές όπως CPU utilization, parallelization, stability, memory usage.

Σε κάποιες άλλες περιπτώσεις βλέπουμε να χρησιμοποιούνται και πιο σύνθετες μέθοδοι μηχανικής μάθησης. Μία από αυτές τις προσπάθειες [10], χρησιμοποιεί ένα νευρωνικό δίκτυο και μια παραλλαγή του αλγορίθμου black hole, που αξιοποιεί την πληροφορία σφάλματος στις προηγούμενες προβλέψεις, ώστε να βελτιώσει την ακρίβεια των επόμενων προβλέψεων. Τα δεδομένα που χρησιμοποιούνται είναι data traces από servers που φιλοξενούν διαφορετικού είδους υπηρεσίες. Στο [11] οι συγγραφείς χρησιμοποιούν transfer learning για να προβλέψουν την απόδοση μιας εφαρμογής στο cloud. Τα προφίλ των εφαρμογών βασίζονται σε εκτελέσεις τους σε διαφορετικά cloud instances, αλλά και σε πληροφορία που έχει συγκεντρωθεί από εκτελέσεις σε offline περιβάλλον.

Ένα εργαλείο που να μπορεί να προβλέψει τις επερχόμενες απαιτήσεις σε πόρους σε περιβάλλον Cloud Computing παρουσιάζεται στο [12]. Σκοπός του είναι να βοηθήσει στην δυναμική κλιμάκωση παρεχόμενων πόρων σε αυτό το περιβάλλον. Αρχικά γίνεται ανάλυση της χρήσης πόρων και έπειτα χρησιμοποιούνται στατιστικά μοντέλα για την διαδικασία της πρόβλεψης. Τα μοντέλα χρησιμοποιούν αλγόριθμους μηχανικής μάθησης. Συγκεκριμένα χρησιμοποιούνται δύο αλγόριθμοι: Error Correction Neural Network (ECNN) και Linear Regression, ενώ παράλληλα αξιοποιούνται οι τεχνικές sliding window και cross-validation στα στάδια της εκπαίδευσης και της πρόβλεψης. Τα δεδομένα που χρησιμοποιούνται για την εκπαίδευση και την εκτίμηση των μοντέλων έχουν παραχθεί από την εκτέλεση του TCP-W benchmark στο Amazon EC2 cloud. Για την τελική αξιολόγηση των μοντέλων χρησιμοποιούνται οι στατιστικές μετρικές Mean Absolute Percentage Error (MAPE), PRED(25), Root Mean Squared Error (RMSE) και R^2 Prediction Accuracy.

Μία παρόμοια προσέγγιση με το [12] βλέπουμε και στο [13], όπου χρησιμοποιείται πάλι ένας συνδυασμός στατιστικών μεθόδων και μεθόδων μηχανικής

μάθησης με στόχο την πρόβλεψη των μελλοντικών απαιτήσεων σε πόρους για βελτίωση της αυτόματης κλιμάκωσης πόρων στο Cloud. Οι συγγραφείς δημιουργούν ένα μοντέλο που χαρακτηρίζουν Advanced Model for Efficient Workload Prediction in the Cloud (AME-WPC) το οποίο διαφοροποιείται από παρεμφερείς προσπάθειες διότι χρησιμοποιεί πληροφορίες συστήματος για να βελτιώσει την ακρίβεια των προβλέψεών του. Παρουσιάζεται μια μέθοδος Two-phase Pattern Matching (TPM), όπου το πρώτο στάδιο αναγνωρίζει μοτίβα στα μεγέθη των φόρτων εργασιών, ενώ στο δεύτερο εντοπίζει μοτίβα στην διακύμανση αυτών. Τελικά το πρόβλημα προσεγγίζεται τόσο με παλινδρόμηση (regression) όσο και με ταξινόμηση (classification), με χρήση του αλγορίθμου μηχανικής μάθησης Random Forest και στις δύο περιπτώσεις. Επίσης, τα μοντέλα εκπαιδεύονται και αξιολογούνται αρχικά με το ακατέργαστο σύνολο δεδομένων, αλλά στη συνέχεια και με ένα επαυξημένο σύνολο που περιέχει χαρακτηριστικά που προέκυψαν με feature extraction. Τελικά, εξετάζεται η λειτουργία του πάνω σε δεδομένα από το Grid Workloads Archive, και στη διαδικασία αξιολόγησης αξιοποιείται η μέθοδος k-NN.

Ένα από τα βασικά κίνητρα για βελτιστοποίηση της αξιοποίησης των πόρων, είναι η μείωση του κόστους λειτουργίας. Στο [14], επιχειρείται η μείωση του κόστους μέσω αυτοματοποίησης της δημιουργίας προφίλ εφαρμογών που μπορούν να βοηθήσουν στην βέλτιστη επιλογή cloud instance, την σύγκριση ανάμεσα σε διαφορετικές εφαρμογές και την βελτίωση των αλγορίθμων δρομολόγησης. Τα προφίλ αυτά είναι ουσιαστικά συνοπτικές περιγραφές της απόδοσης της εφαρμογής και των απαιτήσεών της σε επεξεργαστική ισχύ, μνήμη, δίσκο και δίκτυο, σε διαφορετικά περιβάλλοντα και σενάρια. Συλλέγονται λεπτομερείς στατιστικές από τις εκτελέσεις σε διάφορα cloud instances και χρησιμοποιούνται σε συνδυασμό με το αντίστοιχο κόστος για την επιλογή του βέλτιστου περιβάλλοντος εκτέλεσης.

Σε μία άλλη προσπάθεια παρουσιάζεται το DocLite [15]. Ένα containerized εργαλείο που προσπαθεί να αξιολογήσει εικονικές μηχανές (Virtual Machines / VMs), σε σχεδόν πραγματικό χρόνο. Αποδεικνύει, πως με χρήση lightweight benchmarks σε συνδυασμό με μικρά container παράγονται αποτελέσματα παρόμοια με αυτά που παράγονται από μεγάλα container. Το DocLite υλοποιεί δύο μεθόδους benchmarking. Στην πρώτη χρησιμοποιεί containers για να κάνει benchmarking σε ένα μικρό κομμάτι του VM. Η δεύτερη μέθοδος χρησιμοποιεί ιστορικό από εκτελέσεις benchmark σε συνδυασμό με την πρώτη μέθοδο. Η αξιολόγηση των μεθόδων έδειξε πως είναι 91 φορές πιο γρήγορες από το να κάναμε benchmarking σε όλο το VM. Ισχυρές είναι επίσης οι ενδείξεις ότι οι μετρήσεις μέσω lightweight benchmarks, μέχρι και micro-benchmarks αποτελούν έναν καθόλα έγκριτο τρόπο εξαγωγής συμπερασμάτων για την απόδοση των διάφορων cloud configurations [16].

Το RUBiS [17], είναι μια profiling-as-a-service Web εφαρμογή, η οποία αντιμετωπίζει το πρόβλημα διαχείρισης εφαρμογών στο cloud, με ταυτόχρονο profiling και scaling. Ο χρήστης παρέχει κάποιους περιορισμούς όσων αφορά τις επιδόσεις και το κόστος, οι οποίοι χρησιμοποιούνται για να ρυθμίσουν το scaling. Την ίδια στιγμή, παράγονται γενικευμένα αλλά ακριβή προφίλ για όσο περισσότερους από τους

κόμβους που αποτελούν την εφαρμογή γίνεται. Στο [18], το autoscaling επιτυγχάνεται μέσω offline δημιουργίας προφίλ benchmark. Απλουστευμένα μοντέλα του φόρτου εργασιών της εφαρμογής, προσομοιώνουν την συμπεριφορά ενός τυπικού χρήστη, και χρησιμοποιούνται ως είσοδος στην εφαρμογή, με σκοπό να αντλήσουν πληροφορίες για την απόδοσή της. Έτσι ο διαχειριστής μπορεί να διαμορφώσει την στρατηγική του για το scaling, λαμβάνοντας υπόψιν το κόστος και τις απαιτήσεις της εφαρμογής.

Συμπεριλαμβάνοντας τον παράγοντα background workload, σε διαμοιραζόμενα VMs, οι συγγραφείς στο [19] επιχειρούν, μέσω μίας canonical correlation ανάλυσης, να εντοπίσουν τους παράγοντες του συστήματος που επηρεάζουν την απόδοση περισσότερο. Δεδομένα χρήσης πόρων εφαρμογών, που έχουν συλλεχθεί μέσω online profiling, επιτρέπουν την πρόβλεψη των αναγκών υλικού για ένα συγκεκριμένο επίπεδο υπηρεσιών της εφαρμογής. Μία παρόμοια προσπάθεια συναντάμε στο [20], όπου χρησιμοποιούνται οι Hardware Performer Counters για να εξαχθούν τα προφίλ των εφαρμογών.

2.2 Επιλογή Χαρακτηριστικών

Κατά την ταξινόμηση με χρήση τεχνητής νοημοσύνης συχνά χρησιμοποιούνται μέθοδοι επιλογής χαρακτηριστικών, για την μείωση της διαστασιμότητας και την βελτίωση απόδοσης της ταξινόμησης. Τέτοιου είδους τεχνικές έχουν χρησιμοποιηθεί εκτεταμένα και με επιτυχία σε πολλούς τομείς. Ορισμένοι από αυτούς είναι η ανίχνευση εισβολής (intrusion detection) [21] [22] [23] [24], η κατηγοριοποίηση κειμένου (text categorization) [25] [26], η επεξεργασία εικόνας (image processing) [27] και η ιατρική διάγνωση (medical diagnosis) [28] [29]. Στα πλαίσια αυτού του υποκεφαλαίου όμως επικεντρώνουμε στην χρήση τους στον τομέα της διαχείρισης πόρων και φόρτου εργασιών.

Να βοηθήσει στην καλύτερη δυναμική διαχείριση πόρων στα κέντρα δεδομένων (datacenters) επιχειρεί το [30], ώστε να μπορούν να ανταποκρίνονται στις απαιτήσεις απόδοσης των εφαρμογών (QoS) με έναν οικονομικά αποδοτικό τρόπο. Τα σύνολα δεδομένων που χρησιμοποιούνται είναι δύο workload traces, τα Google Cluster Trace (GCT) και Bit Brains Trace (BBT). Πρώτα, προτείνεται ένα μοντέλο ταξινόμησης εφαρμογών και διεργασιών με βάση τις ανάγκες τους σε υπολογιστικούς πόρους. Έπειτα εξετάζονται επτά αλγόριθμοι μηχανικής μάθησης (SVM, k-NN, Stochastic Gradient Descent, Logistic Regression, Decision Tree, Random Forest, MLP) και αξιολογούνται για την ακρίβεια των προβλέψεων τους στη διαδικασία της ταξινόμησης. Τέλος, προτείνεται ένας αλγόριθμος επιλογής χαρακτηριστικών, που δημιουργεί μια κατάταξη των χαρακτηριστικών του εκάστοτε συνόλου δεδομένων ως προς τη σημαντικότητα τους στην ακρίβεια της ταξινόμησης. Ο αλγόριθμος αυτός

δημιουργεί ενός Decision Tree για κάθε χαρακτηριστικό και μετρά σε κάθε διαίρεση του δέντρου το impurity πριν και μετά τη διαίρεση, όπου το impurity υπολογίζεται με βάση το Mean Squared Error (MSE).

Μία λίγο διαφορετική οπτική χρησιμοποιούν οι συγγραφείς στο [31], που έχουν σκοπό τη μείωση της κατανάλωσης ενέργειας και του κόστους στα datacenters μέσω διαφορετικής διαχείρισης του εκάστοτε είδους εφαρμογής. Για να επιτευχθεί αυτό προτείνουν την ταξινόμηση εφαρμογών με βάση τον τρόπο που χρησιμοποιεί το υποκείμενο υλικό, προσεγγίζοντας το ζήτημα από τη σκοπιά της επιλογής χαρακτηριστικών. Συγκεκριμένα, ο βασικός ισχυρισμός είναι πως μόνο λίγες από τις μετρικές/παραμέτρους περιγραφής της χρήσης των πόρων είναι σημαντικές για την ταξινόμηση των εφαρμογών. Προτείνουν δηλαδή ένα μοντέλο επιλογής χαρακτηριστικών, μέσω θεωρητικής ανάλυσης της αξίας των μετρικών στην διαδικασία ταξινόμησης. Η ανάλυσή τους καταλήγει στην επιλογή εννέα παραγόντων ως χαρακτηριστικά του μοντέλου, το οποίο έχει την ικανότητα ταξινόμησης των εφαρμογών σε τρεις βασικές κατηγορίες: CPU intensive, I/O intensive, Network intensive.

Την ταξινόμηση εφαρμογών σε ορισμένες στοιχειώδεις κατηγορίες βλέπουμε και στα [32] [33] από τους Jian Zhang και R.J. Figueiredo. Για να γίνει ευκολότερη η αποδοτική διαχείριση πόρων στο Cloud , προτείνεται η ταξινόμηση εφαρμογών στις κατηγορίες CPU intensive, I/O and paging intensive, Network intensive και Idle. Οι δύο αυτές προσπάθειες παρουσιάζουν αρκετές ομοιότητες με τη δική μας. Αμφότερες επιχειρούν την ταξινόμηση εφαρμογών με βάση την χρήση του υποκείμενου υλικού, χρησιμοποιώντας και κάποια μέθοδο επιλογής χαρακτηριστικών. Και στις δύο περιπτώσεις, η έμπνευση προκύπτει από την υπηρεσία VMPlant, η οποία βασίζεται στην τεχνολογία των Virtual Machines (VMs). Το VMPlant, παρέχει αυτοματοποιημένη δημιουργία και ευέλικτη διαμόρφωση VMs, με δυνατότητα να είναι ειδικά προσαρμοσμένα στις απαιτήσεις της εκάστοτε εφαρμογής και να μπορούν να κλωνοποιούνται και να εκτελούνται σε ενιαίο περιβάλλον εκτέλεσης πάνω σε κατανεμημένες αρχιτεκτονικές. Τα VMs αποτελούν μια μορφή εικονοποίησης που τεχνολογικά μπορούμε να θεωρήσουμε πρόγονο των containers και το VMPlant αντίστοιχα μπορεί να θεωρηθεί πρόγονος του docker.

Το [33] ουσιαστικά αποτελεί εξέλιξη της προσπάθειας που γίνεται στο [32]. Συγκεκριμένα, στο [32] τα δεδομένα αντλούνται από την εκτέλεση εφαρμογών σε VMs στο περιβάλλον του VMPlant. Έπειτα χρησιμοποιείται η μέθοδος μείωσης διαστάσεων Principal Component Analysis (PCA) και η ταξινόμηση επιτυγχάνεται με χρήση του αλγορίθμου k-NN. Στο τέλος αξιολογείται η ακρίβεια του ταξινομητή. Σε επέκταση αυτής της προσέγγισης, στο [33], προτείνεται ένα μοντέλο δυναμικής αυτόνομης επιλογής χαρακτηριστικών για μείωση διαστάσεων. Χρησιμοποιεί το πιθανοτικό μοντέλο Bayesian Network για να αυτοματοποιήσει την επιλογή των μετρικών που σχετίζονται περισσότερο με τις διαφορετικές κλάσεις εφαρμογών και μεγιστοποιούν την ακρίβεια της ταξινόμησης. Επίσης χρησιμοποιείται το Mahalanobis Distance, για επιλογή των δεδομένων εκπαίδευσης, ώστε να μπορεί το μοντέλο να διαχειριστεί και

δυναμικούς φόρτους εργασιών. Τέλος, χρησιμοποιείται το confusion matrix, για αξιολόγηση του μοντέλου επιλογής χαρακτηριστικών, συγκρίνοντας τις επιδόσεις του απέναντι σε εκείνες του μοντέλου που δημιουργήθηκε στο [32].

3 Περιγραφή Συστήματος

3.1 Αρχική Προσέγγιση Συστήματος

Η αρχική προσέγγιση του συστήματος πάνω στο οποίο δουλεύουμε παρουσιάζεται στο [6]. Πρόκειται για ένα σύστημα δημιουργίας προφίλ χρήσης πόρων containerized εφαρμογών και κατηγοριοποίησής τους σε γνωστά benchmark, με βάση τις υπολογιστικές τους ανάγκες. Το σύστημα αναπτύσσεται με το flow-based προγραμματιστικό εργαλείο Node RED, ενώ το περιβάλλον container που χρησιμοποιείται είναι το Docker. Οι δύο βασικές διαδικασίες του συστήματος είναι η διαδικασία δημιουργίας των προφίλ, και η διαδικασία εκπαίδευσης και αξιολόγησης ενός μοντέλου ταξινομητή.

Η διαδικασία δημιουργίας των προφίλ ξεκινά με την συλλογή των ακατέργαστων δεδομένων. Πρόκειται για μετρικές οι οποίες παρήχθησαν από το Docker και αντλήθηκαν με χρήση του Docker stats. Ορισμένες από τις μετρικές αυτές χρησιμοποιούνται ως έχουν, απευθείας στην δημιουργία προφίλ, ενώ άλλες χρησιμοποιούνται για την εξαγωγή άλλων πιο σύνθετων μετρικών, μέσω ορισμένων υπολογισμών. Τα προφίλ που δημιουργούνται, είναι επί της ουσίας πολυδιάστατα διανύσματα, όπου κάθε διάσταση αναπαριστά μία από τις μετρικές που έχουν επιλεχθεί. Οι διαστάσεις του διανύσματος ονομάζονται και χαρακτηριστικά (features) του προφίλ που δημιουργήθηκε. Τα προφίλ έχουν την μορφή:

$$\vec{V} = \left(V_1, V_2, V_3, \ldots, V_m \right)$$

όπου το V_m είναι το m-οστό χαρακτηριστικό του προφίλ. Κάθε χαρακτηριστικό οφείλει να έχει τα παρακάτω τρία γνωρίσματα για να έχει αξία για την διαδικασία της ταξινόμησης:

1. Σταθερότητα: Αν θέλουμε να έχει νόημα να αναφερόμαστε σε προβλέψεις, θα πρέπει οι μετρικές στις οποίες αυτές βασίζονται, να έχουν μία όσο γίνεται πιο σταθερή συμπεριφορά ανάμεσα στις διαφορετικές εκτελέσεις των ίδιων benchmarks και προγραμμάτων.
2. Διαφοροποίηση: Σε συνέχεια του προηγούμενου, θα πρέπει οι εκτελέσεις των διαφορετικών benchmarks και προγραμμάτων να μπορούν να διαφοροποιούνται. Θα προτιμήσουμε δηλαδή τις μετρικές εκείνες για τις οποίες παρατηρείται μία σταθερότητα στην τιμή για τις εκτελέσεις των ίδιων benchmarks, αλλά και σημαντική διαφορά για τις εκτελέσεις των διαφορετικών. Όσο πιο ευδιάκριτη είναι η διαφορά αυτή, τόσο πιο εύκολο θα είναι για τον αλγόριθμο να αποφασίσει ανάμεσα σε δύο διαφορετικά υποψήφια προφίλ πρόβλεψης.
3. Ανεξαρτησία από το υλικό: Ζητάμε να κρατήσουμε τις μετρικές εκείνες για τις οποίες τα προηγούμενα δύο κριτήρια ικανοποιούνται ακόμα και σε εκτελέσεις διαφορετικών μηχανημάτων. Με τον τρόπο αυτό θα μπορούμε να αποφανθούμε για την αντιστοίχιση του προφίλ της εφαρμογής χωρίς να νοιαζόμαστε για το που αυτή εκτελείται.

Για να δημιουργηθούν οι κατηγορίες στις οποίες ταξινομούνται οι εφαρμογές, δημιουργήθηκαν προφίλ από την εκτέλεση benchmarks σε containers. Τα benchmarks που χρησιμοποιήθηκαν ανήκουν σε δύο πολύ γνωστές οικογένειες benchmark, την Sysbench και την Phoronix. Τα επιλεγμένα benchmarks παρουσιάζουν μεγάλη ετερογένεια μεταξύ τους, ώστε να καλύπτουν όσο το δυνατόν μεγαλύτερο φάσμα εφαρμογών γίνεται. Πρόκειται για 62 benchmarks, εκ των οποίων περίπου 40 ανήκουν στην οικογένεια Phoronix, ενώ τα υπόλοιπα, προκύπτουν από 4 benchmarks της Sysbench, τα οποία επιδέχονται παραμετροποίηση ως προς τον φόρτο εργασίας.

Από τις εκτελέσεις των benchmarks δημιουργήθηκαν τα σύνολα δεδομένων που χρησιμοποιούνται τόσο για την εκπαίδευση όσο και για την αξιολόγηση του μοντέλου. Οι εκτελέσεις έγιναν σε πέντε διαφορετικά υπολογιστικά συστήματα, εκ των οποίων τα τρία διαφοροποιούνται από την υλική τους υποδομή, ενώ τα υπόλοιπα 2 ήταν εικονικές μηχανές. Για το σύνολο δεδομένων που χρησιμοποιείται στο στάδιο της εκπαίδευσης χρησιμοποιήθηκαν δύο φυσικά μηχανήματα και μία εικονική μηχανή που φιλοξενήθηκε στην ακαδημαϊκή υπηρεσία για cloud computing Okeanos. Για το σύνολο δεδομένων που χρησιμοποιείται στο στάδιο της αξιολόγησης χρησιμοποιήθηκαν ένα φυσικό μηχάνημα και μία εικονική μηχανή που φιλοξενήθηκε στο ίδιο μηχάνημα. Τα πέντε συστήματα παρουσιάζουν μεγάλη ετερογένεια στην διαμόρφωση των υλικών τους υποδομών, ώστε να εξερευνηθούν η δυνατότητα της προτεινόμενης μεθοδολογίας να κάνει προβλέψεις ανεξάρτητες του υποκείμενου υλικού.

Configuration A

CPU	Intel® CoreTM i7-3820 CPU @ 3.6GHz
RAM	32 GB (4x Nanya NT8GC64B8HB0N)
Disk	256 GB (ADATA SU800)
OS	Ubuntu LTS distribution (20.04)

Πίνακας 1: Hardware Configuration A

Configuration B

CPU	AMD FX-8370 Eight-Core Processor
RAM	16 GB (2x Mushkin 992125R)
Disk	256 GB (Samsung SSD 850)
OS	Ubuntu LTS distribution (20.04)

Πίνακας 2: Hardware Configuration B

Configuration C

CPU	2 cores with 1 thread from an Intel® Xeon® CPU E5-2650 v3 @ 2.30GHz
RAM	4 GB
OS	Debian 10.11

Πίνακας 3: Hardware Configuration C

Configuration D

CPU	Intel® CoreTM i5-3570 CPU @ 3.4GHz
RAM	16 GB (4x AMI CMZ8GX3M2A1600C9)
Disk	256 GB (Samsung SSD 850)
OS	Ubuntu LTS distribution (20.04)

Πίνακας 4: Hardware Configuration D

Configuration E

CPU	2 out of 4 physical cores
RAM	8 GB out of 16 GB RAM
Disk	100 GB out of 256 GB Disk
OS	Ubuntu LTS distribution (20.04)

Πίνακας 5: Hardware Configuration E

Έγιναν πολλαπλές εκτελέσεις για όλα τα benchmarks σε όλα τα μηχανήματα, μέσα σε Docker containers. Από τα περιβάλλοντα A, B και C σχηματίστηκε το σύνολο δεδομένων που χρησιμοποιείται στην φάση της εκπαίδευσης (training dataset), με λίγο περισσότερες από 1000 εγγραφές. Σε αυτό το σύνολο δεδομένων, περίπου το 70% προέρχεται από το configuration A, το 15% από το B και το υπόλοιπο 15% από το C. Από τα configurations D και E, με ποσοστά 80% και 20% αντίστοιχα, προέκυψε ένα δεύτερο σύνολο δεδομένων, το οποίο χρησιμοποιείται στην φάση της αξιολόγησης (testing dataset), με περίπου 300 εγγραφές.

Training Dataset	External Testing Dataset
Configuration A: 70%	Configuration D: 80%
Configuration B: 15%	Configuration E: 20%
Configuration C: 15%	
Συνολικός αριθμός εγγραφών ~1050 από 62 διαφορετικά benchmarks	Συνολικός αριθμός εγγραφών ~300 από 62 διαφορετικά benchmarks

Πίνακας 6: Training and Testing datasets

Το Docker stats επιστρέφει ένα μεγάλο πλήθος μετρικών, οι οποίες αφορούν στην συμπεριφορά της εκτέλεσης ως προς τη CPU, τη μνήμη, τις διαδικασίες εισόδου/εξόδου και το δίκτυο. Δεδομένου όμως του ότι ο στόχος είναι οι προβλέψεις του μοντέλου να μην εξαρτώνται από το υλικό όπου έγινε η εκτέλεση, πολλές από τις μετρικές που επιστρέφει το Docker stats δεν είναι χρήσιμες για την διαδικασία. Έτσι για παράδειγμα ο χρόνος πραγματικής χρησιμοποίησης της CPU από την εφαρμογή προφανώς εξαρτάται από το μοντέλο της, ωστόσο η μεταβολή στην χρησιμοποίηση, δηλαδή το πότε και πόσο απαιτητική σε υπολογισμούς γίνεται η εφαρμογή θα πρέπει να εμφανίζει κάποια ομοιότητα ανεξαρτήτως του που τρέχει, υπό την προϋπόθεση ότι οι αναγκαίοι πόροι επαρκούν. Ακολουθώντας τη λογική αυτή, οι συγγραφείς στο [6] δημιούργησαν για όσες μετρικές έχει νόημα, τον αριθμητικό μέσο όρο, το γεωμετρικό μέσο όρο, τον αριθμητικό μέσο όρο της μεταβολής και το γεωμετρικό μέσο όρο της μεταβολής, οι οποίοι συμβολίζονται ως avg_(metric_name), geo_avg_(metric_name), avg_delta_(metric_name) και geo_avg_delta_(metric_name) αντίστοιχα. Έτσι, ως μετρικές που περιέχονται τελικά στα σύνολα δεδομένων, δηλαδή ως χαρακτηριστικά του training και του testing dataset επιλέχθηκαν οι εξής:

avg_delta_cpu **geo_avg_delta_cpu**	Η τιμή της μετρικής cpu ισούται κάθε στιγμή με το συνολικό χρόνο σε nanoseconds τον οποίο χρησιμοποιήθηκε η cpu από το container.

avg_tasks **geo_avg_tasks** **avg_delta_tasks**	Η τιμή της μετρικής tasks ισούται κάθε στιγμή με τον αριθμό των νημάτων που λειτουργούν παράλληλα εντός του container.
avg_memory **avg_delta_memory** **geo_avg_memory**	Η τιμή της μετρικής memory ισούται κάθε στιγμή με τον αριθμό των bytes που χρησιμοποιούνται στη μνήμη από το container.
rx_bytes **avg_rx_bytes** **geo_avg_rx_bytes** **avg_delta_rx_bytes** **geo_avg_delta_rx_bytes**	Η τιμή της μετρικής rx_bytes ισούται κάθε στιγμή με τον αριθμό των bytes που έχουν ληφθεί από το container.
tx_bytes **avg_tx_bytes** **avg_delta_tx_bytes** **geo_avg_tx_bytes** **geo_avg_delta_tx_bytes**	Η τιμή της μετρικής tx_bytes ισούται κάθε στιγμή με τον αριθμό των bytes που έχουν σταλεί από το container.
rx_packets **avg_rx_packets** **avg_delta_rx_packets** **geo_avg_rx_packets** **geo_avg_delta_rx_packets**	Η τιμή της μετρικής rx_packets ισούται κάθε στιγμή με τον αριθμό των πακέτων που έχουν ληφθεί από το container.
tx_packets **avg_tx_packets** **avg_delta_tx_packets** **geo_avg_tx_packets** **geo_avg_delta_tx_packets**	Η τιμή της μετρικής tx_packets ισούται κάθε στιγμή με τον αριθμό των πακέτων που έχουν σταλεί από το container.

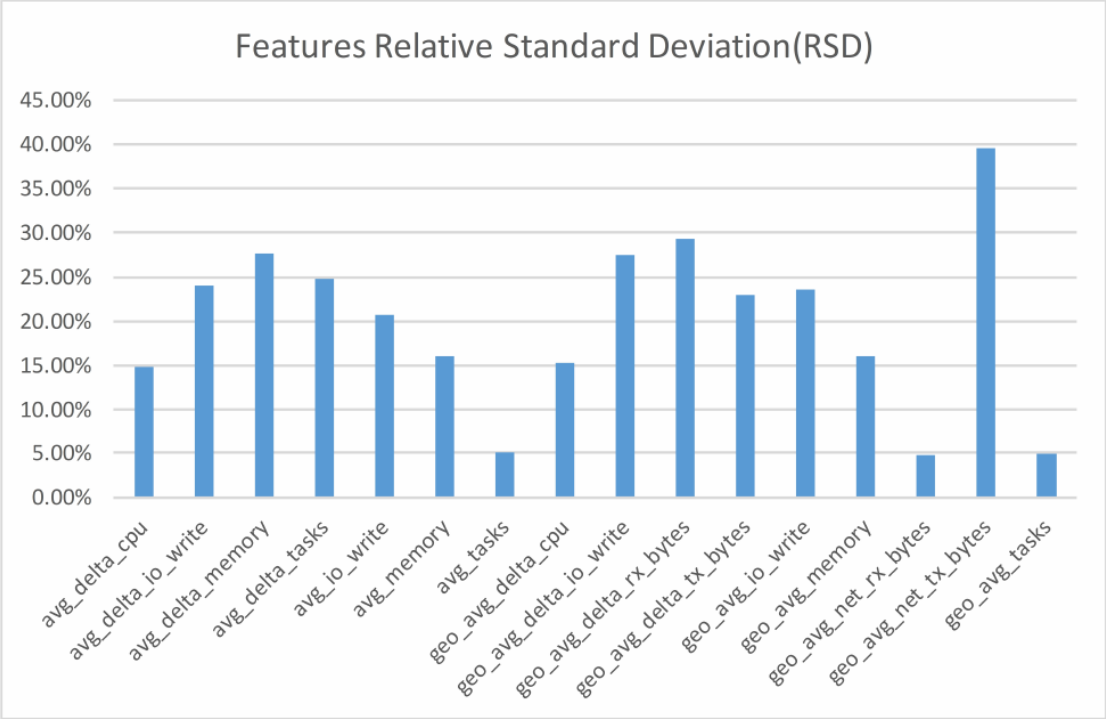
io_read **avg_io_read** **avg_delta_io_read** **geo_avg_io_read** **geo_avg_delta_io_read**	Η τιμή της μετρικής io_read ισούται κάθε στιγμή με το σύνολο των bytes που έχουν γραφτεί στο δίσκο από το container.
io_write **avg_io_write** **geo_avg_io_write** **avg_delta_io_write** **geo_avg_delta_io_write**	Η τιμή της μετρικής io_write ισούται κάθε στιγμή με το σύνολο των bytes που έχουν διαβαστεί από το δίσκο από το container.

Πίνακας 7: Χαρακτηριστικά Προφίλ

Έπειτα, για να επιλεχθούν τα χαρακτηριστικά που απαρτίζουν εν τέλει τα προφίλ και χρησιμοποιούνται στην εκπαίδευση και την αξιολόγηση του μοντέλου ταξινόμησης, έγινε μια στατιστική ανάλυση των δεδομένων. Η ανάλυση έγινε στο training dataset και χρησιμοποιήθηκε ως δείκτης απόδοσης η σχετική τυπική απόκλιση (Relative Standard Deviation – RSD), η οποία ορίζεται ως εξής:

$$RSD = \frac{Τυπική\ Απόκλιση}{Μέση\ Τιμή} \cdot 100\%$$

Το μέγεθος αυτό εκφράζει το κατά πόσο οι τιμές μιας μεταβλητής βρίσκονται κοντά στη μέση τιμή, ή απλώνονται σε ένα ευρύτερο φάσμα τιμών. Μία μικρή σχετική τυπική απόκλιση υποδηλώνει ότι τα σημεία των δεδομένων είναι συγκεντρωμένα και συνεπώς τείνουν πιο πολύ στο να ικανοποιούν το κριτήριο της σταθερότητας. Προκειμένου να υπολογιστεί το συνολικό RSD για κάθε χαρακτηριστικό, πρώτα υπολογίζεται το RSD για κάθε ξεχωριστό συνδυασμό benchmark και φόρτου εργασίας. Στη συνέχεια, υπολογίζεται η μέση τιμή όλων των RSD των διαφορετικών χαρακτηριστικών. Έτσι, επιλέγονται τελικά 16 χαρακτηριστικά με σχετικά μικρές RSD. Στο τέλος, διεξήχθη ένα ANOVA test για τα ήδη επιλεγμένα χαρακτηριστικά. Θα ασχοληθούμε πιο αναλυτικά με αυτό παρακάτω.



Γράφημα 1: RSD Χαρακτηριστικών Προφίλ [6]

Τέλος, στην αρχική προσέγγιση παρουσιάζεται μια διαδικασία επιλογής αλγορίθμου μηχανικής μάθησης, για την διαδικασία της ταξινόμησης. Γίνεται σύγκριση αρκετών διαφορετικών αλγορίθμων χρησιμοποιώντας τη μέθοδο 10-fold cross-validation και τα μεγέθη Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Relative Absolute Error (RAE) και accuracy. Ο αλγόριθμος που επιλέγεται και χρησιμοποιείται τελικά είναι ο Random Forest.

3.2 Παρούσα προσέγγιση

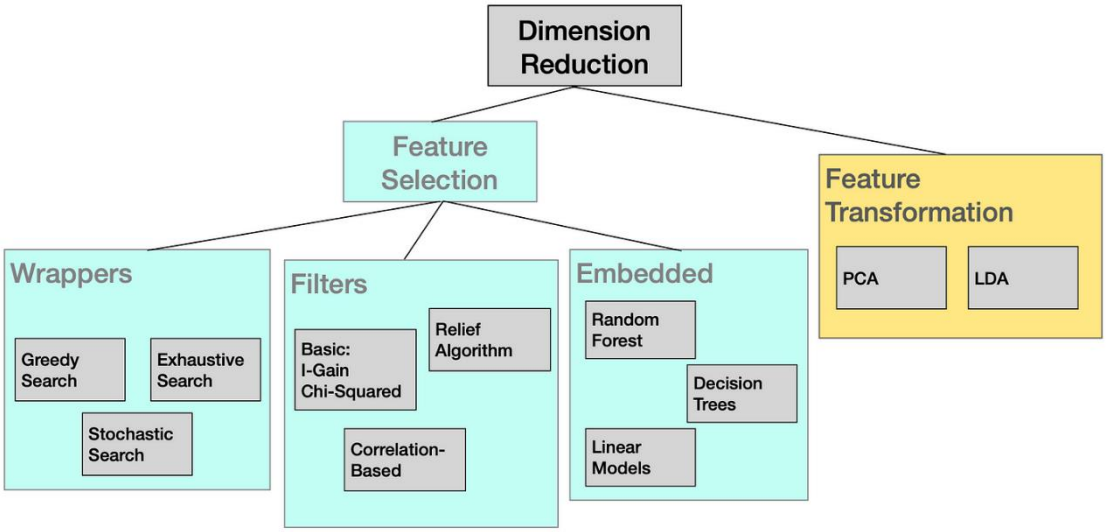
Στην παρούσα προσέγγιση του συστήματος επιχειρούμε να βελτιστοποιήσουμε τα προφίλ των εφαρμογών μέσω επιλογής χαρακτηριστικών. Για να το πετύχουμε αυτό αναπτύξαμε έναν αλγόριθμο επιλογής χαρακτηριστικών αξιοποιώντας στατιστικές μεθόδους. Για την αξιολόγηση των διαφορετικών προφίλ εφαρμογών αλλά και του αλγορίθμου μας δημιουργήσαμε ένα μοντέλο ταξινόμησης ΤΝΔ. Δεδομένου ότι εργαζόμαστε με tabular data το ΤΝΔ ίσως δεν αποτελεί την καλύτερη επιλογή

μοντέλου για την διαδικασία ταξινόμησης. Παρόλα αυτά, επιλέχθηκε ως καταλληλότερη επιλογή για την αξιολόγηση διάφορων υποσυνόλων χαρακτηριστικών, όντας ένα μοντέλο πιο τελειοποιημένο και επομένως πιο ευάλωτο στην οποιαδήποτε αλλαγή της μορφής των δεδομένων.

3.2.1 Βελτιστοποίηση προφίλ με Επιλογή Χαρακτηριστικών

Η κατηγοριοποίηση εφαρμογών, εγείρει μια σημαντική πρόκληση, η οποία αφορά την επιλογή ενός υποσυνόλου από τις μετρικές απόδοσης που έχουν συλλεχθεί με τα εργαλεία παρακολούθησης. Όπως είδαμε, στην αρχική προσέγγιση του συστήματος αυτό έγινε με βάση το RSD της εκάστοτε μετρικής, και επιλέχθηκαν 16 από ένα σύνολο 38 μετρικών. Στην παρούσα προσέγγιση παρουσιάζουμε μια μεθοδολογία επιλογής χαρακτηριστικών, με σκοπό τη βελτιστοποίηση του προφίλ που δημιουργείται, επιλέγοντας τις καταλληλότερες μετρικές για το πρόβλημά μας. Η μεθοδολογία αυτή χρησιμοποιεί πρώτα κάποιες Filter Methods για Feature Selection, έπειτα μία εκδοχή Correlation Coefficient και μία παραλλαγή του αλγορίθμου Forward Selection, που είναι Wrapper Method για Feature Selection. Με αυτήν την μεθοδολογία εξετάζουμε διάφορα υποσύνολα χαρακτηριστικών, για την αξιολόγηση των οποίων χρησιμοποιούμε ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ).

Οι μετρικές που μπορούν να χρησιμοποιηθούν κατά την διαδικασία ταξινόμησης των εφαρμογών είναι πολλές κι αυτό μπορεί να έχει αρνητικές επιπτώσεις στην απόδοση του ταξινομητή. Περισσότερες μετρικές, ή αλλιώς περισσότερα χαρακτηριστικά οδηγούν και σε πιο περίπλοκες αναπαραστάσεις των δεδομένων, με περισσότερες διαστάσεις. Ένα από τα σημαντικότερα ζητήματα που προκύπτουν όταν αυξάνεται η διαστασιμότητα είναι πως ο όγκος του χώρου αναπαράστασης αυξάνεται τόσο γρήγορα, που τα δεδομένα μέσα σε αυτόν κατανέμονται «αραιά» (sparse). Επιπλέον, σε ένα σύνολο δεδομένων με πολλά χαρακτηριστικά, πιθανότατα αρκετά από αυτά να μην έχουν αξία για την διαδικασία της ταξινόμησης, και άλλα να μπορούν να θεωρηθούν περιττά, λόγω του ότι μεταφέρουν πολύ παρόμοια πληροφορία με άλλα χαρακτηριστικά. Η επιλογή χαρακτηριστικών είναι μία μέθοδος μείωσης της διαστασιμότητας (Dimensionality Reduction), η οποία μπορεί να συνδυαστεί και με γνώσεις σχετικές με το αντικείμενο που μελετάται, και να οδηγήσει σε πολύ καλύτερη απόδοση χρησιμοποιώντας ένα υποσύνολο των αρχικών χαρακτηριστικών.



Σχήμα 2: Μέθοδοι Dimensionality Reduction και Feature Selection

Η προτεινόμενη μεθοδολογία αποτελεί μια Hybrid Method για feature selection, και οι μέθοδοι που χρησιμοποιούμε στα πλαίσια αυτής ανήκουν στις γενικότερες κατηγορίες Filter Methods και Wrapper Methods, που με τη σειρά τους ανήκουν στις επιβλεπόμενες (supervised) μεθόδους. Επιβλεπόμενες ονομάζονται οι μέθοδοι που αξιοποιούνται σε σύνολα labeled δεδομένων. Οι μέθοδοι Filter είναι πιο απλές, πιο γρήγορες και λιγότερο απαιτητικές σε πόρους. Επί της ουσίας είναι μέθοδοι αξιολόγησης της χρησιμότητας των χαρακτηριστικών με βάση κάποια στατιστική ανάλυση της σχέσης ανάμεσα στο κάθε χαρακτηριστικό και στην πρόβλεψη του μοντέλου. Οι μέθοδοι Wrapper αξιοποιούν κάποιο μοντέλο για να αξιολογήσουν διαφορετικά υποσύνολα του συνόλου χαρακτηριστικών. Κάθε υποσύνολο χρησιμοποιείται για να εκπαιδευτεί ένα μοντέλο του οποίου η απόδοση αξιολογείται με βάση κάποιο validation dataset, υποσύνολο του αρχικού, που δεν χρησιμοποιήθηκε κατά την διαδικασία της εκπαίδευσης [34].

Όπως αναφέραμε, οι μέθοδοι Filter βασίζονται συνήθως σε κάποια στατιστική μέθοδο. Εδώ χρησιμοποιούμε συνολικά έξι μεθόδους Filter, οι οποίες βασίζονται στα εξής κριτήρια:

1. Information Gain (Mutual Information): Το Mutual Information μετρά την αλληλεξάρτηση μεταξύ δύο μεταβλητών. Πιο συγκεκριμένα, μετρά την ποσότητα πληροφορίας που αντλούμε για μία τυχαία μεταβλητή παρατηρώντας μία άλλη. Το Mutual Information είναι στενά συνδεδεμένο με την έννοια της εντροπίας. Η εντροπία ουσιαστικά είναι ένα μέτρο της αβεβαιότητας. Διαισθητικά, το Mutual Information μετρά πόσο μειώνεται η αβεβαιότητα για μία τυχαία μεταβλητή μέσω της γνώσης μας σχετικά με κάποια άλλη.

2. Fisher’s Score: Η βασική ιδέα εδώ είναι να βρούμε ένα υποσύνολο χαρακτηριστικών, τέτοιο ώστε στον χώρο δεδομένων που σχηματίζεται, οι αποστάσεις μεταξύ σημείων που ανήκουν σε διαφορετικές κλάσεις να είναι όσο γίνεται μεγαλύτερες, και ταυτόχρονα, οι αποστάσεις μεταξύ σημείων που ανήκουν στην ίδια κλάση να είναι όσο πιο μικρές γίνεται.

3. Mean Absolute Difference (MAD): Για να υπολογίσουμε το MAD μεταξύ δύο συνόλων τιμών, υπολογίζουμε τον μέσο της απόλυτης τιμής της διαφοράς μεταξύ του κάθε ζεύγους τιμών των δύο συνόλων. Στην επιλογή χαρακτηριστικών, διαισθητικά, για το κάθε χαρακτηριστικό βρίσκουμε την μέση τιμή του χαρακτηριστικού για την κάθε κλάση και υπολογίζουμε την απόλυτη τιμή της διαφοράς για τις τιμές των διαφορετικών κλάσεων. Η μέση τιμή των διαφορών αυτών είναι και το MAD score του εκάστοτε χαρακτηριστικού.

4. Distance Correlation: Το Distance Correlation μετρά το μέγεθος της αλληλεξάρτησης μεταξύ δύο τυχαίων διανυσμάτων, που δεν είναι απαραίτητο να έχουν τις ίδιες διαστάσεις. Συνήθως βασίζεται σε μετρήσεις Ευκλείδειων αποστάσεων και είναι πολύ ισχυρό κριτήριο, αφού μπορεί να ποσοτικοποιήσει και γραμμικές και μη γραμμικές συσχετίσεις μεταξύ δύο τυχαίων μεταβλητών.

5. Permutation Importance: Το Permutation Importance του κάθε χαρακτηριστικού, ορίζεται ως η μείωση της απόδοσης ενός μοντέλου, όταν οι τιμές του χαρακτηριστικού ανακαταταχθούν τυχαία. Με την τεχνική αυτή δεν μετράμε κάποια σχέση μεταξύ ενός χαρακτηριστικού και των προσδοκώμενων κλάσεων, αλλά τη σημαντικότητα του χαρακτηριστικού για να μπορέσει το μοντέλο να κάνει σωστές προβλέψεις.

6. Kendall Correlation: Το Kendall Correlation είναι μια στατιστική μέθοδος που μετρά τη συσχέτιση μεταξύ δύο μεταβλητών. Αξιολογεί τόσο τη δύναμη όσο και την κατεύθυνση της συσχέτισης. Επί της ουσίας μας δίνει την εικόνα του πως δύο μεταβλητές «κινούνται» η μία σε σχέση με την άλλη.

Οι μέθοδοι Filter μας βοηθάνε να αντιληφθούμε την αξία του κάθε χαρακτηριστικού για την διαδικασία της ταξινόμησης, όμως δεν μπορούν να μας βοηθήσουν στην επιλογή υποσυνόλου χαρακτηριστικών. Για να μπορέσουμε να εξασφαλίσουμε ότι επιλέξαμε το βέλτιστο υποσύνολο χαρακτηριστικών για το πρόβλημά μας, πρέπει να εξετάσουμε όλα τα δυνατά υποσύνολα και να τα αξιολογήσουμε με βάση το επιλεγμένο μοντέλο. Κάτι τέτοιο όμως συνήθως έχει τόσο μεγάλο κόστος σε χρόνο και υπολογιστικούς πόρους, που καθίσταται αδύνατο. Δεδομένου όμως ότι το πρόβλημα που αντιμετωπίζουμε εδώ είναι ένα πρόβλημα επιβλεπόμενης μάθησης (supervised learning), δηλαδή έχουμε labeled δεδομένα, και ότι το πρόβλημα το ίδιο εγείρει εκ φύσεως κάποιους περιορισμούς, είναι εφικτό να

περιορίσουμε σε μεγάλο βαθμό τον αριθμό των δυνατών υποσυνόλων. Για παράδειγμα, γνωρίζουμε ότι είναι σημαντικό τα χαρακτηριστικά που θα αποτελέσουν το προφίλ τελικά να καλύπτουν όλα τα είδη των τυπικών πόρων που μπορεί να χρησιμοποιήσει μία εφαρμογή. Σε αυτούς τους πόρους ανήκουν η CPU, η RAM, ο αριθμός ταυτόχρονων διεργασιών, η εγγραφή / ανάγνωση από τον δίσκο και το δίκτυο. Επομένως μπορούμε να αποκλείσουμε κάθε υποσύνολο που δεν καλύπτει όλους τους πόρους.

Παρόλο που το μέγεθος του συνόλου δεδομένων μας δεν είναι μεγάλο, θα χρειαζόμασταν μεγάλη υπολογιστική ισχύ και πολύ χρόνο για να εξερευνήσουμε όλα τα δυνατά υποσύνολα χαρακτηριστικών. Για την βελτιστοποιημένη επιλογή υποσυνόλου χαρακτηριστικών χρησιμοποιούνται οι μέθοδοι Wrapper. Εμείς χρησιμοποιούμε μία παραλλαγή του αλγορίθμου Forward Selection. Ακολουθώντας την μεθοδολογία που προτείνουμε, κάνουμε μια πληροφορημένη διαλογή υποσυνόλων προς εξερεύνηση. Πρώτα, θεωρώντας αξιωματικά ότι χρειάζεται να έχουμε τουλάχιστον δύο μετρικές για τον κάθε βασικό πόρο, έπειτα, με την απομάκρυνση των χαρακτηριστικών που δεν συνεισφέρουν στην διαδικασία λόγω του ότι μεταφέρουν πολύ παρόμοια πληροφορία με τα ήδη επιλεγμένα. Τέλος, επιλέγοντας σε κάθε βήμα να εξετάσουμε μόνο τα πιο υποσχόμενα από τα υποσύνολα που απομένουν, ακολουθώντας την greedy λογική του forward selection. Έτσι τελικά μπορούμε να είμαστε βέβαιοι ότι έχουμε προσεγγίσει με μεγάλη ακρίβεια το καλύτερο υποσύνολο χαρακτηριστικών, και συνεπώς και την καλύτερη απόδοση, εξετάζοντας ένα μικρό ποσοστό του συνόλου όλων των δυνατών υποσυνόλων.

3.2.2 Τεχνητά Νευρωνικά Δίκτυα

Προτού προχωρήσουμε στην υλοποίηση του αλγορίθμου και του μοντέλου μας, χρειάζεται να κατανοήσουμε καλύτερα την λειτουργία του μοντέλου. Στην παράγραφο αυτή θα εξετάσουμε τη γενική λειτουργία των ΤΝΔ καθώς παρουσιάζουμε και αιτιολογούμε τις δικές μας επιλογές σε ότι αφορά τη δομή του μοντέλου. Ξεκινάμε εξηγώντας πρώτα τι είναι τα Τεχνητά Νευρωνικά Δίκτυα και έπειτα καθένα από τα βασικά τους δομικά χαρακτηριστικά.

Τα Τεχνητά Νευρωνικά Δίκτυα, ή απλώς Νευρωνικά Δίκτυα, αποτελούν ένα υποσύνολο των αλγορίθμων Μηχανικής Μάθησης, που αποτελούν με τη σειρά τους ένα υποσύνολο των μεθόδων Τεχνητής Νοημοσύνης. Το όνομα και η λειτουργία τους είναι εμπνευσμένα από τον ανθρώπινο εγκέφαλο, προσομοιώνοντας τους βιολογικούς νευρώνες. Αποτελούνται από επίπεδα (layers) διασυνδεδεμένων κόμβων, οι οποίοι μοντελοποιούν τους νευρώνες, με τις συνδέσεις μεταξύ τους να αναπαριστούν τις διανευρωνικές συνάψεις. Ο νευρώνας είναι η βασική μονάδα επεξεργασίας σε ένα ΤΝΔ, καθώς δέχεται κάποιες εισόδους, και παράγει μια έξοδο, η οποία εισάγεται σε

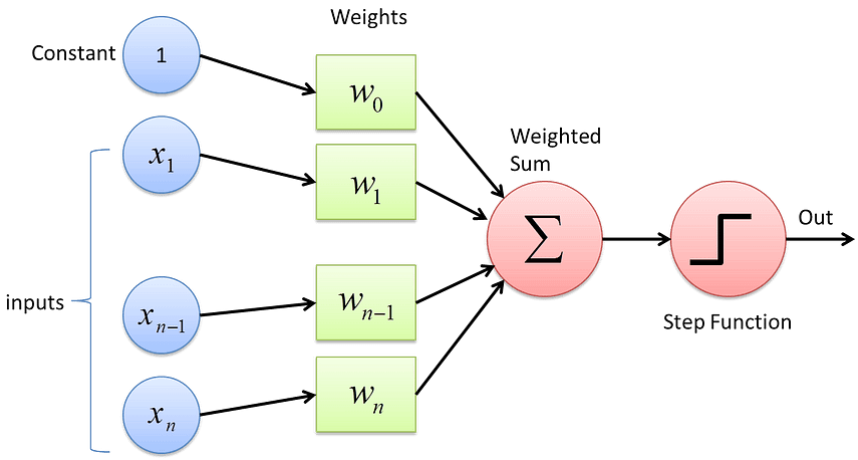
επόμενους νευρώνες σαν είσοδος. Το υπολογιστικό κόστος για κάθε νευρώνα είναι πολύ μικρό, δεδομένου ότι εκτελεί πολύ απλές πράξεις. Παρόλα αυτά, τα ΤΝΔ αποτελούν πανίσχυρα μοντέλα σε διάφορα προβλήματα, καθώς βασίζονται στην συντονισμένη λειτουργία μεγάλου πλήθους νευρώνων στη δομή ενός δικτύου. Μπορούν να βοηθήσουν σημαντικά τους υπολογιστές να λαμβάνουν έξυπνες αποφάσεις, περιορίζοντας τη συμμετοχή του ανθρώπινου παράγοντα, καθώς μπορούν να εκπαιδευτούν σε μεγάλα σύνολα δεδομένων και να μοντελοποιήσουν περίπλοκες σχέσεις.

3.2.2.1 Το Perceptron

Το perceptron είναι η απλούστερη μορφή νευρωνικού δικτύου. Αποτελείται ουσιαστικά από έναν νευρώνα που λαμβάνει ως είσοδο ένα διάνυσμα της μορφής $x = [x_1, x_2, \ldots, x_n]$ και η έξοδος του καθορίζεται από το αποτέλεσμα του σταθμισμένου αθροίσματος των εισόδων με βάρη (weights) $w = [w_1, w_2, \ldots, w_n]$ συν μία σταθερά b που ονομάζεται πόλωση (bias). Πιο συγκεκριμένα έχουμε:

$$f(x, w, b) = \begin{cases} 1 & , w \cdot x + b > 0 \\ 0 & , \text{διαφορετικά} \end{cases}$$

Όπως φαίνεται από τον παραπάνω ορισμό, το perceptron είναι ένας δυαδικός ταξινομητής, εφόσον η έξοδός του μπορεί να πάρει δύο διακριτές τιμές. Μια άλλη παρατήρηση που μπορούμε να κάνουμε είναι ότι πρόκειται για έναν γραμμικό ταξινομητή, καθώς η έξοδος του προκύπτει από γραμμικό συνδυασμό των εισόδων του. Μπορεί να εφαρμοστεί επομένως μόνο στην κατηγορία των προβλημάτων που ικανοποιούν το κριτήριο της γραμμικής διαχωρισιμότητας. Αυτό σημαίνει ότι είναι δυνατόν να οριστεί ένα γραμμικό υπερεπίπεδο, στην περίπτωσή μας το $w \cdot x + b = 0$ το οποίο να αποτελεί το σύνορο απόφασης για τις δύο κλάσεις, δηλαδή τα σημεία των δύο κλάσεων να βρίσκονται ομαδοποιημένα εκατέρωθεν του υπερεπιπέδου.

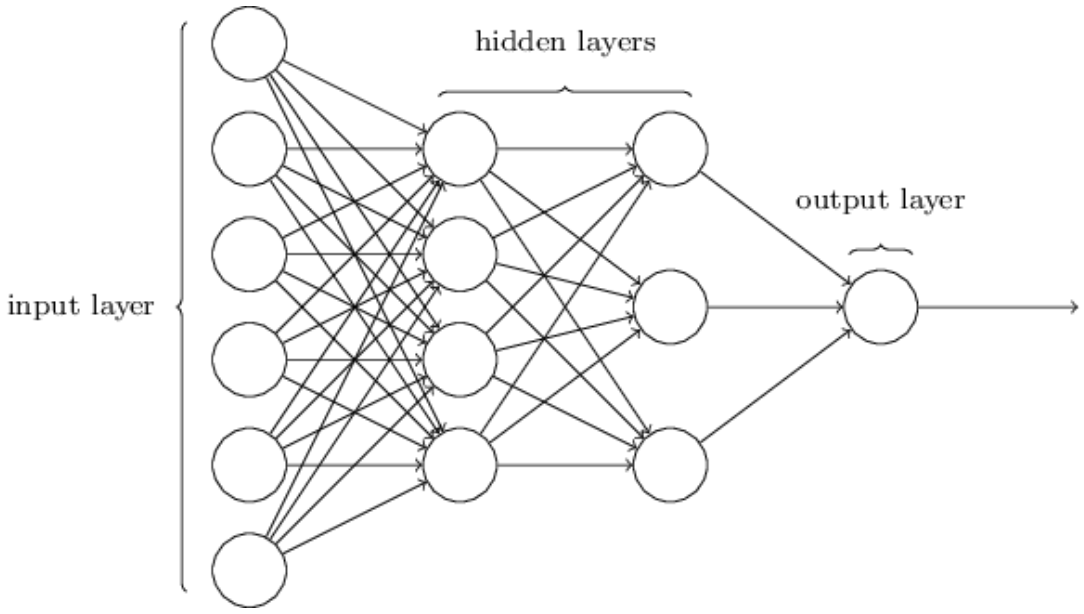


Σχήμα 3: Το perceptron [35]

Επομένως η μοντελοποίηση ενός γραμμικού δυαδικού προβλήματος ταξινόμησης με perceptron συνίσταται στον προσδιορισμό των βαρών ***w*** και της πόλωσης ***b*** τα οποία θα ορίζουν το κατάλληλο υπερεπίπεδο διαχωρισμού για το εκάστοτε πρόβλημα.

3.2.2.2 Το πολυεπίπεδο Perceptron

Το πολυεπίπεδο perceptron (Multilayer Perceptron - MLP) αποτελεί μια γενίκευση του απλού perceptron. Αποτελείται από πλήθος νευρώνων οργανωμένων σε διαδοχικά επίπεδα (layers) στα οποία κάθε νευρώνας σε κάποιο επίπεδο συνδέεται με κάθε νευρώνα του προηγούμενου επιπέδου. Τα επίπεδα που συναντάμε διαδοχικά σε ένα MLP είναι: 1 επίπεδο εισόδου, 1 ή περισσότερα κρυφά επίπεδα και 1 επίπεδο εξόδου. Οι νευρώνες του επιπέδου εισόδου μεταφέρουν αυτούσια την είσοδο στην έξοδό τους ώστε να γίνει η επεξεργασία από τα επόμενα επίπεδα.



Σχήμα 4: Το πολυεπίπεδο perceptron [36]

Η διαδικασία υπολογισμού της εξόδου των κρυφών επιπέδων και των επιπέδων εξόδου αποτελεί μία γενίκευση της διαδικασίας υπολογισμού της εξόδου του απλού perceptron. Αναλυτικότερα, ορίζεται μία συνάρτηση ενεργοποίησης (activation function), η οποία εφαρμόζεται στην τιμή $w \cdot x + b$ και καθορίζει την τελική έξοδο του νευρώνα. Ο υπολογισμός της εξόδου y_i^l του νευρώνα i στο επίπεδο l προκύπτει από την εξίσωση:

$$a_i^l = \sigma\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l \cdot a_j^{l-1} + b_i^l\right)$$

Όπου:

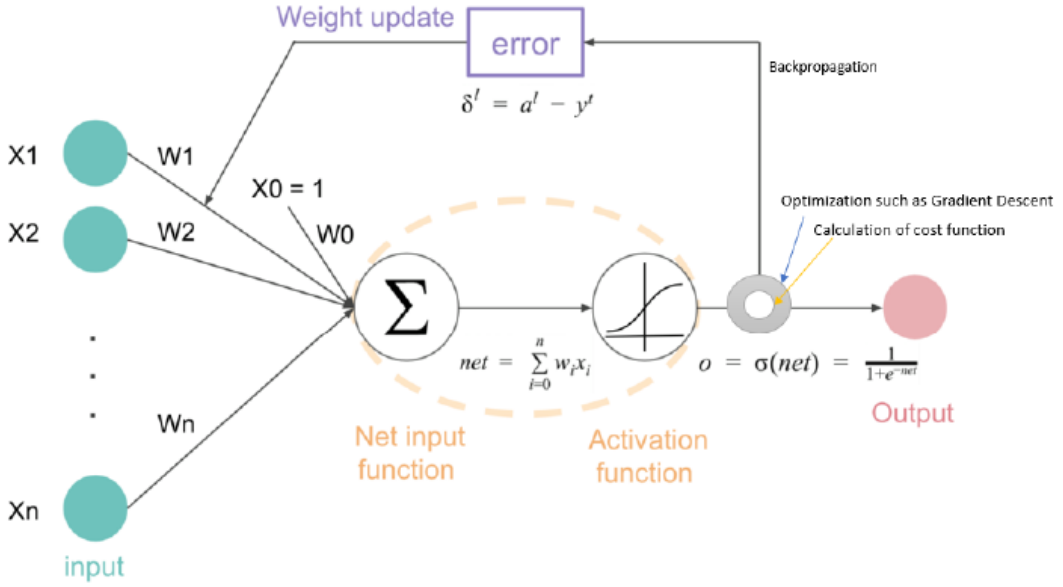
- $\boldsymbol{\sigma}$ είναι η συνάρτηση ενεργοποίησης
- $\boldsymbol{N_{l-1}}$ το πλήθος των νευρώνων στο επίπεδο $l-1$
- $\boldsymbol{b_i^l}$ η πόλωση του νευρώνα i στο επίπεδο l
- $\boldsymbol{w_{ij}^l}$ το βάρος της σύνδεσης του νευρώνα i στο επίπεδο l με τον νευρώνα j στο επίπεδο $l-1$
- $\boldsymbol{a_i^l}$ είναι η ενεργοποίηση του νευρώνα i στο επίπεδο l (δηλαδή η έξοδός του μετά τη συνάρτηση ενεργοποίησης)

Το απλό perceptron θα μπορούσε να θεωρηθεί οριακή περίπτωση στην οποία χρησιμοποιείται αποκλειστικά η βηματική συνάρτηση ως συνάρτηση ενεργοποίησης. Η διαφορά όμως μεταξύ των δύο είναι πιο ουσιαστική από τη δυνατότητα χρήσης αυθαίρετης συνάρτησης ενεργοποίησης στα πολυεπίπεδα perceptron. Σε αντίθεση με το απλό perceptron δεν έχουμε να κάνουμε πλέον με ένα γραμμικό ταξινομητή. Αυτό οφείλεται κατά βάση στη συνάρτηση ενεργοποίησης η οποία εκλέγεται εσκεμμένα να είναι μια μη γραμμική συνάρτηση, ώστε να εισαχθεί η απαραίτητη μη-γραμμικότητα στο μοντέλο. Σε διαφορετική περίπτωση οι έξοδοι θα προκύπταν με γραμμικό τρόπο από τις εισόδους και έτσι θα είχαμε ένα τετριμμένο γραμμικό μοντέλο με περιττή πολυπλοκότητα.

3.2.2.3 Αλγόριθμος Οπισθοδιάδοσης (Backpropagation Algorithm)

Η εκπαίδευση των ΤΝΔ ανήκει στην κατηγορία της επιβλεπόμενης εκπαίδευσης, καθώς γνωρίζουμε τις επιθυμητές εξόδους κάθε διαθέσιμου δείγματος. Η διαδικασία εκπαίδευσης συνίσταται στην επαναληπτική ανανέωση των παραμέτρων του δικτύου, ώστε οι έξοδοι να προσεγγίσουν όσο το δυνατόν καλύτερα τις επιθυμητές. Γι' αυτό το σκοπό χρησιμοποιείται το training set. Στη συνέχεια, με χρήση του testing set μπορούμε να ελέγξουμε την απόδοση του εκπαιδευμένου πλέον ΤΝΔ και να είναι δυνατή η αξιολόγηση του και η σύγκρισή του με άλλα μοντέλα.

Η εκπαίδευση των ΤΝΔ επιτυγχάνεται με τη χρήση του αλγόριθμου οπισθοδιάδοσης (Backpropagation Algorithm) [37]. Οι παράμετροι προς εκμάθηση αρχικοποιούνται συνήθως σε τυχαίες τιμές ή αρχικοποιούνται σε τιμές από μερικώς εκπαιδευμένα δίκτυα. Κατά τη διαδικασία εκμάθησης κάθε δείγμα (ή batch) δίνεται ως είσοδος στο μοντέλο, το οποίο εκτιμάει την έξοδο με βάση τις τιμές των παραμέτρων του. Αυτό αποτελεί την πρώτη φάση του αλγορίθμου, το λεγόμενο πρόσθιο πέρασμα (forward pass). Η έξοδος αυτή χρησιμοποιείται για τον υπολογισμό της συνάρτησης κόστους η οποία πρέπει να ελαχιστοποιηθεί. Για να επιτευχθεί η ελαχιστοποίηση, με βάση τον αλγόριθμο οπισθοδιάδοσης, υπολογίζονται οι μερικές παράγωγοι της συνάρτησης κόστους ως προς κάθε παράμετρο του ΤΝΔ. Ο αλγόριθμος οπισθοδιάδοσης οφείλει το όνομα του στο γεγονός ότι οι κλίσεις υπολογίζονται διαδοχικά ξεκινώντας από το τελευταίο επίπεδο του ΤΝΔ και οι τιμές τους χρησιμοποιούνται για τους αντίστοιχους υπολογισμούς σε προηγούμενα επίπεδα με βάση τον κανόνα της αλυσίδας. Έτσι έχουμε κατά μία έννοια την προς τα πίσω διάδοση της υπολογισμένης κλίσης από το τελευταίο στο πρώτο επίπεδο του ΤΝΔ. Στην τελευταία φάση του αλγορίθμου, οι κλίσεις αυτές χρησιμοποιούνται για την ανανέωση των βαρών στη βάση κάποιου αλγόριθμου ελαχιστοποίησης.



Σχήμα 5: Ο Αλγόριθμος Οπισθοδιάδοσης σε ένα ΤΝΔ

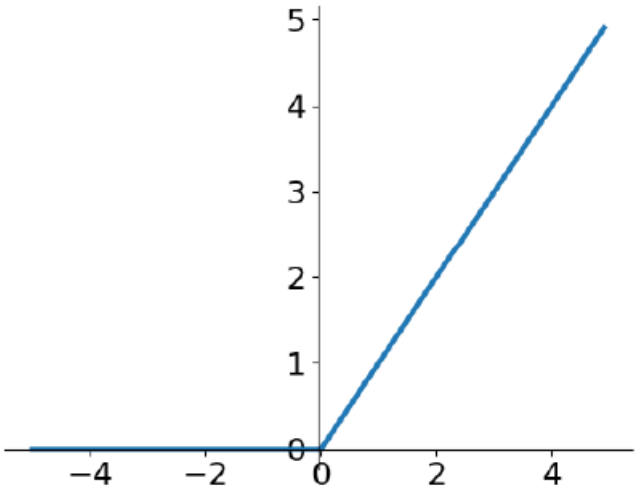
3.2.2.4 Συναρτήσεις Ενεργοποίησης

Προκειμένου να έχει νόημα η τεράστια ισχύς γενίκευσης των μοντέλων ΤΝΔ είναι σημαντικό να υπάρχουν τα λεγόμενα επίπεδα ενεργοποίησης. Ένα ΤΝΔ χωρίς επίπεδα ενεργοποίησης μπορεί να περιγραφεί από πράξεις γραμμικής άλγεβρας (πινάκων) ανάμεσα στις εξόδους του προηγούμενου επιπέδου και στα βάρη-παραμέτρους του παρόντος. Έτσι, ακόμα και η πιο πολύπλοκη αρχιτεκτονική δε θα μπορούσε να είναι παραπάνω εκφραστική από ένα γραμμικό μοντέλο. Επομένως, η χρήση των συναρτήσεων ενεργοποίησης αποσκοπεί κυρίως στην εισαγωγή μη-γραμμικότητας στα μοντέλα, ώστε να αυξήσουμε την εκφραστική ισχύ των μοντέλων μας πέρα από την τετριμμένη γραμμική απεικόνιση. Όπως είναι αναμενόμενο λοιπόν, οι συναρτήσεις ενεργοποίησης είναι μη γραμμικές συναρτήσεις. Στο ΤΝΔ που αναπτύξαμε στα πλαίσια της εργασίας χρησιμοποιούμε δύο από τις πιο ευρέως χρησιμοποιούμενες συναρτήσεις ενεργοποίησης

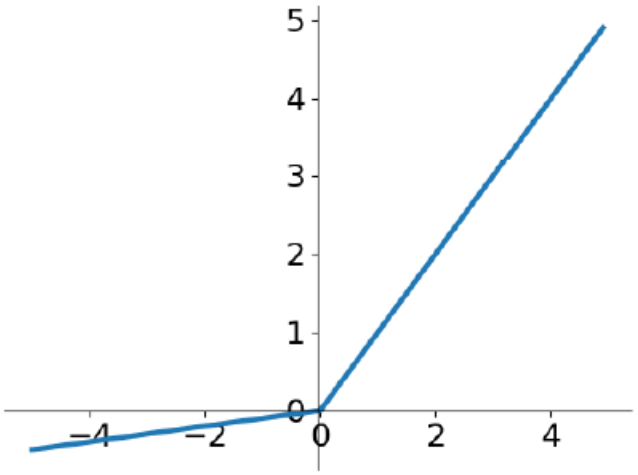
1. ReLU : Η συνάρτηση ReLU (Rectified Linear Unit) αποτελεί την κατεξοχήν επιλογή συνάρτησης ενεργοποίησης σε βαθύτερες αρχιτεκτονικές. Ανορθώνει την είσοδο της, δηλαδή διατηρεί τις θετικές εισόδους και μηδενίζει τις αρνητικές. Μαθηματικά μπορεί να οριστεί ως:

$$f\ (x)\ = max(\ 0,\ x)$$

Η συνάρτηση ReLU χρησιμοποιείται ευρύτατα καθώς έχει σημαντικά πλεονεκτήματα σε σχέση με άλλες. Πιο συγκεκριμένα, πρόκειται για μία μονόπλευρη συνάρτηση η οποία προσομοιάζει καλύτερα τις ενεργοποιήσεις των βιολογικών νευρώνων και επίσης μειώνει τον κίνδυνο εξαφανιζόμενων κλίσεων, καθώς μπορεί να κορεστεί μόνο από τη μία πλευρά. Πρόβλημα εξαφανιζόμενων κλίσεων (vanishing gradients problem) ονομάζεται ο μηδενισμός της κλίσης, που έχει ως αποτέλεσμα να σταματήσει να διαδίδεται το σφάλμα και να είναι αδύνατη η περαιτέρω εκπαίδευση του ΤΝΔ. Επίσης, η ReLU είναι αρκετά απλούστερη υπολογιστικά και έχει αραιή ενεργοποίηση λόγω του μηδενισμού των αρνητικών εισόδων. Παρόλα αυτά, μπορούμε να συναντήσουμε κι εδώ προβλήματα με μηδενισμό κλίσεων. Το πρόβλημα σε αυτή την περίπτωση ονομάζεται dying ReLU problem και αναφέρεται στην κατάσταση κάποιου νευρώνα ο οποίος, λόγω πολύ αρνητικής πόλωσης, μπορεί να μηδενίζεται για κάθε πιθανή είσοδο. Έτσι, καταλήγει να «πεθαίνει», να είναι δηλαδή πλήρως ανενεργός, αφαιρώντας πολυπλοκότητα από το μοντέλο. Μία παραλλαγή που έχει προταθεί για να αντιμετωπιστεί αυτό το πρόβλημα είναι η Leaky ReLU, η οποία αντί να μηδενίζει τις αρνητικές εισόδους, τις απεικονίζει με μία πολύ μικρή αρνητική κλίση.



Σχήμα 6: Η συνάρτηση ReLU



Σχήμα 7: Η συνάρτηση Leaky ReLU

2. Softmax : Η συνάρτηση Softmax δέχεται ως είσοδο ένα διάνυσμα N πραγματικών αριθμών και το κανονικοποιεί, ώστε να προκύψει μία κατανομή πιθανότητας. Η έξοδος επομένως αποτελείται από N πραγματικούς αριθμούς στο διάστημα [0, 1], και των οποίων το άθροισμα ισούται με 1. Η συνάρτηση Softmax ορίζεται από τον παρακάτω τύπο:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Στην περίπτωση της ταξινόμησης μέσω Τεχνητών Νευρωνικών Δικτύων η softmax αποτελεί συνήθως τη συνάρτηση ενεργοποίησης του τελευταίου επιπέδου. Έτσι λοιπόν οι έξοδοι, ίσες στον αριθμό με τις πιθανές κλάσεις εντός των οποίων εκτελείται η ταξινόμηση, μετατρέπονται σε μία κατανομή πιθανότητας η οποία εκφράζει την πιθανότητα το τρέχον δείγμα να ανήκει σε καθεμία από τις διαθέσιμες κλάσεις. Η κατανομή αυτή ουσιαστικά είναι και η τελική εκτίμηση του μοντέλου, η οποία θα αποτελέσει είσοδο στη συνάρτηση ελαχιστοποίησης σφάλματος, ώστε τελικά να εκπαιδευτεί το δίκτυο.

3.2.2.5 Optimizers

Όταν ένα νευρωνικό δίκτυο βρίσκεται στην διαδικασία της εκπαίδευσης χρειάζεται να προσαρμόζει τις εσωτερικές του παραμέτρους για να κάνει τις προβλέψεις ή τις κατηγοριοποιήσεις πιο ακριβείς. Οι optimizers είναι οι τεχνικές και οι μέθοδοι που λένε στο νευρωνικό δίκτυο πως να προσαρμόσει αυτές τις παραμέτρους κατά τη διάρκεια της μάθησης. Ο optimizer που χρησιμοποιούμε εδώ ονομάζεται Adam και βασίζεται στον αλγόριθμο βελτιστοποίησης Adam [38]. Η λειτουργία του optimizer Adam μπορεί να αναλυθεί στα εξής βήματα:

1. Αρχικοποίηση των βαρών του μοντέλου με κάποιες αρχικές τιμές.

2. Για κάθε εποχή εκπαίδευσης:
    a. Υπολογισμός της παραγώγου της συνάρτησης κόστους (loss function) ως προς τις εσωτερικές παραμέτρους του μοντέλου.
    b. Υπολογισμός του εκθετικού κινούμενου μέσου όρου (exponential moving average) των παραγώγων, και του εκθετικού κινούμενου μέσου όρου των τετραγώνων των παραγώγων.
    c. Αξιοποίηση αυτών των εκθετικών κινούμενων μέσων όρων για την προσαρμογή του ρυθμού μάθησης (learning rate) για την τρέχουσα επανάληψη.
    d. Ενημέρωση των εσωτερικών παραμέτρων του μοντέλου χρησιμοποιώντας τον προσαρμοσμένο ρυθμό μάθησης και τις παραγώγους που υπολογίστηκαν προηγουμένως.

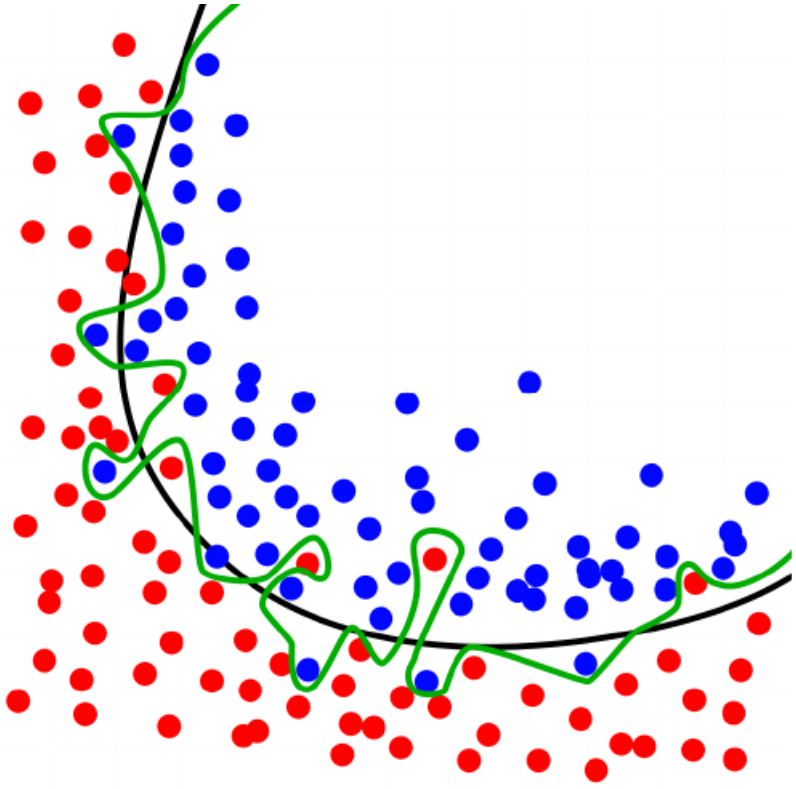
3. Επανάληψη των βημάτων 2 και 3 μέχρι το μοντέλο να συγκλίνει, ή να εξαντληθεί ο μέγιστος αριθμός εποχών.

3.2.2.6 Υπερεκπαίδευση (overfitting) και τεχνικές ομαλοποίησης

Με τον όρο υπερεκπαίδευση (overfitting) αναφερόμαστε στην εκπαίδευση ενός μοντέλου μηχανικής μάθησης με τρόπο τέτοιο ώστε να επιτυγχάνεται πολύ υψηλή απόδοση στο training set, αλλά αρκετά χαμηλή στο testing set. Το πρόβλημα αυτό οφείλεται στην χρήση πολύπλοκων μοντέλων τα οποία με εντατική εκπαίδευση μοντελοποιούν πέρα από τα ουσιαστικά μοτίβα που κρύβονται στα δεδομένα και τον

τυχαίο θόρυβο που αναπόφευκτα υπάρχει. Ως αποτέλεσμα, η υψηλή απόδοση στο training set στο οποίο εκπαιδεύτηκε ένα υπερεκπαιδευμένο μοντέλο, δε μπορεί να γενικευτεί όταν του παρουσιαστούν δεδομένα τα οποία δεν έχει «ξαναδεί». Το πρόβλημα της υπερεκπαίδευσης είναι ιδιαίτερα υπαρκτό στα μοντέλα ΤΝΔ, καθώς πρόκειται για πολύπλοκα μοντέλα που μπορούν να έχουν μέχρι και δισεκατομμύρια παραμέτρους. Γι' αυτό το λόγο έχουν αναπτυχθεί διάφορες τεχνικές ομαλοποίησης με σκοπό να περιορίσουν την ισχύ του προβλήματος.

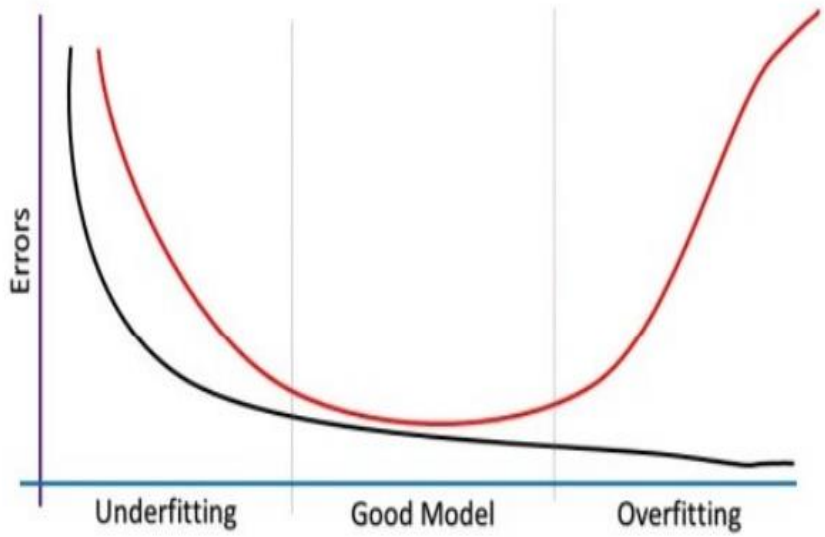
Στο Σχήμα 7 συγκρίνεται ένα ομαλοποιημένο με ένα υπερεκπαιδευμένο μοντέλο. Όπως γίνεται προφανές, η πράσινη γραμμή ακολουθεί με υπερβολικά μεγάλη ακρίβεια τις παρατηρήσεις οδηγώντας σε ένα πιο περίπλοκο μοντέλο. Στην πράξη, αυτό οδηγεί σε αδυναμία γενίκευσης, δηλαδή κακή απόδοση σε καινούρια δεδομένα τα οποία δεν χρησιμοποιήθηκαν κατά την εκπαίδευση. Το ομαλοποιημένο μοντέλο, αντίθετα, κάνει έναν πιο φυσικό διαχωρισμό στα δεδομένα, λαμβάνοντας υπόψιν και σημεία θορύβου που μπορεί να υπάρχουν εκατέρωθεν του συνόρου που ορίζει.



Σχήμα 8: Οπτικοποίηση της υπερεκπαίδευσης

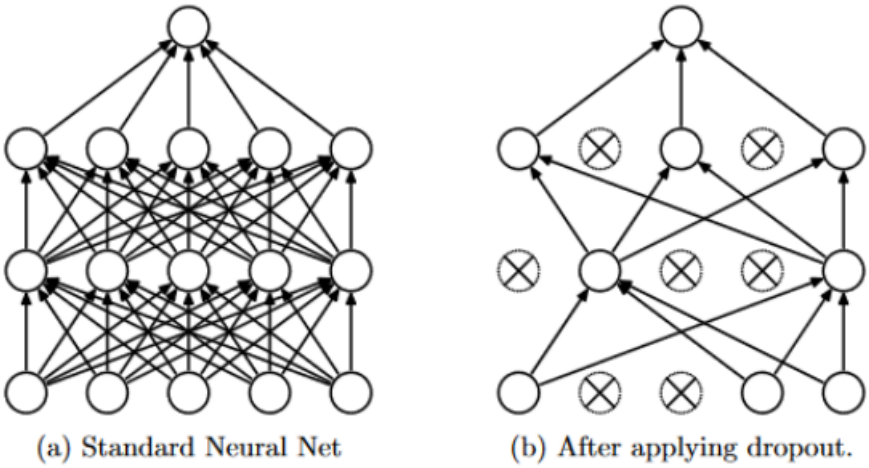
Στο ΤΝΔ που αναπτύχθηκε στα πλαίσια της εργασίας μας χρησιμοποιούμε τρεις από τις πλέον γνωστές τεχνικές ομαλοποίησης:

1. Πρόωρος Τερματισμός (Early Stopping) : Ο πρόωρος τερματισμός (Early Stopping) είναι μία ευρέως χρησιμοποιούμενη τεχνική ομαλοποίησης, καθώς έχει πολύ ικανοποιητική απόδοση με ελάχιστη πολυπλοκότητα υλοποίησης. Βασίζεται στην απλή λογική του περιοδικού ελέγχου του μοντέλου πάνω σε ένα σύνολο δεδομένων επικύρωσης (validation set), διαφορετικό και συνήθως μικρότερο από τα training και testing set. Με βάση την επίδοση του μοντέλου πάνω σε αυτό το σετ δεδομένων μπορεί να ληφθεί απόφαση για πρόωρο τερματισμό της εκπαίδευσης ώστε να αποφευχθεί η περίπτωση της υπερεκπαίδευσης (overfitting). Η απόφαση συνήθως λαμβάνεται όταν η συνάρτηση κόστους δεν έχει βελτιωθεί (δηλαδή δεν έχει βρεθεί σε νέο ελάχιστο) στη διάρκεια ενός συγκεκριμένου αριθμού εποχών εκπαίδευσης. Η χρησιμότητα του πρόωρου τερματισμού μπορεί να γίνει καλύτερα κατανοητή με τη μελέτη της συμπεριφοράς της συνάρτησης κόστους κατά τη διάρκεια της εκπαίδευσης του μοντέλου. Στο Σχήμα 8 φαίνεται η πορεία της συνάρτησης κόστους για τα δεδομένα εκπαίδευσης και επικύρωσης. Στην αρχή της εκπαίδευσης το μοντέλο είναι υποεκπαιδευμένο (underfitted), με το κόστος για τα δεδομένα εκπαίδευσης να είναι ακόμα αρκετά χαμηλότερο από τα δεδομένα επικύρωσης.



Σχήμα 9: Η συνάρτηση κόστους στη διάρκεια της εκπαίδευσης

2. Dropout : Η τεχνική του dropout [39] είναι μία από τις πιο απλές και αποτελεσματικές τεχνικές για την μείωση της υπερεκπαίδευσης σε μοντέλα ΤΝΔ. Εφαρμόζεται σε κάποιο επίπεδο του δικτύου, ορίζοντας μία τιμή $0 < dropout < 1$ η οποία καθορίζει το ποσοστό των νευρώνων του επιπέδου οι οποίοι θα απενεργοποιηθούν με τυχαία επιλογή για κάθε επανάληψη. Προσομοιάζει μια ensemble μέθοδο, καθώς μπορεί να παρουσιαστεί ως ανάλογο της ταυτόχρονης εκπαίδευσης πολλών διαφορετικών αρχιτεκτονικών ΤΝΔ και του μετέπειτα συνδυασμού τους. Τα Τεχνητά Νευρωνικά Δίκτυα αποτελούν ισχυρά μοντέλα, στα οποία έχουμε συνδιαμορφώσεις των κόμβων, με αποτέλεσμα να υπάρχουν κόμβοι που διορθώνουν λάθη προηγουμένων, οδηγώντας σε υποβέλτιστα αποτελέσματα και σε αδυναμία γενίκευσης σε διαφορετικά δεδομένα. Η τεχνική του dropout περιορίζει αυτά τα φαινόμενα, επιτρέποντας τη διέλευση των δεδομένων από το δίκτυο μόνο μέσα από περιορισμένα και λιγότερο πολύπλοκα μονοπάτια διαμέσου του δικτύου. Πρακτικά μειώνεται με τυχαίους τρόπους η πολυπλοκότητα κατά τη διάρκεια της εκπαίδευσης χωρίς να θυσιαστεί η συνολική πολυπλοκότητα του ΤΝΔ ως εκπαιδευμένο μοντέλο προς χρήση. Σημειώνεται ότι το dropout εφαρμόζεται μόνο κατά τη διαδικασία εκπαίδευσης. Στη διαδικασία ελέγχου χρησιμοποιούνται όλοι οι κόμβοι του δικτύου.



Σχήμα 10: Οπτικοποίηση του Dropout

3. Batch Normalization : Το Batch Normalization [40], είναι μία τεχνική κατά την οποία κανονικοποιείται η έξοδος ενός επιπέδου του ΤΝΔ (και επομένως η είσοδος στο επόμενο). Πιο συγκεκριμένα, γίνεται υπολογισμός της μέσης τιμής και διακύμανσης για κάθε διάσταση του διανύσματος εισόδου, κατά μήκος του mini-batch, εκτελείται κανονικοποίηση (αφαίρεση μέσης τιμής και διαίρεση με τη διακύμανση) και η τελική κατανομή προκύπτει από κλιμακοποίηση και μετατόπιση, από δύο παραμέτρους οι οποίες τίθενται προς εκμάθηση. Το batch normalization έχει ιδιαίτερα θετική επίδραση στην ποιότητα και διαδικασία εκπαίδευσης του μοντέλου. Δίνεται η δυνατότητα να χρησιμοποιηθούν υψηλότεροι ρυθμοί εκμάθησης, επιτρέποντας την ταχύτερη σύγκλιση και περιορίζοντας τον κίνδυνο του φαινομένου των exploding gradients. Αυτό συνδέεται και με τη δυνατότητα που δίνει στην εκπαίδευση να είναι ανθεκτικότερη στην κλίμακα των παραμέτρων του δικτύου. Στο άρθρο [40] στο οποίο διατυπώθηκε η συγκεκριμένη τεχνική, οι συγγραφείς υποστηρίζουν ότι η επιτυχία της οφείλεται στο γεγονός ότι μειώνει το λεγόμενο Internal Covariate Shift. Ο όρος αυτός περιγράφει το γεγονός ότι η κατανομή εισόδου σε κάθε στρώμα του ΤΝΔ εξαρτάται από τις παραμέτρους των προηγούμενων στρωμάτων. Έτσι, κάθε ανανέωση των παραμέτρων που εκτελείται κατά τη διαδικασία του backpropagation με σκοπό τη μείωση της συνάρτησης κόστους, έχει ταυτόχρονα ως αποτέλεσμα τη μεταβολή της κατανομής εισόδου με βάση την οποία έγινε η ανανέωση των παραμέτρων. Αυτό με τη σειρά του είναι πιθανό να έχει οδηγήσει σε υποβέλτιστες παραμέτρους για το συγκεκριμένο mini-batch και άρα αύξηση της συνάρτησης κόστους. Παρόλα αυτά, η μείωση του Internal Covariate Shift από το Batch Normalization έχει αμφισβητηθεί. Σε ένα πιο πρόσφατο άρθρο [41], οι συγγραφείς δοκιμάζουν να εισάγουν τεχνητά Internal Covariate Shift μετά τη χρήση batch normalization και διαπιστώνουν ότι και πάλι η επίδοση είναι αρκετά καλύτερη από ένα απλό ΤΝΔ χωρίς batch normalization. Εκτιμούν τελικά ότι η θετική επίδραση του batch normalization πιθανώς να οφείλεται στην λείανση της μορφολογίας της συνάρτησης προς ελαχιστοποίηση, οδηγώντας σε μια πιο σταθερή συμπεριφορά των κλίσεων κατά την κατάβαση δυναμικού και επομένως τη δυνατότητα ταχύτερης εκμάθησης.

Ο βασικός λόγος που επιλέξαμε το ΤΝΔ ως μοντέλο ταξινόμησης, είναι ότι αποτελεί πολύ καλή επιλογή μοντέλου για την διαδικασία επιλογής χαρακτηριστικών. Όσον αφορά το πρόβλημα ταξινόμησης, δεδομένου ότι το σύνολο δεδομένων μας είναι σχετικά μικρό και εργαζόμαστε με tabular data, ένας απλός αλγόριθμος μηχανικής μάθησης, όπως ο Random Forest που χρησιμοποιείται στην αρχική προσέγγιση, πιθανόν να αποτελεί καλύτερη επιλογή από το ΤΝΔ. Το ΤΝΔ όμως, έχει την ικανότητα να μοντελοποιεί πιο περίπλοκες (μη γραμμικές) σχέσεις στα δεδομένα. Είναι πιο τελειοποιημένο σαν μοντέλο και πιο ευάλωτο στην οποιαδήποτε αλλαγή, επομένως

μπορεί να αναδείξει καλύτερα τις διαφορές επίδοσης, κάθε φορά που μεταβάλλεται το υποσύνολο χαρακτηριστικών που χρησιμοποιείται.

4 Υλοποίηση Συστήματος

4.1 Τεχνολογίες υλοποίησης

Η υλοποίηση τόσο για τη μεθοδολογία επιλογής χαρακτηριστικών, όσο και για το ΤΝΔ έγινε σε python. Η βιβλιοθήκη Scikit-learn χρησιμοποιήθηκε σε πολλά σημεία κατά την υλοποίηση. Αρχικά, αξιοποιήθηκε κατά την προετοιμασία των δεδομένων, για την κωδικοποίηση των labels των δεδομένων και τον χωρισμό του συνόλου δεδομένων σε training και validation set. Έπειτα, χρησιμοποιήθηκε για το cross-validation, στην διαδικασία του feature selection και τέλος, για την αξιολόγηση του μοντέλου. Όσο αφορά το μοντέλο, για την ανάπτυξη του ΤΝΔ χρησιμοποιήσαμε το Keras API, ενώ για υπόβαθρο του Keras χρησιμοποιήθηκε το Tensorflow.

Σχήμα 11: Scikit-learn

Η βιβλιοθήκη Scikit-learn είναι μια δημοφιλής βιβλιοθήκη ανοικτού κώδικα της python, που προσφέρει ένα ευέλικτο πλαίσιο για την εφαρμογή αλγορίθμων μηχανικής μάθησης. Έχει ευρεία απήχηση στους τομείς της επιστήμης δεδομένων και της μηχανικής μάθησης, λόγω ευκολίας χρήσης, πλούσιας λειτουργικότητας και καλής τεκμηρίωσης. Αποτελεί ένα ισχυρό εργαλείο για την ανάπτυξη, την εκπαίδευση και την αξιολόγηση αλγορίθμων μηχανικής μάθησης το οποίο βασίζεται πάνω σε άλλες γνωστές τεχνολογίες, όπως είναι οι βιβλιοθήκες NumPy, Pandas και Matplotlib. Η Scikit-learn προσφέρει λειτουργικότητα σε πλήθος διαφορετικών εφαρμογών, όπως προβλήματα πρόβλεψης, ταξινόμησης, συσταδοποίησης, model selection και data preprocessing.

Σχήμα 12: TensorFlow

Το TensorFlow είναι μια end-to-end πλατφόρμα ανοικτού κώδικα για την δημιουργία εφαρμογών μηχανικής μάθησης και βαθιάς μάθησης. Αναπτύχθηκε από την Google, αποτελεί ένα από τα πιο δημοφιλή εργαλεία στον τομέα της τεχνητής νοημοσύνης, και χρησιμοποιείται ευρέως τόσο στην ακαδημαϊκή έρευνα όσο και στη βιομηχανία. Πρόκειται για μια symbolic math βιβλιοθήκη που χρησιμοποιεί dataflow και διαφοροποιήσιμο προγραμματισμό για να εκτελέσει διάφορες λειτουργίες. Δέχεται τα δεδομένα σε μορφή πολυδιάστατων πινάκων που ονομάζονται tensors και λειτουργεί με βάση γράφους ροής δεδομένων.

Σχήμα 13: Keras

Το Keras είναι ένα υψηλού επιπέδου API για την ανάπτυξη νευρωνικών δικτύων που αναπτύχθηκε από την Google. Είναι γραμμένο σε python και χρησιμοποιείται για να διευκολύνει την υλοποίηση νευρωνικών δικτύων. Έχει ένα python frontend που το καθιστά πολύ εύχρηστο και δίνει την δυνατότητα επιλογής μεταξύ διαφορετικών back-end. Το TensorFlow έχει υιοθετήσει το Keras ως το επίσημο υψηλού επιπέδου API του. Το Keras ενσωματώνεται στο TensorFlow και μπορεί να χρησιμοποιηθεί για ταχεία εκτέλεση αλγορίθμων βαθιάς μάθησης, αφού παρέχει ενσωματωμένες δυνατότητες για όλους τους υπολογισμούς νευρωνικών δικτύων.

4.2 Επιλογή Χαρακτηριστικών

Ο αλγόριθμος επιλογής χαρακτηριστικών που υλοποιήσαμε αναπτύσσεται στις εξής πέντε φάσεις:

- Φάση 1η : Χρησιμοποιούμε πέντε μεθόδους επιλογής χαρακτηριστικών τύπου Filter, για να κατατάξουμε τα χαρακτηριστικά με βάση τη σημαντικότητά τους για την πρόβλεψη. Οι μέθοδοι αυτοί είναι:
    - Information Gain (Mutual Information)
    - Fisher's Score
    - Mean Absolute Difference (MAD)
    - Distance Correlation
    - Permutation Importance

- Φάση 2η : Με βάση τις κατατάξεις που δημιουργήσαμε για τα χαρακτηριστικά, επιλέγουμε ένα υποσύνολο αυτών που να καλύπτει όλους τους βασικούς πόρους. Συγκεκριμένα, επιλέγουμε τα δύο πιο «χρήσιμα» χαρακτηριστικά για κάθε πόρο, δημιουργώντας έτσι το αρχικό υποσύνολο χαρακτηριστικών.

- Φάση 3η : Αφού επιλέξουμε τα αρχικά χαρακτηριστικά, τα υπόλοιπα είναι προς το παρόν αναξιοποίητα. Από αυτά όμως ορισμένα μεταφέρουν πληροφορία με μεγάλη συνάφεια με αυτή ενός η περισσότερων από τα αρχικά χαρακτηριστικά. Αξιοποιώντας το Kendall Correlation Coefficient, εντοπίζουμε τα χαρακτηριστικά που έχουν απόλυτη τιμή του Correlation Coefficient μεγαλύτερη του 0.9 (δηλαδή πάνω από 90%) με κάποιο από τα αρχικά, και τα αφαιρούμε από το σύνολο με τα εναπομείναντα.

- Φάση 4η : Σε αυτήν την φάση, υλοποιείται μια παραλλαγή του αλγορίθμου Forward Selection χρησιμοποιώντας ένα ΤΝΔ ως μοντέλο ταξινόμησης. Ο αλγόριθμος μας ξεκινά με ένα σύνολο επιλεγμένων χαρακτηριστικών και ένα σύνολο αναξιοποίητων χαρακτηριστικών. Σε κάθε βήμα, εκπαιδεύει και αξιολογεί τόσα μοντέλα όσα τα αναξιοποίητα χαρακτηριστικά, χρησιμοποιώντας ως υποσύνολο χαρακτηριστικών τα επιλεγμένα χαρακτηριστικά με την προσθήκη ενός από τα αναξιοποίητα κάθε φορά. Η αξιολόγηση γίνεται με βάση το μέσο accuracy ενός ΤΝΔ, με την μέθοδο του 5-fold cross-validation. Για κάθε ένα από αυτά τα μοντέλα, αποθηκεύουμε το υποσύνολο των χαρακτηριστικών που χρησιμοποιήθηκε και το μέσο accuracy του ΤΝΔ. Έπειτα, το αναξιοποίητο χαρακτηριστικό που χρησιμοποιήθηκε στο μοντέλο που έδωσε το καλύτερο accuracy, προστίθεται στα επιλεγμένα χαρακτηριστικά και αφαιρείται από τα αναξιοποίητα.

- Φάση 5η : Στην προηγούμενη φάση εκπαιδεύσαμε ένα μοντέλο για καθένα από τα υποσύνολα χαρακτηριστικών που θεωρήσαμε ότι θα είναι πιο αποδοτικά. Κατατάσσουμε τα αποτελέσματα με βάση το accuracy που πέτυχε το μοντέλο για το κάθε υποσύνολο. Μία προσέγγιση θα ήταν να επιλέξουμε απευθείας το υποσύνολο που έδωσε το καλύτερο accuracy. Μια άλλη προσέγγιση είναι πως οι διαφορές στο accuracy ανάμεσα στα αποδοτικότερα υποσύνολα είναι πολύ μικρές, και αξίζει να γίνει αναλυτικότερη σύγκριση. Με βάση αυτήν την

προσέγγιση, θα μπορούσαμε να επιλέξουμε τα πιο αποδοτικά (π.χ. τα επικρατέστερα 5 ή 10), και να διεξάγουμε περαιτέρω ανάλυση της απόδοσής τους.

Όπως αναφέραμε η υλοποίηση έγινε σε python και χρησιμοποιείται εκτενώς η βιβλιοθήκη Scikit-learn σε όλη τη διαδικασία. Στην προετοιμασία των δεδομένων χρησιμοποιούμε τον *LabelEncoder* για να κωδικοποιήσουμε τα labels των δεδομένων μας καθώς και το *train_test_split* για να χωρίσουμε το training set σε training και validation set, με το validation set να είναι το 10% του αρχικού training set. Παρακάτω χρησιμοποιούμε το *Kfold* και το *cross_val_score* για την διαδικασία του cross-validation. Στη συνέχεια, αξιοποιούμε τα modules *mutual_info_classif* και *permutation_importance* στη διαδικασία του feature selection. Τέλος, για την αξιολόγηση του μοντέλου χρησιμοποιούμε τα modules *classification_report* και *confusion_matrix*. Οι μετρικές που μας απασχολούν είναι το accuracy, το precision, το recall, και το f1-score.

Εκτός των παραπάνω, στην προετοιμασία των δεδομένων χρησιμοποιούμε το *SMOTE* από την βιβλιοθήκη *imblearn* για να κάνουμε oversampling στο training set. Επιπλέον, στο πλαίσιο της παραπάνω διαδικασίας χρησιμοποιούμε και το *KerasClassifier* από τους wrappers της βιβλιοθήκης του Keras. Πρόκειται για ένα wrapper που προσφέρει στο μοντέλο που ορίσαμε με το Keras συμβατότητα με το API του Scikit-learn. Ο Wrapper χρησιμοποιήθηκε αφού πρώτα ορίστηκε το μοντέλο με το Keras και το TensorFlow, ώστε να μπορούμε να αξιοποιήσουμε τα modules της βιβλιοθήκης για cross-validation κατά το feature selection.

4.3 Υλοποίηση ΤΝΔ

Το ΤΝΔ που αναπτύσσουμε στην παρούσα διπλωματική εργασία είναι ένα μοντέλο σχεδιασμένο για την ταξινόμηση εφαρμογών με βάση τον τρόπο που χρησιμοποιούν το υποκείμενο υλικό. Το μοντέλο μας, πέραν της ταξινόμησης, δημιουργήθηκε για να αξιοποιηθεί και στην αξιολόγηση διαφορετικών υποσυνόλων χαρακτηριστικών κατά τη διαδικασία επιλογής χαρακτηριστικών. Έτσι, με βάση την απόδοση του μοντέλου μας αξιολογούμε τόσο τα διαφορετικά υποσύνολα, όσο και την προτεινόμενη μέθοδο αυτή καθαυτή.

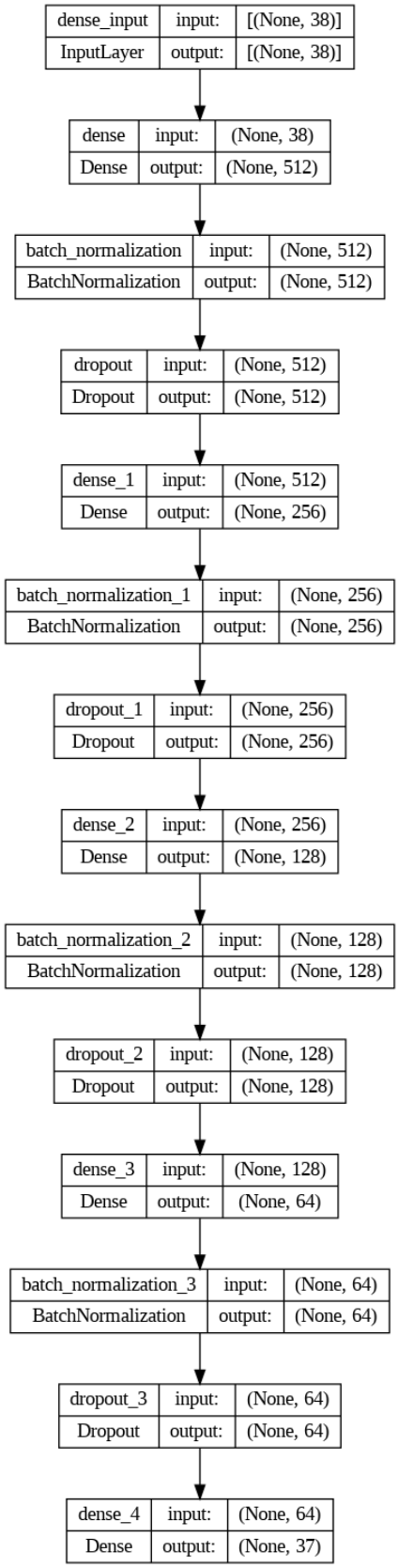
Η αρχιτεκτονική του μοντέλου μας αποτελείται από πέντε επίπεδα (layers), δηλαδή πρόκειται για μια αρχιτεκτονική βαθιάς μάθησης. Συγκεκριμένα έχουμε τα ακόλουθα layers:

- Input Layer: Το πρώτο επίπεδο του ΤΝΔ δέχεται στην είσοδο το σύνολο δεδομένων μας, οπότε η διάσταση εισόδου ισούται με τον αριθμό των χαρακτηριστικών που έχουμε επιλέξει. Έχει 512 νευρώνες και χρησιμοποιεί την συνάρτηση ενεργοποίησης ReLU.

- Hidden Layers: Το μοντέλο μας περιέχει τρία κρυφά επίπεδα (hidden layers). Καθένα από αυτά χρησιμοποιεί την ReLU ως συνάρτηση ενεργοποίησης και έχουν κατά αντιστοιχία 256 νευρώνες, 128 νευρώνες και 64 νευρώνες. Οι λόγοι που χρησιμοποιείται η συνάρτηση ReLU τόσο στα κρυφά όσο και στο επίπεδο εισόδου, είναι τα πολλά πλεονεκτήματα που προσφέρει συγκριτικά με άλλες συναρτήσεις ενεργοποίησης. (βλ. Παράγραφο 3.2.2.4)

- Output Layer: Το τελευταίο επίπεδο του ΤΝΔ αποτελείται από τόσους νευρώνες όσες και οι κλάσεις ταξινόμησης και χρησιμοποιεί την συνάρτηση Softmax ως συνάρτηση ενεργοποίησης. Η Softmax εδώ δίνει μια κατανομή πιθανότητας ώστε κάθε νευρώνας, που εκπροσωπεί μια κλάση, να λαμβάνει μία τιμή στο διάστημα [0,1]. Η τιμή του κάθε νευρώνα δηλώνει την πιθανότητα να ανήκει το δείγμα στην αντίστοιχη κλάση και επομένως το άθροισμα όλων των τιμών ισούται κάθε φορά με 1.

Στο Σχήμα 14 βλέπουμε σχηματικά την αρχιτεκτονική του μοντέλου μας. Όπως διαπιστώνουμε, η διάσταση εισόδου είναι 38 διότι δεν έχει εφαρμοστεί προς το παρόν η επιλογή χαρακτηριστικών, επομένως έχουμε τα 38 αρχικά χαρακτηριστικά του συνόλου δεδομένων μας. Επίσης, παρατηρούμε πως η διάσταση εξόδου είναι 37 και όχι 62 που είναι το πλήθος των benchmark που χρησιμοποιήθηκαν κατά τη διαδικασία δημιουργίας και συλλογής των δεδομένων μας. Αυτό συμβαίνει διότι έχει προηγηθεί μια ομογενοποίηση ορισμένων benchmark για να οδηγηθούμε από τις 62 στις 37 κλάσεις. Ο λόγος που προβήκαμε στην ομογενοποίηση αυτή, καθώς και ο τρόπος με τον οποίο έγινε εξηγούνται αναλυτικά στο επόμενο κεφάλαιο. Επιπλέον, παρατηρούμε πως έπειτα από το Input Layer αλλά και μετά από καθένα από τα Hidden Layers εφαρμόζεται Batch Normalization για να βελτιωθεί η σταθερότητα του μοντέλου μας κατά την εκπαίδευση. Ακόμα, ένα ενδιάμεσο Dropout layer μεσολαβεί ανάμεσα σε κάθε ζεύγος layers. Τα dropout layers έχουν rate 0.2. Δηλαδή, σε κάθε επίπεδο, σε κάθε εποχή, το 20% των νευρώνων του επιπέδου απενεργοποιούνται. Σκοπός των dropout layers είναι να αποφευχθεί το overfitting. Για να καθοριστεί ο αριθμός των επιπέδων και ο αριθμός των νευρώνων σε κάθε επίπεδο, αλλά και για τον προσδιορισμό άλλων τιμών όπως το dropout rate και το learning rate του optimizer, διεξήχθη διαδικασία ρύθμισης υπερπαραμέτρων, όπου αξιοποιήθηκαν στρατηγικές όπως το Grid Search και το Random Search.



Σχήμα 14: Αρχιτεκτονική ΤΝΔ

Κατά την διαδικασία εκπαίδευσης του μοντέλου χρησιμοποιείται ως συνάρτηση κόστους (loss function) το Sparse Categorical Cross-entropy. Η συνάρτηση αυτή είναι κατάλληλη για μοντέλα που εκτελούν ταξινόμηση σε πολλές κλάσεις (multi-class classification). Για την βελτιστοποίηση των βαρών του ΤΝΔ, επιλέχθηκε ο optimizer Adam, και ο ρυθμός εκπαίδευσης (learning rate) ορίστηκε σε 0.001 όπως προέκυψε από τη διαδικασία ρύθμισης υπερπαραμέτρων. Ακόμη, χρησιμοποιούμε πρόωρο τερματισμό (early stopping) για την αποτροπή της υπερεκπαίδευσης. Το early stopping βασίζεται στην παρακολούθηση του validation loss, δηλαδή της τιμής που λαμβάνει σε κάθε εποχή η συνάρτηση κόστους στο validation set. Στόχος είναι η ελαχιστοποίηση της συνάρτησης κόστους και έχουμε ορίσει την παράμετρο patience να ισούται με 50 εποχές. Επομένως, εάν για 50 συνεχόμενες εποχές δεν έχει παρατηρηθεί μείωση του validation loss, τότε ενεργοποιείται ο πρόωρος τερματισμός. Όταν συμβεί αυτό, το μοντέλο μας επαναφέρει τα βάρη με τα οποία είχε πετύχει την καλύτερη απόδοση.

Όπως βλέπουμε και στο Σχήμα 15 το μοντέλο μας έχει συνολικά 198693 παραμέτρους, εκ των οποίων οι 196773 είναι διαμορφώσιμες κατά τη διαδικασία της εκπαίδευσης, ενώ οι υπόλοιπες 1920 δεν είναι.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               19968

 batch_normalization         (None, 512)               2048
(BatchNormalization)

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 batch_normalization_1       (None, 256)               1024
(BatchNormalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 batch_normalization_2       (None, 128)               512
 (BatchNormalization)

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 64)                8256
```

```
 batch_normalization_3       (None, 64)                256
 (BatchNormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_4 (Dense)             (None, 37)                2405

=================================================================
Total params: 198693 (776.14 KB)
Trainable params: 196773 (768.64 KB)
Non-trainable params: 1920 (7.50 KB)
_________________________________________________________________
```

Πίνακας 8: Σύνοψη ΤΝΔ

5 *Πειραματική Αξιολόγηση*

Η πειραματική διαδικασία που διεξήχθη αποτελείται από τρία στάδια. Στο πρώτο στάδιο μελετάμε την απόδοση του ταξινομητή στο σύνολο δεδομένων μας και διερευνώνται η πειραματική προσέγγιση και η διαμόρφωση των δεδομένων με στόχο να βελτιώσουμε την ακρίβεια των προβλέψεων. Στο δεύτερο στάδιο εκτελούμε τον αλγόριθμο επιλογής χαρακτηριστικών και εντοπίζουμε τα υποσύνολα χαρακτηριστικών που δίνουν την μεγαλύτερη ακρίβεια. Τέλος, στο τρίτο στάδιο διεξάγεται περαιτέρω ανάλυση και σύγκριση της απόδοσης του μοντέλου για τα πέντε πιο αποδοτικά υποσύνολα χαρακτηριστικών.

5.1 Μορφοποίηση συνόλου δεδομένων

Αφού υλοποιήσαμε το μοντέλο ταξινόμησης προχωρήσαμε στην διαδικασία της εκπαίδευσης με το training set και στην αξιολόγηση της απόδοσης του στο testing set. Το training set περιέχει περίπου 1050 εγγραφές από εκτελέσεις benchmark που έγιναν σε τρία διαφορετικά υπολογιστικά συστήματα, και το testing set περίπου 300 εγγραφές από δύο άλλα μηχανήματα, όπως περιγράφεται στην παράγραφο 3.1. Ορίστηκε το validation set ως το 10% των εγγραφών του training set και θέσαμε των αριθμό των εποχών να ισούται με 1000, με δυνατότητα όμως και πρόωρου τερματισμού, και το batch size να είναι 128. Μόλις ολοκληρώνεται η εκπαίδευση του μοντέλου, ελέγχουμε το accuracy του ΤΝΔ στο validation set και στο testing set, η οποία προκύπτει:

Validation set	Testing set
73,58%	47,54%

Πίνακας 9: Baseline Model Accuracy

Όπως βλέπουμε σε αυτήν την πρώτη εκδοχή του μοντέλου μας τα αποτελέσματα παρουσιάζουν μεγάλα περιθώρια βελτίωσης. Παρατηρούμε μεγάλη απόκλιση ανάμεσα στο validation accuracy και στο testing accuracy, γεγονός αναμενόμενο αφού τα δεδομένα που βρίσκονται στο training και το validation set έχουν παραχθεί από εκτελέσεις σε διαφορετικά μηχανήματα από αυτά του testing set. Για να μπορέσουμε να κατανοήσουμε καλύτερα πως λειτουργεί το ΤΝΔ με τα δεδομένα μας και που προκύπτουν λανθασμένες προβλέψεις εξάγουμε το confusion matrix και το classification report για τις προβλέψεις του μοντέλου στο testing set.

Το classification report είναι ένα από τα πιο χρήσιμα εργαλεία αξιολόγησης ενός μοντέλου ταξινόμησης. Παρέχει συνοπτικές πληροφορίες σχετικά με την απόδοση του μοντέλου, συνήθως σε μορφή πίνακα, και τα μεγέθη που περιέχει είναι τα precision, recall, f1-score και support. Τα μεγέθη αυτά αναγράφονται για την κάθε κλάση ξεχωριστά, αλλά περιέχει και μεσοκαθορισμένες τιμές όπως το macro average και το weighted average. Αναλυτικότερα, το precision (ακρίβεια) εκφράζει το ποσοστό των δειγμάτων που ανήκουν όντως σε μία κλάση, επί του συνόλου των δειγμάτων που το μοντέλο μας ταξινόμησε σε αυτήν. Το recall (ευαισθησία) είναι το ποσοστό των δειγμάτων που ταξινομήθηκαν σε μία κλάση, επί του συνόλου των δειγμάτων που ανήκουν πραγματικά σε αυτήν. Το f1-score είναι μια μετρική η οποία συνδυάζει το precision και το recall και υπολογίζεται ως:

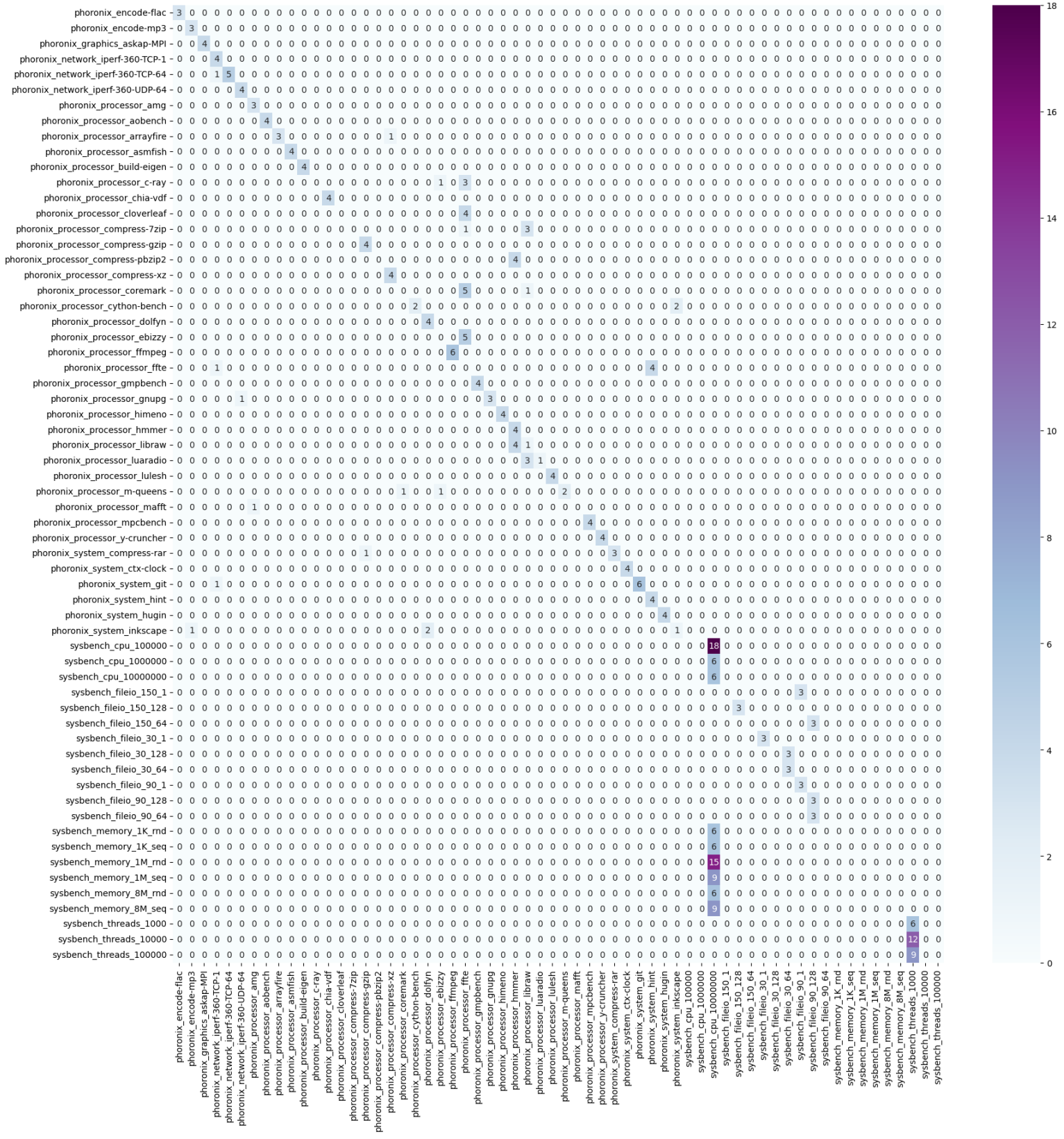
$$f1 = \frac{2 \cdot (precision \cdot recall)}{precision + recall}$$

Τέλος το support δηλώνει το πλήθος των πραγματικών δειγμάτων που ανήκουν στη συγκεκριμένη κλάση. Το classification report από την αξιολόγηση της πρώτης εκδοχής του μοντέλου μας στο testing set προκύπτει ως εξής:

Classification Report				
classes	precision	recall	f1-score	support
phoronix_encode-flac	1.00	1.00	1.00	3
phoronix_encode-mp3	0.75	1.00	0.86	3
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf-360-TCP-1	0.57	1.00	0.73	4
phoronix_network_iperf-360-TCP-64	1.00	0.83	0.91	6
phoronix_network_iperf-360-UDP-64	0.80	1.00	0.89	4
phoronix_processor_amg	0.75	1.00	0.86	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	1.00	0.75	0.86	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	0.00	0.00	0.00	4
phoronix_processor_compress-7zip	0.00	0.00	0.00	4
phoronix_processor_compress-gzip	0.80	1.00	0.89	4
phoronix_processor_compress-pbzip2	0.00	0.00	0.00	4
phoronix_processor_compress-xz	0.80	1.00	0.89	4
phoronix_processor_coremark	0.00	0.00	0.00	6
phoronix_processor_cython-bench	1.00	0.50	0.67	4
phoronix_processor_dolfyn	0.67	1.00	0.80	4
phoronix_processor_ebizzy	0.00	0.00	0.00	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.00	0.00	0.00	5
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.75	0.86	4
phoronix_processor_himeno	1.00	1.00	1.00	4
phoronix_processor_hmmer	0.33	1.00	0.50	4
phoronix_processor_libraw	0.12	0.20	0.15	5
phoronix_processor_luaradio	1.00	0.25	0.40	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	1.00	0.50	0.67	4

phoronix_processor_mafft	0.00	0.00	0.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_compress-rar	1.00	0.75	0.86	4
phoronix_system_ctx-clock	1.00	1.00	1.00	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hint	0.50	1.00	0.67	4
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	0.33	0.25	0.29	4
sysbench_cpu_100000	0.00	0.00	0.00	18
sysbench_cpu_1000000	0.00	0.00	0.00	6
sysbench_cpu_10000000	0.07	1.00	0.14	6
sysbench_fileio_150_1	0.00	0.00	0.00	3
sysbench_fileio_150_128	1.00	1.00	1.00	3
sysbench_fileio_150_64	0.00	0.00	0.00	3
sysbench_fileio_30_1	1.00	1.00	1.00	3
sysbench_fileio_30_128	0.00	0.00	0.00	3
sysbench_fileio_30_64	0.50	1.00	0.67	3
sysbench_fileio_90_1	0.50	1.00	0.67	3
sysbench_fileio_90_128	0.33	1.00	0.50	3
sysbench_fileio_90_64	0.00	0.00	0.00	3
sysbench_memory_1K_rnd	0.00	0.00	0.00	6
sysbench_memory_1K_seq	0.00	0.00	0.00	6
sysbench_memory_1M_rnd	0.00	0.00	0.00	15
sysbench_memory_1M_seq	0.00	0.00	0.00	9
sysbench_memory_8M_rnd	0.00	0.00	0.00	6
sysbench_memory_8M_seq	0.00	0.00	0.00	9
sysbench_threads_1000	0.22	1.00	0.36	6
sysbench_threads_10000	0.00	0.00	0.00	12
sysbench_threads_100000	0.00	0.00	0.00	9
macro avg	0.52	0.57	0.52	305
weighted avg	0.43	0.48	0.42	305

Πίνακας 10: Baseline Model Classification Report



Γράφημα 2: Baseline Model Confusion Matrix

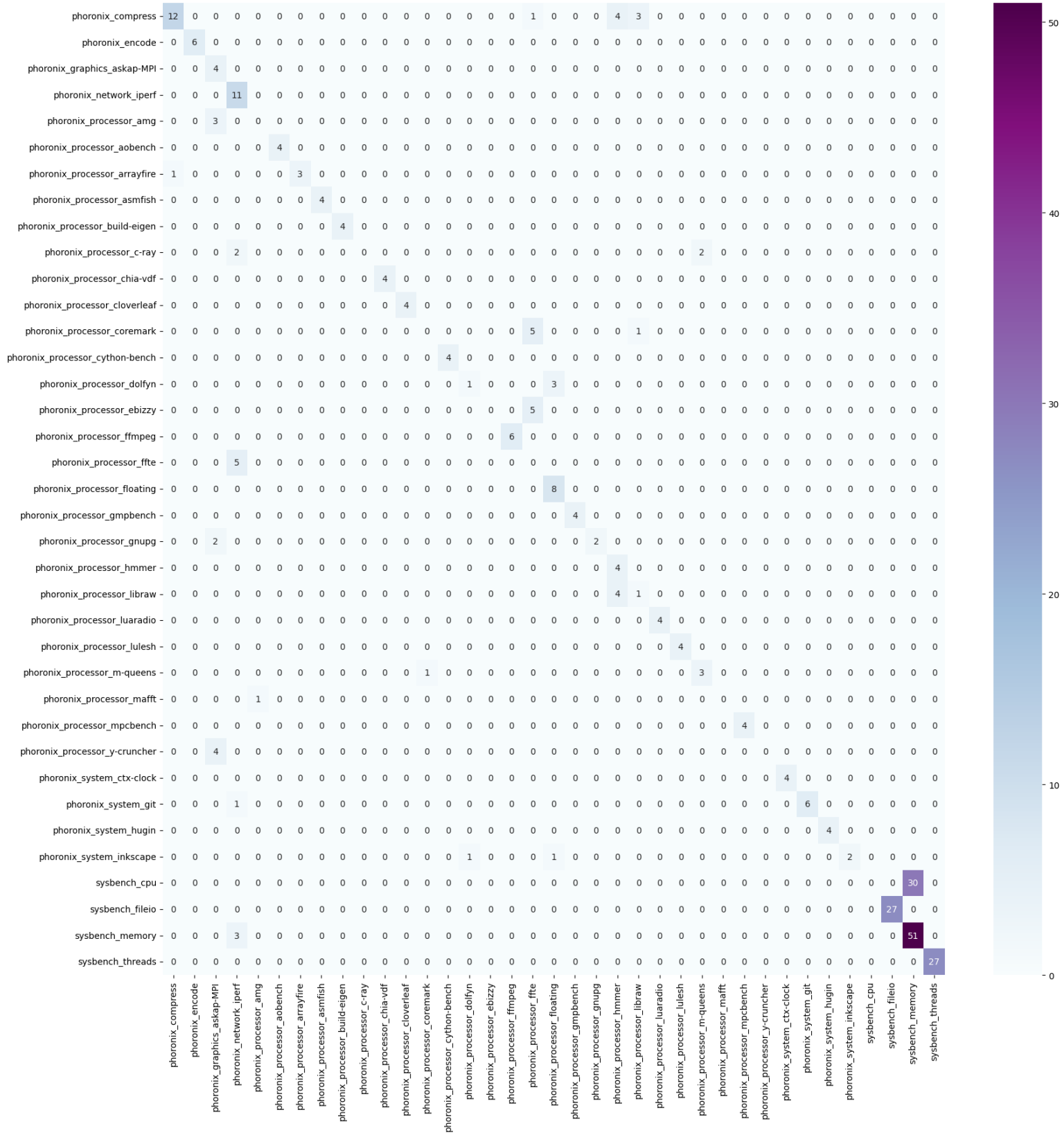
Το confusion matrix από την αξιολόγηση της πρώτης εκδοχής του μοντέλου μας στο testing set φαίνεται στο Σχήμα 15. Το confusion matrix είναι ένας δισδιάστατος πίνακας με γραμμές και στήλες τις κλάσεις ταξινόμησης. Απεικονίζει τον τρόπο με τον οποίο το μοντέλο μας ταξινομεί τα δεδομένα στις διάφορες κατηγορίες και παρέχει μια επισκόπηση των αποτελεσμάτων. Κάθε στοιχείο του πίνακα είναι ένας φυσικός αριθμός που αντιπροσωπεύει το πλήθος των δειγμάτων που ανήκουν στην κλάση της αντίστοιχης γραμμής και ταξινομήθηκαν από το μοντέλο μας στην κλάση της αντίστοιχης στήλης. Γίνεται άμεσα αντιληπτό επομένως, πως από έναν αλάνθαστο ταξινομητή θα προέκυπτε ένα confusion matrix του οποίου όλα τα στοιχεία που δεν ανήκουν στην κύρια διαγώνια θα είναι ίσα με μηδέν.

Από το classification report και το confusion matrix που προέκυψαν παρατηρούμε πως το μοντέλο μας ξεχωρίζει αρκετά καλά τις περισσότερες κλάσεις που αντιστοιχούν σε benchmarks της phoronix. Αντιθέτως, στις κλάσεις που αντιπροσωπεύονται από τα benchmarks της sysbench υπάρχει αρκετή σύγχυση. Αφ' ενός βλέπουμε πως το μοντέλο δεν ξεχωρίζει τις κλάσεις που ανήκουν στο ίδιο benchmark αλλά με διαφορετικό workload και αφ' εταίρου «μπερδεύει» τις κλάσεις του sysbench_cpu με αυτές του sysbench_memory. Αυτό δεν είναι παράξενο, καθώς τα benchmarks της sysbench είναι αρκετά απλά και είναι εύκολο τα cpu και τα memory intensive να παράγουν παρόμοια αποτυπώματα. Όσον αφορά τις κλάσεις που έχουν ίδιο benchmark με διαφορετικό workload, μελετώντας τα προφίλ που παράγουν διαπιστώνουμε ότι είναι σχεδόν πανομοιότυπα, επομένως είναι λογικό το ΤΝΔ να μην τα διαχωρίζει.

Προσπαθώντας να βελτιώσουμε τα παραπάνω αποτελέσματα προβήκαμε στην ομογενοποίηση των datasets μας ως εξής. Για όλα τα προφίλ που παρήχθησαν από τη συλλογή sysbench, δώσαμε το ίδιο όνομα στα προφίλ που προέρχονται από τα ίδια benchmarks με διαφορετικό φόρτο εργασίας. Θεωρήσαμε δηλαδή ότι ένα προφίλ χαρακτηρίζει τη φύση μιας εφαρμογής και δεν διαφοροποιείται από την παραμετροποίηση. Για τα προφίλ που προέρχονται από τη σουίτα Phoronix με την ίδια λογική ομογενοποιήσαμε εφαρμογές που επιτελούν παρόμοιες λειτουργίες, όπως για παράδειγμα συμπίεση αρχείου rar και συμπίεση αρχείου zip ή κωδικοποίηση αρχείου mp3 και κωδικοποίηση αρχείου flac. Έτσι προκύπτει ένα νέο σύνολο δεδομένων με μόνο 37 διαφορετικές κλάσεις. Σε αυτό το (Homogenized Dataset) HD Model το μοντέλο μας έχει αισθητά καλύτερη απόδοση:

Validation set	Testing set
93,40%	72,79%

Πίνακας 11: HD Model Accuracy



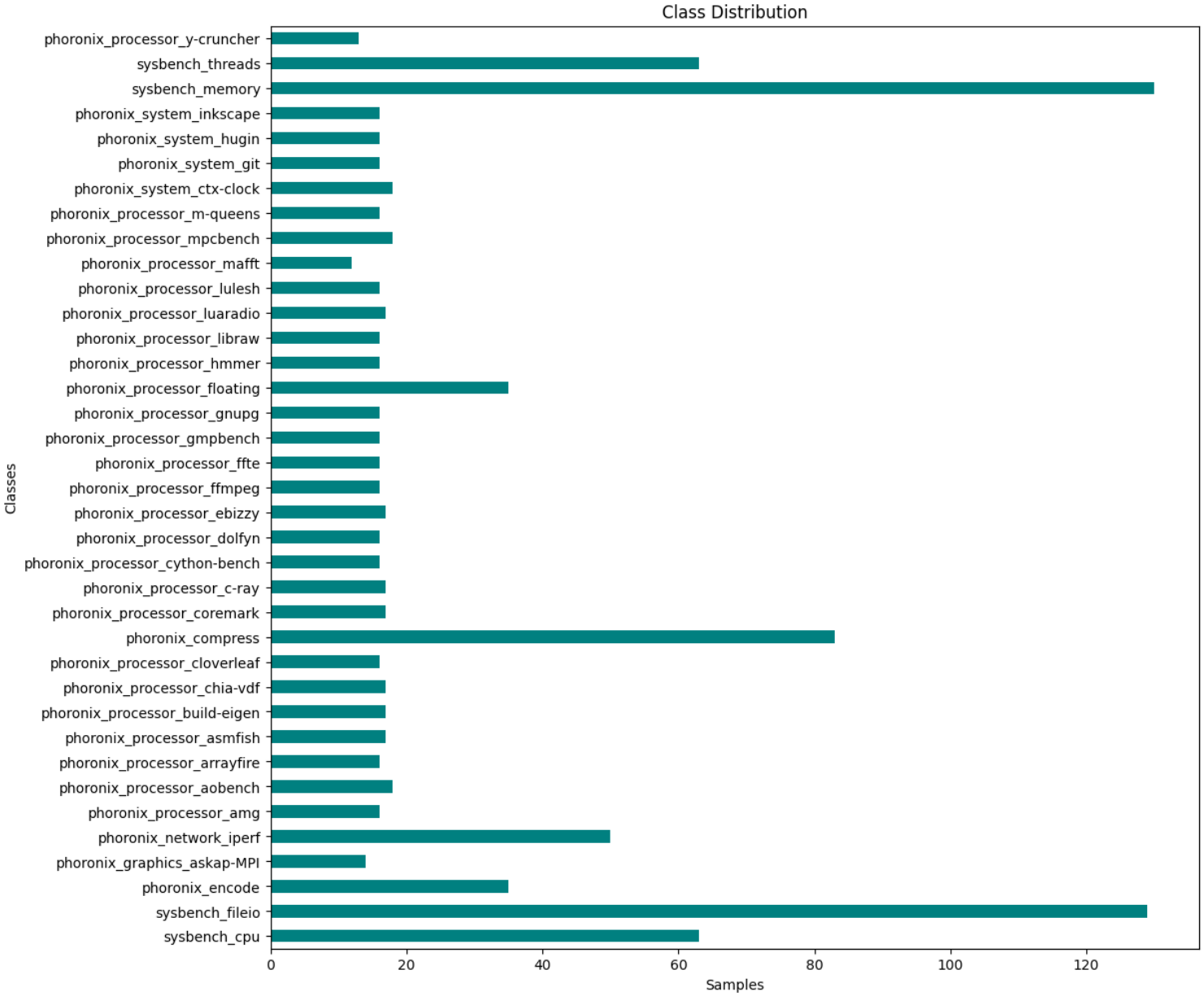
Γράφημα 3: HD Model Confusion Matrix

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	0.92	0.60	0.73	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	0.31	1.00	0.47	4
phoronix_network_iperf	0.50	1.00	0.67	11
phoronix_processor_amg	0.00	0.00	0.00	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	1.00	0.75	0.86	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.00	0.00	0.00	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.50	0.25	0.33	4
phoronix_processor_ebizzy	0.00	0.00	0.00	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.00	0.00	0.00	5
phoronix_processor_floating	0.67	1.00	0.80	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.50	0.67	4
phoronix_processor_hmmer	0.33	1.00	0.50	4
phoronix_processor_libraw	0.20	0.20	0.20	5
phoronix_processor_luaradio	1.00	1.00	1.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4

phoronix_processor_m-queens	0.60	0.75	0.67	4
phoronix_processor_mafft	0.00	0.00	0.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	0.00	0.00	0.00	4
phoronix_system_ctx-clock	1.00	1.00	1.00	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.50	0.67	4
sysbench_cpu	0.00	0.00	0.00	30
sysbench_fileio	1.00	1.00	1.00	27
sysbench_memory	0.63	0.94	0.76	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.67	0.69	0.65	305
weighted avg	0.67	0.73	0.68	305

Πίνακας 12: HD Model Classification Report

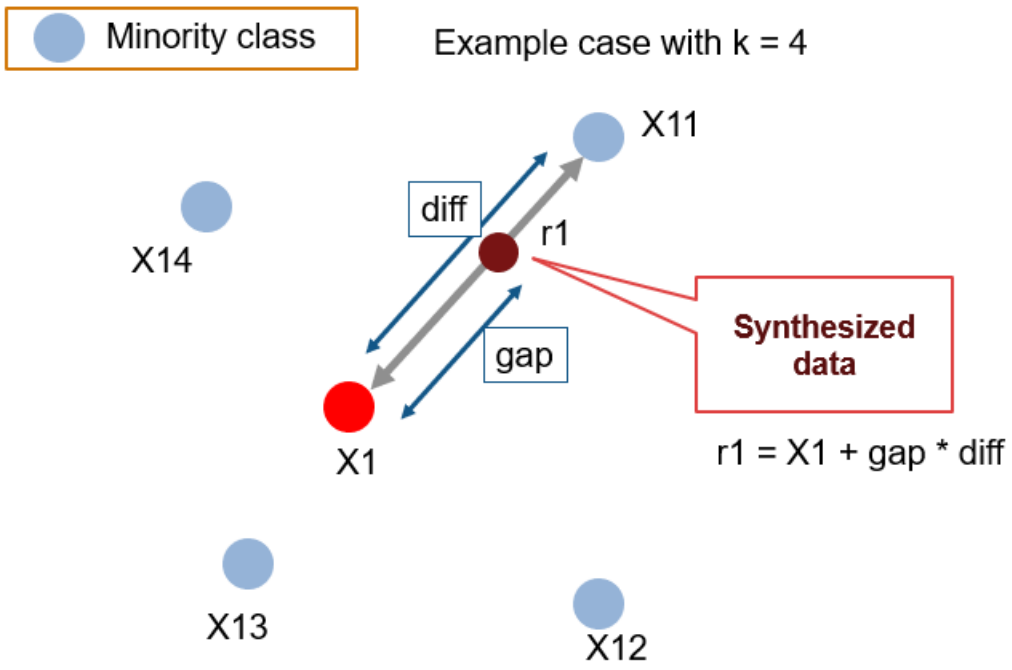
Η ομογενοποίηση όμως των δεδομένων μας δημιούργησε μια νέα πρόκληση. Είχαμε ήδη αρκετές εκτελέσεις από τα benchmarks της sysbench τόσο ανά benchmark όσο και ανά workload. Με την ομαδοποίηση αυτών, στο νέο σύνολο δεδομένων, οι κλάσεις των benchmarks της sysbench καταλήγουν να έχουν πολύ περισσότερα δείγματα από τις υπόλοιπες κλάσεις. Έχουμε λοιπόν ένα μη ισορροπημένο σύνολο δεδομένων, αφού ορισμένες κλάσεις εκπροσωπούνται πολύ περισσότερο από άλλες. Αυτό μπορεί να οδηγήσει το ΤΝΔ σε μία έκφανση της υπερεκπαίδευσης, αφού βάση στατιστικής θα αποδίδει καλύτερα αν επιλέγει συχνότερα τις πολυπληθέστερες κλάσεις. Με την πλειοψηφία των κλάσεων να έχει λιγότερα από 20 δείγματα, και την πιο πολυπληθή κλάση να έχει 130.



Γράφημα 4: Κατανομή δειγμάτων - κλάσεων στο ομογενοποιημένο dataset

Για να αντιμετωπίσουμε το πρόβλημα του μη ισορροπημένου dataset χρησιμοποιούμε το SMOTE (Synthetic Minority Oversampling Technique) για να κάνουμε oversampling στα δεδομένα μας. Το SMOTE είναι ειδικά σχεδιασμένο για την διαχείριση μη ισορροπημένων συνόλων δεδομένων μέσω της δημιουργίας συνθετικών δειγμάτων για τις κλάσεις με λιγότερα δείγματα (oversampling). Για να το πετύχει αυτό, αρχικά επιλέγει τυχαία ένα δείγμα από κάποια από τις κλάσεις με λιγότερα δείγματα. Έπειτα βρίσκει τους k Nearest Neighbors και επιλέγει τυχαία έναν

από αυτούς. Τέλος, σχεδιάζει την ευθεία που ενώνει το επιλεγμένο δείγμα και τον επιλεγμένο γείτονα και δημιουργεί ένα συνθετικό δείγμα σε ένα τυχαίο σημείο της ευθείας, ανάμεσα στα δύο σημεία. Επί της ουσίας υπολογίζει τη διαφορά των feature vectors του δείγματος και του γείτονα και το πολλαπλασιάζει με ένα τυχαίο αριθμό στο διάστημα (0,1].



Σχήμα 15: Synthesized data creation with SMOTE

Έτσι, προέκυψε η τελική μορφή του συνόλου δεδομένων με το οποίο εργαστήκαμε. Όλες οι κλάσεις εκπροσωπούνται εξίσου σε αυτό και ο συνολικός αριθμός δειγμάτων ανέρχεται περίπου στα 4800. Εκπαιδεύουμε και αξιολογούμε ξανά το ΤΝΔ με το νέο μας dataset, το οποίο είναι αρκετά μεγαλύτερο, και για τον λόγο αυτό αυξήσαμε και το batch size σε 512. Αφού εκπαιδεύσουμε το μοντέλο και το αξιολογήσουμε με το validation και το testing set εξάγουμε για το (Homogenized – Oversampled Dataset) HOD Model τις ίδιες πληροφορίες όπως και στο Baseline Model και το HD Model.

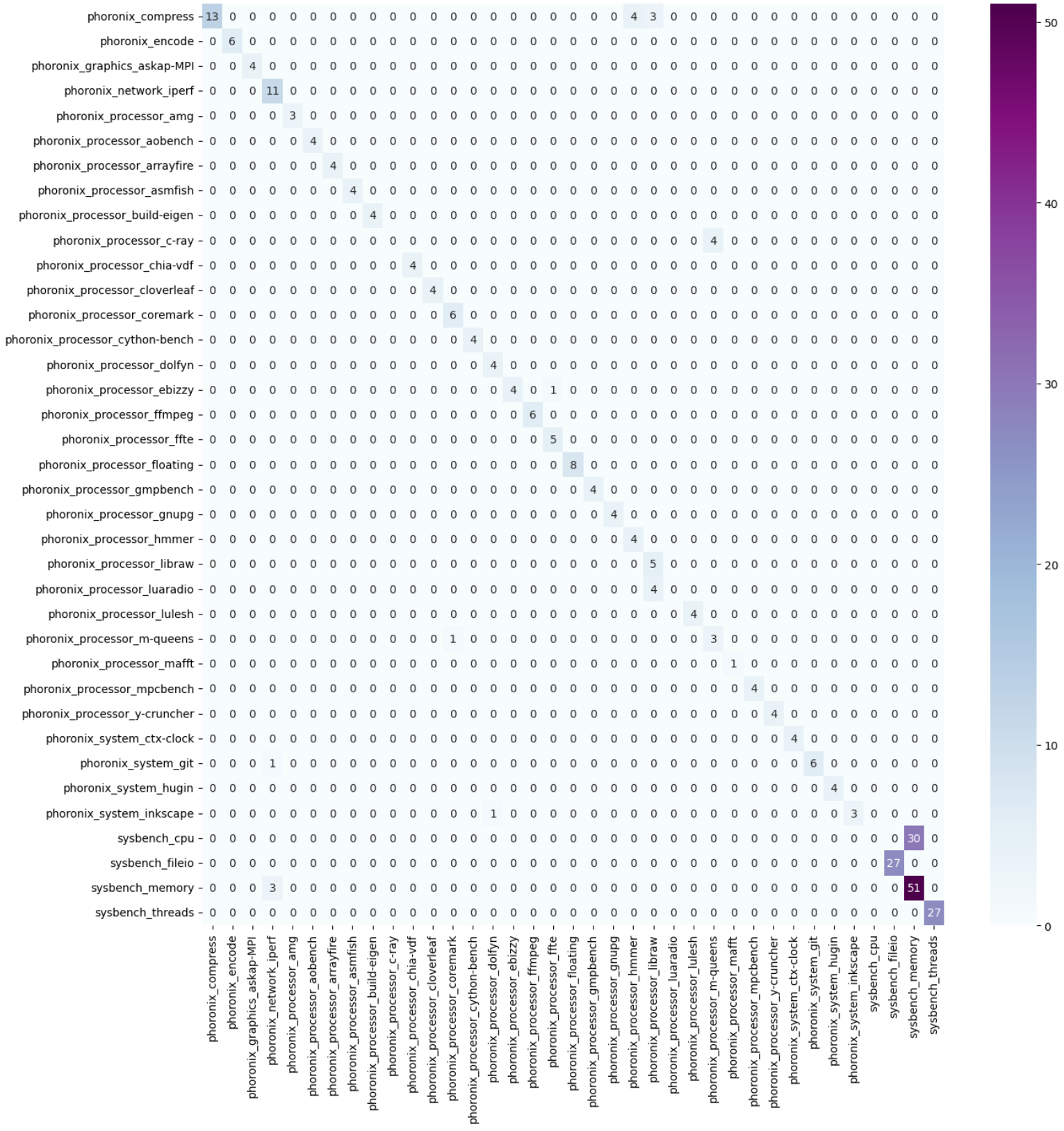
Validation set	Testing set
96,47%	82,95%

Πίνακας 13: HOD Model Accuracy

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.65	0.79	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.73	1.00	0.85	11
phoronix_processor_amg	1.00	1.00	1.00	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	1.00	1.00	1.00	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.86	1.00	0.92	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.80	1.00	0.89	4
phoronix_processor_ebizzy	1.00	0.80	0.89	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.83	1.00	0.91	5
phoronix_processor_floating	1.00	1.00	1.00	8

phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	1.00	1.00	4
phoronix_processor_hmmer	0.50	1.00	0.67	4
phoronix_processor_libraw	0.42	1.00	0.59	5
phoronix_processor_luaradio	0.00	0.00	0.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	1.00	1.00	1.00	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.00	0.00	0.00	30
sysbench_fileio	1.00	1.00	1.00	27
sysbench_memory	0.63	0.94	0.76	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.84	0.89	0.85	305
weighted avg	0.77	0.83	0.79	305

Πίνακας 14: HOD Model Classification Report



Γράφημα 5: HOD Model Confusion Matrix

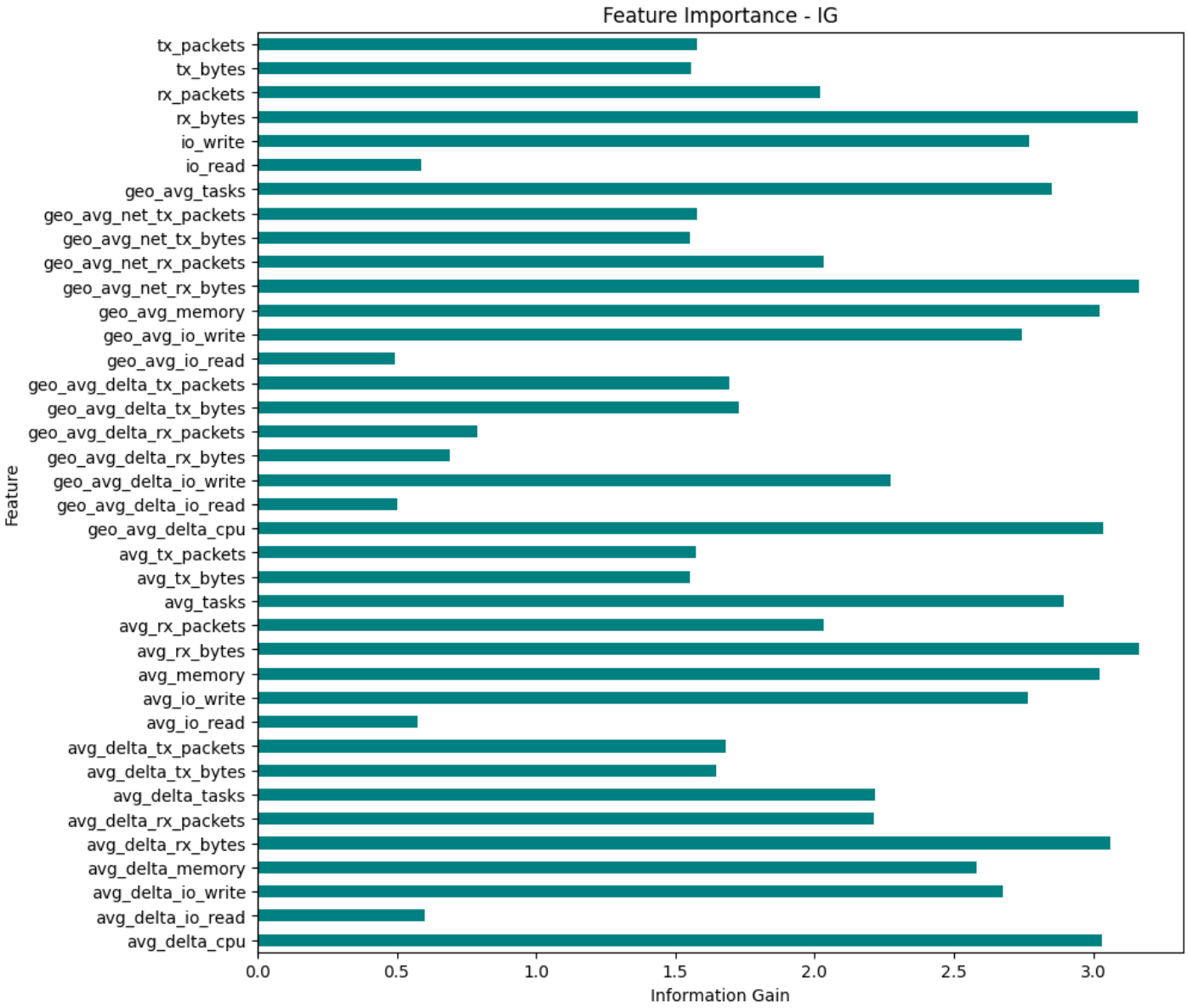
Βλέπουμε πως με την ομογενοποίηση κάποιων κλάσεων πετυχαίνουμε μεγάλη αύξηση στην ακρίβεια των προβλέψεων, όμως ταξινομούνται λάθος αρκετά δείγματα από τις λιγότερο εκπροσωπούμενες κλάσεις. Μετά το oversampling όμως τα περισσότερα από αυτά τα δείγματα ταξινομούνται πλέον σωστά. Όπως μπορούμε να παρατηρήσουμε και από το confusion matrix του HOD Model, τα περισσότερα στοιχεία εκτός της κύριας διαγώνιου είναι μηδενικά. Παρόλα αυτά, υπάρχει ακόμα δυσκολία στο διαχωρισμό των κλάσεων sysbench_memory και sysbench_cpu, οι οποίες έχουν και μεγάλο αριθμό δειγμάτων στο testing set και επομένως επηρεάζουν σε μεγάλο βαθμό τις συνολικές μετρικές απόδοσης του μοντέλου. Για να αντιμετωπίσουμε και αυτό το πρόβλημα επιχειρούμε τη βελτιστοποίηση των προφίλ των εφαρμογών μέσω επιλογής χαρακτηριστικών.

5.2 Επιλογή Χαρακτηριστικών

Αφού διαμορφώσαμε κατάλληλα τις κλάσεις και τα δείγματα του συνόλου δεδομένων μας, ώστε να βελτιώσουμε την απόδοση του ΤΝΔ, ξεκινάμε την εκτέλεση της μεθοδολογίας επιλογής χαρακτηριστικών. Στόχος είναι η εύρεση του βέλτιστου υποσυνόλου χαρακτηριστικών και η διαδικασία εκτελέστηκε όπως έχει περιγραφεί στην παράγραφο 4.2. Ξεκινήσαμε λοιπόν με την 1^{η} φάση, όπου δημιουργήσαμε πέντε κατατάξεις των χαρακτηριστικών, με βάση πέντε ευρέως χρησιμοποιούμενα κριτήρια.

- Information Gain / Mutual Information

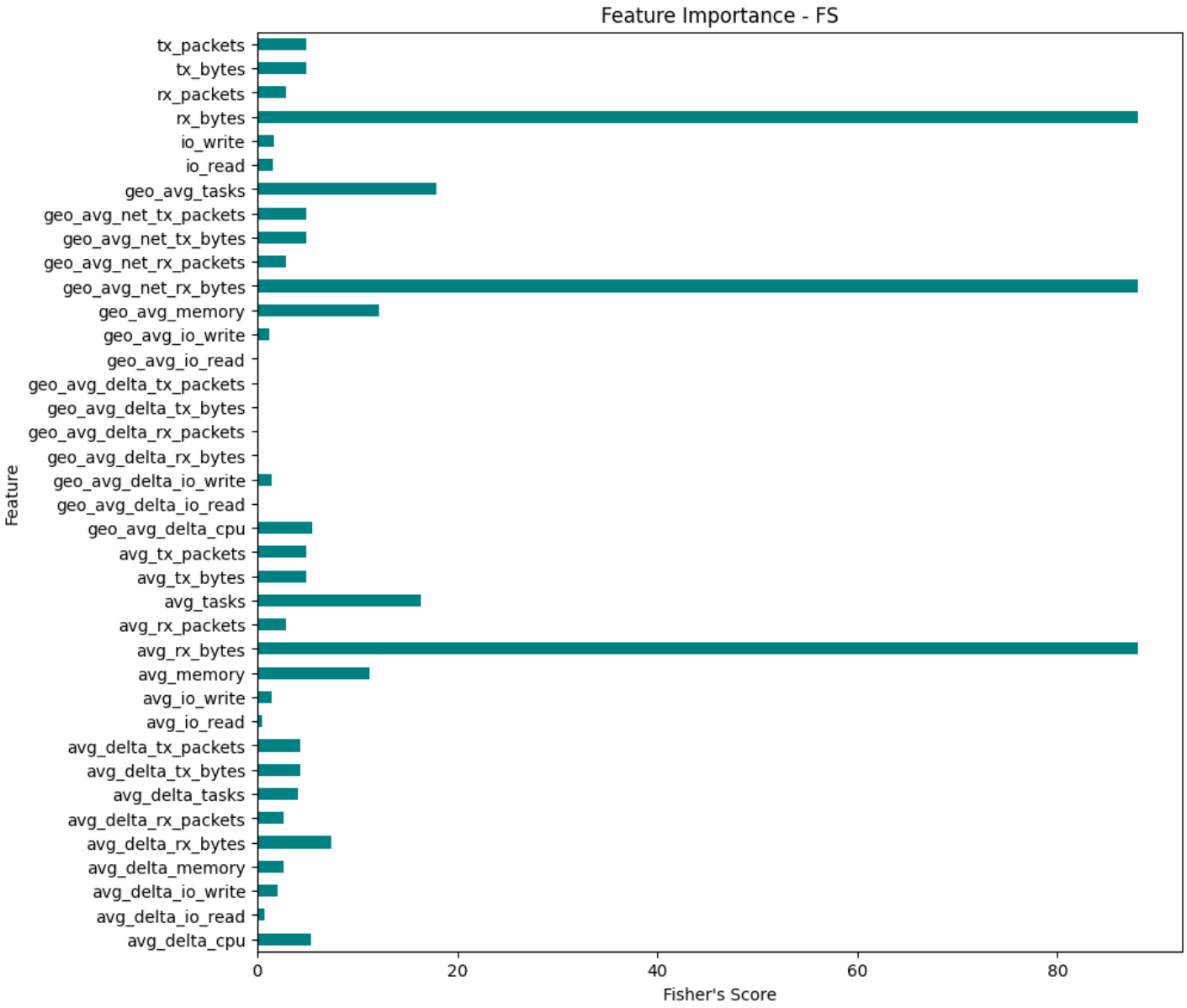
  Στο Γράφημα 6 βλέπουμε την σημαντικότητα του κάθε χαρακτηριστικού με βάση το κριτήριο Mutual Information, δηλαδή το πόσο μειώνει την αβεβαιότητα για την πρόβλεψη η πληροφορία που έχουμε για το εκάστοτε χαρακτηριστικό. Όσο μεγαλύτερη τιμή τόσο πιο σημαντικό είναι το χαρακτηριστικό στην διαδικασία της ταξινόμησης.



Γράφημα 6: Information Gain / Mutual Information Feature Importance

- Fisher's Score

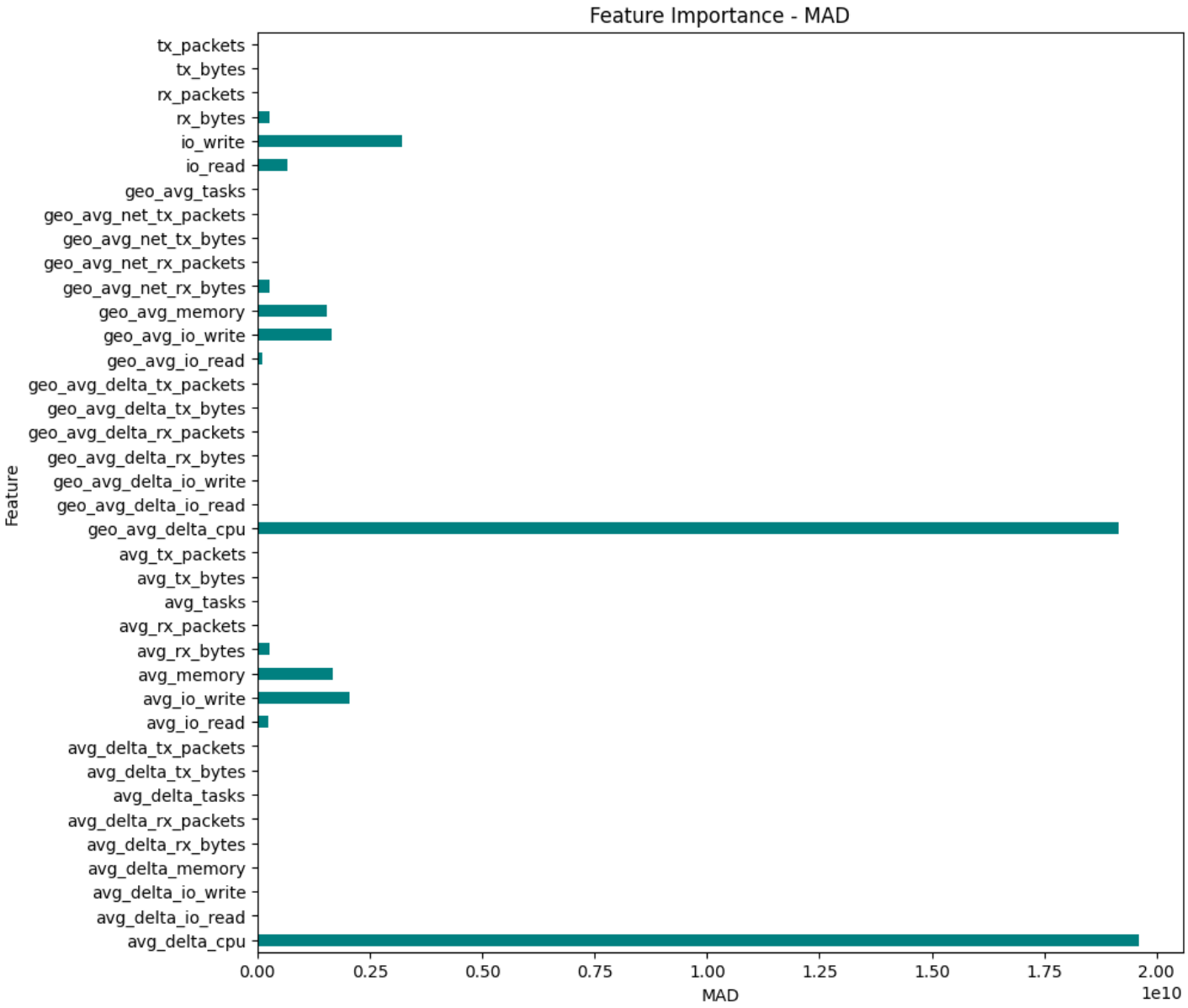
  Στο Γράφημα 7 βλέπουμε την σημαντικότητα του κάθε χαρακτηριστικού με βάση το κριτήριο Fisher's Score, δηλαδή το πόσο βοηθάει στον διαχωρισμό μεταξύ των κλάσεων το εκάστοτε χαρακτηριστικό. Όσο μεγαλύτερη τιμή τόσο πιο σημαντικό είναι το χαρακτηριστικό στην διαδικασία της ταξινόμησης.



Γράφημα 7: Fisher's Score Feature Importance

- MAD

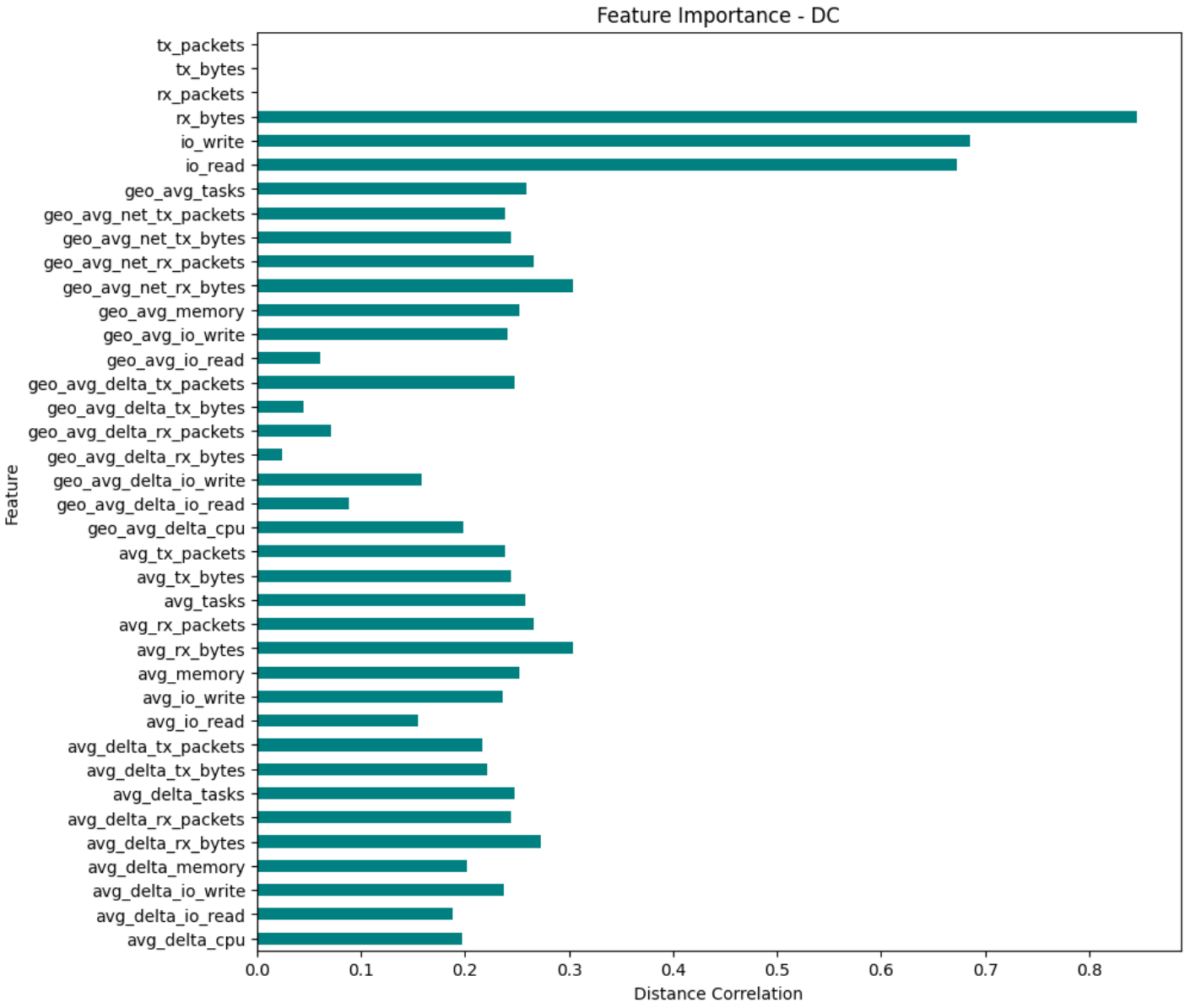
  Στο Γράφημα 8 βλέπουμε την σημαντικότητα του κάθε χαρακτηριστικού με βάση το κριτήριο MAD, δηλαδή με βάση τη μέση τιμή της απόλυτης απόκλισης του διανύσματος του εκάστοτε χαρακτηριστικού από το διάνυσμα των labels. Όσο μεγαλύτερη τιμή τόσο πιο σημαντικό είναι το χαρακτηριστικό στην διαδικασία της ταξινόμησης.



Γράφημα 8: MAD Feature Importance

- Distance Correlation

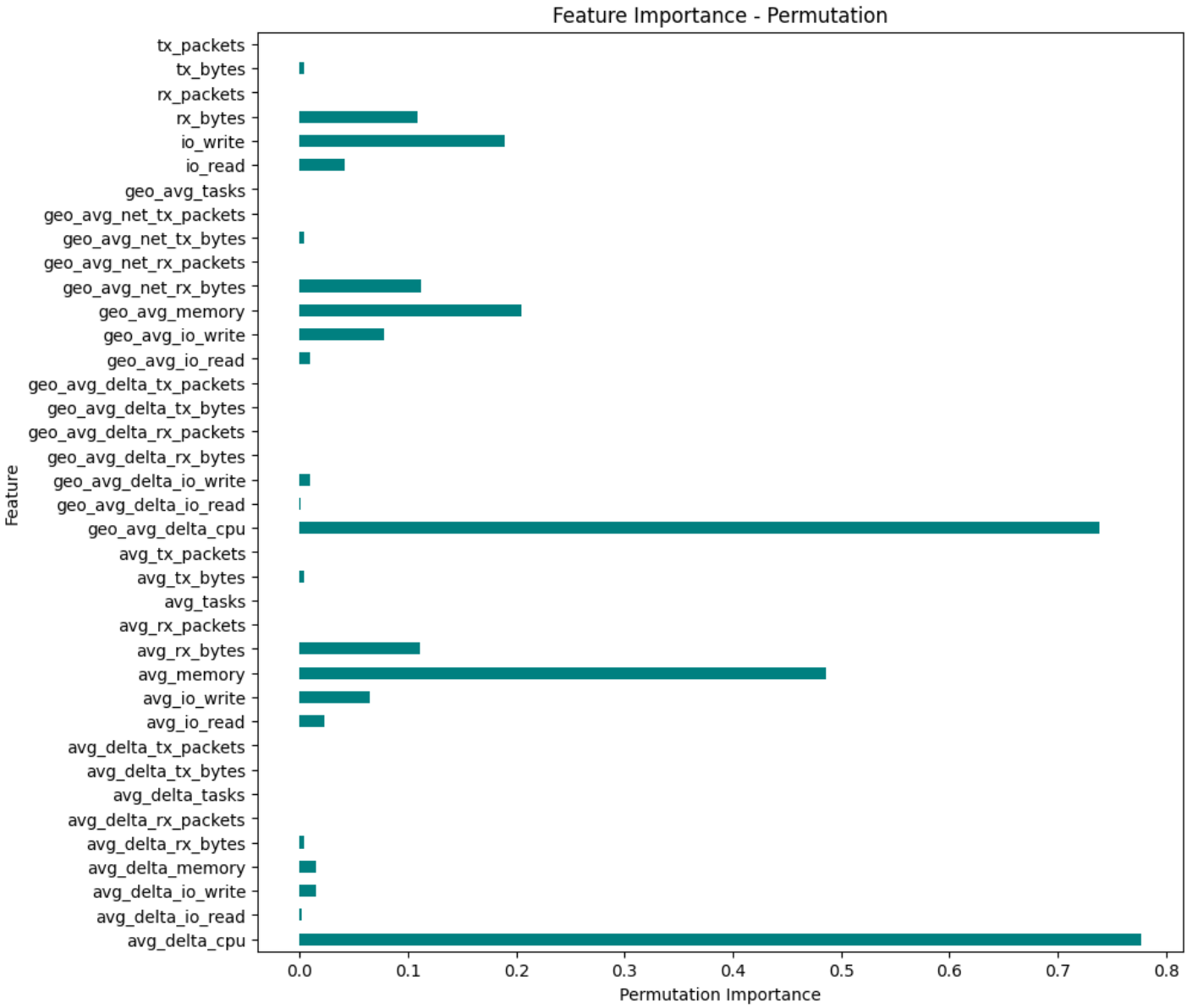
  Στο Γράφημα 9 βλέπουμε την σημαντικότητα του κάθε χαρακτηριστικού με βάση το κριτήριο Distance Correlation, δηλαδή με βάση το μέγεθος της αλληλεξάρτησης του διανύσματος του εκάστοτε χαρακτηριστικού από το διάνυσμα των labels. Όσο μεγαλύτερη τιμή τόσο πιο σημαντικό είναι το χαρακτηριστικό στην διαδικασία της ταξινόμησης.



Γράφημα 9: Distance Correlation Feature Importance

- Permutation Importance

  Στο Γράφημα 10 βλέπουμε την σημαντικότητα του κάθε χαρακτηριστικού με βάση το κριτήριο Permutation Importance, δηλαδή με βάση το πόσο βασίζεται η ακρίβεια των προβλέψεων στο εκάστοτε χαρακτηριστικό. Όσο μεγαλύτερη τιμή τόσο πιο σημαντικό είναι το χαρακτηριστικό στην διαδικασία της ταξινόμησης.



Γράφημα 10: Permutation Feature Importance

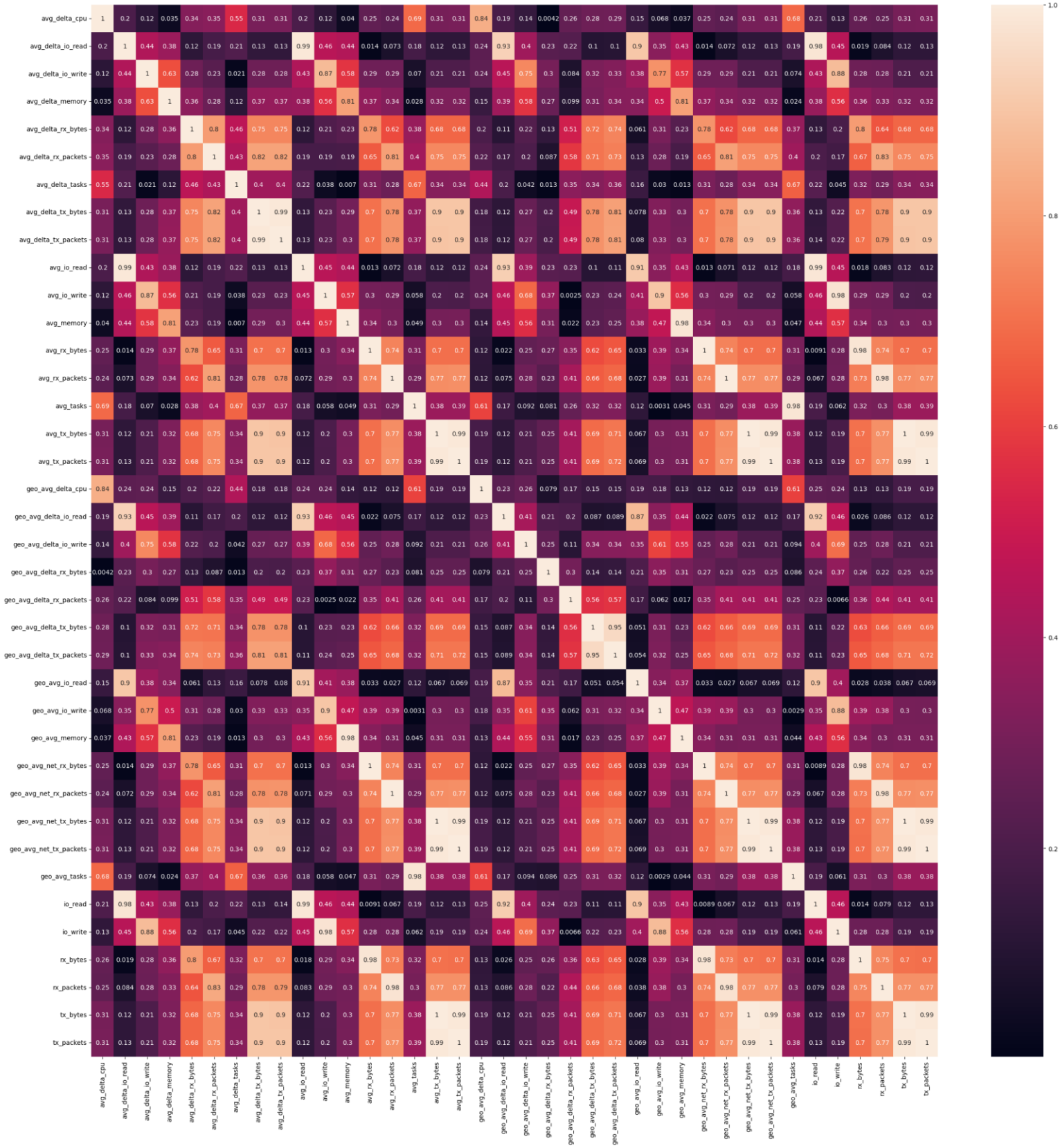
Έχοντας δημιουργήσει πέντε διαφορετικές κατατάξεις της σημαντικότητας του κάθε χαρακτηριστικού, προχωράμε στην $2^{η}$ φάση, την επιλογή ενός αρχικού υποσυνόλου, το οποίο να περιέχει τα δύο «σημαντικότερα» χαρακτηριστικά για τον κάθε πόρο. Για τον πόρο της εγγραφής / ανάγνωσης από τον δίσκο, επιλέχθηκε ένα χαρακτηριστικό για εγγραφή και ένα για ανάγνωση, ενώ για το δίκτυο, επιλέχθηκε ένα χαρακτηριστικό για την λήψη δεδομένων και ένα για την αποστολή. Έτσι σχηματίστηκε το αρχικό υποσύνολο που είναι το ['geo_avg_delta_cpu',

'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read'].

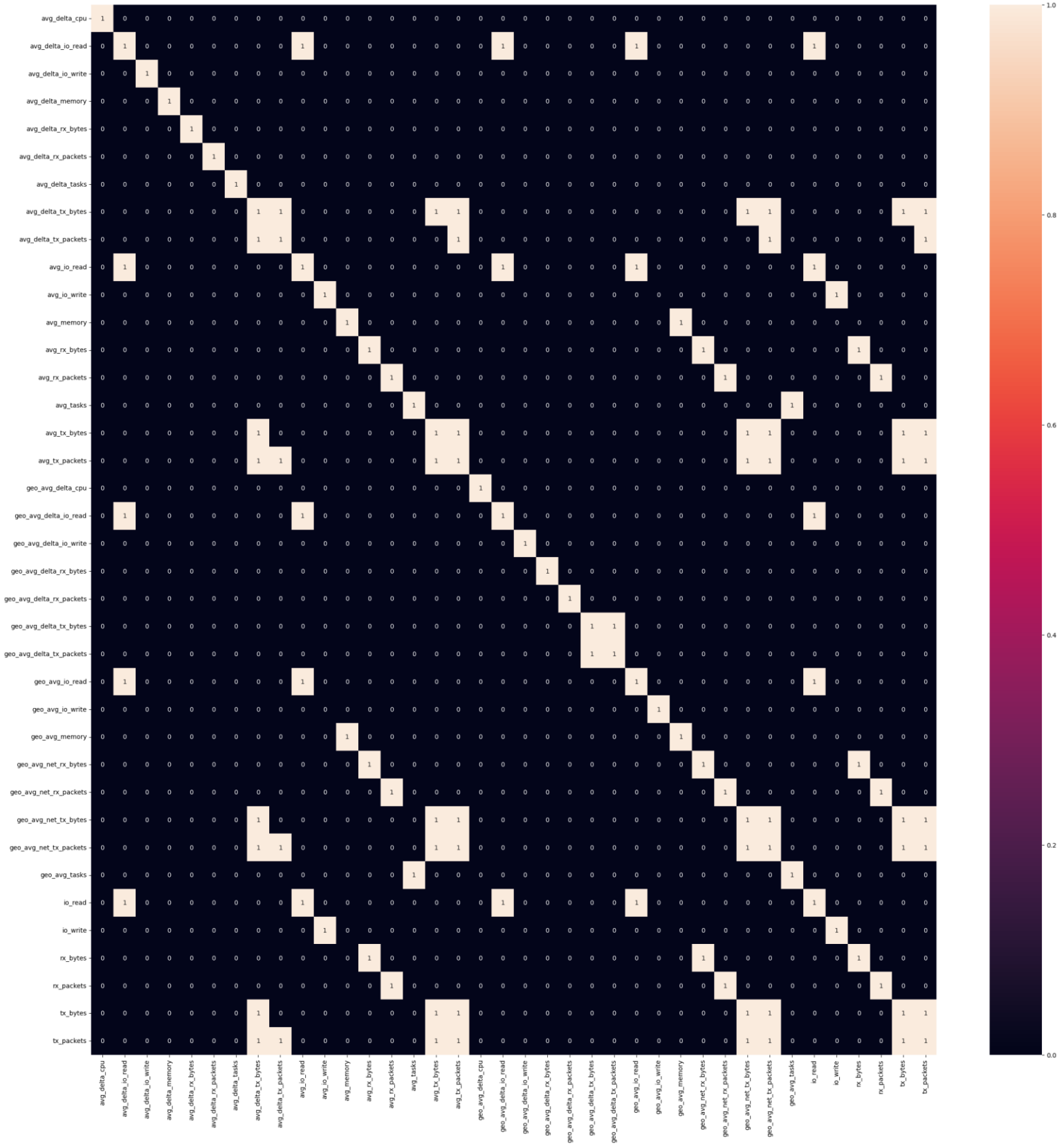
Resource	Features
CPU	geo_avg_delta_cpu avg_delta_cpu
Memory	geo_avg_memory avg_memory
Tasks	avg_tasks geo_avg_tasks
Disk IO	io_write io_read
Network	geo_avg_net_rx_bytes geo_avg_net_tx_bytes

Πίνακας 15: Τα 10 αρχικά χαρακτηριστικά ανά πόρο

Σε αυτό το σημείο έχουμε επιλέξει 10 χαρακτηριστικά από τα 38 που είχαμε αρχικά. Επομένως, έχουμε ένα σύνολο 28 αναξιοποίητων χαρακτηριστικών που θα μπορούσαν να ενταχθούν στο βέλτιστο υποσύνολο. Πριν όμως αρχίσουμε να αξιολογούμε τα διάφορα υποσύνολα, περνάμε στην $3^{η}$ φάση της μεθοδολογίας, όπου εξετάζουμε την συσχέτιση μεταξύ όλων των ζευγών μεταξύ των χαρακτηριστικών, με σκοπό να εντοπίσουμε ποια από τα αναξιοποίητα 28 έχουν μεγάλη συνάφεια με κάποιο ή κάποια από τα αρχικά 10 χαρακτηριστικά. Για τον σκοπό αυτό χρησιμοποιούμε το Kendall Correlation. Με βάση αυτό κάθε ζεύγος χαρακτηριστικών λαμβάνει μία τιμή στο διάστημα [-1,1], με τα -1 και 1 να δηλώνουν τέλεια συσχέτιση, και το 0 να δηλώνει πλήρη ανεξαρτησία μεταξύ των δύο χαρακτηριστικών. Εμείς παίρνουμε την απόλυτη τιμή του Correlation Coefficient, αφού δεν μας ενδιαφέρει η κατεύθυνση της συσχέτισης αλλά μόνο το μέγεθός της. Αυτό φαίνεται στο Γράφημα 11, ενώ στο Γράφημα 12 έχουμε το ίδιο heatmap, όμως για να δώσουμε μία πιο ξεκάθαρη εικόνα έχουμε δώσει την τιμή 1 σε κάθε στοιχείο που ήταν μεγαλύτερο του 0.9 και την τιμή 0 σε όλα τα υπόλοιπα στοιχεία. Δηλαδή στο Γράφημα 12 βλέπουμε τα ζεύγη χαρακτηριστικών με συσχέτιση μεγαλύτερη του 90%.



Γράφημα 11: Kendall Correlation Coefficient



Γράφημα 12: Kendall Correlation Coefficient > 90%

Αφαιρούμε λοιπόν από το σύνολο των αναξιοποίητων χαρακτηριστικών όσα έχουν συσχέτιση μεγαλύτερη του 90% με κάποιο από τα αρχικά 10. Τα χαρακτηριστικά που αφαιρέθηκαν είναι συνολικά 13 και έτσι από τα 28 πέφτουμε στα 15 αναξιοποίητα χαρακτηριστικά. Τα 13 χαρακτηριστικά που αφαιρέθηκαν είναι τα ['avg_rx_bytes', 'rx_bytes','avg_delta_tx_bytes', 'avg_tx_bytes', 'avg_tx_packets', 'geo_avg_net_tx_packets', 'tx_bytes', 'tx_packets', 'avg_delta_io_read', 'avg_io_read', 'geo_avg_delta_io_read', 'geo_avg_io_read','avg_io_write'].

Έχοντας πλέον διαμορφώσει τα σύνολα των 10 αρχικών χαρακτηριστικών και των 15 (υποψήφιων) αναξιοποίητων χαρακτηριστικών, μπορούμε να περάσουμε στην 4η φάση, όπου υλοποιείται μια παραλλαγή του αλγορίθμου Forward Selection χρησιμοποιώντας ένα ΤΝΔ ως μοντέλο ταξινόμησης. Σε αυτή τη φάση καλούμαστε να αξιολογήσουμε σε κάθε βήμα τόσα υποσύνολα χαρακτηριστικών, όσα και τα εναπομείναντα αναξιοποίητα χαρακτηριστικά στο συγκεκριμένο βήμα. Έτσι θα έχουμε 15 υποσύνολα στο 1ο βήμα, 14 στο 2ο , 13 στο 3ο κ.ο.κ. Επομένως θα χρειαστεί να εκπαιδεύσουμε ένα μεγάλο αριθμό μοντέλων, και επιπλέον, για το κάθε ένα από αυτά εφαρμόζουμε αξιολόγηση με 5-fold cross-validation. Η διαδικασία αυτή είναι απαιτητική τόσο σε υπολογιστικούς πόρους όσο και σε χρόνο. Για το λόγο αυτό μειώνουμε τον αριθμό των εποχών σε 250 χωρίς δυνατότητα πρόωρου τερματισμού. Με αυτόν τον τρόπο, πετυχαίνουμε μεγάλη μείωση του χρόνου εκπαίδευσης των μοντέλων και συνεπώς του συνολικού χρόνου εκτέλεσης του αλγορίθμου, με μια μικρή μόνο μείωση στην απόδοση του ΤΝΔ.

5.3 Αξιολόγηση των επικρατέστερων υποσυνόλων

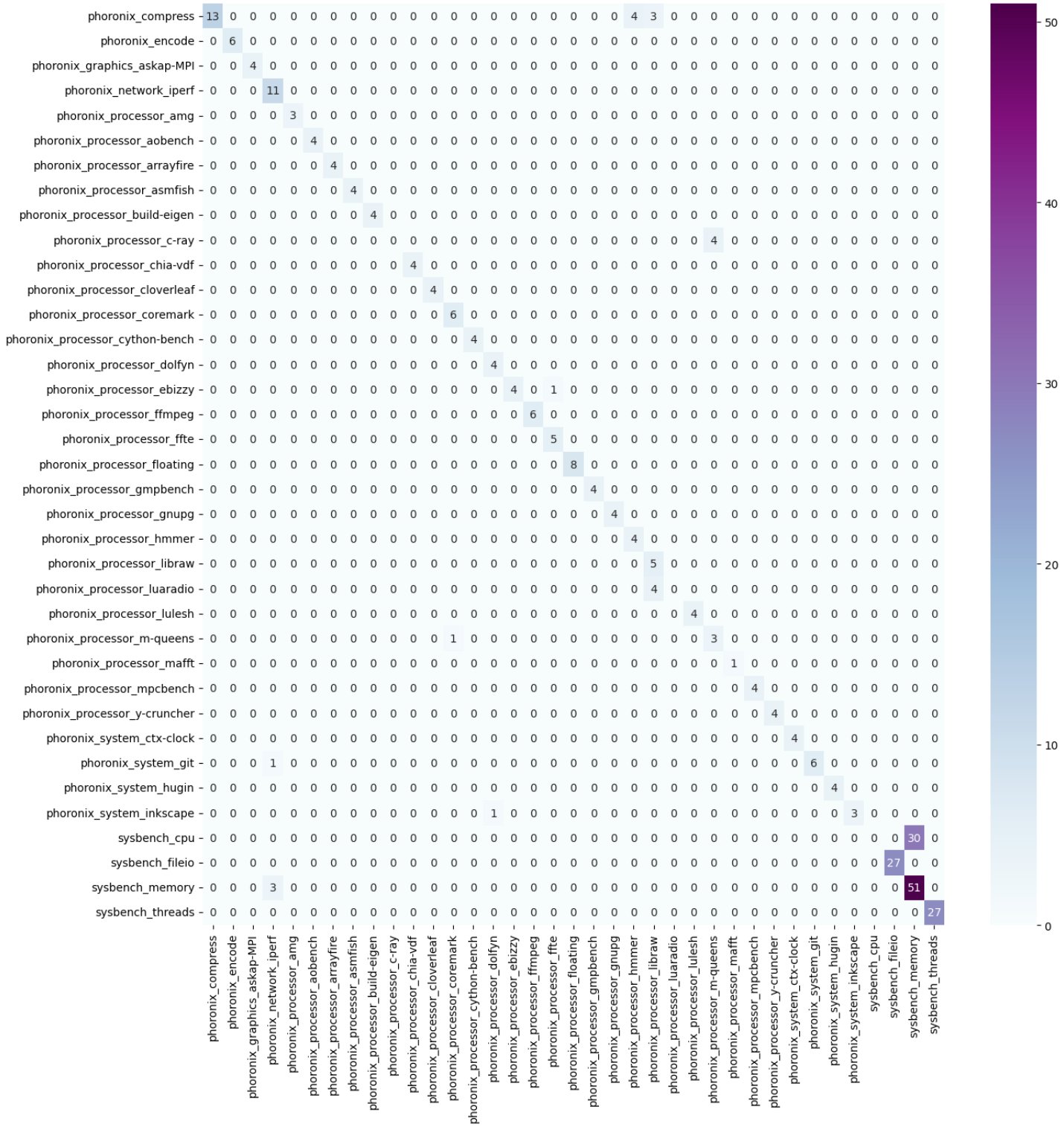
Αφού ολοκληρώνεται η εκτέλεση, οδηγούμαστε στην 5η και τελευταία φάση της μεθοδολογίας, όπου επιλέγουμε τα πέντε πιο αποδοτικά υποσύνολα χαρακτηριστικών για να διεξάγουμε σε αυτά περαιτέρω αξιολόγηση. Επαναφέρουμε τον αριθμό των εποχών σε 1000, καθώς και την δυνατότητα πρόωρου τερματισμού και παράγουμε για το κάθε υποσύνολο το confusion matrix και το classification report. Ο στόχος είναι να συγκρίνουμε τα υποσύνολα αυτά μεταξύ τους, αλλά και με το υποσύνολο με τα 10 αρχικά χαρακτηριστικά, και το σύνολο που περιέχει όλα τα 38 χαρακτηριστικά. Ονομάζουμε subset_0 το υποσύνολο των 10 αρχικών χαρακτηριστικών και subset_1 έως subset_5 τα 5 πιο αποδοτικά υποσύνολα, τα οποία είναι τα εξής:

Subset	Features
Subset_0 (10 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read']
Subset_1 (17 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write', 'avg_delta_tx_packets', 'avg_delta_io_write', 'geo_avg_io_write', 'rx_packets']
Subset_2 (18 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write', 'avg_delta_tx_packets', 'avg_delta_io_write', 'geo_avg_io_write', 'rx_packets', 'avg_delta_rx_bytes']
Subset_3 (22 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write', 'avg_delta_tx_packets', 'avg_delta_io_write', 'geo_avg_io_write', 'rx_packets', 'avg_delta_rx_bytes', 'geo_avg_delta_rx_bytes', 'avg_delta_tasks', 'geo_avg_delta_tx_packets', 'geo_avg_net_rx_packets']
Subset_4 (17 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write', 'avg_delta_tx_packets', 'avg_delta_io_write', 'geo_avg_io_write', 'geo_avg_delta_tx_packets']
Subset_5 (13 features)	['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write']

Πίνακας 16: Best Feature Subsets

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.40	0.57	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.50	0.73	0.59	11
phoronix_processor_amg	1.00	1.00	1.00	3
phoronix_processor_aobench	0.80	1.00	0.89	4
phoronix_processor_arrayfire	1.00	1.00	1.00	4
phoronix_processor_asmfish	0.57	1.00	0.73	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	1.00	1.00	1.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.75	1.00	0.86	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	1.00	1.00	1.00	4
phoronix_processor_ebizzy	1.00	0.80	0.89	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.50	0.20	0.29	5
phoronix_processor_floating	0.88	0.88	0.88	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.75	0.86	4
phoronix_processor_hmmer	0.44	1.00	0.62	4
phoronix_processor_libraw	0.45	1.00	0.62	5
phoronix_processor_luaradio	1.00	0.25	0.40	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.50	0.75	0.60	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	0.80	1.00	0.89	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.00	0.00	0.00	30
sysbench_fileio	1.00	1.00	1.00	27
sysbench_memory	0.63	0.94	0.76	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.86	0.87	0.84	305
weighted avg	0.77	0.80	0.76	305

Πίνακας 17: Subset_0 Classification Report



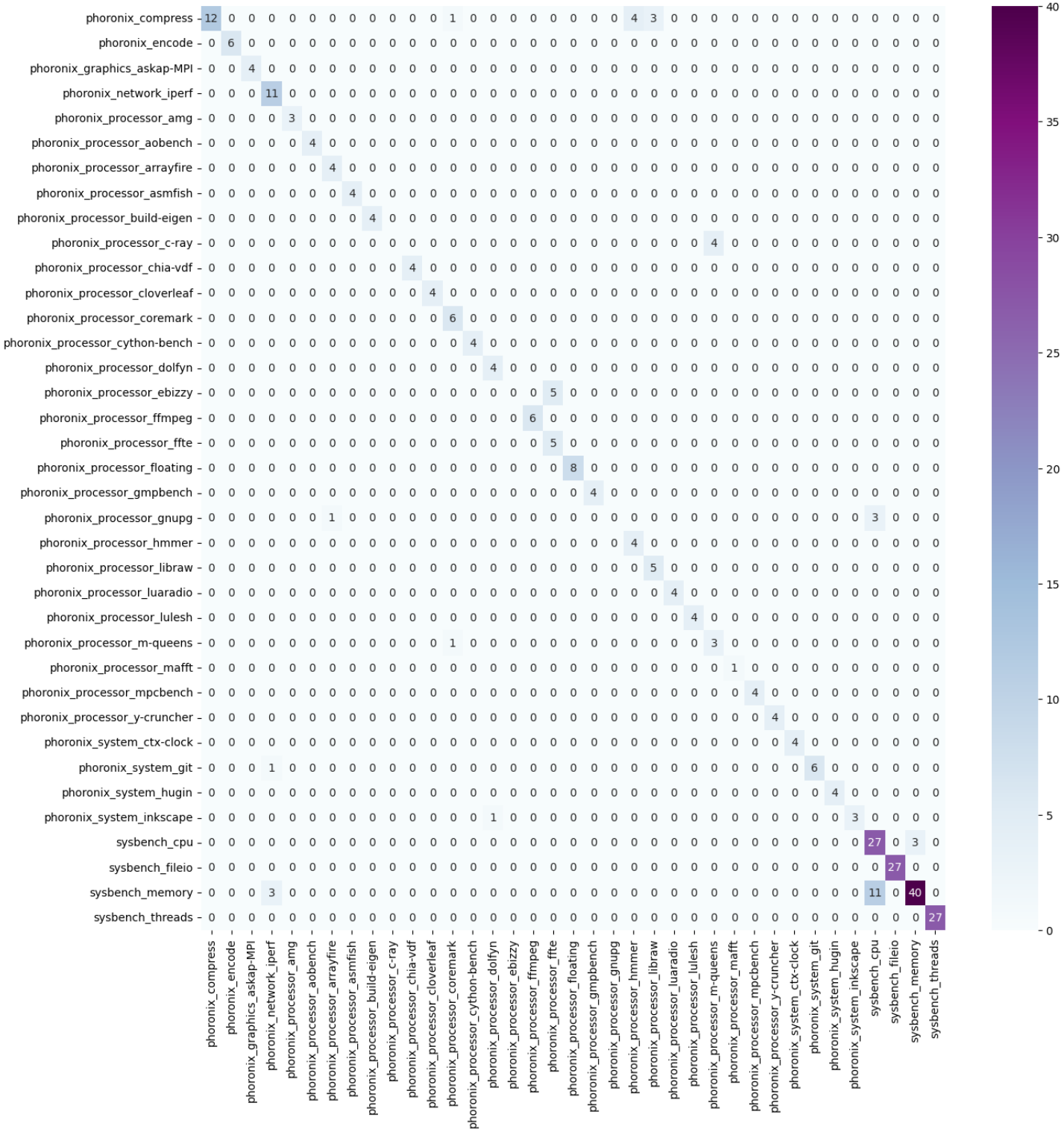
Γράφημα 13: Subset_0 Confusion Matrix

Παραπάνω βλέπουμε το classification report και το confusion matrix για το subset_0. Παρατηρούμε πως η απόδοση του μοντέλου για το subset_0 είναι πολύ κοντά σε αυτήν που έχουμε χρησιμοποιώντας όλα τα χαρακτηριστικά. Βλέπουμε δηλαδή πως το ΤΝΔ αντλεί από αυτά τα δέκα χαρακτηριστικά πληροφορία σχεδόν ισάξια με αυτήν που αντλούσε από τα αρχικά 38 χαρακτηριστικά. Το γεγονός αυτό ήταν αναμενόμενο, δεδομένου ότι επιλέξαμε χαρακτηριστικά για όλους τους πόρους και μάλιστα επιλέχθηκαν αυτά με την μεγαλύτερη αξία για την διαδικασία της ταξινόμησης.

Στη συνέχεια έχουμε το classification report και το confusion matrix για το subset_1. Το subset_1 είναι το υποσύνολο χαρακτηριστικών με το καλύτερο accuracy μέσω της διαδικασίας 5-fold cross-validation. Με άλλα λόγια είναι το υποσύνολο χαρακτηριστικών που θα επιλεγόταν εάν ο αλγόριθμός μας επέλεγε αυτόματα το πιο αποδοτικό υποσύνολο. Όπως είναι λογικό έχει πολύ καλύτερη απόδοση τόσο από το subset_0 όσο και από το σύνολο όλων των χαρακτηριστικών. Παρατηρούμε μάλιστα πως καταφέρνει να ταξινομήσει με επιτυχία τα περισσότερα από τα δείγματα των benchmarks sysbench_cpu και sysbench_memory παρότι αυτά παράγουν προφίλ με πολύ μικρές διαφοροποιήσεις.

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.60	0.75	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.73	1.00	0.85	11
phoronix_processor_amg	1.00	1.00	1.00	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	0.80	1.00	0.89	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.75	1.00	0.86	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.80	1.00	0.89	4
phoronix_processor_ebizzy	0.00	0.00	0.00	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.50	1.00	0.67	5
phoronix_processor_floating	1.00	1.00	1.00	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	0.00	0.00	0.00	4
phoronix_processor_hmmer	0.50	1.00	0.67	4
phoronix_processor_libraw	0.62	1.00	0.77	5
phoronix_processor_luaradio	1.00	1.00	1.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	1.00	1.00	1.00	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.66	0.90	0.76	30
sysbench_fileio	1.00	1.00	1.00	27
sysbench_memory	0.93	0.74	0.82	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.83	0.88	0.84	305
weighted avg	0.86	0.87	0.85	305

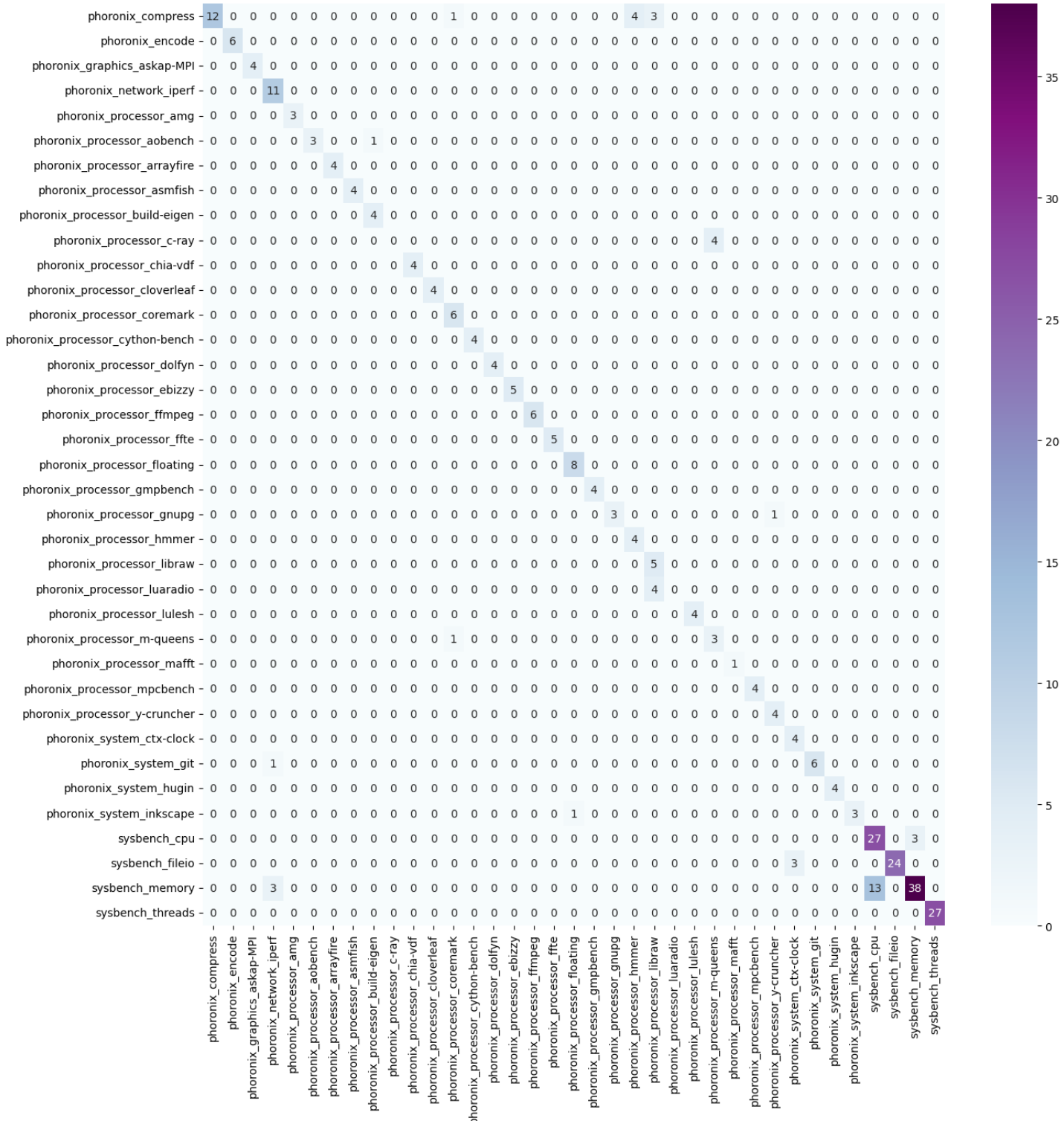
Πίνακας 18: Subset_1 Classification Report



Γράφημα 14: Subset_1 Confusion Matrix

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.60	0.75	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.73	1.00	0.85	11
phoronix_processor_amg	1.00	1.00	1.00	3
phoronix_processor_aobench	1.00	0.75	0.86	4
phoronix_processor_arrayfire	1.00	1.00	1.00	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	0.80	1.00	0.89	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.75	1.00	0.86	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	1.00	1.00	1.00	4
phoronix_processor_ebizzy	1.00	1.00	1.00	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	1.00	1.00	1.00	5
phoronix_processor_floating	0.89	1.00	0.94	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.75	0.86	4
phoronix_processor_hmmer	0.50	1.00	0.67	4
phoronix_processor_libraw	0.42	1.00	0.59	5
phoronix_processor_luaradio	0.00	0.00	0.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	0.80	1.00	0.89	4
phoronix_system_ctx-clock	0.57	1.00	0.73	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.68	0.90	0.77	30
sysbench_fileio	1.00	0.89	0.94	27
sysbench_memory	0.93	0.70	0.80	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.85	0.89	0.86	305
weighted avg	0.88	0.86	0.85	305

Πίνακας 19: Subset_2 Classification Report



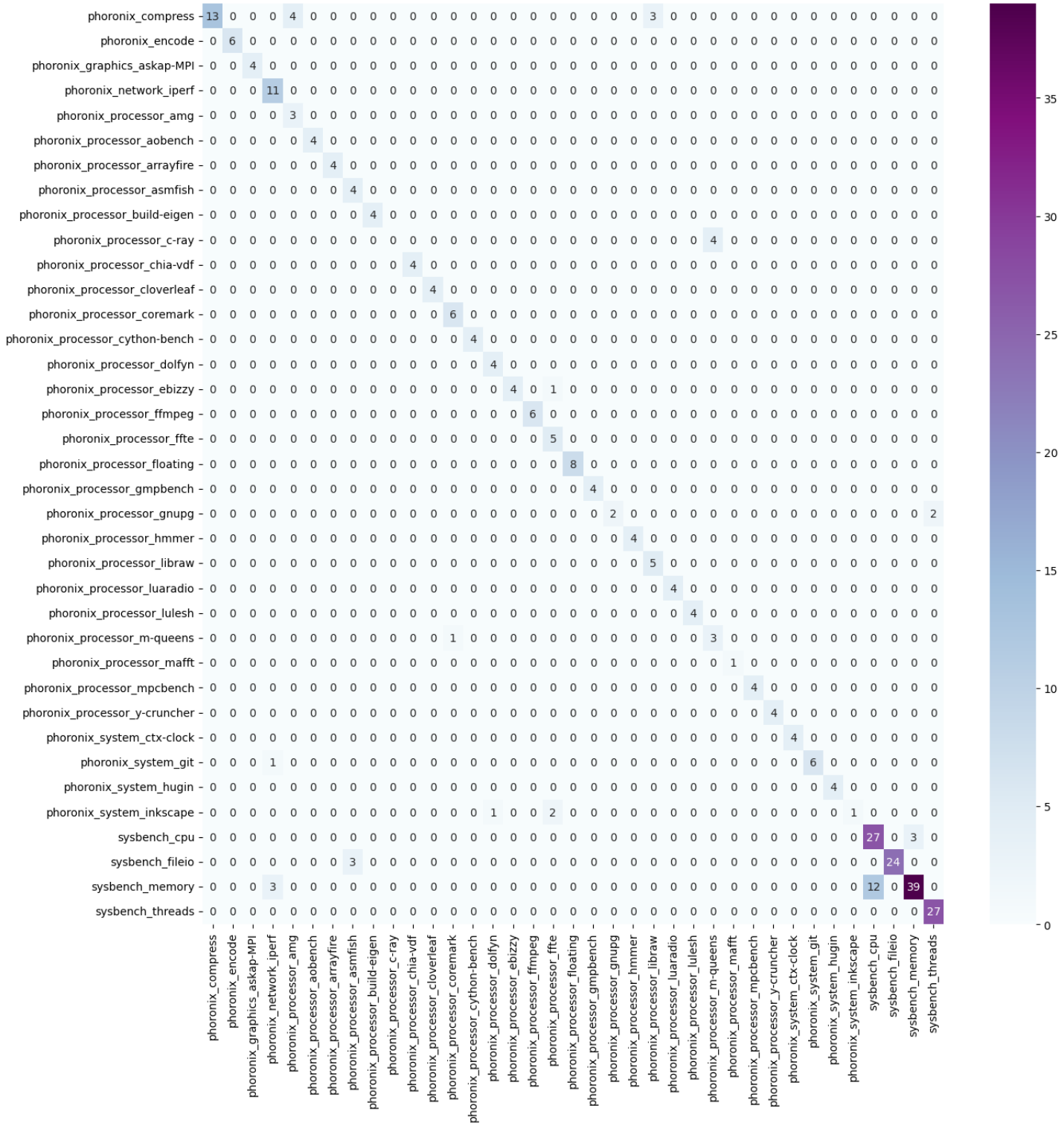
Γράφημα 15: Subset_2 Confusion Matrix

Παραπάνω βλέπουμε το classification report και το confusion matrix για το subset_2. Παρατηρούμε πως παρότι έχουμε κατ' ελάχιστον χαμηλότερο 5-fold cross-validation accuracy, έχουμε καλύτερη συνολική απόδοση στο testing set, με το subset_2 να ξεπερνά το subset_1 τόσο σε precision όσο και σε recall.

Ακολουθούν το classification report και το confusion matrix για το subset_3. Το subset_3 πετυχαίνει ακόμη καλύτερη απόδοση από το subset_2 τόσο σε accuracy όσο και σε precision και recall. Αποδεικνύεται επομένως πως το υποσύνολο χαρακτηριστικών με το μεγαλύτερο 5-fold cross-validation accuracy δεν είναι απαραίτητα και το πιο αποδοτικό.

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.65	0.79	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.73	1.00	0.85	11
phoronix_processor_amg	0.43	1.00	0.60	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	1.00	1.00	1.00	4
phoronix_processor_asmfish	0.57	1.00	0.73	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.86	1.00	0.92	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.80	1.00	0.89	4
phoronix_processor_ebizzy	1.00	0.80	0.89	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.62	1.00	0.77	5
phoronix_processor_floating	1.00	1.00	1.00	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.50	0.67	4
phoronix_processor_hmmer	1.00	1.00	1.00	4
phoronix_processor_libraw	0.62	1.00	0.77	5
phoronix_processor_luaradio	1.00	1.00	1.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	1.00	1.00	1.00	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.25	0.40	4
sysbench_cpu	0.69	0.90	0.78	30
sysbench_fileio	1.00	0.89	0.94	27
sysbench_memory	0.93	0.72	0.81	54
sysbench_threads	0.93	1.00	0.96	27
macro avg	0.88	0.90	0.87	305
weighted avg	0.89	0.87	0.86	305

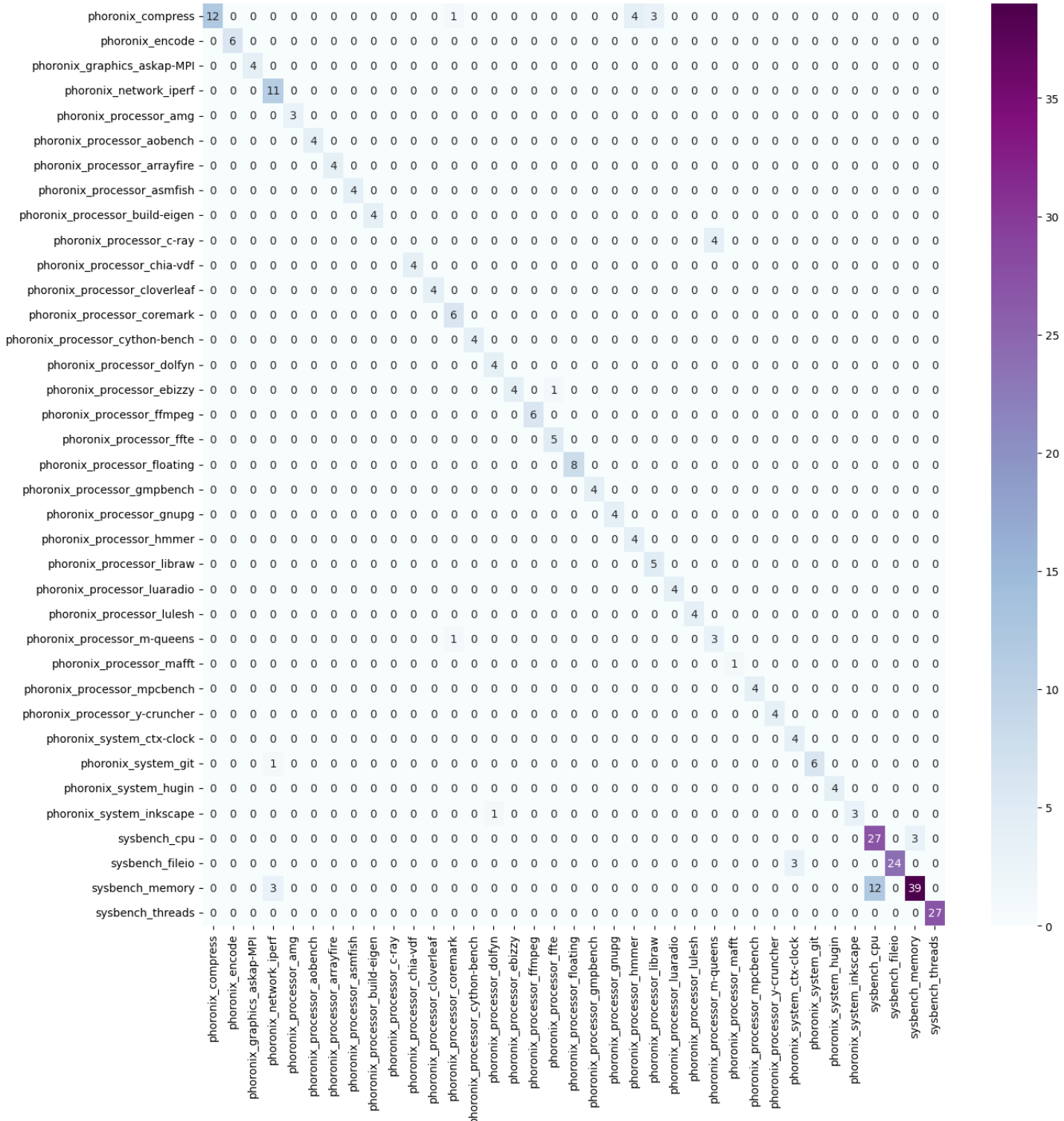
Πίνακας 20: Subset_3 Classification Report



Γράφημα 16: Subset_3 Confusion Matrix

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	1.00	0.60	0.75	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.73	1.00	0.85	11
phoronix_processor_amg	1.00	1.00	1.00	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	1.00	1.00	1.00	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	1.00	1.00	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.75	1.00	0.86	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.80	1.00	0.89	4
phoronix_processor_ebizzy	1.00	0.80	0.89	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	0.83	1.00	0.91	5
phoronix_processor_floating	1.00	1.00	1.00	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	1.00	1.00	4
phoronix_processor_hmmer	0.50	1.00	0.67	4
phoronix_processor_libraw	0.62	1.00	0.77	5
phoronix_processor_luaradio	1.00	1.00	1.00	4
phoronix_processor_lulesh	1.00	1.00	1.00	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	0.57	1.00	0.73	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.69	0.90	0.78	30
sysbench_fileio	1.00	0.89	0.94	27
sysbench_memory	0.93	0.72	0.81	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.89	0.93	0.90	305
weighted avg	0.90	0.88	0.88	305

Πίνακας 21: Subset_4 Classification Report



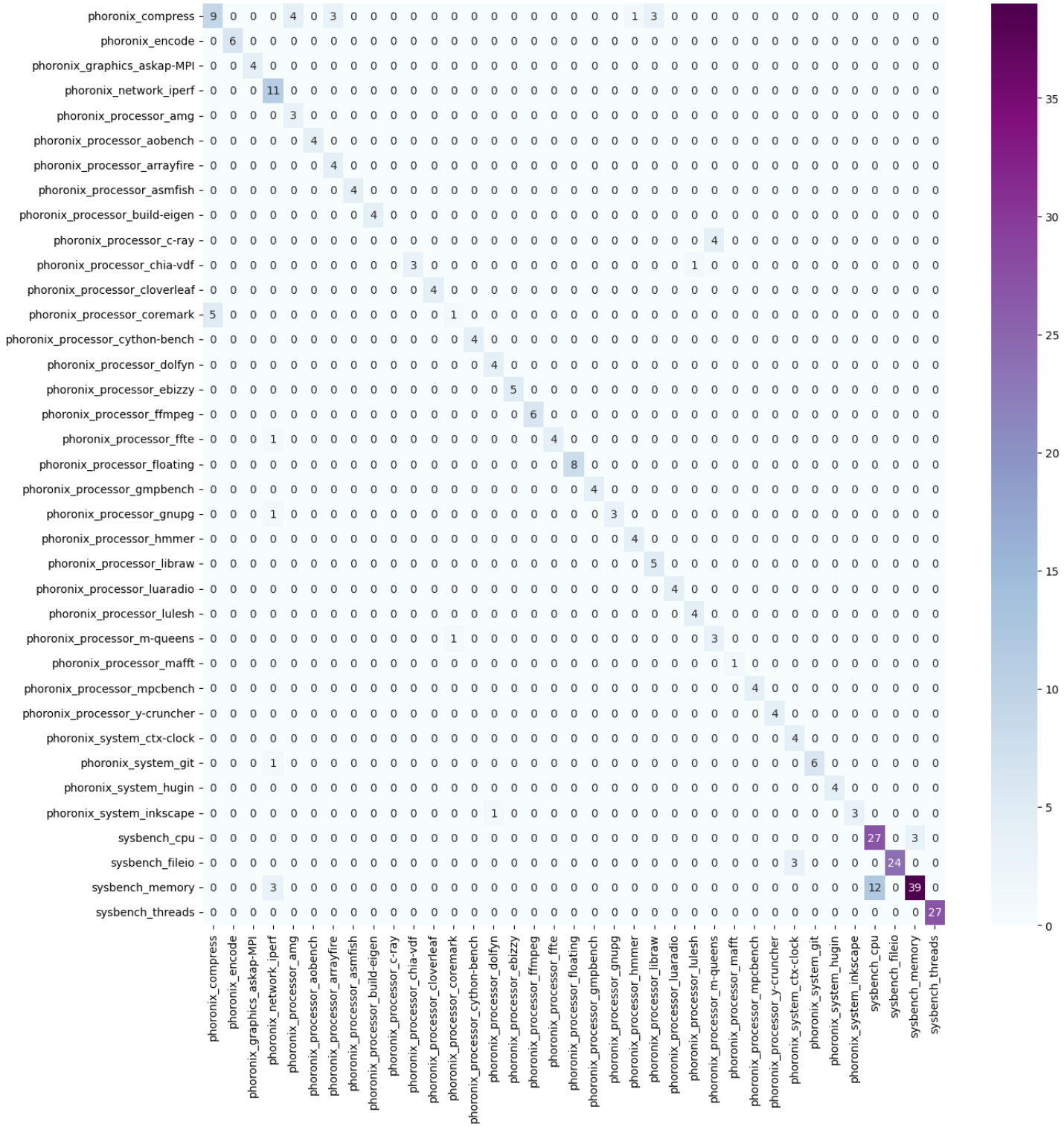
Γράφημα 17: Subset_4 Confusion Matrix

Στον Πίνακα 21 και το Γράφημα 17 έχουμε το classification report και το confusion matrix για το subset_4. Το subset_4 βλέπουμε ότι πετυχαίνει την καλύτερη απόδοση από όσα εξετάσαμε, φέρνοντας αισθητά καλύτερα αποτελέσματα στο testing set accuracy. Ταυτόχρονα έχει καλύτερες τιμές από κάθε άλλο υποσύνολο σε precision, recall και f1-score. Έτσι καθίσταται το καλύτερο υποσύνολο χαρακτηριστικών για το σύνολο δεδομένων μας.

Τέλος, παρακάτω βλέπουμε το classification report και το confusion matrix για το subset_5, όπου παρατηρούμε αρκετά μειωμένη απόδοση σε σχέση με το subset_4. Παρότι τα αποτελέσματα εδώ δεν απέχουν πολύ από αυτά του subset_1, αν συνεχίσουμε να εξετάζουμε υποσύνολα μπορούμε να περιμένουμε περαιτέρω πτώση των τιμών των μετρικών που μας ενδιαφέρουν.

Classification Report				
classes	precision	recall	f1-score	support
phoronix_compress	0.64	0.45	0.53	20
phoronix_encode	1.00	1.00	1.00	6
phoronix_graphics_askap-MPI	1.00	1.00	1.00	4
phoronix_network_iperf	0.65	1.00	0.79	11
phoronix_processor_amg	0.43	1.00	0.60	3
phoronix_processor_aobench	1.00	1.00	1.00	4
phoronix_processor_arrayfire	0.57	1.00	0.73	4
phoronix_processor_asmfish	1.00	1.00	1.00	4
phoronix_processor_build-eigen	1.00	1.00	1.00	4
phoronix_processor_c-ray	0.00	0.00	0.00	4
phoronix_processor_chia-vdf	1.00	0.75	0.86	4
phoronix_processor_cloverleaf	1.00	1.00	1.00	4
phoronix_processor_coremark	0.50	0.17	0.25	6
phoronix_processor_cython-bench	1.00	1.00	1.00	4
phoronix_processor_dolfyn	0.80	1.00	0.89	4
phoronix_processor_ebizzy	1.00	1.00	1.00	5
phoronix_processor_ffmpeg	1.00	1.00	1.00	6
phoronix_processor_ffte	1.00	0.80	0.89	5
phoronix_processor_floating	1.00	1.00	1.00	8
phoronix_processor_gmpbench	1.00	1.00	1.00	4
phoronix_processor_gnupg	1.00	0.75	0.86	4
phoronix_processor_hmmer	0.80	1.00	0.89	4
phoronix_processor_libraw	0.62	1.00	0.77	5
phoronix_processor_luaradio	1.00	1.00	1.00	4
phoronix_processor_lulesh	0.80	1.00	0.89	4
phoronix_processor_m-queens	0.43	0.75	0.55	4
phoronix_processor_mafft	1.00	1.00	1.00	1
phoronix_processor_mpcbench	1.00	1.00	1.00	4
phoronix_processor_y-cruncher	1.00	1.00	1.00	4
phoronix_system_ctx-clock	0.57	1.00	0.73	4
phoronix_system_git	1.00	0.86	0.92	7
phoronix_system_hugin	1.00	1.00	1.00	4
phoronix_system_inkscape	1.00	0.75	0.86	4
sysbench_cpu	0.69	0.90	0.78	30
sysbench_fileio	1.00	0.89	0.94	27
sysbench_memory	0.93	0.72	0.81	54
sysbench_threads	1.00	1.00	1.00	27
macro avg	0.85	0.89	0.85	305
weighted avg	0.86	0.85	0.84	305

Πίνακας 22: Subset_5 Classification Report



Γράφημα 18: Subset_5 Confusion Matrix

Παρατηρούμε πως τα υποσύνολα αυτά δεν έχουν μεγάλες αποκλίσεις στην απόδοση τους, γεγονός αναμενόμενο. Γίνεται όμως ξεκάθαρο πως η επιλογή του υποσυνόλου χαρακτηριστικού παίζει μεγάλο ρόλο στην ικανότητα διάκρισης ανάμεσα σε κοντινές κλάσεις. Βλέπουμε πως τα 5 πιο αποδοτικά υποσύνολα μπορούν να ξεχωρίσουν αρκετά καλά τα δείγματα που ανήκουν στην κλάση sysbench_cpu από αυτά που ανήκουν στην sysbench_memory. Στον παρακάτω πίνακα εμφανίζεται μια συνολική εικόνα της απόδοσης του κάθε υποσυνόλου.

Subset	macro_ avg precision *	macro _avg recall *	macro_ avg f1-score *	weighted _avg precision *	weighted _avg recall *	weighted _avg f1-score *	5-fold cv accuracy	validation set accuracy	testing set accuracy
All_feats	0.84	0.89	0.85	0.77	0.83	0.79	92,91%	96,47%	82,95%
Subset_0	0.86	0.87	0.84	0.77	0.80	0.76	91,41%	96,05%	80,00%
Subset_1	0.83	0.88	0.84	0.86	0.87	0.85	93,76%	97,09%	86,56%
Subset_2	0.85	0.89	0.86	0.88	0.86	0.85	93,66%	97,09%	85,90%
Subset_3	0.88	0.90	0.87	0.89	0.87	0.86	93,66%	95,84%	86,89%
Subset_4	0.89	0.93	0.90	0.90	0.88	0.88	93,64%	96,67%	87,87%
Subset_5	0.85	0.89	0.85	0.86	0.85	0.84	93,64%	96,05%	84,59%

* Αφορά την απόδοση στο testing set

Πίνακας 23: Συνολική απόδοση των επικρατέστερων υποσυνόλων

Όπως βλέπουμε, το subset_1 που πέτυχε την καλύτερη επίδοση με το 5-fold cross-validation και προέκυψε ως πιο αποδοτικό από τη διαδικασία feature selection, δεν έχει την καλύτερη απόδοση συνολικά. Το υποσύνολο που αποδεικνύεται καλύτερο τελικά είναι το subset_4. Όχι μόνο έχει την μεγαλύτερη ακρίβεια προβλέψεων στο testing set (accuracy = 87.87%), αλλά έχει τις καλύτερες τιμές και στα: macro_avg precision, macro_avg recall, macro_avg f1-score, weighted_avg precision, weighted_avg recall και weighted_avg f1-score. Επομένως, είναι και το υποσύνολο που επιλέγουμε τελικά, με τα εξής 17 χαρακτηριστικά: ['geo_avg_delta_cpu', 'avg_delta_cpu', 'geo_avg_memory', 'avg_memory', 'avg_tasks', 'geo_avg_tasks', 'geo_avg_net_rx_bytes', 'geo_avg_net_tx_bytes', 'io_write', 'io_read', 'avg_delta_memory', 'avg_rx_packets', 'geo_avg_delta_io_write', 'avg_delta_tx_packets', 'avg_delta_io_write', 'geo_avg_io_write', 'geo_avg_delta_tx_packets'].

6 *Συμπεράσματα και μελλοντικές κατευθύνσεις*

Έμπνευση για αυτήν την διπλωματική εργασία αποτέλεσε το σύστημα δημιουργίας προφίλ φόρτου εργασιών εφαρμογών και ταξινόμησης τους με χρήση αλγορίθμων μηχανικής μάθησης που παρουσιάζεται στο [6]. Η συνεισφορά της παρούσης αποτελείται από δύο βασικά σκέλη:

1. Την ανάπτυξη και παραμετροποίηση ενός μοντέλου ταξινόμησης και συγκεκριμένα ενός ΤΝΔ βαθιάς αρχιτεκτονικής. Το μοντέλο αυτό αναλαμβάνει την ταξινόμηση εφαρμογών σε γνωστά benchmarks με βάση το φόρτο εργασιών τους.
2. Την υλοποίηση ενός αλγορίθμου επιλογής χαρακτηριστικών για την βελτιστοποίηση προφίλ φόρτου εργασιών εφαρμογών. Ο αλγόριθμος που προτείνεται αξιοποιεί διάφορες γνωστές μεθόδους feature selection καθώς και το ΤΝΔ που αναπτύξαμε, με σκοπό την επιλογή ενός βέλτιστου υποσυνόλου χαρακτηριστικών που απαρτίζουν τα προφίλ των εφαρμογών.

Εργαστήκαμε με ένα σύνολο δεδομένων αποτελούμενο από περίπου 1300 εγγραφές, εκ των οποίων περίπου οι 1000 χρησιμοποιήθηκαν ως training set ενώ οι υπόλοιπες ως testing set. Οι εγγραφές αυτές προέρχονται από τις εκτελέσεις benchmarks σε πέντε διαφορετικά υπολογιστικά συστήματα. Έπειτα από ομογενοποίηση ορισμένων benchmarks καταφέραμε να αυξήσουμε την ακρίβεια των προβλέψεων του μοντέλου μας από 47,54% σε 72,79%. Μία ακόμα μεγάλη βελτίωση της πετύχαμε κάνοντας oversampling στο training set οδηγώντας την ακρίβεια στο

82,95%. Στη συνέχεια, επιχειρήσαμε περαιτέρω βελτίωση μέσω επιλογής χαρακτηριστικών. Επιλέξαμε ένα αρχικό υποσύνολο βασιζόμενοι σε στατιστικές μεθόδους και έπειτα αξιολογήσαμε διαφορετικά υποσύνολα με στόχο να εντοπίσουμε το βέλτιστο. Τελικά, επιλέγοντας το κατάλληλο υποσύνολο χαρακτηριστικών, το οποίο περιέχει 17 από τα συνολικά 38 features του αρχικού προφίλ, το ΤΝΔ που αναπτύξαμε έφτασε σε ακρίβεια προβλέψεων ίση με 87,87%.

Η μεγάλη αύξηση που παρατηρήσαμε με την ομογενοποίηση των benchmarks αλλά και με το oversampling οφείλεται σε δύο λόγους. Αφ' ενός, κάποια benchmarks παράγουν πολύ παρόμοια αποτυπώματα, και αφ' εταίρου, το αρχικό σύνολο δεδομένων έχει πολλές διαφορετικές κλάσεις και πολλά χαρακτηριστικά συγκριτικά με τον αριθμό των εγγραφών του, καθότι ο απαιτούμενος αριθμός εγγραφών αυξάνεται εκθετικά με την αύξηση των διαστάσεων του χώρου αναπαράστασης. Με βάση αυτό αντιλαμβανόμαστε πως το σύστημα θα μπορούσε να επωφεληθεί από την επαύξηση του συνόλου δεδομένων. Στην διαδικασία δημιουργίας και συλλογής νέων δειγμάτων θα ήταν καλό να χρησιμοποιηθούν benchmarks που να εκτελούν εξειδικευμένες εργασίες ώστε να μην δημιουργούν παρόμοια αποτυπώματα με άλλα. Επίσης, το σύνολο δεδομένων μας προέρχεται από 5 μηχανήματα (3 για το training set και 2 για το testing set). Θα είχε αξία να εντάξουμε εκτελέσεις από περισσότερες διαμορφώσεις υλικών υποδομών στα δεδομένα μας, καθώς όσο περισσότερα διαφορετικά συστήματα χρησιμοποιήσουμε, τόσο πιο εύκολα θα μπορεί να γενικεύσει και να κάνει hardware-agnostic προβλέψεις το μοντέλο μας.

Στη διαδικασία βελτιστοποίησης των προφίλ των εφαρμογών ο μόνος τρόπος για να είμαστε βέβαιοι ότι επιλέξαμε το πιο αποδοτικό υποσύνολο χαρακτηριστικών, θα έπρεπε να εξετάσουμε όλα τα δυνατά υποσύνολα ένα προς ένα. Από ένα σύνολο όμως 38 χαρακτηριστικών μπορούμε να σχηματίσουμε $2^{38} \approx 274,877,906,944$ διαφορετικά υποσύνολα. Υπολογιστικοί και χρονικοί περιορισμοί καθιστούν αδύνατον το να αξιολογήσουμε όλα αυτά τα υποσύνολα. Με την μεθοδολογία που εφαρμόσαμε όμως εξετάζουμε ένα μικρό αριθμό από αυτά. Αρχικά, με την επιλογή των αρχικών 10 χαρακτηριστικών αξιωματικά, μειώνουμε τον αριθμό των υποσυνόλων σε $2^{28} \approx 268,435,456$. Έπειτα, με την απόρριψη των συναφών χαρακτηριστικών πετυχαίνουμε περαιτέρω μείωση στα $2^{15} = 32,768$. Τέλος, με το να επιλέγουμε να εξετάζουμε σε κάθε βήμα μόνο τα υποσύνολα που προκύπτουν από την επιλογή του πιο αποδοτικού χαρακτηριστικού στο συγκεκριμένο βήμα, εξετάζουμε μονάχα $15+14+13+\ldots+1=120$ υποσύνολα. Καταφέρνουμε λοιπόν να μειώσουμε κατά πολύ τον αριθμό των υποσυνόλων που χρειάζεται να εξετάσουμε μέσω πληροφορημένης επιλογής των υποσυνόλων που δύνανται να δώσουν βέλτιστα αποτελέσματα όσον αφορά την ακρίβεια των προβλέψεων.

Σε μία λίγο διαφορετική προσέγγιση της επιλογής χαρακτηριστικών θα μπορούσε να μην εξερευνούμε μόνο τα υποσύνολα που δημιουργούνται με το πιο αποδοτικό χαρακτηριστικό σε κάθε βήμα. Θα μπορούσαμε να εξετάζουμε Όλα τα υποσύνολα που δημιουργούνται από τα αποδοτικότερα δύο ή τρία χαρακτηριστικά, καθώς η διαφορά στην πληροφορία που περιέχεται σε τόσο συγγενικά υποσύνολα είναι

μικρή και οι διαφορές στην απόδοση είναι συχνά αμελητέες. Επιπλέον, όπως είδαμε και στην πειραματική αξιολόγηση των επικρατέστερων υποσυνόλων, ένα υποσύνολο που πέτυχε λίγο μικρότερη ακρίβεια προβλέψεων με το 5-fold cross-validation, έχει τη δυναμική να ξεπεράσει σε απόδοση στο testing set κάποια «καλύτερα» υποσύνολα όταν δεχτεί πιο ολοκληρωμένη εκπαίδευση.

Μία ακόμη ιδέα για μελλοντική επέκταση της παρούσης θα ήταν η δημιουργία γενικών κατηγοριών εφαρμογών και η ταξινόμηση τους σε δύο βήματα. Στο πρώτο βήμα, ένα απλό μοντέλο θα αναλάμβανε την ταξινόμηση της εφαρμογής σε μία γενική κατηγορία. Στο δεύτερο βήμα, ένα πιο εξειδικευμένο μοντέλο θα ταξινομούσε πλέον την εφαρμογή σε ένα συγκεκριμένο benchmark. Έτσι, θα μπορούσαμε να επωφεληθούμε από τα χαρακτηριστικά που προσφέρουν καλύτερο διαχωρισμό των κλάσεων για την κάθε κατηγορία εφαρμογών. Δηλαδή, η κάθε κατηγορία να έχει διαφορετικό προφίλ εφαρμογών για το οποίο θα έχει γίνει βελτιστοποίηση με βάση τα δείγματα που ανήκουν στην εκάστοτε κατηγορία.

Βιβλιογραφία

[1] P. Rani and M. Sabri, “CLOUD COMPUTING: A NEW REVOLUTION IN INFORMATION TECHNOLOGY SECTOR,” *SSRN Electronic Journal*, vol. 10, pp. 53–57, Aug. 2020.

[2] P. Kansal, M. Kumar, and O. P. Verma, “Classification of Resource Management Approaches in Fog/Edge Paradigm and Future Research Prospects: A Systematic Review,” *J Supercomput*, vol. 78, no. 11, pp. 13145–13204, Jul. 2022, doi: 10.1007/s11227-022-04338-1

[3] X. Li, L. Pan, and S. Liu, “A survey of resource provisioning problem in cloud brokers,” *Journal of Network and Computer Applications*, vol. 203, p. 103384, Jul. 2022, doi: 10.1016/j.jnca.2022.103384

[4] L. Chen, “Curse of Dimensionality,” in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds., Boston, MA: Springer US, 2009, pp. 545–546. doi: 10.1007/978-0-387-39940-9_133. Available: https://doi.org/10.1007/978-0-387-39940-9_133. [Accessed: Aug. 19, 2023]

[5] S. Karanam, “Curse of Dimensionality — A ‘Curse’ to Machine Learning,” *Medium*, Aug. 11, 2021. Available: https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb. [Accessed: Aug. 19, 2023]

[6] A. Psychas, P. Dadamis, N. Kapsoulis, A. Litke, and T. Varvarigou, “Containerised Application Profiling and Classification Using Benchmarks,” *Applied Sciences*, vol. 12, no. 23, Art. no. 23, Jan. 2022, doi: 10.3390/app122312374

[7] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, “Predicting cloud performance for HPC applications before deployment,” *Future Generation Computer Systems*, vol. 87, pp. 618–628, Oct. 2018, doi: 10.1016/j.future.2017.10.048

[8] D. Wu and A. Gokhale, “A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration,” in *20th Annual International Conference on High Performance Computing*, Dec. 2013, pp. 89–98. doi: 10.1109/HiPC.2013.6799133

[9] A. Martin and V. Marangozova-Martin, “Automatic benchmark profiling through advanced workflow-based trace analysis,” *Software: Practice and Experience*, vol. 48, no. 6, pp. 1195–1217, 2018, doi: 10.1002/spe.2570

[10] J. Kumar, A. K. Singh, and R. Buyya, “Self directed learning based workload forecasting model for cloud resource management,” *Information Sciences*, vol. 543, pp. 345–366, Jan. 2021, doi: 10.1016/j.ins.2020.07.012

[11] M. Baughman, R. Chard, L. Ward, J. Pitt, K. Chard, and I. Foster, “Profiling and Predicting Application Performance on the Cloud,” in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2018, pp. 21–30. doi: 10.1109/UCC.2018.00011
[12] S. Islam, J. Keung, K. Lee, and A. Liu, “Empirical prediction models for adaptive resource provisioning in the cloud,” *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, Jan. 2012, doi: 10.1016/j.future.2011.05.027
[13] K. Cetinski and M. B. Juric, “AME-WPC: Advanced model for efficient workload prediction in the cloud,” *Journal of Network and Computer Applications*, vol. 55, pp. 191–201, Sep. 2015, doi: 10.1016/j.jnca.2015.06.001
[14] R. Chard *et al.*, “An Automated Tool Profiling Service for the Cloud,” in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 223–232. doi: 10.1109/CCGrid.2016.57
[15] B. Varghese, L. T. Subba, L. Thai, and A. Barker, “DocLite: A Docker-Based Lightweight Cloud Benchmarking Tool,” in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 213–222. doi: 10.1109/CCGrid.2016.14
[16] J. Scheuner and P. Leitner, “Estimating Cloud Application Performance Based on Micro-Benchmark Profiling,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, Jul. 2018, pp. 90–97. doi: 10.1109/CLOUD.2018.00019
[17] K. Xu, F. Wang, and L. Gu, “Profiling-as-a-Service in Multi-tenant Cloud Computing Environments,” in *2012 32nd International Conference on Distributed Computing Systems Workshops*, Jun. 2012, pp. 461–465. doi: 10.1109/ICDCSW.2012.88
[18] M. Catillo, L. Ocone, M. Rak, and U. Villano, “Auto-scaling Applications in the Cloud by Simple Indexes with Complex Loads,” in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Sep. 2020, pp. 76–81. doi: 10.1109/WETICE49692.2020.00023
[19] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, and B. B. Zhou, “Profiling Applications for Virtual Machine Placement in Clouds,” in *2011 IEEE 4th International Conference on Cloud Computing*, Jul. 2011, pp. 660–667. doi: 10.1109/CLOUD.2011.75
[20] A. Kandalintsev, R. Lo Cigno, D. Kliazovich, and P. Bouvry, “Profiling cloud applications with hardware performance counters,” in *The International Conference on Information Networking 2014 (ICOIN2014)*, Feb. 2014, pp. 52–57. doi: 10.1109/ICOIN.2014.6799664
[21] A. Ghosh, A. Schwartzbard, and M. Schatz, “Learning Program Behavior Profiles for Intrusion Detection,” presented at the 1st Workshop on Intrusion Detection and Network Monitoring (ID 99), 1999. Available: https://www.usenix.org/conference/id-99/learning-program-behavior-profiles-intrusion-detection. [Accessed: Aug. 20, 2023]
[22] S. M. Kasongo and Y. Sun, “Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset,” *J Big Data*, vol. 7, no. 1, p. 105, Nov. 2020, doi: 10.1186/s40537-020-00379-6

[23] S. C. Lee and D. V. Heinbuch, “Training a neural-network based intrusion detector to recognize novel attacks,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 31, no. 4, pp. 294–299, Jul. 2001, doi: 10.1109/3468.935046
[24] W. Lee, S. Stolfo, and K. Mok, “Adaptive Intrusion Detection: A Data Mining Approach,” *Artificial Intelligence Review*, vol. 14, May 2001, doi: 10.1023/A:1006624031083
[25] G. Forman, “An extensive empirical study of feature selection metrics for text classification [J],” *Journal of Machine Learning Research - JMLR*, vol. 3, Mar. 2003.
[26] R. J. Cascaro, B. D. Gerardo, and R. P. Medina, “Filter Selection Methods for Multiclass Classification,” in *Proceedings of the 2nd International Conference on Computing and Big Data*, in ICCBD 2019. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 27–31. doi: 10.1145/3366650.3366655. Available: https://doi.org/10.1145/3366650.3366655. [Accessed: Aug. 20, 2023]
[27] T. S. Levitt, J. M. Agosta, and T. O. Binford, “Model-Based Influence Diagrams For Machine Vision,” in *Machine Intelligence and Pattern Recognition*, M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, Eds., in Uncertainty in Artificial Intelligence, vol. 10. North-Holland, 1990, pp. 371–388. doi: 10.1016/B978-0-444-88738-2.50036-1. Available: https://www.sciencedirect.com/science/article/pii/B9780444887382500361. [Accessed: Aug. 20, 2023]
[28] D. Heckerman, “Probabilistic Similarity Networks.” arXiv, Nov. 06, 2019. doi: 10.48550/arXiv.1911.06263. Available: http://arxiv.org/abs/1911.06263. [Accessed: Aug. 20, 2023]
[29] D. J. Spiegelhalter, R. C. G. Franklin, and K. Bull, “Assessment, criticism and improvement of imprecise subjective probabilities for a medical expert system,” in *Machine Intelligence and Pattern Recognition*, M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, Eds., in Uncertainty in Artificial Intelligence, vol. 10. North-Holland, 1990, pp. 285–294. doi: 10.1016/B978-0-444-88738-2.50029-4. Available: https://www.sciencedirect.com/science/article/pii/B9780444887382500294. [Accessed: Aug. 20, 2023]
[30] V. S. Shekhawat, A. Gautam, and A. Thakrar, “Datacenter Workload Classification and Characterization: An Empirical Approach,” in *2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS)*, Dec. 2018, pp. 1–7. doi: 10.1109/ICIINFS.2018.8721402
[31] J. Peng, J. Chen, X. Zhi, M. Qiu, and X. Xie, “Research on application classification method in cloud computing environment,” *J Supercomput*, vol. 73, no. 8, pp. 3488–3507, Aug. 2017, doi: 10.1007/s11227-016-1663-5
[32] J. Zhang and R. J. Figueiredo, “Application classification through monitoring and learning of resource consumption patterns,” in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, Apr. 2006, p. 10 pp.-. doi: 10.1109/IPDPS.2006.1639378
[33] J. Zhang and R. J. Figueiredo, “Autonomic Feature Selection for Application Classification,” in *2006 IEEE International Conference on Autonomic Computing*, Jun. 2006, pp. 43–52. doi: 10.1109/ICAC.2006.1662380

[34] M. Oleszak, “Feature Selection Methods and How to Choose Them,” *neptune.ai*, Sep. 09, 2022. Available: https://neptune.ai/blog/feature-selection-methods. [Accessed: Sep. 07, 2023]
[36] “SLDM IV: Deep Learning in H2O.” Available: http://htmlpreview.github.io/?https://github.com/ledell/sldm4-h2o/blob/master/sldm4-deeplearning-h2o.html. [Accessed: Sep. 08, 2023]
[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, Art. no. 6088, Oct. 1986, doi: 10.1038/323533a0
[38] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” arXiv, Jan. 29, 2017. doi: 10.48550/arXiv.1412.6980. Available: http://arxiv.org/abs/1412.6980. [Accessed: Sep. 08, 2023]
[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.
[40] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” arXiv, Mar. 02, 2015. doi: 10.48550/arXiv.1502.03167. Available: http://arxiv.org/abs/1502.03167. [Accessed: Sep. 09, 2023]
[41] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How Does Batch Normalization Help Optimization?” arXiv, Apr. 14, 2019. doi: 10.48550/arXiv.1805.11604. Available: http://arxiv.org/abs/1805.11604. [Accessed: Sep. 11, 2023]