ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών
Προηγμένα Συστήματα Πληροφορικής
Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Σύστημα Πρόβλεψης Τρύγου Μελιού βασισμένο σε Spring Boot Framework** **Honey Harvesting Predictor with Spring Boot Framework**
Ονοματεπώνυμο Φοιτητή	Τζεμπελίκου Σοφία
Πατρώνυμο	Ευάγγελος
Αριθμός Μητρώου	ΜΠΣΠ19049
Επιβλέπων	Ευθύμιος Αλέπης, Αναπληρωτής Καθηγητής

Ημερομηνία Παράδοσης **07 / 2022**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Ευχαριστίες

Για την υλοποίηση της παρούσας διπλωματικής διατριβής, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Ευθύμιο Αλέπη για την βοήθεια και την καθοδήγησή του καθ' όλη τη διάρκεια της εκπόνησης της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλη την οικογένειά μου για την στήριξή τους κατά την διάρκεια των σπουδών μου αλλά ιδιαιτέρως τον πατέρα μου γιατί χάρη στο μελισσοκομικό του υπόβαθρο είχε καθοριστικό ρόλο ως προς το κομμάτι της έρευνας που σχετίζεται με τη μελισσοκομεία.

Περίληψη

Καθώς η ανάγκη για ανάπτυξη της επιστήμης και της τεχνολογίας αυξάνεται καθημερινά, αυξάνεται επίσης και η ανάγκη ανάπτυξης μοντέλων που μιμούνται την ανθρώπινη λογική και την ανθρώπινη κρίση. Ένα κομμάτι στο οποίο εστιάζουν πολλές έρευνες είναι η χρήση μηχανικής μάθησης με στόχο την εξαγωγή συμπερασμάτων όμοιων με την ανθρώπινη λογική. Στην παρούσα διατριβή αναλύεται η χρήση της μηχανικής μάθησης στην γεωργία και πιο συγκεκριμένα στη μελισσοκομία με σκοπό την πρόβλεψη της εποχής τρύγους των μελισσιών και στη συνέχεια γίνεται στατιστική ανάλυση με σκοπό τη σύγκριση της παραγωγής ανάμεσα σε χρονικές περιόδους αλλά και το αντίκτυπο των καιρικών συνθηκών στην παραγωγή του μελιού.

Abstract

As the need for the development of science and technology grows daily, so does the need to develop models that mimic human reason and human judgment. One area that much research focuses on is the use of machine learning to draw conclusions similar to human logic. This dissertation analyzes the use of machine learning in agriculture and more specifically in beekeeping in order to predict the harvest season of bees and then makes a statistical analysis in order to compare production between time periods and the impact of weather conditions on production. of honey.

Πήνακας Περιεχομένων

ΕΙΣΑΓΩΓΗ - ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΉ ΑΝΤΙΚΕΙΜΈΝΟΥ ---------- 6

ΕΡΕΥΝΗΤΙΚΌ ΠΕΔΊΟ ---------- 7

Συνέντευξη μελισσοκόμου ---------- 9
Τα επίπεδα δημιουργίας λογισμικού μηχανικής μάθησης. ---------- 11
Data Engineering Pipelines[7] ---------- 11
Απορρόφηση δεδομένων (Data Ingestion)[7] ---------- 12
Εξερεύνηση και Επικύρωση(Exploration and Validation)[7] ---------- 12
Καθαρισμός δεδομένων(Data Wrangling)[7] ---------- 13
Διαχωρισμός δεδομένων(Data Spliting)[7] ---------- 13
Machine Learning Pipelines[8] ---------- 13
Εκπαίδευση μοντέλου(Model Training)[7] ---------- 14
Αξιολόγηση Μοντέλου (Model Evaluation)[7] ---------- 14
Δοκιμή μοντέλου (Model Testing)[7] ---------- 14
Model Packaging[7] ---------- 15
Διαφορετικές μορφές ροών εργασίας μηχανικής μάθησης[7] ---------- 15
Forecast ---------- 16
Web-Service ---------- 16
Online Learning ---------- 17
AutoML ---------- 18
Perceptron και Multi Layer Perceptron (MLP) *---------- 19*
Perceptron ---------- 19
*Perceptron για δυαδική ταξινόμηση (**Binary Classification**) ---------- 19*
Πολυστρωματικός Perceptron (Multilayer Perceptron(MLP)) ---------- 23
Classification[17] ---------- 24

ΣΥΝΤΟΜΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΥΣΗΣ ---------- 24

ΑΝΑΛΥΤΙΚΗ ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΛΥΣΗΣ ---------- 25

Συλλογή και διαχείριση δεδομένων. ---------- 25
Υλοποίηση του DataCollectorAPI *---------- 26*
Υλοποίηση του HoneyBees API *---------- 34*
Υλοποίηση του Classifier Service ---------- 40
Accuracy[19] ---------- 41
Precision[19] ---------- 41
Recall[19] ---------- 41
F1 Scope[19] ---------- 42
Υλοποίηση του UI Service ---------- 42

ΣΥΜΠΕΡΑΣΜΑΤΑ ---------- 47

Αποτελέσματα Classifier ---------- 51

ΒΙΒΛΙΟΓΡΑΦΙΑ ---------- 52

Εισαγωγή - Σύντομη Περιγραφή Αντικειμένου

Η μέλισσα θεωρείται το πιο εργατικό πλάσμα στον πλανήτη Γη. Η κοινωνία της μέλισσας, η δομή και η λειτουργία όλων των μελών της μέσα στη κυψέλη θεωρούνται, ιδανικό μοντέλο βιώσιμης κοινωνίας. Τα μέλη της κοινωνίας των μελισσών είναι διαχωρισμένα σε τρία διαφορετικά είδη, τις εργάτριες, τους κηφήνες και τη βασίλισσα. Η βασίλισσα είναι μοναδική σε κάθε μελίσσι και ο ρόλος της είναι να είναι η μάνα του μελισσιού, είναι η μοναδική που μπορεί να γονιμοποιηθεί, άρα και να παράγει τον υπόλοιπο πληθυσμό της μικρής αυτής κοινωνίας. Η μέλισσα εργάτρια ζει κατά μέσο όρο σαράντα πέντε ημέρες και ανάλογα με την ηλικία της έχει διαφορετικούς ρόλους μέσα στην κοινωνία (καθαρίστρια, φρουρός, ανιχνευτής τροφής, ακόλουθος της βασίλισσας και νταντά για νέα μέλη της κυψέλης). Οι κηφήνες αντιθέτως έχουν μία και μοναδική εργασία κατά τη διάρκεια της ζωής τους και αυτή είναι να γονιμοποιήσουν την βασίλισσα έτσι ώστε να μεγαλώσει ο αριθμός μελισσών στη κυψέλη.

Γιατί όμως η μέλισσα είναι τόσο σημαντική για τον πλανήτη και κατά συνέπεια τον άνθρωπό;

Εκτός από την παραγωγή μελιού και άλλων προϊόντων (όπως το κερί, η γύρη και ο βασιλικός πολτός) που καταναλώνονται από τον άνθρωπο, η μέλισσα έχει ανεκτίμητη αξία καθώς έχει την ιδιότητα του γονιμοποιητή. Με τον όρο γονιμοποιητής ορίζεται η ιδιότητα της μέλισσας να μεταφέρει την γύρη από το ένα φυτό στο άλλο. Η μεταφορά αυτή αποτελεί ακρογωνιαίο λίθο της αναπαραγωγικής δραστηριότητας, επιβίωσης και εξέλιξης των φυτικών ειδών. Πάνω από το 35% της παγκόσμιας παραγωγής τροφής εξαρτάται από έντομα-επικονιαστές[1]. Καρποί όπως τα μήλα, οι φράουλες και τα αμύγδαλα θα εμφανίσουν απότομη πτώση παραγωγής σε περίπτωση που οι μέλισσες εξαφανιστούν. Η μεταφορά γύρης ενός άνθους σε ένα άλλο άνθος αποτελεί το πρώτο βήμα για την αναπαραγωγή του φυτού. Η διαδικασία αυτή ονομάζεται επικονίαση[2]. Περισσότερα από 80% των φυτών του πλανήτη είναι εντομόφυλα και η αναπαραγωγή τους εξαρτάται από γονιμοποιητές όπως οι μέλισσες. Κάθε μέλισσα κατά τη διαδικασία εύρεσης τροφής μπορεί να επισκεφθεί 50-1000 διαφορετικά φυτά ανάλογα με τη διάρκεια του ταξιδιού της, έτσι δικαιολογημένα έχει τον τίτλο του πιο αποδοτικού γονιμοποιητή.

Η μέλισσα δεν χρειάζεται τον άνθρωπο για να επιβιώσει, πολλές φορές όμως χρειάζεται την βοήθειά του καθώς οι συνθήκες επιβίωσης τους γίνονται όλο και δυσκολότερες τα τελευταία χρόνια. Η μεγαλύτερη απειλή για τις μέλισσες είναι τα πολύ τοξικά χημικά φάρμακα που χρησιμοποιούνται στις βιομηχανικές καλλιέργειες. Επίσης, οι μονοκαλλιέργειες, που καταστρέφουν πλούσια οικοσυστήματα και απειλούν πολλά είδη ανθοφόρων φυτών με εξαφάνιση, αποτελούν μεγάλο κίνδυνο για τις μέλισσες, αφού για να επιβιώσουν πρέπει να τρέφονται από μία μεγάλη ποικιλία ανθοφόρων φυτών. Οι μελισσοκόμοι δεν θα μπορούσαν να βοηθήσουν τόσο στο κομμάτι της απειλής των φαρμάκων και των μονοκαλλιεργειών καθώς θα ήταν αδύνατο ένας μελισσοκόμος να επηρεάσει ολόκληρες τις βιομηχανικές καλλιέργειες εάν δεν υπάρχει μία κοινή ευαισθητοποίηση όλων των αγροτών και προς επέκταση, ολόκληρης της κοινωνίας, προς το κομμάτι αυτό[3].

Αντιθέτως, υπάρχουν και άλλοι παράγοντες που συμβάλλουν στην εξαφάνιση των μελισσών, όπως για παράδειγμα οι κλιματικές συνθήκες, στις οποίες οι μελισσοκόμοι θα μπορούσαν να βοηθήσουν. Ένας μελισσοκόμος βοηθάει τα μελίσσια να επιβιώσουν από το κρύο του χειμώνα και την αφόρητη ζέστη του καλοκαιριού μεταφέροντας τα σε διαφορετικές περιοχές ανάλογα με την εποχή που διανύουν. Για παράδειγμα το καλοκαίρι συνήθως τα πηγαίνουν σε υψηλό υψόμετρο όπου η θερμοκρασία είναι χαμηλότερη, ενώ το χειμώνα προτιμούν περιοχές κοντά στη θάλασσα όπου και η θερμοκρασία είναι όσο το δυνατόν ψηλότερη.

Για να βοηθήσουν οι μελισσοκόμοι τις μέλισσες στην επιβίωσή τους, και κατ' επέκταση τα περισσότερα φυτά του πλανήτη που εξαρτώνται από αυτές, έχουν ως κίνητρο την παραγωγή μελιού και των υπόλοιπων προϊόντων που παράγονται από αυτήν, έτσι ώστε να μπορέσουν και εκείνοι να ωφεληθούν οικονομικά και να συνεχίσουν να τις βοηθούν. Σε αυτό το σημείο έρχεται η

παρούσα έρευνα να βοηθήσει τον σύγχρονο μελισσοκόμο στην παραγωγή του μελιού έτσι ώστε και εκείνος να συνεχίσει τις δράσεις του προς το καλό των μελισσών.

Όπως αναφέρθηκε παραπάνω, ένας μελισσοκόμος εξαρτάται οικονομικά από την παραγωγή μελιού που θα του αποφέρουν τα μελίσσια πού κατέχει στην περιουσία του. Για το λόγο αυτό θα ήταν για εκείνον πολύ χρήσιμο να μπορούσε ιδανικά να έχει στα χέρια του κάποια στατιστική ανάλυση σχετική με το κλήμα που έχει η κάθε περιοχή αλλά και την ποσότητα μελιού που έχει παράγει σε προηγούμενες χρονιές , με σκοπό την πρόβλεψη της παραγωγής της εκάστοτε χρονιάς, ίσως και τρόπους για να αυξήσει την παραγωγή αυτή. Το πρόβλημα αυτό έρχεται να λύσει η παρούσα έρευνα όπου μέσω των νέων τεχνολογιών και συγκεκριμένα της μηχανικής μάθησης, θα προσπαθήσει να κάνει μία στατιστική ανάλυση των δεδομένων που έχουν συλλεγεί από προηγούμενες χρονιές και να μας δώσει αποτελέσματα αρχικά για το εάν μία περίοδος είναι περίοδος τρύγους ή όχι και δευτερευόντως θα προσπαθήσει να παράγει στατιστικά σχετικά με το κλήμα και την απόδοση των μελισσών στην παραγωγή μελιού ανάλογα με την κάθε περίοδο.

Ερευνητικό πεδίο

Για περισσότερο από έναν αιώνα η παρακολούθηση της κοινωνίας των μελισσών έχει αποτελέσει αντικείμενο προσοχής για μελισσοκόμους αλλά και ερευνητές. Πολλές μεταβολικές περιβαλλοντικές παράμετροι έχουν θεωρηθεί ότι συμβάλλουν στην σωστή λειτουργεία αλλά και υγεία της κοινωνίας των μελισσών. Κάποιες απ' αυτές τις παραμέτρους είναι η συγκέντρωση διοξειδίου του άνθρακα (CO2) στην ατμόσφαιρα για τον προσδιορισμό του μεταβολικού ρυθμού (Seeley 1974), η θερμοκρασία και η υγρασία της κυψέλης για τη διερεύνηση της μεταβολικής δραστηριότητας και της ομοιόστασης (Southwick and Moritz 1987; Human et al. 2006), το βάρος της κυψέλης για να αποδειχθεί το μέγεθος της αποικίας και τα τροφικά αποθέματα (Meikle et al. 2008), η δόνηση και ο ήχος για να υποδείξουν την δυναμική του σμήνους (Ferrari et al. 2008) και τη δραστηριότητα αναζήτησης τροφής των μελισσών ως μέθοδος συμπεριφοράς για την αξιολόγηση των επιπτώσεων της έκθεσης σε φυτοφάρμακα (Pham-Delegue et al. 2002· Schneider et al. . 2012).

Σε προηγούμενες έρευνες[4] έχει αποδειχθεί ότι οι μέλισσες επηρεάζονται από τις καιρικές συνθήκες με διάφορους τρόπους. Για παράδειγμα, οι αγριομέλισσες συλλέγουν γύρη αντί για νέκταρ την εποχή που η θερμοκρασία είναι σε υψηλά επίπεδα, υπάρχει ξηρασία ή και θυελλώδης άνεμοι ενώ στην αντίθετη περίπτωση προτιμούν τη συγκομιδή του νέκταρ (Peat and Goulson 2005). Οι μέλισσες έχουν τη δυνατότητα να προβλέψουν την πιθανότητα βροχής στο μέλλον και αυτό αποδεικνύεται από το γεγονός ότι τις ημέρες πριν τη βροχόπτωση, οι ποσότητες συγκομιδής τροφής είναι αισθητά μεγαλύτερες απ' ότι τις υπόλοιπες ημέρες (He et al. 2016). Το γεγονός αυτό δεν καλύπτει όλα τα είδη μελισσών καθώς διαφορετικά είδη μελισσών προτιμούν να μαζεύουν την τροφή τους σε διαφορετικές θερμοκρασίες (Vicens και Bosch 2000). Η κατανόηση της σχέσης ανάμεσα στις προτιμήσεις συγκομιδής τροφής των μελισσών ανάλογα με τον καιρό θα μπορούσε αν αποτελέσει σημαντικό παράγοντα για τους μελισσοκόμους με σκοπό να διαλέξουν το καταλληλότερο είδος μελισσών για το κοπάδι τους, δεδομένου του γεωγραφικού πλάτους, της εποχής ανθοφορίας και του τοπικού κλίματος.

Η μελέτη Burrill και Dietz 1981[4] ερεύνησε την επιρροή των μελισσών στην επίτευξη μεγαλύτερων ποσοστών κατά τη συλλογή τροφής ως προς τις εξωτερικές συνθήκες με βάση δύο μεταβλητές, τη θερμοκρασία και την ηλιακή ακτινοβολία (SR). Η δραστηριότητα των μελισσών μετρήθηκε χρησιμοποιώντας ηλεκτρο-οπτικούς μετρητές που κατέγραφαν κάθε είσοδο και έξοδο των μελισσών από την κυψέλη. Η έρευνα αυτή ανέφερε θετική συσχέτιση μεταξύ της θερμοκρασίας και της δραστηριότητας των μελισσών. Σχετικά με την ηλιακή ακτινοβολία, τα δεδομένα της έρευνας αυτής δείχνουν μια θετική συσχέτιση μέχρι ένα ορισμένο όριο ακτινοβολίας (0,66 Langleys ή 460 W/m2) ακολουθούμενη από μια αρνητική συσχέτιση για υψηλότερα επίπεδα SR (> 460 W/m2). Μέσα από τα αποτελέσματα της έρευνας φαίνεται ότι ο ρυθμός αναζήτησης τροφής των μελισσών έχει άμεση σχέση με τις εξωτερικές συνθήκες και γι' αυτό το λόγο οι

μέλισσες πρέπει να είναι σε θέση να προβλέψουν τις καιρικές συνθήκες που θα επικρατήσουν στο κοντινό τους μέλλον.

Μία δεύτερη έρευνα (Automatic localization and decoding of honeybee markers using deep convolutional neural networks)[5] βασίζεται σε δύο βαθιά νευρωνικά δίκτυα για τον εντοπισμό και αποκωδικοποίηση προσαρμοσμένων δυαδικών δεικτών που φέρουν οι μέλισσες από την πρώτη τους μέρα μέχρι την τελευταία μέρα ζωή τους. Μέσα από την έρευνα αυτή, φαίνεται ότι τα σύνολα δεδομένων που προκύπτουν θα βοηθήσουν στην προώθηση της κατανόησης της συλλογικής νοημοσύνης των μελισσών.

Η παρούσα έρευνα ως επι το πλείστων είναι βασισμένη στην έρευνα Predictive modelling of honey bee foraging activity using local weather conditions, Dominic Clark 2018[4]. Στην έρευνα αυτή παρουσιάζεται μια μέθοδος που καταγράφει τη δραστηριότητα των μελισσών ανάλογα με τη βροχή, την ηλιακή ακτινοβολία, τη θερμοκρασία, την υγρασία, την ταχύτητα του ανέμου και την κατεύθυνση του ανέμου συνεχώς σε χρονική ανάλυση 1 δείγμα/λεπτό. Επίσης, στην έρευνα αυτή επισυνάψαν έναν ειδικά κατασκευασμένο ηλεκτρο-οπτικό μετρητή μελισσών πολλαπλών καναλιών σε μια εθνική κυψέλη μελισσών που βρίσκεται σε ένα χωράφι στο North Somerset, UK (γεωγραφικό πλάτος: 51,4237, γεωγραφικό μήκος: −2,6711). Ανέπτυξαν, έναν μετεωρολογικό σταθμό για την καταγραφή των μετεωρολογικών μεταβλητών στη θέση της κυψέλης. Και οι δύο συσκευές τροφοδοτούσαν δεδομένα σε μια κεντρική βάση δεδομένων για αποθήκευση δεδομένων και επακόλουθη ποσοτική ανάλυση. Οι στόχοι αυτής της μελέτης ήταν να διερευνήσει την επίδραση του καιρού στη δραστηριότητα αναζήτησης τροφής και τη χρησιμότητα της προγνωστικής μοντελοποίησης για την εμπορική παρακολούθηση της υγείας των μελισσών.

Επιπρόσθετα, παρουσιάζονται δεδομένα από πειραματικές περιόδους που καλύπτουν δύο περιόδους αναζήτησης τροφής: Ιούλιος-Σεπτέμβριος 2013 και Ιούνιος-Σεπτέμβριος 2014, χρησιμοποιώντας διαφορετική αποικία μελισσών για καθεμία. Κάθε αποικία στεγαζόταν σε μια τυπική εθνική κυψέλη με ένα μεγάλο κουτί γόνου και ένα τυπικό για αποθήκευση μελιού. Αυτές οι περίοδοι περιλαμβάνουν αρκετές ημέρες που προορίζονται για τη συντήρηση της κυψέλης (θεραπεία βαρρόα και άλλες μελισσοκομικές δραστηριότητες). Τα δεδομένα από αυτές τις μέρες παραλείφθηκαν από την ανάλυση. Συνολικά, υπάρχουν 42 πλήρεις ημέρες δεδομένων από το 2013 και 74 πλήρεις ημέρες από το 2014. Οι ημέρες που παραλείφθηκαν ήταν η 25η Ιουλίου, η 19η Αυγούστου και η 17η Σεπτεμβρίου 2013 και η 1η Αυγούστου 2014.

Ακόμη, προσδιόρισαν τη συσχέτιση μεταξύ κάθε μετρούμενης μετεωρολογικής μεταβλητής και του ρυθμού εξόδου των μελισσών (ER). Στη συνέχεια προσάρμοσαν ένα γενικευμένο γραμμικό μοντέλο (GLM) στα δεδομένα. Αξιολόγησαν τη δύναμη του μοντέλου τους να προβλέψει τη δραστηριότητα των μελισσών όταν δοθούν νέα δεδομένα, που δεν χρησιμοποιούνται στην προσαρμογή του μοντέλου. Η προγνωστική ισχύς του μοντέλου αξιολογήθηκε εξετάζοντας τον συντελεστή προσδιορισμού μεταξύ προβλεπόμενων και παρατηρούμενων τιμών από το νέο σύνολο δεδομένων.

Μέσω της έρευνάς τους συμπέραναν ότι η προσπάθεια αναζήτησης τροφής, που μετράται ως ο αριθμός των μελισσών που φεύγουν από την κυψέλη με σκοπό την αναζήτηση τροφής κάθε λεπτό, είναι στενά συνδεδεμένη με τις συμπίπτουσες μετεωρολογικές συνθήκες στην περιοχή όπου βρίσκεται η κυψέλη. Οι πιο σημαντικές από αυτές τις συνθήκες είναι η θερμοκρασία και η ηλιακή ακτινοβολία. Η περίοδος αντίδρασης μεταξύ των αλλαγών στις καιρικές συνθήκες και των αλλαγών στη δραστηριότητα αναζήτησης τροφής είναι μικρότερη από 1 λεπτό. Αυτά τα αποτελέσματα υποδηλώνουν ότι οι μέλισσες παρακολουθούν συνεχώς το εξωτερικό τους περιβάλλον και χρησιμοποιούν τις πληροφορίες που αποκτούν για να λάβουν αποφάσεις σχετικά με το αν θα ξεκινήσουν ταξίδια αναζήτησης τροφής. Οι αισθητηριακοί βιολογικοί μηχανισμοί που μεσολαβούν σε αυτό το φαινόμενο σε ατομικό επίπεδο μέλισσας ή/και επίπεδο κυψέλης, δεν είναι καλά κατανοητοί και δικαιολογούν περαιτέρω διερεύνηση.

Η παρούσα εργασία έχει σκοπό τη δημιουργία μίας ολοκληρωμένης εφαρμογής στατιστικών σχετικών με την παραγωγή μελιού κατά τους καλοκαιρινούς μήνες. Κατά την έρευνα ως μέσο συλλογής δεδομένων χρησιμοποιήθηκαν ηλεκτρονικές ζυγαριές τις οποίες χρησιμοποιούν οι μελισσοκόμοι για τον υπολογισμό τού μελιού που παράγεται καθημερινά. Στην

περίπτωσή μας, για τη σωστή εκπόνηση της έρευνας χρειάστηκε να συνεργαστούμε με έναν συγκεκριμένο μελισσοκόμο για να μπορέσουμε να βγάλουμε συμπεράσματα για τα κοπάδια του. Ο μελισσοκόμος αυτός χρησιμοποιεί ενδεικτικά μία ζυγαριά σε κάθε κοπάδι του κατά τους καλοκαιρινούς μήνες. Η ζυγαριά αυτή έχει τη δυνατότητα να αποστέλλει μία φορά την ημέρα στο κινητό του μελισσοκόμου ένα SMS το οποίο περιέχει το βάρος της κυψέλης στην οποία είναι τοποθετημένη, την εξωτερική θερμοκρασία, την υγρασία και την ημερομηνία κατά την οποία έγινε η μέτρηση.

Συνέντευξη μελισσοκόμου

Εκτός από τις πληροφορίες που αποσπάσαμε μέσω της ζυγαριάς, αποφασίστηκε πως θα ήταν σωστό κατά τη διαδικασία συλλογής των δεδομένων να διεξαχθεί μία συνέντευξη με τον ίδιο τον μελισσοκόμο έτσι ώστε να διασφαλιστούν όσο το δυνατόν περισσότερες πληροφορίες σχετικά με τις διεργασίες που γίνονται στα μελίσσια την περίοδο πού μελετάμε και την γενική του άποψη, όπως και τις παρατηρήσεις του σχετικά με το θέμα. Παρακάτω παρατίθεται το ερωτηματολόγιο καθώς και οι απαντήσεις του ειδικού:

Ερώτηση 1: Πόσα μελίσσια έχετε στην κατοχή σας (συνολικό αριθμό);

Απάντηση : Ο συνολικός αριθμός κυψελών που κατέχω αυτή την περίοδο είναι τριακόσια είκοσι (320). Αυτό δεν σημαίνει ότι θα κατέχω τον ίδιο αριθμό και την επόμενη χρονιά. Τα μελίσσια είναι ζωντανός οργανισμός και υπάρχει πιθανότητα κάποια από αυτά να πεθάνουν, είτε να μπορέσουμε την άνοιξη να συλλέξουμε τους γόνους τους, οπότε ο αριθμός αυτός αλλάζει από χρονιά σε χρονιά.

Ερώτηση 2: Οι κυψέλες αυτές είναι οργανωμένες σε κοπάδια και αν ναι πώς είναι χωρισμένα αριθμητικά;

Απάντηση : Ναι αυτός ο αριθμός των κυψελών είναι χωρισμένος σε τέσσερα κοπάδια συνολικά για να μπορούμε να τα διαχειριζόμαστε καλύτερα. Έχουν χωριστεί σε τρία μεγάλα κοπάδια και ένα μικρότερο. Τα τρία μεγάλα κοπάδια αποτελούνται από 90 κυψέλες το καθένα ενώ το τελευταίο αποτελείται από 50 κυψέλες.

Ερώτηση 3: Από πόσες μέλισσες αποτελείται περίπου το κάθε μελίσσι;

Απάντηση: Το κάθε μελίσσι αποτελείται περίπου από σαράντα χιλιάδες (40.000) μέλισσες.

Ερώτηση 4: Ποιες εργασίες γίνονται στα μελίσσια κατά τους καλοκαιρινούς μήνες και ποιες από αυτές βοηθούν στην αύξηση της παραγωγής του μελιού;

Απάντηση : Η όλη διαδικασία ξεκινά από τον χειμώνα, για να έχουμε μεγαλύτερη παραγωγή μελιού θα πρέπει για αρχή να έχουμε δυνατά μελίσσια. Μιλώντας για δυνατά μελίσσια εννοούμε μελίσσια χωρίς ασθένειες, με σωστές βασίλισσες τα οποία βρίσκονται σε μέρη που μπορούν να βρουν μόνα τους τροφή κατά τον χειμώνα. Οπότε τον χειμώνα μεταφέρουμε τα μελίσσια σε ένα μέρος με χαμηλό υψόμετρο, συνήθως κοντά στη θάλασσα έτσι ώστε να ανταπεξέλθουν στις καιρικές συνθήκες.

Η βασίλισσα της κάθε κυψέλης το καλοκαίρι θα πρέπει να είναι δυνατή και νέα, για το λόγο αυτό συνήθως τον Μάιο, ίσως και λίγο νωρίτερα (εξαρτάται από της καιρικές συνθήκες της κάθε χρονιάς και τη δυναμική του εκάστοτε κοπαδιού) ξεκινάμε να κάνουμε βασιλοτροφία. Βασιλοτροφία είναι η διαδικασία κατά την οποία το κάθε μελίσσι δημιουργεί έναν αριθμό από βασίλισσες. Το κάθε μελίσσι όμως χρειάζεται μόνο μία βασίλισσα. Όταν η νέα βασίλισσα γεννηθεί και γίνει αρκετά δυνατή γίνεται μάχη ανάμεσα στην νέα βασίλισσα και στην παλιά, η βασίλισσα

που θα χάσει την μάχη παίρνει έναν αριθμό μελισσών της κυψέλης και φεύγει με σκοπό να βρουν ένα νέο μέρος για να μείνουν. Έτσι λοιπόν δημιουργείται ο γόνος που αναφέρθηκε προηγουμένως.

Από τη στιγμή που οι βασίλισσα η οποία έχασε τη μάχη, παίρνει τους ακόλουθούς της και απομακρύνεται μέχρι τη στιγμή που θα βρεθεί το νέο τους σπίτι, μαζεύονται όλες οι μέλισσες μαζί σε κάποιο από τα δέντρα γύρω από την παλιά τους κυψέλη μέχρι οι μέλισσες ανιχνευτές της βασίλισσας να βρουν το νέο τους σπίτι. Κατά την περίοδο αυτή, η οποία διαρκεί κάποιες ώρες, εμείς οι μελισσοκόμοι μπορούμε να βρούμε τον γόνο αυτό που βρίσκεται σε κατάσταση εύρεσης νέου σπιτιού και να τον μαζέψουμε βάζοντάς τον σε μία νέα κυψέλη. Έτσι δημιουργείται ένα νέο μελίσσι για το κοπάδι μας. Ένας άλλος τρόπος να αλλάξεις τη βασίλισσα για να κάνεις δυνατότερο το μελίσσι σου είναι να σκοτώσεις την παλιά βασίλισσα έτσι ώστε τα μελίσσια να δημιουργήσουν από μόνα τους μία νέα.

Μία άλλη διαδικασία για την αύξηση της παραγωγής μελιού και ίσως η πιο σημαντική είναι οι νομές. Αφού όλο το χειμώνα τα μελίσσια να μας ανταπεξήλθαν από τις χειμερινές καιρικές συνθήκες και έγιναν όλες οι διαδικασίες για να δυναμώσουν την άνοιξη, έρχεται ο καιρός να μπουν δυναμικά στη συλλογή μελιού. Τους μήνες από τον Ιούνιο μέχρι τον Αύγουστο, δηλαδή καθ'όλη την καλοκαιρινή περίοδο εμείς οι μελισσοκόμοι θα πρέπει να μεταφέρουμε τα κοπάδια μας σε μέρη τα οποία θα ευνοήσουν την παραγωγή. Για αρχή ένα τέτοιο μέρος θα πρέπει να βρίσκεται σε ένα μεγάλο υψόμετρο, σε αντίθεση με τον χειμώνα. Αυτό το κάνουμε γιατί το καλοκαίρι σε χαμηλά υψόμετρα έχει καύσωνα και υγρασία, κάτι που δεν βοηθάει τα μελίσσια όχι μόνο στο να παράγουν μέλι αλλά ακόμα και στο να επιζήσουν.

Συνεχίζοντας με τον τρόπο που διαλέγουμε το μέρος όπου θα μεταφέρουμε τα μελίσσια, προσέχουμε την ανθοφορία του μέρους αυτού. Ένας μελισσοκόμος θα πρέπει να γνωρίζει την εποχή ανθοφορίας του κάθε φυτού από το οποίο θέλει να μαζέψει μέλι. Για παράδειγμα εάν θέλεις να έχεις παραγωγή από μέλι ελάτης θα πρέπει να γνωρίζεις ότι το έλατο έχει εποχές ανθοφορίας την άνοιξη, δηλαδή περίπου στα μέσα Μαΐου, η εποχή αυτή συμπίπτει με την εποχή αλλαγής βασιλισσών.

Και τέλος άλλος ένας βασικός παράγοντας είναι ο παράγοντας της τύχης. Αφού ο μελισσοκόμος έχει κάνει όλες τις δυνατές διαδικασίες για να βοηθήσει τα μελίσσια του να εξελιχθούν έτσι ώστε να παράγουν περισσότερο μέλι, υπάρχουν και οι αστάθμητοι παράγοντες τους οποίους δεν μπορεί να ρυθμίσει όπως για παράδειγμα ο καιρός.

Ερώτηση 5: Ποιες από τις εργασίες που γίνονται στα μελίσσια εκτιμάτε ότι μπορεί να μειώνουν την παραγωγή μελιού;

Απάντηση: Συνήθως η εποχή που έχουμε κάποια παραγωγή μελιού αλλά όχι την ποσότητα που θα έπρεπε είναι η άνοιξη αλλά και ο χειμώνας που δεν έχουμε καθόλου παραγωγή. Το χειμώνα είναι φυσιολογικό να μην υπάρχει παραγωγή λόγω των κακών καιρικών συνθηκών αλλά και των εναργειών που γίνονται από τους μελισσοκόμους για την αντιμετώπιση των ασθαινιών. Κάποιες από τις ασθένειες που προσβάλουν τις μέλισσες είναι η Βαρόα ή αλλιως άκαρη που είναι και η πιο συνηθισμένη, η νοζεμίαση , όπως και κάποιες από τις ασθένειες του γόνου οι οποίες καταστρέφουν ολοκληρωτικά το μελίσσι αφού σκοτώνουν τα νεογνά.

Αντίθετα όμως την άνοιξη λόγω της αυξημένης ανθοφορίας θα έπρεπε να υπήρχε μεγαλύτερη παραγωγή. Η μειωμένη παραγωγή οφείλεται στις εργασίες που γίνονται την άνοιξη με σκοπό να δυναμώσουν τα μελίσσια αλλά και στην παραγωγή βασιλικού πολτού που τα εξαντλεί.

Άλλος ένας σημαντικός παράγοντας που συμβάλλει στη μείωση της απόδοσης παραγωγής είναι η περίοδος τρύγου. Τρύγος ονομάζεται η περίοδος όπου ο μελισσοκόμος θα πρέπει να συλλέξει από την κάθε κυψέλη το μέλι. Αυτή η περίοδος μειώνει την παραγωγή γιατί οι μέλισσες καταλαβαίνοντας ότι κάποιος "κλέβει" την τροφή τους, στρεσάρονται με αποτέλεσμα να μην έχουν την απόδοση που θα έπρεπε μετά από την περίοδο αυτή. Τέλος, κατά τον τρύγο,

πολλές από τις μέλισσες χάνονται ή πεθαίνουν με αποτέλεσμα μετά από αυτή την περίοδο, ο πληθυσμός της κυψέλης να είναι ορατά μειωμένος.

Ερώτηση 6: Ποια είναι τα φάρμακα που χορηγούνται στα μελίσσια για να αναρρώσουν από τις ασθένειες και πόσο αυτά επηρεάζουν την ποσότητα της παραγωγής αλλά και την ποιότητα του μελιού;

Απάντηση: Όπως ανέφερα και νωρίτερα, οι εργασίες που σχετίζονται με την προετοιμασία των μελισσιών για την παραγωγική περίοδο, δηλαδή το καλοκαίρι, γίνονται συνήθως κατά τη διάρκεια της άνοιξης αλλά και του χειμώνα. Ειδικότερα, εργασίες όπως η καταπολέμηση των ιών που πλήττουν τα μελίσσια γίνονται το χειμώνα και κατά προτίμηση πριν ακόμα έρθει η άνοιξη. Αυτό δεν γίνεται τυχαία, με τη λογική ότι τα μελίσσια το χειμώνα δεν παράγουν μέλι, αποφασίζουμε να χορηγήσουμε εκείνη την εποχή όποιο φάρμακο και αν χρειάζονται για να αναρρώσουν με σκοπό οι ουσίες του φαρμάκου αυτού να μην περάσουν στο μέλι που θα παραχθεί. Πιο απλά από τη στιγμή που χορηγούμε τα φάρμακα το χειμώνα μέχρι τη στιγμή που θα παράξουν οι μέλισσες το μέλι, οι ουσίες των φαρμάκων έχουν εξαλειφθεί και έτσι καταφέρνουμε να παράξουμε ένα καθαρό προϊόν.

Ερώτηση 7: Ποια είναι η ποσότητα μελιού ανα τρύγο το καλοκαιρί (περίπου);

Απάντηση: Δεν μπορώ να απαντήσω αυτή την ερώτηση με σιγουριά, η κάθε χρονιά είναι διαφορετική. Μπορώ όμως να σου πω ότι στατιστικά ένα δυνατό μελίσσι σε μία καλή χρονιά (από την άποψη ότι υπήρχαν σωστές καιρικές συνθήκες και δεν χάσαμε πολλά μελίσσια από τα ραντίσματα των γεωργών), μπορεί να παράγει κατά μέσω όρο 40 κιλά μελιού σε κάθε τρύγο. Αντίθετα σε μία χρονιά που δεν είναι τόσο παραγωγική, αυτός ο αριθμός είναι ορατά αισθητός. Για παράδειγμα το 2021 που δεν ήταν καλή χρονιά , το κάθε μελίσσι παρήγαγε περίπου 15 κιλά ανά τρύγο.

Τα επίπεδα δημιουργίας λογισμικού μηχανικής μάθησης.

Το Machine Learning υιοθετείται γρήγορα από νέες εφαρμογές και βιομηχανίες. Ο στόχος ενός έργου μηχανικής μάθησης είναι η κατασκευή ενός στατιστικού μοντέλου χρησιμοποιώντας συλλεγόμενα δεδομένα και την εφαρμογή αλγορίθμων μηχανικής μάθησης. Ωστόσο, η δημιουργία επιτυχημένων έργων λογισμικού που βασίζονται σε Machine Learning εξακολουθεί να είναι δύσκολη, επειδή κάθε λογισμικό που βασίζεται σε Machine Learning πρέπει να διαχειρίζεται τρία κύρια στοιχεία: Δεδομένα, Μοντέλο και Κώδικα. Το MLOps, ως επέκταση του DevOps, καθιερώνει αποτελεσματικές πρακτικές και διαδικασίες γύρω από το σχεδιασμό, την κατασκευή και την ανάπτυξη μοντέλων Machine Learning στην παραγωγή. Παρακάτω περιγράφονται βασικές τεχνικές μεθοδολογίες, οι οποίες εμπλέκονται στην ανάπτυξη του λογισμικού που βασίζεται στη Μηχανική Εκμάθηση και έχουν. Χρησιμοποιηθεί στην παρούσα έρευνα, δηλαδή Data Engineering, ML Model Engineering και Software Release Engineering.[6]

Data Engineering Pipelines[7]

Το θεμελιώδες κομμάτι οποιασδήποτε ροής εργασίας μηχανικής εκμάθησης είναι τα Δεδομένα. Η συλλογή καλών συνόλων δεδομένων έχει τεράστιο αντίκτυπο στην ποιότητα και την απόδοση του μοντέλου Machine Learning. Η περίφημη παραπομπή «Garbage In, Garbage Out», στο πλαίσιο μηχανικής μάθησης σημαίνει ότι το μοντέλο Machine Learning είναι τόσο καλό όσο τα δεδομένα που του παρέχονται. Επομένως, τα δεδομένα, τα οποία έχουν χρησιμοποιηθεί για την εκπαίδευση του μοντέλου Machine Learning, επηρεάζουν έμμεσα τη συνολική απόδοση του συστήματος παραγωγής. Η ποσότητα και η ποιότητα του συνόλου δεδομένων είναι συνήθως συγκεκριμένου προβλήματος και μπορούν να ανακαλυφθούν εμπειρικά.

Ως σημαντικό βήμα, η μηχανική δεδομένων αναφέρεται ως πολύ χρονοβόρα. Μπορεί να ξοδεύουμε τον περισσότερο χρόνο σε ένα έργο μηχανικής εκμάθησης για την κατασκευή συνόλων δεδομένων, τον καθαρισμό και τη μετατροπή δεδομένων.

Ο αγωγός μηχανικής δεδομένων περιλαμβάνει μια ακολουθία λειτουργιών στα διαθέσιμα δεδομένα. Ο τελικός στόχος αυτών των λειτουργιών είναι η δημιουργία συνόλων δεδομένων εκπαίδευσης και δοκιμής για τους αλγόριθμους Machine Learning. Στη συνέχεια, περιγράφουμε κάθε στάδιο του αγωγού μηχανικής δεδομένων, όπως η απορρόφηση δεδομένων, η εξερεύνηση και η επικύρωση, η διαμάχη δεδομένων (καθαρισμός) και ο διαχωρισμός δεδομένων.

Απορρόφηση δεδομένων (Data Ingestion)[7]

Απορρόφηση δεδομένων είναι η συλλογή δεδομένων με χρήση διαφόρων συστημάτων, πλαισίων και μορφών, όπως internal/external databases, data marts, OLAP cubes, data warehouses, OLTP systems, Spark, HDFS κ.λπ. Αυτό το βήμα μπορεί επίσης να περιλαμβάνει παραγωγή συνθετικών δεδομένων ή εμπλουτισμό δεδομένων. Οι βέλτιστες πρακτικές για αυτό το βήμα περιλαμβάνουν τις ακόλουθες ενέργειες που θα πρέπει να αυτοματοποιηθούν στο μέγιστο:

- Ταυτοποίηση πηγών δεδομένων: Εύρεση δεδομένων και τεκμηρίωση της προέλευσής τους.
- Εκτίμηση χώρου: Έλεγχος του χώρου αποθήκευσης που θα χρειαστεί.
- Τοποθεσία χώρου: Δημιουργία ενός χώρου εργασίας με αρκετό χώρο αποθήκευσης.
- Λήψη δεδομένων: Λήψη δεδομένων και μετατροπή τα σε μορφή που είναι εύκολα διαχειρήσιμη χωρίς να αλλάξουν τα ίδια τα δεδομένα.
- Δημιουργία αντιγράφων ασφαλείας δεδομένων: Η επεξεργασία θα πρέπει πάντα να γίνεται σε ένα αντίγραφο των δεδομένων και να διατηρείται ανέγγιχτο το αρχικό σύνολο δεδομένων.
- Συμμόρφωση με το απόρρητο: Εάν τα δεδομένα περιέχουν ευαίσθητες πληροφορίες, αυτές θα πρέπει να διαγράφονται ή να προστατεύονται (π.χ. ανωνυμοποιούνται) για να διασφαλιστεί η συμμόρφωση με τον GDPR.
- Κατάλογος μεταδεδομένων: Τεκμηρίωση των μεταδεδομένων του συνόλου δεδομένων καταγράφοντας τις βασικές πληροφορίες σχετικά με το μέγεθος, τη μορφή, τα ψευδώνυμα, τον χρόνο τελευταίας τροποποίησης και τις λίστες ελέγχου πρόσβασης.

Εξερεύνηση και Επικύρωση(Exploration and Validation)[7]

Η διαδικασία αυτή περιλαμβάνει τη δημιουργία προφίλ δεδομένων για τη λήψη πληροφοριών σχετικά με το περιεχόμενο και τη δομή των δεδομένων. Η έξοδος αυτού του βήματος είναι ένα σύνολο μεταδεδομένων, όπως το μέγιστο, το ελάχιστο και ο μέσος όρος των τιμών. Οι λειτουργίες επικύρωσης δεδομένων είναι συναρτήσεις ανίχνευσης σφαλμάτων που ορίζονται από το χρήστη, οι οποίες σαρώνουν το σύνολο δεδομένων για να εντοπίσουν ορισμένα σφάλματα. Η επικύρωση είναι μια διαδικασία αξιολόγησης της ποιότητας των δεδομένων εκτελώντας ρουτίνες επικύρωσης δεδομένων (μέθοδοι ανίχνευσης σφαλμάτων). Για παράδειγμα, για τα χαρακτηριστικά "διεύθυνση", είναι συνεπή τα στοιχεία διεύθυνσης; Συσχετίζεται ο σωστός ταχυδρομικός κώδικας με τη διεύθυνση; Λείπουν τιμές στα σχετικά χαρακτηριστικά; Οι βέλτιστες πρακτικές για αυτό το βήμα περιλαμβάνουν τις ακόλουθες ενέργειες:

- Χρήση των εργαλείων RAD: Η χρήση φορητών υπολογιστών Jupyter είναι ένας καλός τρόπος για να διατηρούνται αρχεία εξερεύνησης και πειραματισμών δεδομένων.
- Χαρακτηριστικό προφίλ: Λήψη και τεκμηρίωση μεταδεδομένων για κάθε χαρακτηριστικό, όπως π.χ : Όνομα, Αριθμός Εγγραφών, Τύπος δεδομένων (κατηγορικός, αριθμητικός, int/float, κείμενο, δομημένο κ.λπ.), Αριθμητικά μέτρα (ελάχιστο, μέγιστο, μέσος όρος, διάμεσος κ.λπ. για αριθμητικά δεδομένα), Ποσότητα τιμών που λείπουν (ή "αναλογία

τιμής που λείπει" = Αριθμός απόντων τιμών/ Αριθμός εγγραφών), Τύπος κατανομής (Gaussian, ομοιόμορφη, λογαριθμική κ.λπ.)

- Αναγνώριση χαρακτηριστικού ετικέτας: Για εποπτευόμενες μαθησιακές εργασίες, προσδιορίστε το(τα) χαρακτηριστικό(α) στόχο(ους).
- Οπτικοποίηση δεδομένων: Δημιουργία μιας οπτικής αναπαράστασης για διανομή αξίας.
- Συσχέτιση χαρακτηριστικών: Υπολογισμός και ανάυση των συσχετίσεων μεταξύ των χαρακτηριστικών.
- Πρόσθετα δεδομένα: Προσδιορισμός δεδομένων που θα ήταν χρήσιμα για τη δημιουργία του μοντέλου.

Καθαρισμός δεδομένων(Data Wrangling)[7]

Ο καθαρισμός δεδομένων είναι ένα βήμα προετοιμασίας δεδομένων όπου μπερδεύουμε δεδομένα μέσω προγραμματισμού, π.χ. επαναμορφοποιώντας ή αναδιαρθρώνοντας συγκεκριμένα χαρακτηριστικά που ενδέχεται να αλλάξουν τη μορφή του σχήματος των δεδομένων. Η συνηστόμενη τακτική είναι να γραφτούν σενάρια ή συναρτήσεις για όλους τους μετασχηματισμούς δεδομένων, για να χρησιμοποιήθούν ξανά όλες αυτές τις λειτουργίες σε μελλοντικά δεδομένα.

- Μετασχηματισμοί: Προσδιορισμός όλων των πολλά υποσχόμενων μετασχηματισμων που μπορεί να χρειαστεί να εφαρμοστούν .
- Outliers: Διόρθωση ή αφαίρεση των ακραίων σημείων (προαιρετικά).
- Τιμές που λείπουν: Συμπλήρωση των τιμων που λείπουν (π.χ. με μηδέν, μέσο όρο, διάμεσος) ή διαγραφή των σειρων ή των στήλων τους.
- Μη σχετικά Δεδομένα: Κατάργηση των χαρακτηριστικων που δεν παρέχουν χρήσιμες πληροφορίες για την εργασία.
- Δεδομένα αναδιάρθρωσης: Ενδέχεται να περιλαμβάνουν τις ακόλουθες λειτουργίες (από το βιβλίο "Αρχές διαμάχης δεδομένων")
    i. Αναδιάταξη πεδίων εγγραφής με μετακίνηση στηλών
    ii. Δημιουργία νέων πεδίων εγγραφής μέσω εξαγωγής τιμών
    iii. Συνδυασμός πολλαπλών πεδίων εγγραφής σε ένα ενιαίο πεδίο εγγραφής
    iv. Φιλτράρισμα συνόλων δεδομένων με κατάργηση συνόλων εγγραφών
    v. Μετατόπιση της ευκρίνειας του συνόλου δεδομένων και των πεδίων που σχετίζονται με εγγραφές μέσω συναθροίσεων και περιστροφών.

Διαχωρισμός δεδομένων(Data Spliting)[7]

Διαχωρισμός των δεδομένων σε σύνολα δεδομένων εκπαίδευσης και δοκιμής που θα χρησιμοποιηθούν κατά τα βασικά στάδια μηχανικής εκμάθησης για την παραγωγή του μοντέλου Machine Learning.

Machine Learning Pipelines[8]

Ο πυρήνας της ροής εργασίας Machine Learning είναι η φάση της σύνταξης και της εκτέλεσης αλγορίθμων μηχανικής μάθησης για την απόκτηση ενός μοντέλου Machine Learning. Ο αγωγός μηχανικής μοντέλων χρησιμοποιείται συνήθως από μια ομάδα επιστήμης δεδομένων και περιλαμβάνει έναν αριθμό λειτουργιών που οδηγούν σε ένα τελικό μοντέλο. Αυτές οι λειτουργίες περιλαμβάνουν την εκπαίδευση του Μοντέλου, την αξιολόγηση του μοντέλου, τη δοκιμή του μοντέλου και το packaging του Μοντέλου.

Εκπαίδευση μοντέλου(Model Training)[7]

Εκπαίδευση μοντέλου (Model training) είναι η διαδικασία εφαρμογής του αλγορίθμου μηχανικής μάθησης σε δεδομένα εκπαίδευσης για την εκπαίδευση ενός μοντέλου Machine Learning. Περιλαμβάνει επίσης μηχανική χαρακτηριστικών (feature engineering), δηλαδή τον συντονισμό υπερπαραμέτρων για τη δραστηριότητα εκπαίδευσης του μοντέλου. Η ακόλουθη λίστα υιοθετήθηκε από το «Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow» του Aurélien Géron
Η μηχανική χαρακτηριστικών μπορεί να περιλαμβάνει:

- Διακριτικοποίηση συνεχών χαρακτηριστικών
- Αποσύνθεση χαρακτηριστικών (π.χ. κατηγορικές, ημερομηνία/ώρα κ.λπ.)
- Προσθήκη μετασχηματισμών χαρακτηριστικών (π.χ. log(x), sqrt(x), x2, κ.λπ.)
- Συγκέντρωση χαρακτηριστικών σε πολλά υποσχόμενα νέα χαρακτηριστικά
- Κλιμάκωση χαρακτηριστικών: Τυποποίηση ή κανονικοποίηση χαρακτηριστικών

Η Μηχανική Μοντέλων μπορεί να είναι μια επαναληπτική διαδικασία και να περιλαμβάνει την ακόλουθη ροή εργασίας:

- Κάθε προδιαγραφή μοντέλου Machine Learning (κώδικας που δημιουργεί ένα μοντέλο Machine Learning) πρέπει να υποβάλλεται σε έλεγχο κώδικα και να εκδοθεί.
- Εκπαίδευση πολλών μοντέλων Machine Learning από διαφορετικές κατηγορίες (π.χ. γραμμική παλινδρόμηση, λογιστική παλινδρόμηση, k-means, naive Bayes, SVM, Random Forest, κ.λπ.) χρησιμοποιώντας τυπικές παραμέτρους.
- Μέτρηση και σύγκρηση της απόδοσής τους. Για κάθε μοντέλο, χρήση διασταυρούμενης επικύρωσης N-fold και υπολογίσμός της μέσης και τυπικής απόκλισης του μέτρου απόδοσης στις πτυχές N.
- Ανάλυση σφαλμάτων: ανάλυση των τύπων σφαλμάτων που κάνουν τα μοντέλα Machine Learning.
- Περαιτέρω εξέταση της επιλογής χαρακτηριστικών και τη μηχανικής.
- Προσδιορισμός των κορυφαίων τρίων έως πέντε μοντέλων με τα περισσότερα υποσχόμενα μοντέλα, προτιμώντας μοντέλα που κάνουν διαφορετικούς τύπους σφαλμάτων.
- Συντονισμός υπερπαραμέτρων με χρήση διασταυρούμενης επικύρωσης (cross-validation). Λαμβάνοντας υπόψη ότι οι επιλογές μετασχηματισμού δεδομένων είναι επίσης υπερπαράμετροι.

Αξιολόγηση Μοντέλου (Model Evaluation)[7]

Αξιολόγηση του μοντέλου είναι η επικύρωση του εκπαιδευόμενου μοντέλου για την βεβαίωση ότι πληρεί τους αρχικούς επιχειρηματικούς στόχους πριν από την προώθησή του στην παραγωγή και από εκεί στον τελικό χρήστη.

Δοκιμή μοντέλου (Model Testing)[7]

Μόλις εκπαιδευτεί το τελικό μοντέλο μηχανικής μάθησης, η απόδοσή του πρέπει να μετρηθεί χρησιμοποιώντας το σύνολο δεδομένων για την εκτίμηση του σφάλματος γενίκευσης εκτελώντας την τελική «Δοκιμή αποδοχής μοντέλου» (Model Acceptance Test).

Model Packaging[7]

Model Packaging είναι η διαδικασία εξαγωγής του τελικού μοντέλου Machine Learning σε μια συγκεκριμένη μορφή (π.χ. PMML, PFA ή ONNX), η οποία περιγράφει το μοντέλο που θα καταναλωθεί από την επιχειρηματική εφαρμογή.

Διαφορετικές μορφές ροών εργασίας μηχανικής μάθησης[7]

Η λειτουργία ενός μοντέλου μηχανικής μάθησης μπορεί να λάβει πολλά αρχιτεκτονικά στυλ. Παρακάτω, συζητάμε τέσσερα αρχιτεκτονικά μοτίβα που ταξινομούνται σε δύο διαστάσεις:

1. Machine Learning Model Training

2. Machine Learning Model prediction

Στο σημείο αυτό θα πρέπει να ληφθεί υπόψη ότι για λόγους απλότητας, θα πρέπει να αγνοηθεί η τρίτη διάσταση 3. Machine Learning Model type, που υποδηλώνει τον τύπο του αλγόριθμου μηχανικής εκμάθησης που χρησιμοποιείται, όπως εποπτευόμενη, μη εποπτευόμενη, ημιεποπτευόμενη και ενισχυτική μάθηση.

Υπάρχουν δύο τρόποι με τους οποίους εκτελούμε την Εκπαίδευση Μοντέλου μηχανικής μάθησης:

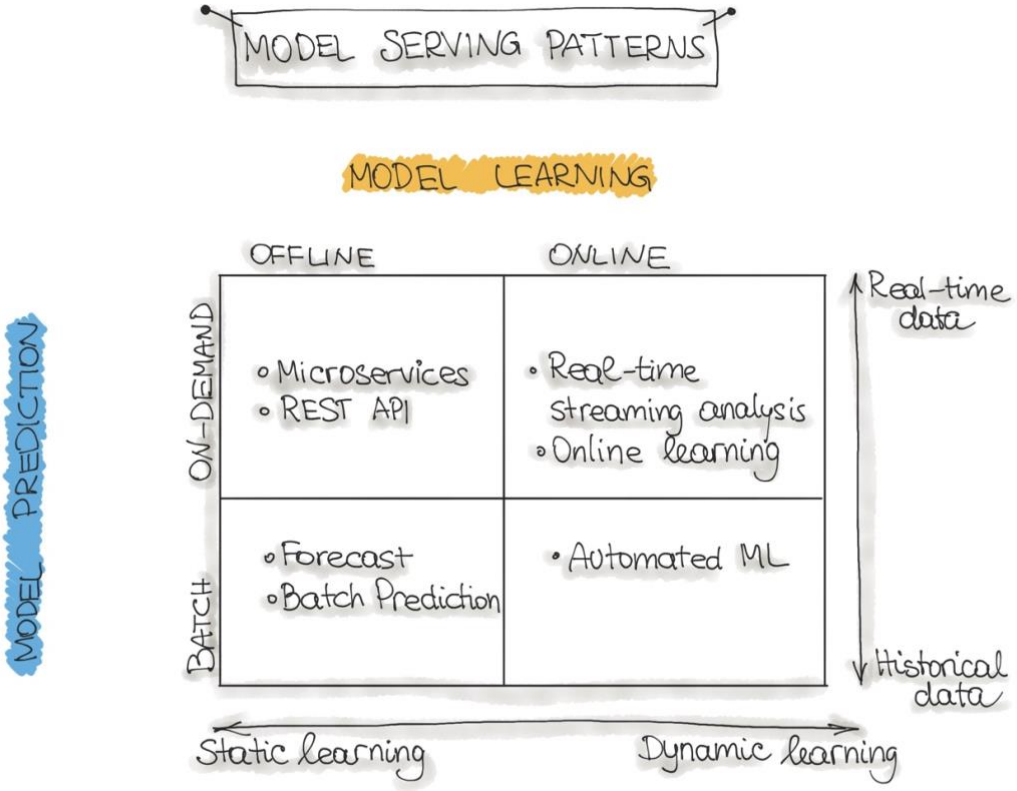
Εκμάθηση εκτός σύνδεσης (γνωστή και ως στατική μάθηση): Το μοντέλο εκπαιδεύεται σε ένα σύνολο δεδομένων που έχουν ήδη συλλεχθεί. Μετά την ανάπτυξη στο περιβάλλον παραγωγής, το μοντέλο μηχανικής μάθησης παραμένει σταθερό έως ότου εκπαιδευτεί εκ νέου. Η εκ νέου εκπαίδευση του μοντέλου πραγματοποιείται επειδή θα υπάρξουν νέα πραγματικά δεδομένα και έτσι κάποια από τα δεδομένα αυτά θα πρέπει να συμπεριληφθούν στη διαδικασία της εκπαίδευσης. Αυτό το φαινόμενο ονομάζεται «διάσπαση του μοντέλου» (model decay) και θα πρέπει να παρακολουθείται προσεκτικά.

Διαδικτυακή μάθηση (γνωστή και ως δυναμική μάθηση): Το μοντέλο επανεκπαιδεύεται τακτικά καθώς έρχονται νέα δεδομένα, π.χ. ως ροές δεδομένων. Αυτό ισχύει συνήθως για συστήματα μηχανικής μάθησης που χρησιμοποιούν δεδομένα χρονοσειράς, όπως αισθητήρες ή δεδομένα συναλλαγών μετοχών για να προσαρμόσουν τα χρονικά αποτελέσματα στο μοντέλο.

Η δεύτερη διάσταση είναι η Πρόβλεψη Μοντέλου μηχανικής μάθησης (Machine Learning Model prediction), η οποία υποδηλώνει τη μηχανική του μοντέλου ML για να κάνει προβλέψεις. Εδώ διακρίνουμε επίσης δύο τρόπους:

- Προβλέψεις παρτίδας: Το αναπτυγμένο μοντέλο μηχανικής μάθησης κάνει ένα σύνολο προβλέψεων με βάση τα ιστορικά δεδομένα εισόδου. Αυτό είναι συχνά αρκετό για δεδομένα που δεν εξαρτώνται από το χρόνο ή όταν δεν είναι κρίσιμο να ληφθούν προβλέψεις σε πραγματικό χρόνο ως έξοδο.
- Προβλέψεις σε πραγματικό χρόνο (γνωστοί και ως προβλέψεις κατ' απαίτηση): Οι προβλέψεις δημιουργούνται σε πραγματικό χρόνο χρησιμοποιώντας τα δεδομένα εισόδου που είναι διαθέσιμα τη στιγμή της αίτησης.

Αφού προσδιορίσουμε αυτές τις δύο διαστάσεις (model training και model prediction), μπορούμε να ταξινομήσουμε τη λειτουργικότητα των μοντέλων μηχανικής μάθησης σε τέσσερα μοτίβα αρχιτεκτονικής [Εικόνα 1]:



Εικόνα 1: Μοτίβα αρχιτεκτονικής της λειτουργικότητας μοντέλων μηχανικής μάθησης

Στη συνέχεια, παρουσιάζεται μια περιγραφή των μοντέλων αρχιτεκτονικών προτύπων όπως Forecast, Web-Service, Online Learning και AutoML.

Forecast

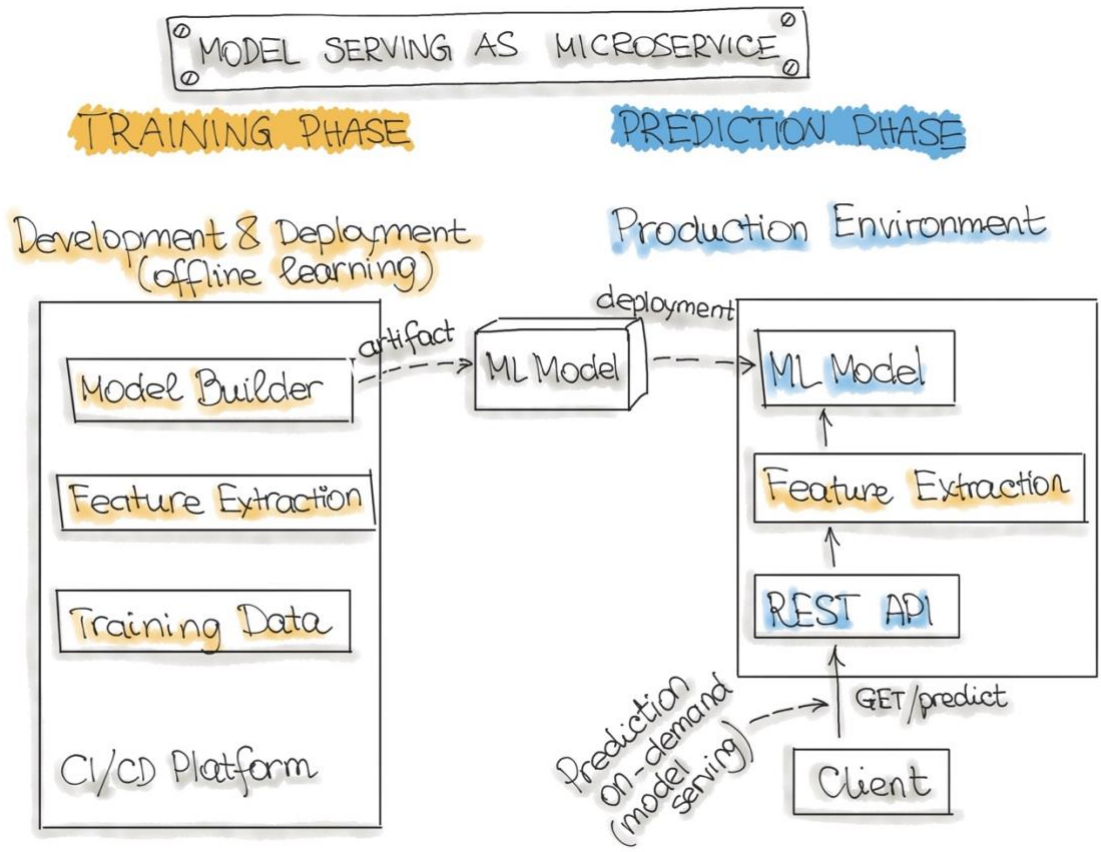
Αυτός ο τύπος ροής εργασιών μηχανικής μάθησης είναι ευρέως διαδεδομένος στην ακαδημαϊκή έρευνα ή στην εκπαίδευση της επιστήμης των δεδομένων. Αυτή η φόρμα χρησιμοποιείται για πειραματισμό με αλγόριθμους και δεδομένα μηχανικής μάθησης καθώς είναι ο ευκολότερος τρόπος δημιουργίας ενός συστήματος μηχανικής μάθησης. Συνήθως, χρησιμοποιείται ένα διαθέσιμο σύνολο δεδομένων, εκπαιδεύεται το μοντέλο, μετά εκτελείται αυτό το μοντέλο σε άλλα (κυρίως ιστορικά) δεδομένα και τέλος το μοντέλο κάνει προβλέψεις. Με αυτόν τον τρόπο, δημιουργείται μια πρόβλεψη. Αυτή η ροή εργασίας μηχανικής μάθησης δεν είναι πολύ χρήσιμη και, ως εκ τούτου, δεν είναι συνηθισμένη σε μια βιομηχανία για συστήματα παραγωγής (π.χ. εφαρμογές για κινητές συσκευές).

Web-Service

Η πιο συχνά περιγραφόμενη αρχιτεκτονική ανάπτυξης για μοντέλα μηχανικής μάθησης είναι μια υπηρεσία ιστού (microservise). Η υπηρεσία Ιστού λαμβάνει δεδομένα εισόδου και εξάγει μια πρόβλεψη για τα σημεία δεδομένων εισόδου. Το μοντέλο εκπαιδεύεται εκτός σύνδεσης σε

ιστορικά δεδομένα, αλλά χρησιμοποιεί πραγματικά δεδομένα για να κάνει προβλέψεις. Η διαφορά από μια πρόβλεψη (προβλέψεις παρτίδας) είναι ότι το μοντέλο μηχανικής μάθησης εκτελείται σχεδόν σε πραγματικό χρόνο και χειρίζεται μία μόνο εγγραφή κάθε φορά αντί να επεξεργάζεται όλα τα δεδομένα ταυτόχρονα. Η υπηρεσία Ιστού χρησιμοποιεί δεδομένα σε πραγματικό χρόνο για να κάνει προβλέψεις, αλλά το μοντέλο παραμένει σταθερό μέχρι να εκπαιδευτεί εκ νέου και να αναπτυχθεί εκ νέου στο σύστημα παραγωγής.

Το παρακάτω σχήμα απεικονίζει την αρχιτεκτονική για την περιτύλιξη εκπαιδευμένων μοντέλων ως υπηρεσίες ανάπτυξης. [Εικόνα 2]



Εικόνα 2: Αρχιτεκτονική για την περιτύλιξη εκπαιδευμένων μοντέλων ως υπηρεσίες ανάπτυξης

Online Learning

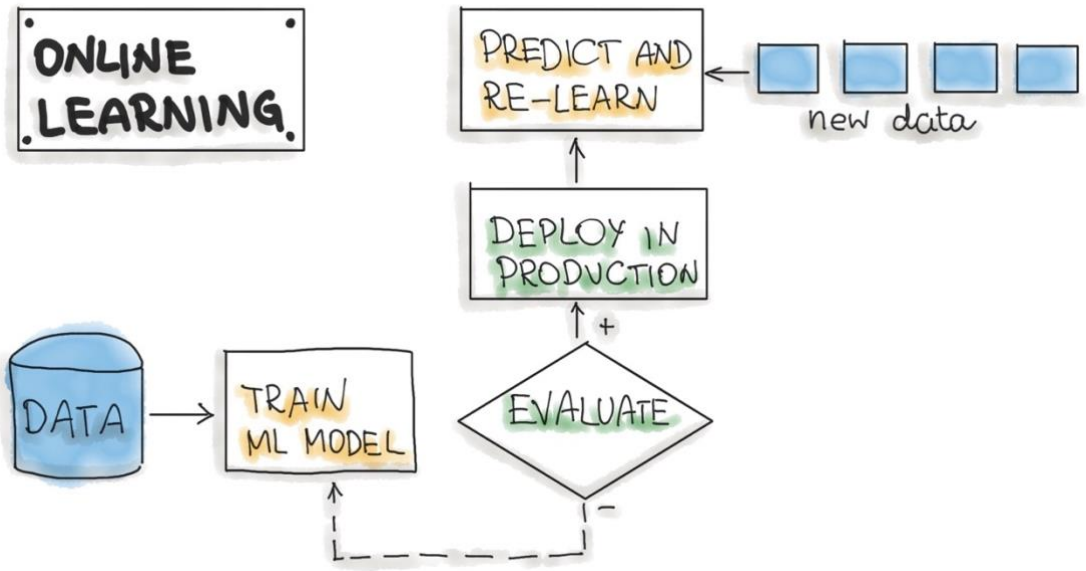
Ο πιο δυναμικός τρόπος για την ενσωμάτωση της μηχανικής μάθησης σε ένα σύστημα παραγωγής είναι η εφαρμογή της διαδικτυακής μάθησης (Online Learning), η οποία είναι επίσης γνωστή ως ανάλυση ροής σε πραγματικό χρόνο. Η διαδικτυακή εκμάθηση μπορεί να είναι ένα συγκεχυμένο όνομα, επειδή η εκπαίδευση βασικής μάθησης ή μοντέλου μηχανικής μάθησης συνήθως δεν εκτελείται στο ζωντανό σύστημα. Θα πρέπει να το ονομάσουμε σταδιακή μάθηση. Ωστόσο, ο όρος διαδικτυακή μάθηση έχει ήδη καθιερωθεί στην κοινότητα μηχανικής μάθησης.

Σε αυτόν τον τύπο ροής εργασίας μηχανικής μάθησης, ο αλγόριθμος εκμάθησης λαμβάνει συνεχώς μια ροή δεδομένων, είτε ως μεμονωμένα σημεία δεδομένων είτε σε μικρές ομάδες που

ονομάζονται μίνι παρτίδες (mini-batches). Το σύστημα μαθαίνει για νέα δεδομένα εν κινήσει καθώς φτάνει, επομένως το μοντέλο μηχανικής μάθησης να επανεκπαιδεύεται σταδιακά με νέα δεδομένα. Αυτό το συνεχώς επανεκπαιδευόμενο μοντέλο είναι άμεσα διαθέσιμο ως διαδικτυακή υπηρεσία.

Τεχνικά, αυτός ο τύπος συστήματος μηχανικής μάθησης λειτουργεί καλά με την αρχιτεκτονική λάμδα σε συστήματα μεγάλων δεδομένων. Συνήθως, τα δεδομένα εισόδου είναι μια ροή γεγονότων και το μοντέλο μηχανικής μάθησης λαμβάνει τα δεδομένα καθώς εισέρχεται στο σύστημα, παρέχει προβλέψεις και μαθαίνει ξανά για αυτά τα νέα δεδομένα. Το μοντέλο θα λειτουργούσε συνήθως ως υπηρεσία σε ένα σύμπλεγμα Kubernetes ή παρόμοιο [Εικόνα 3].

Μια μεγάλη δυσκολία με το διαδικτυακό σύστημα εκμάθησης στην παραγωγή είναι ότι εάν εισέρχονται κακά δεδομένα στο σύστημα, το μοντέλο μηχανικής μάθησης, καθώς και η απόδοση ολόκληρου του συστήματος, θα μειώνονται ολοένα και περισσότερο.



Εικόνα 3: Online Learning Pipeline

AutoML

Μια ακόμη πιο εξελιγμένη έκδοση της διαδικτυακής μάθησης είναι η αυτοματοποιημένη μηχανική εκμάθηση ή η AutoML.

Το AutoML τραβάει μεγάλη προσοχή και θεωρείται η επόμενη πρόοδος για την μηχανική μάθηση που χρισιμοποιείται από τις εταιρείες. Η AutoML υπόσχεται εκπαίδευση μοντέλων μηχανικής μάθησης με ελάχιστη προσπάθεια και χωρίς εξειδίκευση στη μηχανική εκμάθηση. Ο χρήστης πρέπει να παρέχει δεδομένα και το σύστημα AutoML επιλέγει αυτόματα έναν αλγόριθμο μηχανικής μάθησης, όπως η αρχιτεκτονική νευρωνικών δικτύων, και διαμορφώνει τον επιλεγμένο αλγόριθμο.

Αντί να ενημερωθεί το μοντέλο, εκτελείται ένας ολόκληρος αγωγός εκπαίδευσης μοντέλου μηχανικής μάθησης στην παραγωγή που οδηγεί σε νέα μοντέλα εν κινήσει. Προς το παρόν, αυτός είναι ένας πολύ πειραματικός τρόπος για την υλοποίηση ροών εργασίας μηχανικής μάθησης. Το AutoML παρέχεται συνήθως από μεγάλους παρόχους cloud, όπως η Google ή το MS Azure. Ωστόσο, τα μοντέλα που κατασκευάζονται με AutoML πρέπει να φτάσουν το επίπεδο ακρίβειας που απαιτείται για την επιτυχία στον πραγματικό κόσμο.

Perceptron και Multi Layer Perceptron (MLP)

Για τις ανάγκες της παρούσας έρευνας, χρησιμοποιήθηκε εποπτευόμενη μηχανική μάθηση και πιο συγκεκριμένα ο αλγόριθμος Multi Layer Perceptron (MLP). Για πιο εύκολη κατανόηση θα πρέπει αρχικά να εξηγήσουμε την χρήση του απλού Perceptron και στη συνέχεια τη χρήση του Multi Layer Perceptron.

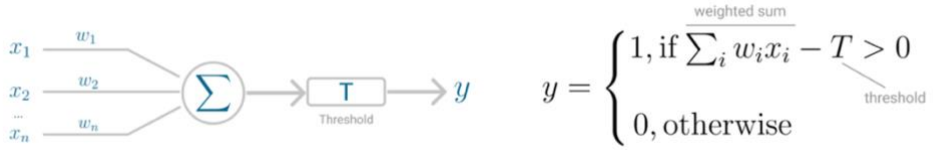
Perceptron

Αν και σήμερα ο Perceptron αναγνωρίζεται ευρέως ως αλγόριθμος, αρχικά προοριζόταν ως μηχανή αναγνώρισης εικόνας. Πήρε το όνομά του από την εκτέλεση της ανθρώπινης λειτουργίας της αντίληψης, της θέασης και της αναγνώρισης εικόνων.

Ειδικότερα, το ενδιαφέρον έχει επικεντρωθεί στην ιδέα μιας μηχανής που θα μπορούσε να εννοιολογήσει εισόδους που προσπίπτουν απευθείας από το φυσικό περιβάλλον του φωτός, του ήχου, της θερμοκρασίας κ.λπ. — ο «φαινομενικός κόσμος» με τον οποίο όλοι είμαστε εξοικειωμένοι — αντίθετα, απαιτεί την παρέμβαση ενός ανθρώπινου παράγοντα για να αφομοιώσει και να κωδικοποιήσει τις απαραίτητες πληροφορίες.

Η μηχανή perceptron του Rosenblatt βασίστηκε σε μια βασική μονάδα υπολογισμού, τον νευρώνα. Όπως και στα προηγούμενα μοντέλα, κάθε νευρώνας έχει ένα κύτταρο που λαμβάνει μια σειρά από ζεύγη εισόδων και βαρών.

Η κύρια διαφορά στο μοντέλο του Rosenblatt είναι ότι οι είσοδοι συνδυάζονται σε ένα σταθμισμένο άθροισμα και, εάν το σταθμισμένο άθροισμα υπερβαίνει ένα προκαθορισμένο όριο, ο νευρώνας ενεργοποιείται και παράγει μια έξοδο.



Εικόνα 4: Νευρωνικό δίκτυο Perceptron (Αριστερά), Λογική Threshold (Δεξιά)

Το Threshold T αντιπροσωπεύει τη συνάρτηση ενεργοποίησης. Εάν το σταθμισμένο άθροισμα των εισόδων είναι μεγαλύτερο από το μηδέν, ο νευρώνας εξάγει την τιμή 1, διαφορετικά η τιμή εξόδου είναι μηδέν [Εικόνα 4].

Perceptron για δυαδική ταξινόμηση (Binary Classification)

Με αυτή τη διακριτή έξοδο, που ελέγχεται από τη συνάρτηση ενεργοποίησης, ο perceptron μπορεί να χρησιμοποιηθεί ως μοντέλο δυαδικής ταξινόμησης (Binary Classification), ορίζοντας ένα γραμμικό όριο απόφασης. Βρίσκει το διαχωριστικό υπερεπίπεδο που ελαχιστοποιεί την απόσταση μεταξύ λανθασμένων σημείων και του ορίου απόφασης.

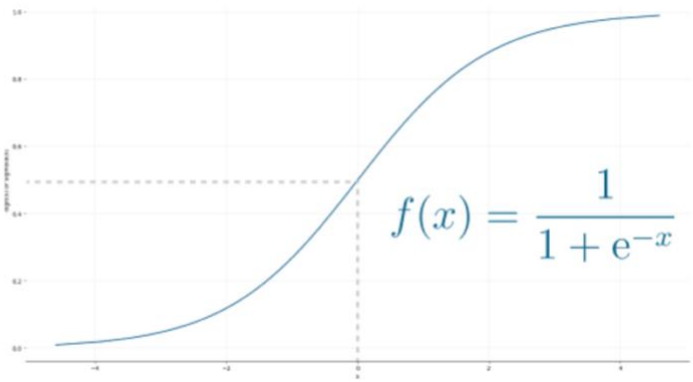
$$\underbrace{D(w,c)}_{\text{distance}} = -\sum_{i \in \mathrm{M}} \overset{\text{output}}{y_i} \left(x_i w_i + c\right)$$

misclassified observations

Εικόνα 5: Συνάρτηση της απώλειας (Perceptron's Loss Function)

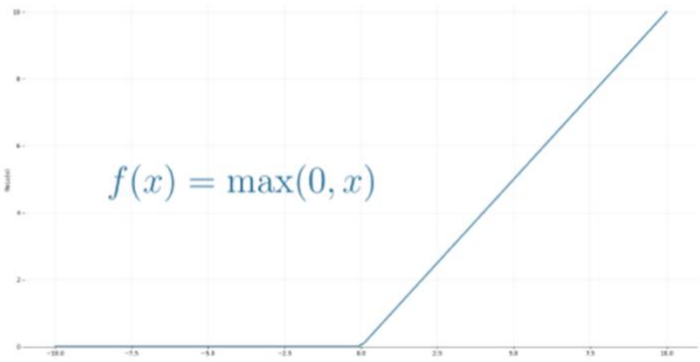
Για να ελαχιστοποιήσει αυτή την απόσταση, το Perceptron χρησιμοποιεί τη Στοχαστική Κάθοδος Κλίσης (Stochastic Gradient Descent)[9] ως συνάρτηση βελτιστοποίησης. Εάν τα δεδομένα είναι γραμμικά διαχωρισμένα, είναι εγγυημένο ότι η Στοχαστική Κάθοδος Κλίσης θα συγκλίνει σε έναν πεπερασμένο αριθμό βημάτων. Το τελευταίο κομμάτι που χρειάζεται το Perceptron είναι η συνάρτηση ενεργοποίησης (activation function), η συνάρτηση που καθορίζει αν ο νευρώνας θα πυροδοτηθεί ή όχι.

Τα αρχικά μοντέλα Perceptron χρησιμοποιούσαν σιγμοειδή συνάρτηση (sigmoid function)[10]. Η σιγμοειδής συνάρτηση αντιστοιχίζει οποιαδήποτε πραγματική είσοδο σε μια τιμή που είναι είτε 0 είτε 1 και κωδικοποιεί μια μη γραμμική συνάρτηση. Ο νευρώνας μπορεί να λάβει αρνητικούς αριθμούς ως είσοδο και θα εξακολουθεί να μπορεί να παράγει μια έξοδο που είναι είτε 0 είτε 1 [Εικόνα 6].



Εικόνα 6: Σιγμοειδής συνάρτηση (Sigmoid function)

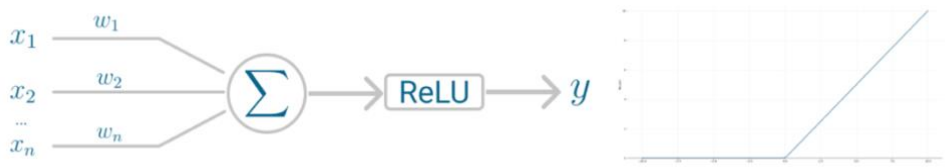
Ωστόσο, αν κοιτάξετε τα έγγραφα και τους αλγόριθμους Deep Learning της τελευταίας δεκαετίας, θα δείτε ότι οι περισσότεροι από αυτούς χρησιμοποιούν τη Διορθωμένη Γραμμική Μονάδα (Rectified Linear Unit (ReLU))[11] ως συνάρτηση ενεργοποίησης (activation function) του νευρώνα [Εικόνα 7].



Εικόνα 7: Διορθωμένη γραμμική μονάδα (Rectified Linear Unit (ReLU))

Ο λόγος για τον οποίο το ReLU υιοθετήθηκε περισσότερο είναι ότι επιτρέπει καλύτερη βελτιστοποίηση με χρήση Stochastic Gradient Descent, πιο αποτελεσματικό υπολογισμό και είναι αμετάβλητο σε κλίμακα, που σημαίνει ότι τα χαρακτηριστικά του δεν επηρεάζονται από την κλίμακα της εισόδου.

Συνοψίζοντας, ο νευρώνας λαμβάνει εισόδους και επιλέγει ένα αρχικό σύνολο βαρών τυχαία. Αυτά συνδυάζονται σε σταθμισμένο άθροισμα και στη συνέχεια το ReLU, η συνάρτηση ενεργοποίησης, καθορίζει την τιμή της εξόδου. [Εικόνα 8]



Εικόνα 8: Μοντέλο νευρώνων Perceptrons (Αριστερά) και συνάρτηση ενεργοποίησης (Δεξιά)

Αλλά μία ερώτηση θα μπορούσε να είναι : ο Perceptron δεν μαθαίνει τα πραγματικά βάρη;

Ο Perceptron χρησιμοποιεί τη Στοχαστική Κλίση για να βρει, ή καλύτερα να μάθει, το σύνολο βάρους που ελαχιστοποιεί την απόσταση μεταξύ των λανθασμένων σημείων και του ορίου απόφασης. Μόλις συγκλίνει η Στοχαστική Κάθοδος Κλίσης, το σύνολο δεδομένων χωρίζεται σε δύο περιοχές με ένα γραμμικό υπερ-επίπεδο. [Εικόνα 9]

Κανόνας ενημέρωσης βαρών Perceptron

- Σφάλμα στο **τρέχον** παράδειγμα εκπαίδευσης:

$$E_i(\vec{w}) = \frac{1}{2}(t^{(i)} - o^{(i)})^2 = \frac{1}{2}(t^{(i)} - \Phi\left(\sum w_l x_l^{(i)}\right))^2$$

Ξεκινώ με τυχαία βάρη. Μετράω σφάλμα $E_i(\vec{w})$ στο τρέχον παράδειγμα εκπαίδευσης με τα τρέχοντα βάρη $\vec{w}$. Προς τα πού να μεταβάλω τα βάρη;

Πηγή εικόνας: http://en.wikipedia.org/wiki/Gradient_descent

Η κλίση $\nabla E_i(\vec{w})$ είναι ένα διάνυσμα που δείχνει προς την κατεύθυνση μεταβολής των βαρών που οδηγεί στη μεγαλύτερη **αύξηση** του E_i. Το $-\nabla E_i(\vec{w})$ δείχνει προς την μεγαλύτερη **μείωση** του E_i.

Τροποποιούμε τα w_i κατά η προς την κατεύθυνση που προκαλεί τη μεγαλύτερη μείωση του E_i:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E_i(\vec{w})$$

Στοχαστική κατάβαση κλίσης.

16

Κανόνας ενημέρωσης βαρών Perceptron

$$E_i(\vec{w}) = \frac{1}{2}(t^{(i)} - o^{(i)})^2 = \frac{1}{2}(t^{(i)} - \Phi\left(\sum_{l=1}^{n} w_l x_l^{(i)}\right))^2$$

$$\nabla E_i(\vec{w}) = \left\langle \frac{\partial E_i(\vec{w})}{\partial w_1}, \ldots, \frac{\partial E_i(\vec{w})}{\partial w_l}, \ldots, \frac{\partial E_i(\vec{w})}{\partial w_n} \right\rangle$$

$$\frac{\partial E_i}{\partial w_l} = (t^{(i)} - \Phi\left(\sum_l w_l x_l^{(i)}\right)) \cdot \frac{\partial (t^{(i)} - \Phi\left(\sum_l w_l x_l^{(i)}\right))}{\partial w_l} = -(t^{(i)} - o^{(i)}) \cdot \Phi'\left(\sum_l w_l x_l^{(i)}\right) \cdot x_l^{(i)}$$

Άρα: $\nabla E_i(\vec{w}) = -(t^{(i)} - o^{(i)}) \cdot \Phi'\left(\sum_l w_l x_l^{(i)}\right) \cdot \left\langle x_1^{(i)}, \ldots, x_n^{(i)} \right\rangle$

$$= -(t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot \vec{x}^{(i)}$$

Κανόνας ενημέρωσης βαρών:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E_i(\vec{w}) = \vec{w} + \eta \cdot (t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot \vec{x}^{(i)}$$

Οπότε: $w_l \leftarrow w_l + \eta \cdot (t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot x_l^{(i)}$

17

Εικόνα 9: Κανόνας ενημέρωσης βαρών

Αν και ειπώθηκε ότι ο Perceptron μπορεί να αντιπροσωπεύει οποιοδήποτε κύκλωμα και λογική, η μεγαλύτερη κριτική ήταν ότι δεν μπορεί να αναπαραστήσει την πύλη XOR[12], αποκλειστικό OR, όπου η πύλη επιστρέφει μόνο 1 εάν οι είσοδοι είναι διαφορετικές.
Αυτό αποδείχθηκε σχεδόν μια δεκαετία αργότερα από τους Minsky και Papert, το 1969[13] και υπογραμμίζει το γεγονός ότι ο Perceptron, με μόνο έναν νευρώνα, δεν μπορεί να εφαρμοστεί σε μη γραμμικά δεδομένα.

Πολυστρωματικός Perceptron (Multilayer Perceptron(MLP))

Ο Multilayer Perceptron αναπτύχθηκε για να αντιμετωπίσει αυτόν τον περιορισμό. Είναι ένα νευρωνικό δίκτυο όπου η αντιστοίχιση μεταξύ εισόδων και εξόδων είναι μη γραμμική.

Ένα Multilayer Perceptron έχει στρώματα εισόδου και εξόδου, καθώς και ένα ή περισσότερα κρυφά στρώματα με πολλούς νευρώνες στοιβαγμένους μεταξύ τους. Και ενώ στον Perceptron ο νευρώνας πρέπει να έχει μια συνάρτηση ενεργοποίησης που επιβάλλει ένα threshold, όπως το ReLU ή το Sigmoid, οι νευρώνες σε ένα Πολυστρωματικό Perceptron μπορούν να χρησιμοποιήσουν οποιαδήποτε αυθαίρετη συνάρτηση ενεργοποίησης. [Εικόνα 10]

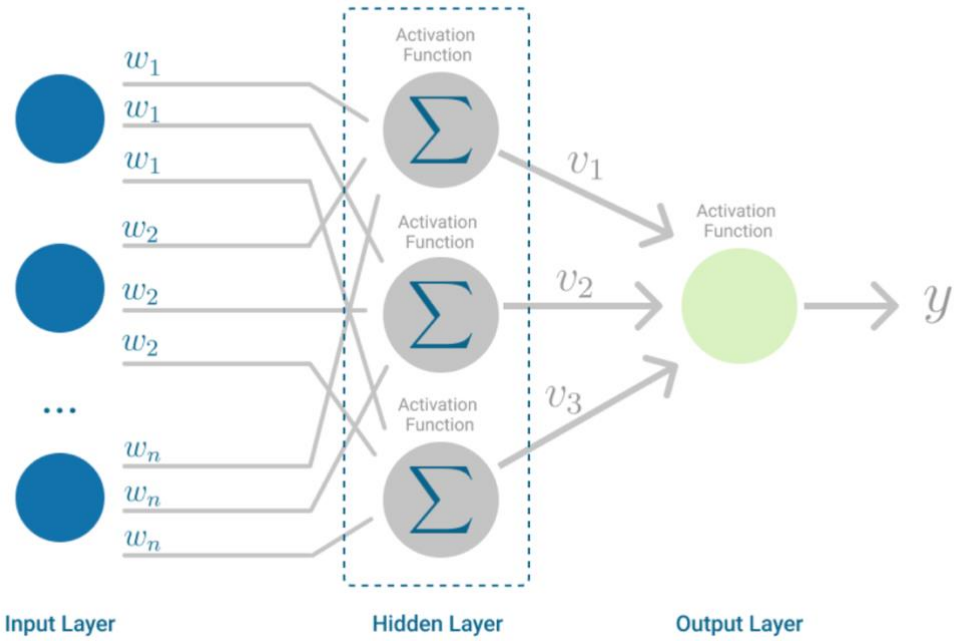


Figure 10: Πολυστρωματικό Perceptron (Multy Layer Perceptron)

Ο Multi layer Perceptron εμπίπτει στην κατηγορία των αλγορίθμων προώθησης τροφοδοσίας, επειδή οι είσοδοι συνδυάζονται με τα αρχικά βάρη σε ένα σταθμισμένο άθροισμα και υπόκεινται στη συνάρτηση ενεργοποίησης, όπως ακριβώς στο Perceptron. Αλλά η διαφορά είναι ότι κάθε γραμμικός συνδυασμός διαδίδεται στο επόμενο στρώμα.

Κάθε επίπεδο τροφοδοτεί το επόμενο με το αποτέλεσμα του υπολογισμού του, την εσωτερική του αναπαράσταση των δεδομένων. Αυτό περνάει από τα κρυφά επίπεδα μέχρι το επίπεδο εξόδου. Εάν ο αλγόριθμος υπολόγιζε μόνο τα σταθμισμένα αθροίσματα σε κάθε νευρώνα, διέδιδε τα αποτελέσματα στο επίπεδο εξόδου και σταματούσε εκεί, δεν θα μπορούσε να μάθει τα βάρη που ελαχιστοποιούν τη συνάρτηση κόστους. Δηλαδή, εάν ο αλγόριθμος υπολόγιζε μόνο μία επανάληψη, δεν θα υπήρχε πραγματική μάθηση.

Classification[17]

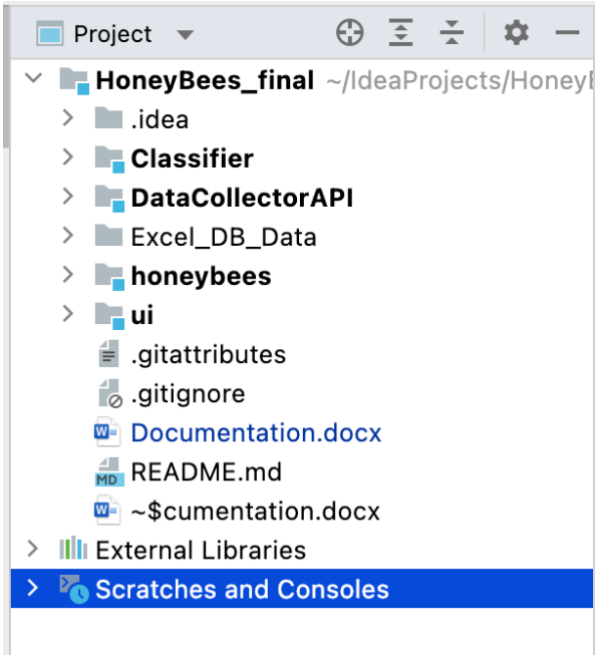
Στη παρούσα έρευνα, ο Multi layer Perceptron χρησιμοποιήθηκε κατά τη διαδικασία του Classification. Classification στη στατιστική είναι η διαδικασία κατηγοριοποίησης. Δηλαδή αν υποθέσουμε ότι υπάρχει ένα σύνολο κατηγοριών, σε ποια από αυτές τις κατηγορίες ανήκει η παρατήρηση που έχει γίνει. Ένα παράδειγμα τέτοιας κατηγοριοποίησης είναι η αντιστοίχιση ενός μηνύματος ηλεκτρονικού ταχυδρομείου στην κατηγορία "spam" ή "non-spam". Ένας αλγόριθμος που εφαρμόζει classification, ειδικά σε μια συγκεκριμένη υλοποίηση, είναι γνωστός ως classifier. Ο όρος "classifier" μερικές φορές αναφέρεται επίσης στη μαθηματική συνάρτηση, που υλοποιείται από έναν αλγόριθμο ταξινόμησης, που αντιστοιχίζει τα δεδομένα εισόδου σε μια κατηγορία.

Σύντομη περιγραφή της λύσης

Σκοπός της παρούσας έρευνας είναι να δημιουργηθεί μία εφαρμογή μηχανικής μάθησης η οποία θα έχει τη δυνατότητα να αναγνωρίζει μέσα από έναν όγκο δεδομένων, τις περιόδους κατά τις οποίες οι μέλισσες κάνουν συγκομιδή μελιού. Δηλαδή τις περιόδους που δεν γίνεται τρύγος από τον μελισσοκόμο και έτσι οι μέλισσες συνεχίζουν να μαζεύουν το μέλι. Επίσης θα γίνεται παραγωγή στατιστικών δεδομένων σχετικά με το κλήμα που επικρατούσε κατά την συγκομιδή του μελιού αλλά και την ποσότητα του μελιού κατά την συγκομιδή. Για την επίτευξη του στόχου αυτού, έχουν χρησιμοποιηθεί τα επίπεδα δημιουργίας λογισμικού μηχανικής μάθησης που αναφέρθηκαν στο ερευνητικό πεδίο παραπάνω. Η στατιστική ανάλυση έγινε με τη βοήθεια ερωτημάτων (queries).

Η εφαρμογή έχει δημιουργηθεί με τη γλώσσα προγραμματισμού Java και τη βοήθεια του Spring Boot Framework ενώ η βάση δεδομένων που έχει χρησιμοποιηθεί είναι υλοποιημένη με MySQL. Τα δεδομένα προέρχονται από δύο ηλεκτρονικές ζυγαριές που είναι τοποθετημένες σε δύο κυψέλες από τα κοπάδια του μελισσοκόμου. Οι ζυγαριές αυτές είναι προγραμματισμένες να αποστέλλουν ένα SMS στο κινητό τηλέφωνο του μελισσοκόμου μία φορά την ημέρα. Στη συγκεκριμένη εργασία γίνεται προσομοίωση της διαδικασίας αυτής προσπαθώντας να αντλήσουμε τα δεδομένα της ζυγαριάς από μία διεύθυνση ηλεκτρονικού ταχυδρομείου και όχι SMS για το λόγο ότι η εφαρμογή μας είναι μία web εφαρμογή. Στην περίπτωση που οι ζυγαριές μπορέσουν να αποστείλουν τα δεδομένα με email, τότε η εφαρμογή μας μπορεί να τα συλλέξει και να τα αποθηκεύσει στη βάση δεδομένων. Αντίθετα όταν η ζυγαριές δεν έχουν αυτή τη δυνατότητα, τότε η εφαρμογή μπορεί να αποστέλλει πλασματικά δεδομένα στο email μία φορά την ημέρα και στη συνέχεια να τα συλλέγει και να τα αποθηκεύει στη βάση δεδομένων.
Στη συνέχεια, η εφαρμογή έχει τη δυνατότητα να διαβάζει από τη βάση δεδομένων, μέσω ενός API που δημιουργήθηκε και να τα χρησιμοποιεί για να μπορέσει μέσω της μηχανικής μάθησης να κάνει τον διαχωρισμό για το αν τα δεδομένα που του δίνονται είναι μία περίοδος συλλογής μελιού από τις μέλισσες ή τρύγος. Για την αναγνώριση και το διαχωρισμό των δεδομένων με τη χρήση μηχανικής μάθησης χρησιμοποιήθηκε εποπτευόμενη μηχανική μάθηση (Supervised Machine Learning) και πιο συγκεκριμένα, ο αλγόριθμος ταξινόμησης (Classification).

Η δομή της εφαρμογής αποτελείται από τέσσερις διαφορετικές ενότητες (modules) οι οποίες συνδέονται μεταξύ τους για την επίτευξη του στόχου. Η πρώτη ενότητα ονομάζεται DataCollectorAPI και είναι υπεύθυνη για τη συλλογή των δεδομένων από μία συγκεκριμένη διεύθυνση ηλεκτρονικού ταχυδρομείου και την αποστολή τους στο κεντρικό API. Το κεντρικό API ονομάζεται honeybees και είναι αυτό που λαμβάνει και στέλνει δεδομένα στη βάση δεδομένων. Η ενότητα Classifier είναι εκείνη στην οποία έχει υλοποιηθεί το κομμάτι της μηχανικής μάθησης και είναι υπεύθυνη να παράξει τα αποτελέσματα που χρειαζόμαστε σχετικά με την έρευνα. Και τέλος η ενότητα UI η οποία είναι υπεύθυνη για την παρουσίαση των παραγόμενων δεδομένων στο χρήστη. [Εικόνα 11]



Εικόνα 11: Δομή της εφαρμογής

Αναλυτική περιγραφή της λύσης

Συλλογή και διαχείριση δεδομένων.

Όπως αναφέρθηκε και στο ερευνητικό πεδίο, ίσως το σημαντικότερο κομμάτι σε μία έρευνα που σχετίζεται με τη μηχανική μάθηση, είναι τα δεδομένα. Αυτό συμβαίνει γιατί όσο σωστότερα είναι τα δεδομένα με τα οποία τροφοδοτούμε το νευρωνικό δίκτυο, τόσο σωστότερα θα είναι και τα αποτελέσματα που θα παραχθούν από αυτό.

Πως όμως συλλέχθηκαν και διαχειρίστηκαν τα δεδομένα στην συγκεκριμένη ερευνά;

Για τη συλλογή των δεδομένων αυτών, χρησιμοποιήθηκε η μηχανική ροής δεδομένων (data engineering pipeline). Η μηχανική ροής δεδομένων έχει περιγραφεί ενδελεχώς στο ερευνητικό πεδίο παραπάνω. Σύμφωνα με τη ροή αυτή, η πρώτη ενέργεια που έπρεπε να γίνει ήταν η ταυτοποίηση της πηγής των δεδομένων. Στο σημείο αυτό έγινε επικοινωνία με τον μελισσοκόμο για να διαπιστωθεί ο όγκος τον δεδομένων που μπορεί να μας παρέχει αλλά και η ποιότητα τους. Ρωτήθηκε δηλαδή εκτός από την ποσότητα των δεδομένων και για τα σημεία που βρίσκονταν οι ζυγαριές κατά τις περιόδους που παράγονταν τα δεδομένα, καθώς και το είδος του μελιού που παράγεται από την κάθε τοποθεσία.

Στη συνέχεια καθώς δεν υπήρχε κάποιος αυτοματοποιημένος τρόπος για να συλλέξουμε τα δεδομένα από το κινητό τηλέφωνο του μελισσοκόμου απ' όπου ήταν αποθηκευμένα, χειροκίνητα αντλήθηκαν από εκεί και αποθηκεύτηκαν σε ένα αρχείο excel (η τελική έκδοση του αρχείου αυτού θα βρίσκεται στα παραδοτέα της υλοποίησης της εφαρμογής).

Στη συνέχεια, αποφασίστηκε η βάση δεδομένων που θα χρησιμοποιηθεί. Καθώς ο όγκος των δεδομένων δεν ήταν ιδιαίτερα μεγάλος, δημιουργήθηκε ένας και μόνο πίνακας σε μία βάση MySQL ο οποίος περιέχει τις εξής στήλες :

- Id : Ο αναγνωριστικός αριθμός της κάθε εγγραφής στη βάση.
- Date : Η ημερομηνία κατά την οποία παράχθηκε η κάθε εγγραφή
- Hum : Η υγρασία
- Scale : Το βάρος
- TemperatureOUT : Η θερμοκρασία του εξωτερικού περιβάλλοντος
- Location : Η τοποθεσία όπου βρίσκεται το κοπάδι
- eidosMeliou : Το είδος μελιού που παράγεται ως επι το πλείστο από την τοποθεσία αυτή
- zigaria_id : Ο αναγνωριστικός αριθμός της ζυγαριάς.

Πριν την καταγραφή των δεδομένων στη βάση και ενώ τα δεδομένα βρίσκονται στο υπολογιστικό φύλλο (excel) έγινε η διαδικασία καθαρισμού δεδομένων (Data Wrangling). Κατά τη διαδικασία αυτή έγινε καθαρισμός των δεδομένων, δηλαδή αφαιρέθηκαν δεδομένα που περιείχαν ανακρίβειες και δημιουργήθηκε το χαρακτηριστικό προφίλ για την κάθε στήλη δεδομένων (Όνομα, Αριθμός εγγραφών, τύπος δεδομένων κλπ). Έγινε αναδιάταξη πεδίων εγγραφής με μετακίνηση στηλών και δημιουργία νέων πεδίων εγγραφής μέσω εξαγωγής τιμών.

Όταν τα δεδομένα πέρασαν τις παραπάνω διαδικασίες και ήταν έτοιμα προς χρήση, έγιναν εισαγωγή στη βάση δεδομένων. Το υπολογιστικό φύλλο χρησιμοποιήθηκε ως αντίγραφο ασφαλείας για όλα τα δεδομένα.

Υλοποίηση του DataCollectorAPI

Ο DataCollectorAPI είναι η πρώτη ενότητα (module) που υλοποιήθηκε και σκοπός της είναι να μπορεί να διαβάζει μηνύματα ηλεκτρονικού ταχυδρομείου αλλά και να στέλνει μηνύματα μέσω αυτού. Αφού αρχικά έγινε έρευνα σχετικά με το ποιος ήταν ο σωστότερος τρόπος να υλοποιηθεί η λειτουργεία αυτή, καταλήξαμε στο γεγονός ότι θα έπρεπε να χρησιμοποιηθεί το GmailAPI[16] της Google.

Το Gmail API είναι ένα RESTful API που μπορεί να χρησιμοποιηθεί για πρόσβαση στα γραμματοκιβώτια του Gmail και αποστολή αλληλογραφίας. Για τις περισσότερες εφαρμογές web, το Gmail API είναι η καλύτερη επιλογή για εξουσιοδοτημένη πρόσβαση στα δεδομένα Gmail ενός χρήστη και είναι κατάλληλο για διάφορες εφαρμογές, όπως:

- Εξαγωγή αλληλογραφίας, δημιουργία ευρετηρίου και δημιουργία αντιγράφων ασφαλείας μόνο για ανάγνωση
- Αυτοματοποιημένη ή μέσω προγραμματισμού αποστολή μηνυμάτων
- Μετεγκατάσταση λογαριασμού email
- Οργάνωση email, συμπεριλαμβανομένου του φιλτραρίσματος και της ταξινόμησης μηνυμάτων
- Τυποποίηση των υπογραφών email σε έναν οργανισμό

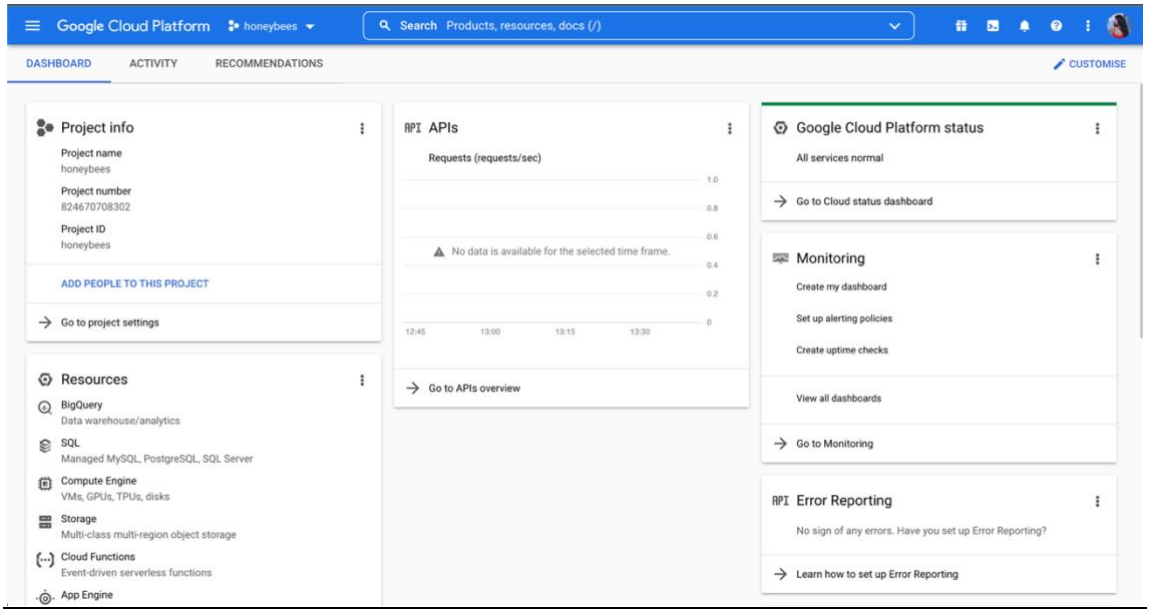
Το API παρέχει πρόσβαση σε λογαριασμούς Gmail μέσω του OAuth[20]. Αυτό επιτρέπει την ασφαλή, ανακλητή πρόσβαση σε λογαριασμούς Gmail που δεν απαιτεί από τους χρήστες να μοιράζονται τους κωδικούς πρόσβασης ηλεκτρονικού ταχυδρομείου τους με τους παρόχους εφαρμογών.

Με το Gmail API, γίνεται δυνατή η αποστολή και η ανάκτηση μηνυμάτων, η εφαρμογή ετικετών, η εύρεση μηνυμάτων με ετικέτες και η διαχείριση μηνυμάτων ηλεκτρονικού ταχυδρομείου μέσα σε νήματα (threads[21]).

Το ιστορικό λογαριασμών του API παρέχει ένα αρχείο καταγραφής δραστηριότητας μηνυμάτων για αναφορά στις εφαρμογές, μειώνοντας την πολυπλοκότητα της παρακολούθησης αλλαγών στους λογαριασμούς IMAP[22] σε πραγματικό χρόνο.

Ακολουθώντας τις προτεινόμενες οδηγίες που δίνονται από την Google με σκοπό την δημιουργία ενός project που χρησιμοποιεί το GmailAPI, έπρεπε να δημιουργηθεί ένα Google Cloud Project. Ένα έργο Google Cloud[23] είναι απαραίτητο για τη χρήση των API του Google Workspace και τη δημιουργία πρόσθετων ή εφαρμογών του Google Workspace. Αυτό το έργο αποτελεί τη βάση για τη δημιουργία, την ενεργοποίηση και τη χρήση όλων των υπηρεσιών Google Cloud, συμπεριλαμβανομένης της διαχείρισης API, της ενεργοποίησης της χρέωσης, της προσθήκης και της αφαίρεσης συνεργατών και της διαχείρισης αδειών.
Για να δημιουργηθεί ένα έργο Google Cloud θα πρέπει να ακολουθηθούν τα παρακάτω βήματα:

- Ανοίξτε την Κονσόλα Google Cloud.
- Επάνω αριστερά, κάντε κλικ στο μενού Μενού > IAM & Διαχειριστής > Δημιουργία έργου.
- Στο πεδίο Όνομα έργου, εισαγάγετε ένα περιγραφικό όνομα για το έργο σας.
- Προαιρετικά: Για να επεξεργαστείτε το αναγνωριστικό του έργου, κάντε κλικ στην Επεξεργασία. Το αναγνωριστικό του έργου δεν μπορεί να αλλάξει μετά τη δημιουργία του έργου, επομένως επιλέξτε ένα αναγνωριστικό που καλύπτει τις ανάγκες σας για τη διάρκεια ζωής του έργου.
- Στο πεδίο Τοποθεσία, κάντε κλικ στην Αναζήτηση για να εμφανίσετε πιθανές τοποθεσίες για το έργο σας. Στη συνέχεια, κάντε κλικ στην επιλογή Επιλογή.
- Κάντε κλικ στην επιλογή Δημιουργία. Η κονσόλα μεταβαίνει στη σελίδα του Πίνακα ελέγχου και το έργο σας δημιουργείται μέσα σε λίγα λεπτά.



Εικόνα 12: Google Cloud Dashboard

Στη συνέχεια, για να μπορεί η εφαρμογή να έχει πρόσβαση στο email είναι απαραίτητο να γίνει δημιουργία των διαπιστευτηρίων πρόσβασης (access credentials). Υπάρχουν τρείς τύποι διαπιστευτηρίων πρόσβασης:

- API key – Χρησιμοποιείται για να αποκτηθεί πρόσβαση σε δημόσια διαθέσιμα δεδομένα.

- OAuth client ID – Χρησιμοποιείται για έλεγχο ταυτότητας του τελικού χρήστη και πρόσβαση στα δεδομένα του. Στην περίπτωση αυτή απαιτήται από τη εφαρμογή να ζητά και να λαμβάνει τη συγκατάθεση του χρήστη.
- Service account – Χρησιμοποιείται για έλεγχο ταυτότητας ως λογαριασμό υπηρεσίας ρομπότ ή για πρόσβαση σε πόρους εκ μέρους των χρηστών του Google Workspace

Στην περίπτωση της παρούσας εργασίας επιλέχθηκε να χρησιμοποιηθεί ο τύπος διαπιστευτηρίων πρόσβασης με OAuth client ID για τον λόγο ότι η εφαρμογή μας θα πρέπει να έχει πρόσβαση σε έναν λογαριασμό ηλεκτρονικού ταχυδρομείου ενός συγκεκριμένου χρήστη. Η Google Cloud Platfom έχει την επιλογή «Help me choose» και στο σημείο αυτό βοηθάει τους χρήστες της να διαλέξουν τον τύπο διαπιστευτηρίων πρόσβασης με βάση τις ανάγκες της εφαρμογής που επιθυμούν να υλοποιήσουν.
Για να δημιουργηθεί το OAuth client ID ακολουθήθηκαν οι παρακάτω οδηγίες:

- Άνοιξε η Κονσόλα Google Cloud.
- Επάνω αριστερά, έγινε κλικ στο μενού Μενού > API και υπηρεσίες > Διαπιστευτήρια.
- Έγινε κλικ στην επιλογή Δημιουργία διαπιστευτηρίων > Αναγνωριστικό πελάτη OAuth.
- Έγινε κλικ στην επιλογή Τύπος εφαρμογής > Εφαρμογή Ιστού.
- Στο πεδίο "Όνομα", πληκτρολογήθηκε ένα όνομα για το διαπιστευτήριο. Αυτό το όνομα εμφανίζεται μόνο στην Κονσόλα Cloud.
- Προστέθηκαν εξουσιοδοτημένα URI που σχετίζονται με την εφαρμογή:
- Στην ενότητα "Εξουσιοδοτημένα URI ανακατεύθυνσης", κάντε κλικ στην Προσθήκη URI. Στη συνέχεια, εισαγάγετε ένα URI τελικού σημείου στο οποίο ο διακομιστής OAuth 2.0 μπορεί να στείλει απαντήσεις.
- Έγινε κλικ στην επιλογή Δημιουργία. Εμφανίστηκε η οθόνη που δημιουργήθηκε από το πρόγραμμα-πελάτη OAuth, η οποία εμφανίζει το νέο Client ID και το Client Secret.
- Σημειώθηκε το Client ID γιατί θα χρειαστεί κατά τη δημιουργία της εφαρμογής. Τα Client Secret δεν χρησιμοποιούνται για εφαρμογές Ιστού.
- Το διαπιστευτήριο που δημιουργήθηκε πρόσφατα εμφανίζεται στην περιοχή "OAuth 2.0 Client ID."

Ο DataCollectorAPI είναι μία εφαρμογή ιστού (web) η οποία χρησιμοποιεί Spring Boot Framework και MVC Design Pattern. Η δομή της εφαρμογής περιλαμβάνει εκτός από τα βασικά πακέτα του Spring Boot Framework και τρία ακόμα πακέτα τα οποία ονομάζονται Models, Services και Controllers.[Εικόνα 13]

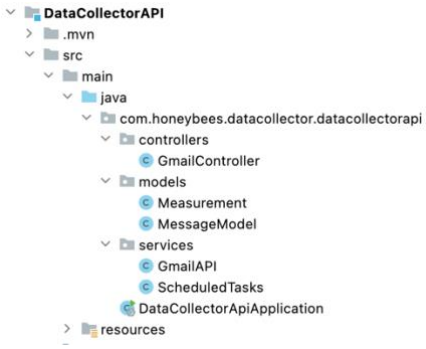


Figure 13: Δομή του DataCollectorAPI

Ανοίγοντας τον GmailController βλέπουμε πως περιέχει δύο μόνο μεθόδους. Οι μέθοδοι αυτοί ειίναι η getMail και η sendmail.[Εικόνα 14]

```java
@CrossOrigin("*")
@RestController
@RequestMapping("/api/data-collector")
public class GmailController {

    @Autowired
    public GmailAPI service;

    @RequestMapping("/getMail")
    public ResponseEntity<MessageModel> getMail() throws GeneralSecurityException, IOException {

        try {
            MessageModel mm = service.getEmail();
            RestTemplate restTemplate = new RestTemplate();
            restTemplate.postForObject( url: "http://localhost:9000/api/measurements/saveMeasurement", mm, MessageModel.class);
            return ResponseEntity.ok(mm);
        } catch(NullPointerException e) {
            return ResponseEntity.ok(new MessageModel());
        }
    }

    @RequestMapping("/sendMail")
    public void sendmail() throws MessagingException, GeneralSecurityException, IOException {
        service.sendEmail( to: "sophiajeb93@gmail.com",  title: "HoneyBees Data",  body: "2022-01-15, 48.9, 49.479, 19.2");
    }

}
```

Εικόνα 14: GmailController Class

Εξετάζοντας λοιπόν την πρώτη μέθοδο (getMail) διακρίνεται ότι το πρώτο της βήμα είναι να καλέσει την μέθοδο getEmail που βρίσκεται μέσα στο GmailAPI Service και στη συνέχεια να αποθηκεύσει το αποτέλεσμα της μεθόδου αυτής μέσα στο MessageModel.

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class MessageModel {

    private String currentDate;
    private float hum;
    private float temperature;
    private float scale;
}
```

Figure 15: MessageModel

Το MessageModel[Εικόνα 15] είναι το μοντέλο που περιέχει τα στοιχεία που έχουν ληφθεί από το ηλεκτρονικό ταχυδρομείο. Δηλαδή τα στοιχεία που έχει αποστείλει η ηλεκτρονική ζυγαριά.

Η μέθοδος getEmail αρχικά καλεί την μέθοδο isMailExists στην οποία μπαίνει ως παράμετρος μία μεταβλητή String. Η μεταβλητή αυτή περιέχει το λεκτικό του τίτλου του email που θα πρέπει να περιέχεται στο ηλεκτρονικό ταχυδρομείο για να μπορέσει η εφαρμογή να αναγνωρίσει ότι πρόκειται για το μήνυμα ηλεκτρονικού ταχυδρομείου που θα πρέπει να διαχειριστεί. Έπειτα, μέσα στη μέθοδο getEmail [Εικόνα 16] γίνεται ένας έλεγχος. Στην περίπτωση που το μήνυμα ηλεκτρονικού ταχυδρομείου με αυτόν τον συγκεκριμένο τίτλο δεν υπαρχει στη λίστα μηνυμάτων του email, τότε εμφανίζει ένα μήνυμα που αναφέρει ότι το email με τον τίτλο αυτό δεν βρέθηκε. Στην αντίθετη περίπτωση, δηλαδή στην περίπτωση που έχει αναγνωρίσει το μήνυμα καλεί την μέθοδο getGmailData με παράμετρο ένα query το οποίο ελέγχει αν το email περιέχει το σωστό λεκτικό στον τίτλο του και αν είναι αδιάβαστο. Τα αποτελέσματα της μεθόδου αυτής, τα αποθηκεύει σε ένα HashMap. Στην περίπτωση που το HashMap είναι άδειο, η μέθοδος επιστρέφει null. Αν όμως το HashMap έχει περιεχόμενο, δημιουργεί ένα νέο αντικείμενο MessageModel, το γεμίζει με το περιεχόμενο του email και μας το επιστρέφει για να το λάβει ο controller. Τέλος, καλεί την μέθοδο deleteMessage ώστε να διαγραφεί το συγκεκριμένο μήνυμα ηλεκρονικού ταχυδρομείου από τα εισερχόμενα και την επόμενη φορά που η εφαρμογή θα χρειαστεί τα ψάξει για νέα μηνύματα, να υπάρχει στη σελίδα με τα εισερχόμενα μόνο το τελευταίο μήνυμα το οποίο έχει αποσταλεί από τη ζυγαριά.

```java
public MessageModel getEmail() throws IOException, GeneralSecurityException {

    String expectedSubject = "HoneyBees Data";

    boolean exist = GmailAPI.isMailExist(expectedSubject);

    if (!exist) {
        System.out.println("Email with the subject " + expectedSubject + " not found");
    } else {
        HashMap<String, String> hm = GmailAPI.getGmailData( query: "subject:" + expectedSubject + " AND is:unread", expectedSubject);
        if(hm.isEmpty()) return null;
        if (hm.get("subject").equalsIgnoreCase(expectedSubject)) {
            MessageModel messageModel = new MessageModel();
            String str = hm.get("body");
            String[] arrOfStr = str.split( regex: ", ");

            messageModel.setCurrentDate(arrOfStr[0]);
            messageModel.setHum(Float.parseFloat(arrOfStr[1]));
            messageModel.setScale(Float.parseFloat(arrOfStr[2]));
            messageModel.setTemperature(Float.parseFloat(arrOfStr[3]));
            GmailAPI.deleteMessage( query: "subject:" + expectedSubject + " AND is:unread");
            return messageModel;
        } else {
            System.out.println("Failed verification - Email content / subject mismatching !");
        }
    }
    return null;
}
```

Figure 16: Μέθοδος getEmail

Σε μία πιο διεξοδική εξήγηση, θα πρέπει να αναφέρουμε τί κάνουν στην πραγματικότητα οι μέθοδοι isMailExist, getGmailData και DeleteMessage που προαναφέρθηκαν και ουσιαστικά μας δημιουργούν το αποτέλεσμα που παρέχεται από την μέθοδο getEmail.

Η μέθοδος isMailExist αρχικά καλέι την μέθοδο getService η οποία δημιουργεί και μας παρέχει ένα νέο αυθεντικοποιημένο Gmail service. Η μέθοδος αυτή καταφέρνει να κάνει την αυθεντικοποίηση με τα στοιχεία του χρήστη που χρειάζεται για να έχει πρόσβαση στο προσωπικό του λογαριασμό καλώντας την μέθοδο getCredentials. Η μέθοδος getCredentials, χρησιμοποιόντας το filepath που της έχουμε ορίσει, λαμβάνει από το αρχείο credentials.json τα στοιχεία του χρήστη που χρειάζεται τα οποία δημιουργήθηκαν στο Google Cloud πρίν ξεκινήσουμε την εφαρμογή.

Στη συνέχεια, η μέθοδος isMailExist [Εικόνα 17] καλεί την μέθοδο listMessagiesMachingQuery η οποία δέχεται ως παραμέτρους το Gmail service που έχει δημιουργηθεί από την μέθοδο getService [Εικόνα 18], το userId και φυσικά το query που χρησιμοποιείται για να μπορέσει να αναγνωρίσει το σωστό μήνυμα ηλεκτρονικού ταχυδρομείου και επιστρέφει τη λίστα με τα μηνύματα τα οποία διάβασε από το email του συγκεκριμένου χρήστη. Τέλος, αφού αποθηκεύσει τα μηνύματα σε μία μεταβλητή λίστας, ονομαζόμενη messages, ελέγχει αν η λίστα αυτή περιέχει μηνύματα και επιστρέφει μία μεταβλητή Boolean. Αν η λίστα είναι άδεια επιστρέφει false, αντίθετα αν περιέχονται μηνύματα επιστρέφει true.

```java
public static boolean isMailExist(String messageTitle) {
    try {

        Gmail service = getService();
        List<Message> messages = listMessagesMatchingQuery(service, userId: "me",  query: messageTitle + " AND is:unread");
        if(messages.isEmpty()) {
            return false;
        } else {
            return true;
        }
    } catch (IOException | GeneralSecurityException e) {
        System.out.println("Exception log" + e);
        return false;
    }
}
```

Εικόνα 17: Μέθοδος isMailExist

```java
private static Gmail getService() throws IOException, GeneralSecurityException {
    // Build a new authorized API client service.
    final NetHttpTransport HTTP_TRANSPORT = GoogleNetHttpTransport.newTrustedTransport();
    Gmail service = new Gmail.Builder(HTTP_TRANSPORT, JSON_FACTORY, getCredentials(HTTP_TRANSPORT))
            .setApplicationName(APPLICATION_NAME)
            .build();
    System.out.println(service);
    return service;
}
```

Εικόνα 18: Μέθοδος getService

```java
private static Credential getCredentials(final NetHttpTransport HTTP_TRANSPORT) throws IOException {
    // Load client secrets.
    InputStream in = new FileInputStream(new File(CREDENTIALS_FILE_PATH));
    if (in == null) {
        throw new FileNotFoundException("Resource not found: " + CREDENTIALS_FILE_PATH);
    }
    GoogleClientSecrets clientSecrets = GoogleClientSecrets.load(JSON_FACTORY, new InputStreamReader(in));
    // Build flow and trigger user authorization request.
    GoogleAuthorizationCodeFlow flow = new GoogleAuthorizationCodeFlow.Builder(
            HTTP_TRANSPORT, JSON_FACTORY, clientSecrets, SCOPES)
            .setDataStoreFactory(new FileDataStoreFactory(new java.io.File(TOKENS_DIRECTORY_PATH)))
            .setAccessType("offline")
            .build();
    LocalServerReceiver receiver = new LocalServerReceiver.Builder().setPort(9999).build();
    return new AuthorizationCodeInstalledApp(flow, receiver).authorize( userId: "user");
}
```

Εικόνα 19: Μέθοδος getCreadentials

```java
private static List<Message> listMessagesMatchingQuery(Gmail service, String userId,
                                                       String query) throws IOException {
    ListMessagesResponse response = service.users().messages().list(userId).setQ(query).execute();
    List<Message> messages = new ArrayList<>();
    while (response.getMessages() != null) {
        messages.addAll(response.getMessages());
        if (response.getNextPageToken() != null) {
            String pageToken = response.getNextPageToken();
            response = service.users().messages().list(userId).setQ(query)
                    .setPageToken(pageToken).execute();
        } else {
            break;
        }
    }
    return messages;
}
```

Εικόνα 20: Μέθοδος listMessagesMachingQuery

Η μέθοδος getGMailData [Εικόνα 21] που επίσης καλείται από την getEmail ξεκινάει όμοια με την μέθοδο isMailExist. Καλεί δηλαδή τις μεθόδους getService και listMessagesMachingQuery [Εικόνα 20] που έχουν εξηγηθεί παραπάνω και έπειτα ελέγχει εάν το μέγεθος της λίστας που προέρχεται από την μέθοδο listMessagesMachingQuery είναι μεγαλύτερο του μηδενός. Αν λοιπον η λίστα αυτή περιέχει μηνύματα, επιστρέφει το περιεχόμενο των μηνυμάτων αυτών σε μορφή HashMap. Στην αντίθετη περίπτωση επιστρέφει ένα άδειο HashMap.

```
public static HashMap<String, String> getGmailData(String query, String expectedSubject) {
    try {
        Gmail service = getService();
        List<Message> messages = listMessagesMatchingQuery(service, USER_ID, query);
        // check here for messages.size()
        if(messages.size() > 0) {
            Message message = getMessage(service, USER_ID, messages, index: 0);
            JsonPath jp = new JsonPath(message.toString());
            String subject = expectedSubject;
            String data = jp.getString( path: "payload.body.data");//"payload.parts[0].body.data");
            byte[] byteString = data.getBytes(StandardCharsets.UTF_8);
            String body = new String(Base64.getDecoder().decode(byteString));
            String link = null;
            String[] arr;
            arr = body.split( regex: "\n");
            for (String s : arr) {
                s = s.trim();
                if (s.startsWith("http") || s.startsWith("https")) {...}
            }
            HashMap<String, String> hm = new HashMap<>();
            hm.put("subject", subject);
            hm.put("body", body);
            hm.put("link", link);
            return hm;
        } else {
            return new HashMap<>();
        }
    } catch (IOException | GeneralSecurityException e) {
        System.out.println("Some problem exists....");
        System.out.println(e.getMessage());
        throw new RuntimeException(e);
    }
}
```

Εικόνα 21: Μέθοδος getGmailData

Για να ολοκληρωθεί η διαδικασία η μέθοδος getMail καλεί την μέθοδο deleteMessage [Εικόνα 22]. Η μέθοδος αυτή καλεί και πάλι τις μεθόδους getService και listMessagesMachingQuery και στη συνέχεια διαγράφει από το inbox του Gmail του χρήστη τα μηνύματα με τον συγκεκριμένο τίτλο τα οποία έχει επεξεργαστεί.

```
public static void deleteMessage(String query)
        throws IOException, GeneralSecurityException {
    Gmail service = getService();
    List<Message> messages = listMessagesMatchingQuery(service, USER_ID, query);
    service.users().messages().trash(USER_ID, messages.get(0).getId()).execute();
}
```

Εικόνα 22: Μέθοδος DeleteMessage

Επιστρέφοντας στον controller και έχοντας πια ένα γεμάτο αντικείμενο MessgeModel με τα στοιχεία που έχουν προκύψει από τη διαεύθυνση ηλεκτρονικού ταχυδρομείου του χρήστη,

χρησιμοποιείται RestTempelate για να μπορέσει ο controller να επικοινωνήσει με το κεντρικό API το οποίο είναι υπεύθυνο για την αποθήκευση των δεδομένων στη βάση.

Η δεύτερη μέθοδος που υπάρχει μέσα στον controller, είναι η sendMail [Εικόνα 23]. Η μέθοδος αυτή καλεί την μέθοδο sendEmail[Εικόνα 24] η οποία βρίσκεται στο service και εκείνη με τη σειρά της καλεί τη μέθοδο sendMailViaGmail [Εικόνα 25]. Η μέθοδος sendMailViaGmail καλεί τη μέθοδο getService που αναλύθηκε παραπάνω για να μπορέσει να συνδεθεί στην ηλεκτρονική διεύθυνση του χρήστη.

Στη συνέχεια καλεί τη μέθοδο createEmail [Εικόνα 26] η οποία δημιουργεί το μήνυμα και το επιστρέφει. Όλες οι μέθοδοι αυτές έχουν τρείς String παραμέτρους. Το to που αναφέρεται στο ποιος είναι ο παραλήπτης του μηνύματος (δηλαδή στη διεύθυνση ηλεκτρονικού ταχυδρομείου του παραλήπτη), το title που αναφέρεται στον τίτλο του μηνύματος και το body που αναφέρεται στο περιεχόμενο του μηνύματος. Τέλος, η μέθοδος sendMailViaGmail καλεί τη sendMessage [Εικόνα 27] και μέσω της μεθόδου αυτής γίνεται η αποστολή του μηνύματος στη συγκεκριμένη διεύθυνση ηλεκτρονικού ταχυδρομείου.

```
@RequestMapping("/sendMail")
public void sendmail() throws MessagingException, GeneralSecurityException, IOException {
    service.sendEmail( to: "sophiajeb93@gmail.com",  title: "HoneyBees Data",  body: "2022-01-15, 48.9, 49.479, 19.2");
}
```

Εικόνα 23: Μέθοδος sendMail

```
public void sendEmail(String to, String title, String body) throws IOException, GeneralSecurityException, MessagingException {
    sendEmailViaGmail(to, title, body);
}
```

Εικόνα 24: Μέθοδος sendEmail

```
public void sendEmailViaGmail(String to, String title, String body) throws IOException, GeneralSecurityException, MessagingException {
    Gmail service = getService();
    MimeMessage mimeMessage = createEmail(to, USER_ID, title, body);
    sendMessage(service, USER_ID, mimeMessage);
}
```

Εικόνα 25: Μέθοδος sendEmailViaGmail

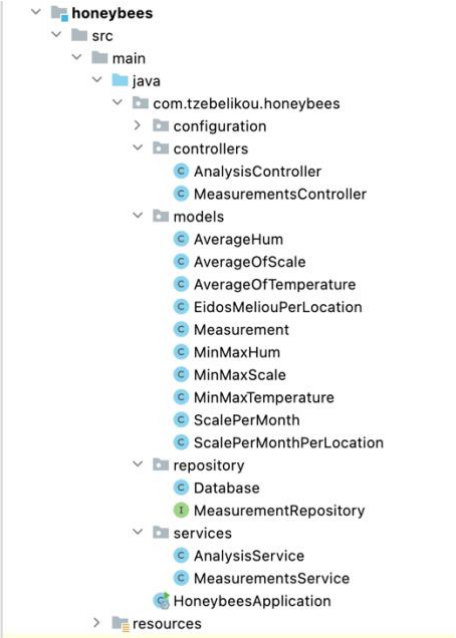
```
private static MimeMessage createEmail(String to,
                                       String from,
                                       String subject,
                                       String bodyText)
        throws MessagingException {
    Properties props = new Properties();
    Session session = Session.getDefaultInstance(props,  authenticator: null);
    MimeMessage email = new MimeMessage(session);
    email.setFrom(new InternetAddress(from));
    email.addRecipient(javax.mail.Message.RecipientType.TO,
            new InternetAddress(to));
    email.setSubject(subject);
    email.setText(bodyText.trim());
    return email;
}
```

Εικόνα 26: Μέθοδος createEmail

Οι δύο αυτοί controllers που αναλύθηκαν παραπάνω, καλούνται μέσω των scheduled tasks. Τα scheduled tasks είναι προγραμματισμένες διαδικασίες που γίνονται ανάλογα με τον χρόνο που τις έχουμε προγραμματίσει. Το Spring Boot framework μας επιτρέπει να προγραμματίσουμε διαδικασίες προσθέτοντας απλά το annotation @Scheduled και δηλώνοντας τον χρόνο που θέλουμε να πραγματοποιούνται.

Υλοποίηση του HoneyBees API

Η ενότητα (module) honeybees είναι το βασικό API όλης της εφαρμογής και είναι υπεύθυνο για την επικοινωνία των δύο άλλων modules με τη βάση δεδομένων. Χρησιμοποιεί επίσης το Spring Boot Framework και το MVC Design pattern έτσι ώστε να υπάρχει ομοιομορφία στις τεχνολογίες που χρησιμοποιούνται για την παρούσα εφαρμογή. Από τη στιγμή που το design pattern είναι ίδιο με το DataCollectorAPI, τότε υπάρχουν και τα ίδια πακέτα (models, services, controllers) αλλά προστίθεται ακόμα ένα πακέτο, το repository. Το πακέτο repository περιέχει μόνο ένα interface το MeasurementRepository [Εικόνα 28] το οποίο χρησιμοποιείται με μοναδικό σκοπό να κληρονομήσει το CrudRepository που παρέχεται από το Spring Boot Framework. Το CrudRepository παρέχει έτοιμες μεθόδους που βοηθούν στην αυτοματοποίηση των βασικών λειτουργειών μίας εφαρμογής, δηλαδή στη δημιουργία, τη διόρθωση και τη διαγραφή μίας εγγραφής στη βάση δεδομένων.



Εικόνα 27: Δομή honeybees module

```
public interface MeasurementRepository extends CrudRepository<Measurement, Long> {

}
```

Εικόνα 28: Interface MeasurementRepository

Ξεκινώντας από τον MeasurementController [Εικόνα 29] βλέπουμε ότι περιέχει τρείς μεθόδους. Οι δύο από αυτές παίρνουν δεδομένα από τη βάση δεδομένων ενώ η τρίτη κάνει εγγραφή στη βάση δεδομένων.

```java
@CrossOrigin("*")
@RestController
@RequestMapping("/api/measurements")
public class MeasurementsController {

    @Autowired
    public MeasurementsService service;

    @ExceptionHandler(Exception.class)
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR, reason = "Error message")
    public void handleError() {
    }

    @GetMapping("/getTrainData")
    public List<Measurement> getTrainData(){ return service.findTrainData();}

    @GetMapping("/getTestDataForSugkomidh")
    public List<Measurement> getTestDataForSugkomidh(){ return service.findTestDataForSugkomidh();}

    @GetMapping("/getTestDataForTrugos")
    public List<Measurement> getTestDataForTrugos(){ return service.findTestDataForTrugos();}

    @PostMapping("/saveMeasurement")
    public Measurement saveMeasurement(@RequestBody Measurement measurement) {
        return service.saveMeasurement(measurement);
    }
```

Εικόνα 29: MeasurementController class

Η μέθοδος saveMeasurement είναι αυτή η οποία καλείται από το DataCollectorAPI. Αφού το DataCollectorAPI έχει συνδεθεί με το ηλεκτρονικό ταχυδρομείο του χρήστη και έχει συλλέξει τα δεδομένα τα οποία έχουν αποσταλεί από τη ζυγαριά, τότε καλεί από τον controller του, μέσω RestTemplate, τη μέθοδο saveMeasurement [Εικόνα 30]. Η μέθοδος saveMeasurement κάλεί την μέθοδο saveMeasurement που βρίσκεται μέσα στο MeasurementService με παράμετρο ένα αντικείμενο Measurement. Το αντικείμενο αυτό είναι γεμάτο με την πληροφορία που έχει συλλεχθεί από το ηλεκτρονικό ταχυδρομείο. Με τη σειρά της η saveMeasurement του MeasurementService καλεί τη μέθοδο save του MeasurementRepository η οποία ουσιαστικά κληρονομείται από το CrudRepository το οποίο παρέχεται από το Spring Boot Framework και έτσι τα δεδομένα αποθηκεύονται στη βάση δεδομένων.

```java
    public Measurement saveMeasurement(Measurement measurement){

        return repository.save(measurement);
    }
```

Εικόνα 30: Μέθοδος saveMeasurement από το MeasurementService

Ο controller getTrainData [Εικόνα 31] δημιουργήθηκε για να επιστρέφει από τη βάση δεδομένων ένα σύνολο δεδομένων με σκοπό να χρησιμοποιηθεί από την ενότητα (module) Classifier ως δεδομένα εκπαίδευσης. Η μέθοδος getTrainData του controller καλεί την μέθοδο findTrainData από το MeasurementService [Εικόνα 32].

```java
    @GetMapping("/getTrainData")
    public List<Measurement> getTrainData(){ return service.findTrainData();}
```

Εικόνα 31: Controller getTrainData

```
public List<Measurement> findAll() {
    return (List<Measurement>) repository.findAll();
}

public List<Measurement> findTrainData() {
    List<Measurement> measurements = findAll();
    List<Measurement> trainData = new ArrayList<>();

    measurements.forEach(measurement -> {
        if (measurement.getId() < 50 ){
            trainData.add(measurement);
        }
    });
    return trainData;
}
```

Εικόνα 32: Μέθοδοι findTrainData και findAll του MeasurementService

Η μέθοδος findTrainData η οποία βρίσκεται στο MeasurementService αρχικά καλεί την μέθοδο findAll. Η μέθοδος findAll καλεί την μέθοδο findAll που βρίσκεται στο CrudRepository και κληρονομείται από το MeasurementRepository και στη συνέχεια επιστρέφει από τη βάση δεδομένων μία λίστα από αντικείμενα Measurement. Η λίστα αυτή περιέχει όλα τα δεδομένα που περιέχονται στη βάση δεδομένων. Στη συνέχεια η findTrainData δημιουργεί μία νέα λίστα, την trainData, από αντικείμενα Measurement και την γεμίζει με τα πενήντα πρώτα αντικείμενα της λίστας measurements (ο αριθμός πενήντα επιλέχθηκε για τον λόγο ότι είναι περίπου το 1/3 των συνολικών δεδομένων που περιέχονται στη βάση δεδομένων). Και τέλος η λίστα αυτή επιστρέφεται στον controller.

Αντίστοιχα η ίδια διαδικασία χρησιμοποιείται και στους controllers getTestDataForSugkomidh και getTestDataForTrugos [Εικόνα 33] οι οποίοι επιστρέφουν όλα τα υπόλοιπα δεδομένα που περιέχονται στη βάση δεδομένων εκτός από τα πρώτα πενήντα δεδομένα. Οι δύο αυτοί controllers επιστρέφουν τα δεδομένα σε μορφή json αφού είναι rest controllers.

```
@GetMapping("/getTestDataForSugkomidh")
public List<Measurement> getTestDataForSugkomidh(){ return service.findTestDataForSugkomidh();}

@GetMapping("/getTestDataForTrugos")
public List<Measurement> getTestDataForTrugos(){ return service.findTestDataForTrugos();}
```

Εικόνα 33: Controller getTestData

```
public List<Measurement> findTestDataForSugkomidh() {
    List<Measurement> measurements = findAll();
    List<Measurement> testData = new ArrayList<>();

    measurements.forEach(measurement -> {
        if (measurement.getId() > 63 && measurement.getId() < 132 ){
            testData.add(measurement);
        }
    });
    System.out.println(testData);
    return testData;
}

public List<Measurement> findTestDataForTrugos() {
    List<Measurement> measurements = findAll();
    List<Measurement> testData = new ArrayList<>();

    measurements.forEach(measurement -> {
        if (measurement.getId() >= 131 && measurement.getId() <= 133 ){
            testData.add(measurement);
        }
    });
    System.out.println(testData);
    return testData;
}
```

Εικόνα 34: Μέθοδος findTestData στο MeasurementService

Τέλος, με σκοπό την επίτευξη στατιστικής ανάλυσης σχετικά με τα δεδομένα που έχουν συλλεχθεί, έχει δημιουργηθεί μέσα στο πακέτο repository, η τάξη Database. Η τάξη Database ουσιαστικά δημιουργεί ένα JDBC Connection [Εικόνα 35] με την βάση δεδομένων της εφαρμογής και από εκεί με queries επιστρέφει δεδομένα βασισμένα σε συγκεκριμένα ερωτήματα οι απαντήσεις των οποίων είναι ωφέλιμες προς τους ανθρώπους που ασχολούνται με την μελισσοκομεία.

Τα ερωτήματα αυτά είναι :

1. Επέστρεψε τον μέσο όρο της εξωτερικής υγρασίας ανά τοποθεσία και ζυγαριά: location, zigaria_id, AVG(hum) as average FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
2. Επέστρεψε την μικρότερη και τη μεγαλύτερη τιμή της εξωτερικής υγρασίας ανά τοποθεσία και ζυγαριά : SELECT location, zigaria_id, MIN(hum) as minHum, MAX(hum) as maxHum FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
3. Επέστρεψε τον μέσο όρο της εξωτερικής θερμοκρασίας ανά τοποθεσία και ζυγαριά: SELECT location, zigaria_id, AVG(temperatureout) as averageOfTemperature FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
4. Επέστρεψε την μικρότερη και τη μεγαλύτερη τιμή της εξωτερικής θερμοκρασίας ανά τοποθεσία και ζυγαριά : SELECT location, zigaria_id, MIN(temperatureout) as minTemperature, MAX(temperatureout) as maxTemperature FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
5. Επέστρεψε τον μέσο όρο του βάρους της κυψέλης ανά τοποθεσία και ζυγαριά : SELECT location, zigaria_id, AVG(scale) as averageOfScale FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
6. Επέστρεψε την μικρότερη και τη μεγαλύτερη τιμή του βάρους της κυψέλης ανά τοποθεσία και ζυγαριά : SELECT location, zigaria_id, MIN(scale) as minScale, MAX(scale) as maxScale FROM ptyxiakh_db.measurements GROUP BY location, zigaria_id ORDER BY location, zigaria_id;
7. Επέστρεψε τον μέσο όρο του βάρους της κυψέλης ανά μήνα: SELECT DATE_FORMAT(STR_TO_DATE(`date`, '%d/%m/%Y'),'%m') as monthOfTheYear,

AVG(scale) as averageOfScalePerMonth FROM ptyxiakh_db.measurements GROUP BY DATE_FORMAT(STR_TO_DATE(`date`, '%d/%m/%Y'),'%m')

8. Επέστρεψε τον μέσο όρο του βάρους της κυψέλης ανά μήνα και τοποθεσία: SELECT DATE_FORMAT(STR_TO_DATE(`date`, '%d/%m/%Y'),'%m') as monthOfYear, location, AVG(scale) as averageOfScale FROM ptyxiakh_db.measurements GROUP BY DATE_FORMAT(STR_TO_DATE(`date`, '%d/%m/%Y'),'%m'), location ORDER BY DATE_FORMAT(STR_TO_DATE(`date`, '%d/%m/%Y'),'%m')

```
public List<AverageHum> getAverageHumidityPerLocationPerZigaria() {

    String query = "SELECT location, zigaria_id, AVG(hum) as average FROM ptyxiakh_db.measurements\n" +
            "GROUP BY location, zigaria_id\n" +
            "ORDER BY location, zigaria_id;";

    List<AverageHum> arraylist = new ArrayList<>();

    try {
        Connection connection = DriverManager.getConnection(FULL_DB_URL, DB_USER, DB_PASSWD);
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(query);

        while (rs.next()){
            String location = rs.getString( columnLabel: "location");
            int zigaria_id = rs.getInt( columnLabel: "zigaria_id");
            float average = rs.getFloat( columnLabel: "average");

             AverageHum result = new AverageHum(location, zigaria_id, average);
             arraylist.add(result);
        }
        rs.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        System.out.println("Something went wrong: " + e.getMessage());
    }
    return arraylist;
}
```

Εικόνα 35: Παράδειγμα JDBC Connection και επιστροφή δεδομένων μέσω ερωτημάτων

Αφού έχει γίνει η διαδικασία εξόρυξης δεδομένων μέσω JDBC Connection, το AnalysisService παίρνει τα δεδομένα αυτά από την εκάστωτε μέθοδο της τάξης Database και τα προοθεί στον AnalysisController ο οποίος με τη σειρά του θα τα προοθήσει σε μορφή json σε όποια διαδικασία τον καλέσει.

```java
@Service
public class AnalysisService {

    Database database = new Database();

    public List<AverageHum> getAverageHum() { return database.getAverageHumidityPerLocationPerZigaria(); }

    public List<MinMaxHum> getMinMaxHum() { return database.getMinMaxHumidityPerLocationPerZigaria(); }

    public List<AverageOfTemperature> getAverageOfTemperature(){
        return database.getAverageTemperaturePerLocationPerZigaria();
    }

    public List<MinMaxTemperature> getMinMaxTemperature(){
        return database.getMinMaxTemperaturePerLocationPerZigaria();
    }

    public List<AverageOfScale> getAverageOfScale() { return database.getAverageOfScalePerLocationPerZigaria();

    public List<MinMaxScale> getMinMaxScale() { return database.getMinMaxScalePerLocationPerZigaria(); }

    public List<EidosMeliouPerLocation> getEidosMeliouPerLocation() { return database.getEidosMeliouPerLocation(

    public List<ScalePerMonth> getScalePerMonth(){
        return database.getAverageOfScalePerMonth();
    }

    public List<ScalePerMonthPerLocation> getScalePerMonthPerLocation(){
        return database.getAverageOfScalePerMonthPerLOcation();
    }
```

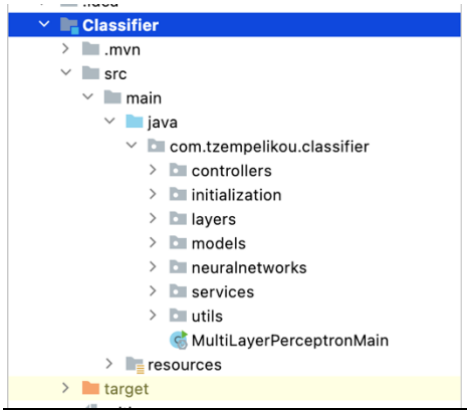
Εικόνα 36: AnalysisService

```java
@RestController
@RequestMapping("/api/analysis")
public class AnalysisController {

    @Autowired
    private AnalysisService service;

    @ExceptionHandler(Exception.class)
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR, reason = "Error message")
    public void handleError() {
    }

    @GetMapping("/averageHum")
    public List<AverageHum> getAverageHum() { return service.getAverageHum(); }

    @GetMapping("/minMaxHum")
    public List<MinMaxHum> getMinMaxHum() { return service.getMinMaxHum(); }

    @GetMapping("/averageOfTemperature")
    public List<AverageOfTemperature> getAverageOfTemperature() { return service.getAverageOfTemperature(); }

    @GetMapping("/minMaxTemperature")
    public List<MinMaxTemperature> getMinMaxTemperature() { return service.getMinMaxTemperature(); }

    @GetMapping("/averageOfScale")
    public List<AverageOfScale> getAverageOfScale() { return service.getAverageOfScale(); }

    @GetMapping("/minMaxScale")
    public List<MinMaxScale> getMinMaxScale() { return service.getMinMaxScale(); }
```

Εικόνα 37: AnalysisController

Υλοποίηση του Classifier Service

Το service classifier αποτελεί το κομμάτι μηχανικής μάθησης που περιέχεται στην εφαρμογή και χρησιμοποιείται για να την ταξινόμηση των δεδομένων σε περίοδο συγκομιδής ή περίοδο τρύγους. Όπως και τα τρία προηγούμενα services χρησιμοποιεί Spring Framework και MVC Design Pattern.



Εικόνα 38: Δομή Classifier Service

Για να καταφέρει το Classifier Service να κάνει την ταξινόμηση χρειάζεται να εισαχθούν σε αυτό τα testData και τα trainData που έχουν παραχθεί στο honeyBees service που εξηγήθηκε παραπάνω. Για τη διαδικασία αυτή δημιουργήθηκε η τάξη GetaDataService η οποία με τη βοήθεια του RestTemplate καλεί το honeyBees service για να μπορέσει να λάβει τα δεδομένα.

```
public class GetDataService {

    // of implemented tests
    public int noOfTestData = 2;

    public List<Measurement> getTrainData(){
        RestTemplate restTemplate = new RestTemplate();
        Measurement[] measurementContent = restTemplate.getForObject( url: "http://localhost:9000/api/measurements/getTrainData", Measurement[].class);
        return Arrays.asList(measurementContent);

    }

    public List<Measurement> getTestDataForSugkomidh(){
        RestTemplate restTemplate = new RestTemplate();
        Measurement[] measurementContent = restTemplate.getForObject( url: "http://localhost:9000/api/measurements/getTestDataForSugkomidh", Measurement[].class);
        return Arrays.asList(measurementContent);

    }

    public List<Measurement> getTestDataForTrugos(){
        RestTemplate restTemplate = new RestTemplate();
        Measurement[] measurementContent = restTemplate.getForObject( url: "http://localhost:9000/api/measurements/getTestDataForTrugos", Measurement[].class);
        return Arrays.asList(measurementContent);

    }
}
```

Εικόνα 39: GetDataService

Στη συνέχεια το GetDataService [Εικόνα 39] καλείται από το MultiLayerPerceptronService. Η τάξη MultiLayerPerceptronService κατά τη διαδικασία ενός object χρειάζεται να γνωρίζει τον αριθμό των inputs, τον αριθμό των hidden layers, τον αριθμό των outputs(patterns) και προεραιτικά έναν random αριθμό και την συνάρτηση ενεργοποίησης (activation method). Κατά τη δημιουργία του Multi Layer Perceptron

δημιουργούνται τα απαραίτητα hidden layers του δικτύου και τα multi class logistic layers (output patterns) που απαιτούνται. Τα hidden ή denseLayer ορίζουν για κάθε νευρώνα το weight και το bias που θα χρησιμοποιήσει το activation function το οποίο για αυτό το δίκτυο είναι Tanh.

Η hyperbolic tangent activation function αναφέρεται επίσης ως συνάρτηση Tanh, μοιάζει πολύ με την συνάρτηση ενεργοποίησης σιγμοειδούς που αναφέρθηκε στο κομμάτι της έρευνας και μάλιστα έχει το ίδιο σχήμα. Η συνάρτηση αυτή παίρνει οποιαδήποτε πραγματική τιμή ως είσοδο και εξάγει τιμές στην περιοχή από -1 έως 1. Όσο μεγαλύτερη είναι η είσοδος (πιο θετική), τόσο πιο κοντά στο 1, 0 θα είναι η τιμή εξόδου, ενώ όσο μικρότερη είναι η είσοδος(πιο αρνητική), η τιμή εξόδου θα είναι πιο κοντά στο -1, 0. Η συνάρτηση ενεργοποίησης Tanh υπολογίζεται ως εξής: (e^x – e^-x) / (e^x + e^-x). Όπου e είναι μια μαθηματική σταθερά που είναι η βάση του φυσικού λογάριθμου.[18]

Η διαδικασία του training χωρίζει τα δεδομένα σε mini batches για να μπορέσει να στείλει το ίδιο δεδομένο σε όλα τα hidden layers. Για το συγκεκριμένο νευρωνικό δίκτυο υπάρχουν εφτά (7) διαφορετικά inputs, όπου τα δίνουμε σε 4 hidden layers. Ο αριθμός epochs (εποχών) είναι 5000 και ο ρυθμός μάθησης είναι 0,1. Αφού χωριστούν τα inputs σε mini batches ξεκινά η διαδικασία μάθησης του Multi Layer Perceptron με τα δεδομένα που έρχονται από το GetDataService. Όταν ολοκληρωθεί η διαδικασία εκμάθησης του MLP μπορεί να γίνει το train ξανά με δεδομένα που έρχονται από το honeyBees service. Κατά τη διαδικασία οπου έχουμε testData, τα αποτελέσματα που είναι το accuracy, το precision και το recall αποστέλονται στο UI Service για οπτικοποίηση σε διαφόρων ειδών charts.

Accuracy[19]

Το accuracy (ακρίβεια) μπορεί να μας πει αμέσως εάν ένα μοντέλο εκπαιδεύεται σωστά και πώς μπορεί να αποδώσει γενικά. Ωστόσο, δεν δίνει λεπτομερείς πληροφορίες σχετικά με την εφαρμογή του στο πρόβλημα. Το πρόβλημα με τη χρήση της ακρίβειας ως κύριας μέτρησης απόδοσης είναι ότι δεν τα πάει καλά όταν υπάρχει σοβαρή ανισορροπία κατηγορίας.

Precision[19]

Το precision βοηθάει όταν το κόστος των ψευδώς θετικών είναι υψηλό. Ας υποθέσουμε λοιπόν ότι το πρόβλημα αφορά την ανίχνευση του καρκίνου του δέρματος. Εάν έχουμε ένα μοντέλο που έχει πολύ χαμηλό precision, τότε σε πολλούς ασθενείς θα πουν ότι έχουν μελάνωμα και αυτό θα περιλαμβάνει κάποιες λανθασμένες διαγνώσεις. Πολλές επιπλέον δοκιμές και άγχος διακυβεύονται. Όταν τα ψευδώς θετικά αποτελέσματα είναι πολύ υψηλά, όσοι παρακολουθούν τα αποτελέσματα θα μάθουν να τα αγνοούν αφού βομβαρδιστούν με ψευδείς συναγερμούς.

$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

Recall[19]

Το recall (επανάκληση) βοηθά όταν το κόστος των ψευδών αρνητικών αποτελεσμάτων είναι υψηλό. Δηλαδή είναι το αντίθετο του Precision.

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

F1 Scope[19]

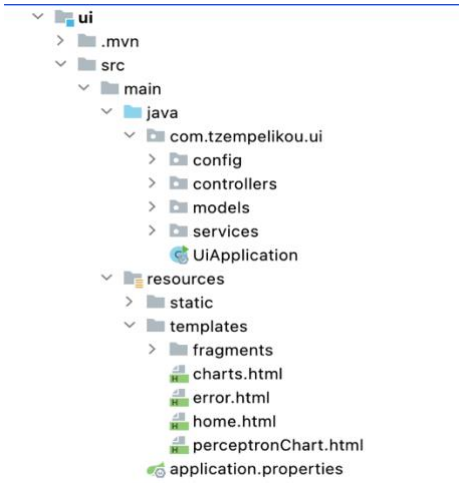
Το F1 scope είναι ένα συνολικό μέτρο της ακρίβειας ενός μοντέλου που συνδυάζει precosion και recall, με αυτόν τον περίεργο τρόπο που η πρόσθεση και ο πολλαπλασιασμός απλώς αναμειγνύουν δύο συστατικά για να φτιάξουν ένα ξεχωριστό πιάτο συνολικά. Δηλαδή, μια καλή βαθμολογία F1 σημαίνει ότι έχετε χαμηλά ψευδώς θετικές και χαμηλά ψευδώς αρνητικές τιμές, επομένως εντοπίζετε σωστά τις πραγματικές απειλές και δεν ενοχλείστε από ψευδείς συναγερμούς. Μια βαθμολογία F1 θεωρείται τέλεια όταν είναι 1, ενώ το μοντέλο είναι ολική αποτυχία όταν είναι 0.

Θυμηθείτε: Όλα τα μοντέλα είναι λάθος, αλλά μερικά είναι χρήσιμα. Δηλαδή, όλα τα μοντέλα θα δημιουργήσουν κάποια ψευδώς αρνητικά, κάποια ψευδώς θετικά και πιθανώς και τα δύο. Ενώ μπορείτε να συντονίσετε ένα μοντέλο για να ελαχιστοποιήσετε το ένα ή το άλλο, συχνά αντιμετωπίζετε μια αντιστάθμιση, όπου η μείωση των ψευδώς αρνητικών οδηγεί σε αύξηση των ψευδώς θετικών ή το αντίστροφο. Θα χρειαστεί να κάνετε βελτιστοποίηση για τις μετρήσεις απόδοσης που είναι πιο χρήσιμες για το συγκεκριμένο πρόβλημά σας.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Υλοποίηση του UI Service

Το ui service είναι υπεύθυνο για την κατασκευή γραφημάτων έτσι ώστε να γίνει η παρουσίαση των αποτελεσμάτων της έρευνας στο χρήστη. Είναι δομημένο με τον ίδιο τρόπο όπως και τα υπόλοιπα services της εφαρμογής, δηλαδή με το Spring framework και MVC Design Pattern με τη διαφορά ότι χρησιμοποιεί Thymelief για τη δημιουργία templates, δηλαδή τη δομή των σελίδων που φαίνονται στον ιστό.



Εικόνα 40: Δομή UI Service

Το Thymelief είναι μία σύγχρονη server-side template μηχανή για περιβάλλοντα ιστού και για αυτόνομα περιβάλλοντα. Ο κύριος στόχος του Thymeleaf είναι να φέρει κομψά φυσικά πρότυπα στη ροή εργασιών ανάπτυξής — HTML που μπορούν να εμφανιστούν σωστά σε προγράμματα περιήγησης και επίσης να λειτουργήσουν ως στατικά πρωτότυπα, επιτρέποντας ισχυρότερη συνεργασία σε ομάδες ανάπτυξης λογισμικού.

Ξεκινώντας παρατηρούμε ότι το module ui περιέχει δύο controllers. Ο κάθε controller αντιστοιχεί σε μία σελίδα HTML που παρέχεται στον χρήστη. Οι controllers αυτοί ονομάζονται HomeController [Εικόνα 41] και DataController αντίστοιχα. Ο HomeController περιέχει μεθόδους GetMapping οι οποίες χρησιμοποιούνται από την σελίδα home.html [Εικόνα 42]. Οι μέθοδοι αυτοί χρησιμοποιούνται για τη δημιουργία δύο κουμπιών από τη σελίδα home.html τα οποία με το πάτημά τους κάνουν redirect τον χρήστη στην σελίδα που επιθυμεί να κατευθυνθεί.

```
@Controller
@Slf4j
@RequiredArgsConstructor
public class HomeController {

    @GetMapping(value = "/")
    public String home(Model model) {
        model.addAttribute( attributeName: "serverTime", new Date());
        return "home";
    }

    @GetMapping(value = "/charts")
    public String chartsHome(Model model) { return "charts"; }

    @GetMapping(value = "/perceptronChart")
    public String perceptronChart(Model model) { return "perceptronChart"; }
}
```

Εικόνα 41: HomeController

```
<div>

    <div style="color: midnightblue; margin-top: 30px">
        <h2 style="margin-left: 38%;">ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ</h2>
        <h1 style="margin-left: 32%;"><strong>ΠΜΣ : </strong>Προηγμένα Συστήματα</h1>
        <h2 style="margin-left: 26%;">Ανάπτυξης Λογισμικού και Τεχνητής Νοημοσύνης</h2>
        <h3 style="margin-left: 40%;"><strong>Τζεμπελίκου Σοφία</strong></h3>
        <img src="images/unipi.jpg" alt="logo" class="center">
        <a href="/charts" class="btn btn-outline-primary btn-lg center" tabindex="-1" role="button" aria-disabled="true">Στατιστική Ανάλυση</a>
        <a href="/perceptronChart" class="btn btn-outline-primary btn-lg center" tabindex="-1" role="button" aria-disabled="true">Αποτελέσματα Classifier</a>
    </div>
```

Εικόνα 42: home.html

Σχετικά με τον DataController έχει δημιουργηθεί μία πιο περίπλοκη λογική. Ο controller αυτός είναι υπεύθυνος να παραλάβει τα δεδομένα από το honeyBees service και να προσαρμόσει τη μορφή τους (για παράδειγμα να δημιουργήσει λίστες) έτσι ώστε να μπορούν εύκολα να χρησιμοποιηθούν από το αρχείο charts.js το οποίο δημιουργεί τα charts. Για να μπορέσει να το κάνει αυτό, αρχικά κάνει dependency injection στο service ResponceEntityToListConverter [Εικόνα 44]. Το service αυτό με τη βοήθεια του RestTemplate καλεί τον AnalysisController που βρίσκεται στο honeyBees service και έτσι τα δεδομένα περνάνε στο ui service.

```java
@Service
public class ResponceEntityToListConverter {

    public List<AverageHum> convertAverageHumRestTemplateToList(){
        RestTemplate restTemplate = new RestTemplate();
        AverageHum[] averageHumArray = restTemplate.getForObject( url: "http://localhost:9000/api/analysis/averageHum", AverageHum[].class);
        return Arrays.asList(averageHumArray);
    }

    public List<MinMaxHum> convertMinMaxHumRestTemplateToList(){
        RestTemplate restTemplate = new RestTemplate();
        MinMaxHum[] minMaxHum = restTemplate.getForObject( url: "http://localhost:9000/api/analysis/minMaxHum", MinMaxHum[].class);
        return Arrays.asList(minMaxHum);
    }

    public List<AverageOfTemperature> convertAverageTemperatureRestTemplateToList(){
        RestTemplate restTemplate = new RestTemplate();
        AverageOfTemperature[] AverageOfTemperatureArray =
                restTemplate.getForObject( url: "http://localhost:9000/api/analysis/averageOfTemperature", AverageOfTemperature[].class);
        return Arrays.asList(AverageOfTemperatureArray);
    }

    public List<MinMaxTemperature> convertMinMaxTemperatureRestTemplateToList(){
        RestTemplate restTemplate = new RestTemplate();
        MinMaxTemperature[] minMaxTemperatureArray =
                restTemplate.getForObject( url: "http://localhost:9000/api/analysis/minMaxTemperature", MinMaxTemperature[].class);
        return Arrays.asList(minMaxTemperatureArray);
    }
```

Εικόνα 43: ResponceentityToListConverter

Στη συνέχεια ο DataController λαμβάνει τα δεδομένα και τα προσαρμόζει στη μορφή που ζητείται από το αρχείο JavaScript.

```java
@RestController
@Slf4j
@RequiredArgsConstructor
public class DataController {

    @Autowired
    private ResponceEntityToListConverter converter;

    // pie. line, bar    ---> pieData
    // stack             ---> columnData

    // average humidity per location and per zigaria_id
    @GetMapping(value = "/rest/DataForAverageOfHumidity")
    public List getDataForAverageOfHumidity() {

        List<AverageHum> averageHumList = converter.convertAverageHumRestTemplateToList();
        List pieData = new ArrayList<>();
        List dataElement = new ArrayList();
        List lableElement = new ArrayList();
        for (AverageHum averageHumModel : averageHumList){
            lableElement.add(averageHumModel.getLocation()+ " " + averageHumModel.getZigaria_id());
            dataElement.add(averageHumModel.getAverage());
        }
        pieData.add(lableElement);
        pieData.add(dataElement);
        return pieData;
    }
```

Εικόνα 44: Παράδειγμα προσαρμογής.δεδομένων απο τον DataController

Υπάρχουν δύο αρχεία javascript, το charts.js και το perceptronChart.js. Το πρώτο αφορά τα αποτελέσματα των ερωτημάτων για την στατιστική ανάλυση και το δεύτερο στα αποτελέσματα που προέρχονται από το Classifier service. Τα αρχεία αυτά, χρησιμοποιώντας functions δημιουργούν το καθένα από τα charts έχοντας τα δεδομένα που τους παράγονται από τον Controller.

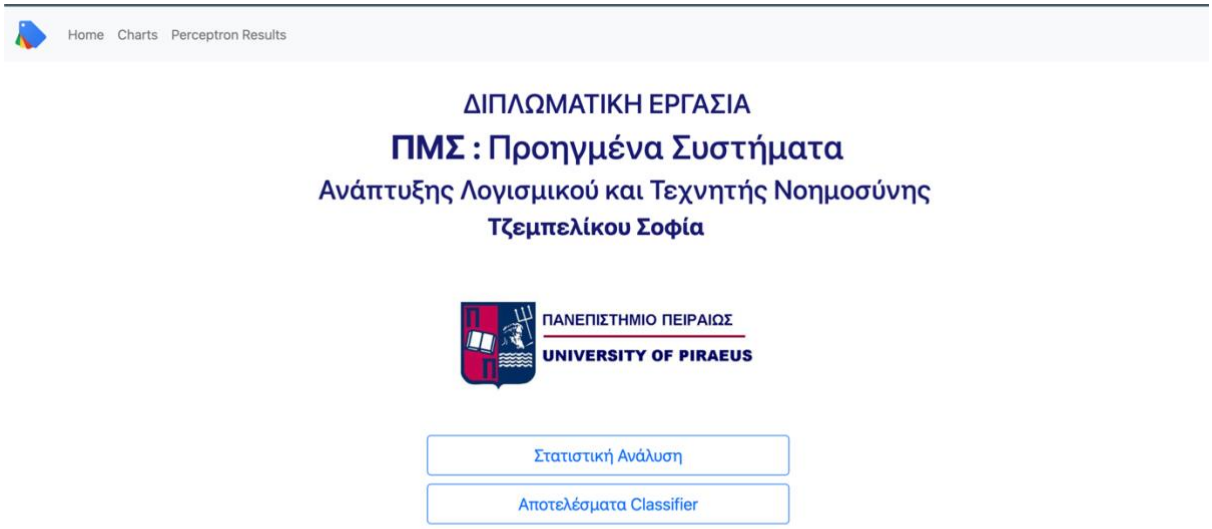
```
$.getJSON( url: "/rest/DataForAverageOfScale",  data: function(json) {
    new Chart(document.getElementById( elementId: "barchartContainerForAverageOfScale"), {
        type: 'bar',
        data: {
            labels: json[0],
            datasets: [{
                label: 'Μέση ποσότητα μελιού της κάθε τοποθεσίας και ζυγαριας',
                backgroundColor: backgroundColors,
                borderColor: borderColors,
                borderWidth: 1,
                hoverOffset: 4,
                data: json[1]
            }]
        },
        options: {
            title: {
                display: true,
                text: 'Bar Chart'
            },
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                yAxes: [{
                    ticks: {
                        beginAtZero:true
                    }
                }]
            }
        }
    });
});
```

Εικόνα 45: Παράδειγμα function απο το αρχείο charts.js

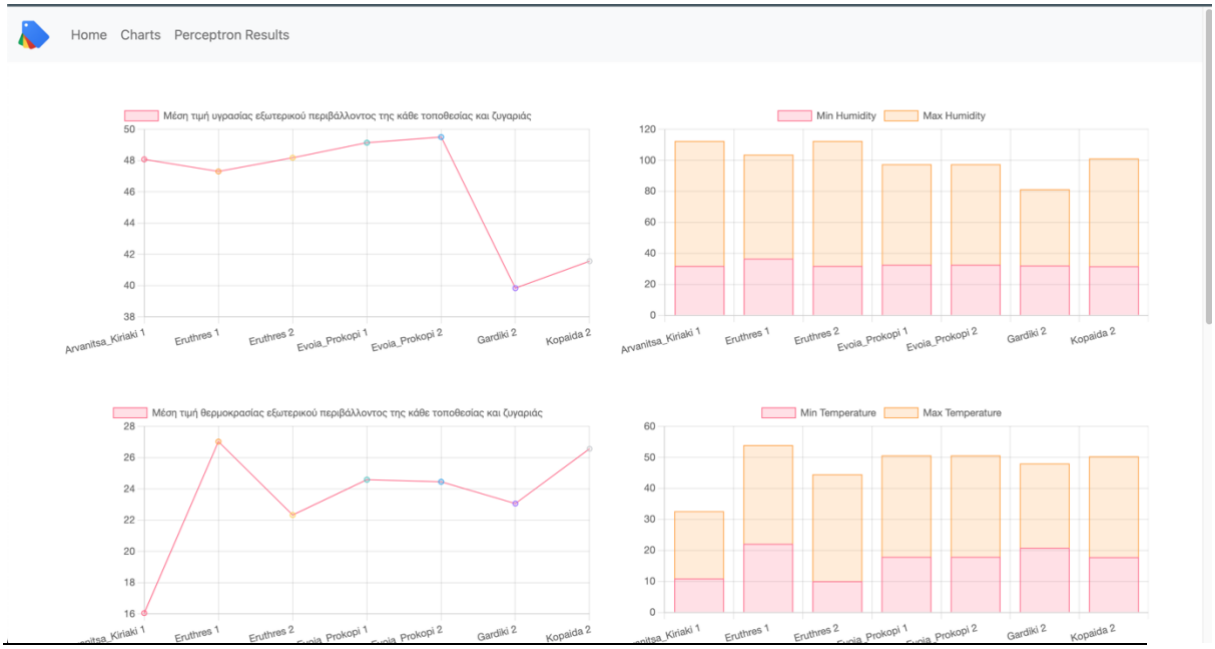
Και τέλος, τα αρχεία html είναι αυτά που προωθούν την πληροφορία στον τελικό χρήστη της εφαρμογής καλώντας το κάθε chart από το αρχείο javascript.

```
<div class="container">

    <br/>
    <br/>
    <div class="row">
        <div class="col">
            <canvas id="linechartContainer" width="200" height="300"></canvas>
        </div>
        <div class="col">
            <canvas id="stackchartContainer" width="200" height="300"></canvas>
        </div>
    </div>
    <br/>
    <br/>
    <div class="row">
        <div class="col">
            <canvas id="linechartContainerTemperature" width="200" height="300"></canvas>
        </div>
        <div class="col">
            <canvas id="stackchartContainerForMinMaxTemperature" width="200" height="300"></canvas>
        </div>
    </div>
    <br/>
    <br/>
```

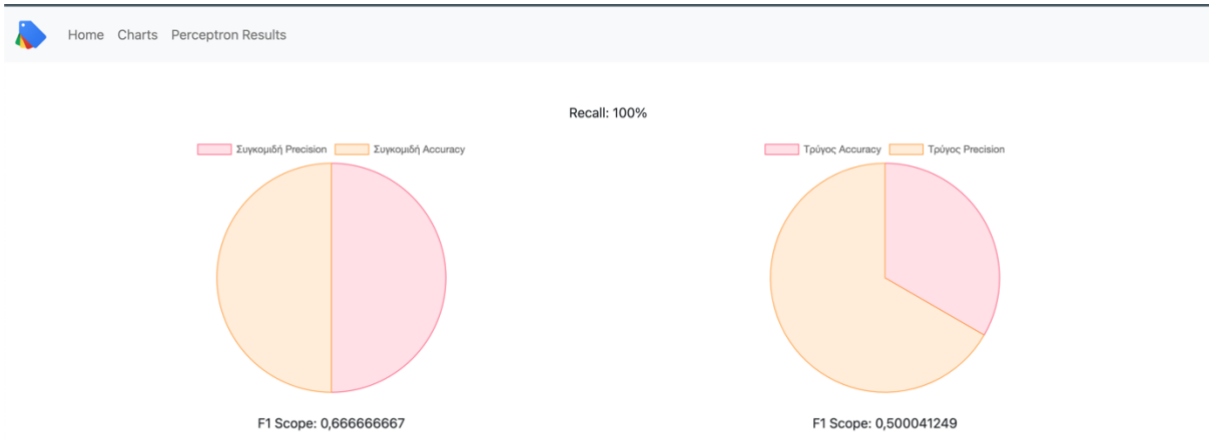
Figure 46: Charts.html



Εικόνα 47: Home.html στον Browser



Εικόνα 48: Charts.html στον Browser



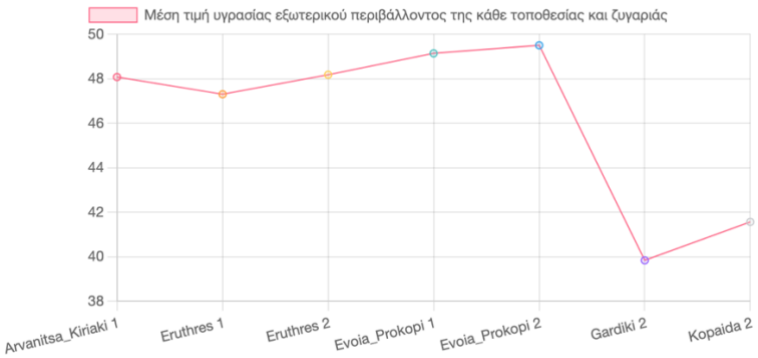
Εικόνα 49: Perceptrons Results στον Browser

Συμπεράσματα

Το αποτέλεσμα της παρούσας έρευνας είναι το γεγονός ότι έχει δημιουργηθεί μία ολοκληρωμένη λύση λογισμικού το οποίο με τη βοήθεια μηχανικής μάθησης αλλά και ερωτημάτων στη βάση δεδομένων παράγει στατιστικά αποτελέσματα σχετικά με την παραγωγή του μελιού, των καιρικών φαινομένων αλλά καταφέρνει να κάνει και ταξινόμηση των εποχών τρύγους και συγκομιδής.

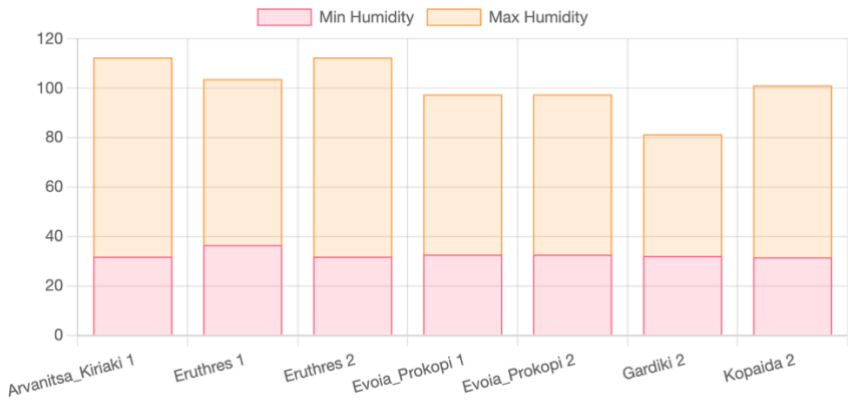
Ξεκινώντας από τη διαδικασία των ερωτημάτων στη βάση δεδομένων, μπορούμε από το ui να δούμε τα αποτελέσματα για το κάθε ερώτημα ξεχωριστά.

Ερώτημα 1 : Μέση τιμή υγρασίας εξωτερικού περιβάλλοντος της κάθε τοποθεσίας και ζυγαριάς.



Στο αποτέλεσμα του ερωτήματος αυτού σύμφωνα με το διάγραμμα παρατηρείται ότι η μεγαλύτερη τιμή υγρασίας βρίσκεται στο Προκόπι Ευβοιας με μέση τιμή 49,5 ενώ η μικρότερη βρίσκεται στη περιοχή Γαρδίκι με μέση τιμή 39,833.

Ερώτημα 2: Μικρότερη και μεγαλύτερη τιμή υγρασίας ανά περιοχή και ζυγαριά

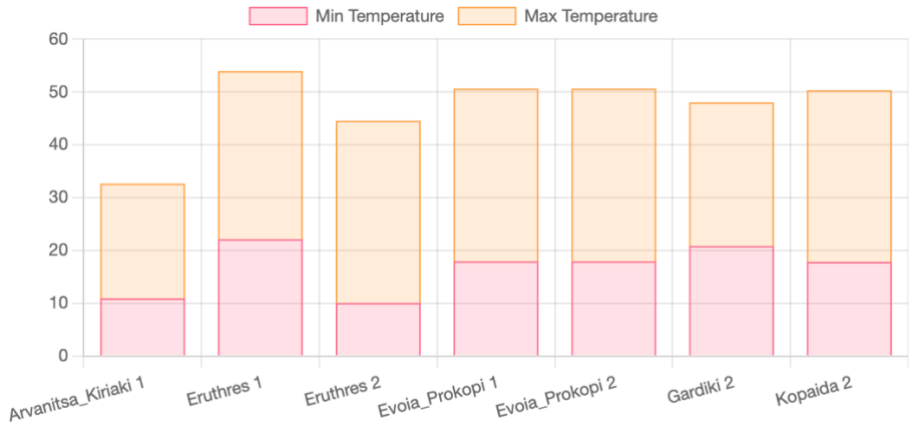


1. Αρβανίτσα-Κυριάκι: min: 31.8 max: 80.6
2. Ερυθρές ζυγαριά 1: min: 36.5 max: 67.1
3. Ερυθρές ζυγαριά 2: min: 31.8 max: 80.6
4. Προκόπι Ευβοίας ζυγαριά 1: min: 32.6 max: 64.8
5. Προκόπι Ευβοίας ζυγαριά 2: min: 32.6 max: 64.8
6. Γαρδίκι ζυγαριά 2: min: 32.1 max: 49.2
7. Κοπαϊδα ζυγαριά 2: min: 31.6 max: 69.5

Ερώτημα 3 : Μέση τιμή θερμοκρασίας εξωτερικού περιβάλλοντος της κάθε τοποθεσίας και ζυγαριάς.

Σύμφωνα με το διάγραμμα, η μεγαλύτερη τιμή θερμοκρασίας βρίσκεται στις Ερυθρές και συγκεκριμένα στη ζυγαριά με id = 1, με μ΄έση τιμή 27.2. Ενώ η μικρότερη μέση τιμή βρίσκεται στην Αρβανίτσα με τιμή 16,05.

Ερώτημα 4:Μικρότερη και μεγαλύτερη τιμή Θερμοκρασίας ανά περιοχή και ζυγαριά

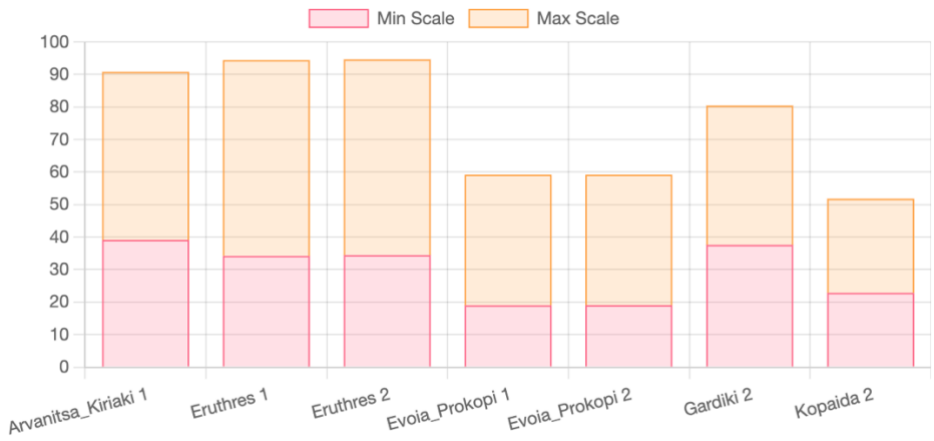


1. Αρβανίτσα-Κυριάκι: min: 10.9 max: 21.7
2. Ερυθρές ζυγαριά 1: min: 22.1 max: 31.8
3. Ερυθρές ζυγαριά 2: min: 10 max: 34.5
4. Προκόπι Ευβοίας ζυγαριά 1: min: 17.9 max: 32.7
5. Προκόπι Ευβοίας ζυγαριά 2: min: 17.9 max: 32.7
6. Γαρδίκι ζυγαριά 2: min: 20.8 max: 27.2
7. Κοπαϊδα ζυγαριά 2: min: 17.5 max: 32.5

Ερώτημα 5: Μέση ποσότητα μελιού της κάθε τοποθεσίας και ζυγαριάς

1. Αρβανίτσα-Κυριάκι: 45.055 kg
2. Ερυθρές ζυγαριά 1: 44.983 kg
3. Ερυθρές ζυγαριά 2: 44.863 kg
4. Προκόπι Ευβοίας ζυγαριά 1: 35.46 kg
5. Προκόπι Ευβοίας ζυγαριά 2: 34.525 kg
6. Γαρδίκι ζυγαριά 2: 38.889 kg
7. Κοπαϊδα ζυγαριά 2: 25.727 kg

Ερώτημα 6: Μικρότερη και μεγαλύτερη τιμή ποσότητας μελιού ανά περιοχή και ζυγαριά

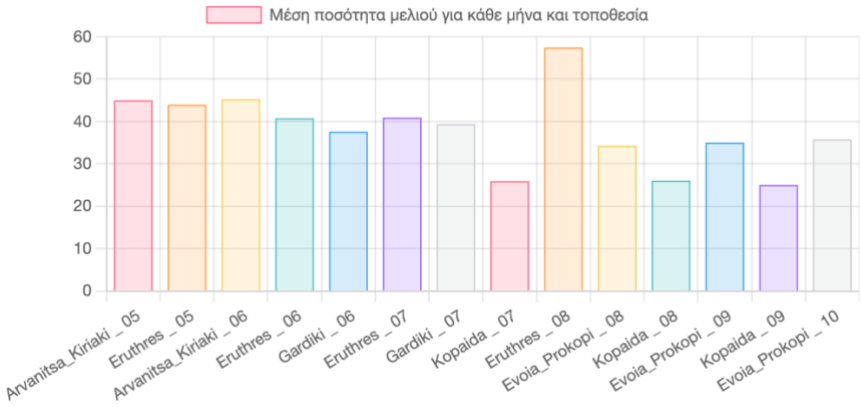


1. Αρβανίτσα-Κυριάκι: min: 39.138 kg max: 51.66 kg
2. Ερυθρές ζυγαριά 1: min: 34.237 kg max: 60.186 kg
3. Ερυθρές ζυγαριά 2: min: 34.465 kg max: 60.186 kg
4. Προκόπι Ευβοίας ζυγαριά 1: min: 19 kg max: 40.176 kg
5. Προκόπι Ευβοίας ζυγαριά 2: min: 19.037 kg max: 40.176 kg
6. Γαρδίκι ζυγαριά 2: min: 37.621 kg max: 42.801 kg
7. Κοπαϊδα ζυγαριά 2: min: 22.836 kg max: 28.957 kg

Ερώτημα 7: Μέση ποσότητα μελιού για κάθε μήνα παραγωγής

1. Μάιος: 44.475 kg
2. Ιούνιος: 41.356 kg
3. Ιούλιος: 37.398 kg
4. Αύγουστος: 39.832 kg
5. Σεπτέμβρης: 30.777 kg
6. Οκτώβρης: 35.761 kg

Ερώτημα 8: Μέση ποσότητα μελιού για κάθε μήνα και τοποθεσία



Μάιος
- Αρβανίτσα : 44.986 kg
- Ερυθρές : 43,964 kg

Ιούνιος
- Αρβανίτσα: 45,215 kg
- Ερυθρές: 40.731 kg

Ιούλιος
- Ερυθρές: 40.731 kg
- Γαρδίκι : 39.385 kg
- Κοπαϊδα: 25.941 kg

Αύγουστος
- Ερυθρές: 57,436 kg
- Προκόπι Ευβοίας: 34.262 kg
- Κοπαϊδα: 26.067 kg

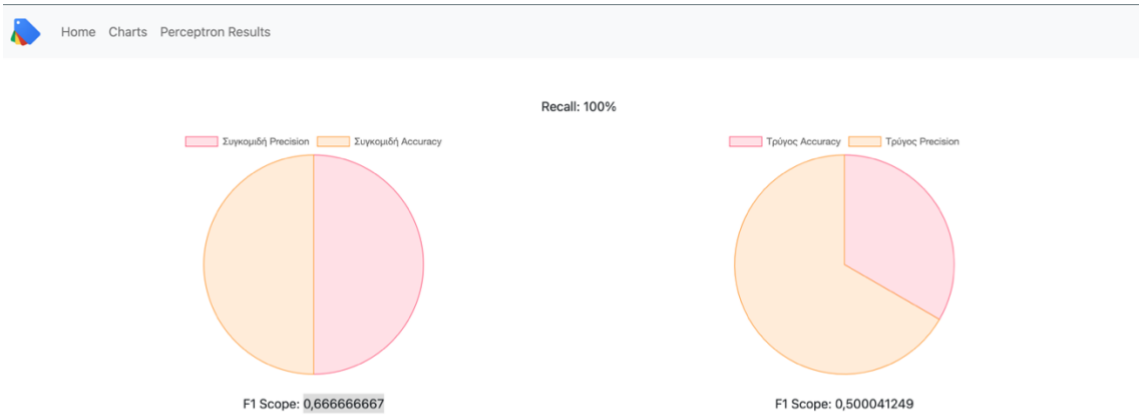
Σεπτέμβρης
- Προκόπι Ευβοίας: 34.999 kg
- Κοπαϊδα: 25 kg

Οκτώβρης
- Προκόπι Ευβοίας: 35.761 kg

Αποτελέσματα Classifier

Σύμφωνα με τα αποτελέσματα του Classifier που φαίνονται στο ui, φαίνεται ότι στην περίπτωση της συγκομιδής, επειδή είμαστε πολύ κοντά στη διαδικασία τρύγου αλλά όχι ακόμα εκεί, το F1 Score υπολογίζεται να είναι μεγαλύτερο του 0.5 συγκεκριμένα 0,666666667 που σημαίνει για τον μελισσοκόμο ότι μπορεί να συνεχίσει τη συγκομιδή.

Όταν το F1 Score φτάσει τη τιμή 0.5, όπως στην περίπτωση του τρύγου, τότε είναι ενδεικτικό για τον μελισσοκόμο ότι πρέπει να τρυγίσει και άρα γνωρίζει με ακρίβεια μίας ημέρας (την προηγούμενη του τρύγου), πότε πρέπει να μαζέψει το μέλι.



Εικόνα 50: Αποτελέσματα Classifier

Βιβλιογραφία

[1]http://melissomania.gr/%CE%B5%CF%81%CE%B3%CE%AC%CF%84%CF%81%CE%B9%CE%B1/%CE%B5%CF%80%CE%B9%CE%BA%CE%BF%CE%BD%CE%AF%CE%B1%CF%83%CE%B7

[2]https://el.wikipedia.org/wiki/%CE%95%CF%80%CE%B9%CE%BA%CE%BF%CE%BD%CE%AF%CE%B1%CF%83%CE%B7

[3]https://www.greenpeace.org/greece/epirease/prostateuoume-melisses/

[4]https://link.springer.com/content/pdf/10.1007/s13592-018-0565-3.pdf

[5]https://arxiv.org/pdf/1802.04557.pdf?fbclid=IwAR2a2g0EPrXVAYnqbk8fNMetmDFrfpblll_mgigBN3sriquZ9RXXYEU_OE0

[6]https://dataengineering.wiki/Concepts/Data+Pipeline

[7]https://ml-ops.org/content/three-levels-of-ml-software?fbclid=IwAR19Q47aXiA65MRh4pw8pKCDTCSFMZf0gOGUWfYZEH2LrbtAYRusndb3Z28

https://www.datarobot.com/blog/what-a-machine-learning-pipeline-is-and-why-its-important/

[8]https://towardsdatascience.com/stochastic-gradient-descent-explained-in-real-life-predicting-your-pizzas-cooking-time-b7639d5e6a32

[9]https://en.wikipedia.org/wiki/Sigmoid_function

[10]https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[11]https://en.wikipedia.org/wiki/XOR_gate

[12]Minsky M. L. and Papert S. A. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

[13]https://eclass.aueb.gr/modules/document/file.php/INF153/slides_2021_22/ai_lecture19.pdf?fbclid=IwAR11fSoq2tTKyK4VqINa-qLcBxrp_hct7S-xXnqTCzfUjWbS5ySlJiDpgKI

[14]http://aibook.csd.auth.gr/include/slides/Chap19.pdf

[15]https://developers.google.com/gmail/api

[16]https://en.wikipedia.org/wiki/Statistical_classification

[17]https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[18]https://wiki.pathmind.com/accuracy-precision-recall-f1

[19]https://developers.google.com/identity/protocols/oauth2

[20]https://www.w3schools.com/java/java_threads.asp

[21]https://developers.google.com/gmail/imap/imap-smtp

[22]https://cloud.google.com/gcp?utm_source=google&utm_medium=cpc&utm_campaign=emea-gr-all-en-bkws-all-all-trial-e-gcp-1011340&utm_content=text-ad-none-any-DEV_c-CRE_501834844520-ADGP_Hybrid%20%7C%20BKWS%20-%20EXA%20%7C%20Txt%20~%20GCP%20~%20General%23v1-KWID_43700061559683578-aud-1649240961595%3Akwd-6458750523-userloc_9061578&utm_term=KW_google%20cloud-NET_g-PLAC_&gclid=Cj0KCQjwwJuVBhCAARIsAOPwGAQ-3oaN_HASznNu6OMQkNDCUcGAhZSAodLTFluiIm1UqgmeW-Vy61EaAne8EALw_wcB&gclsrc=aw.ds