NATIONAL TECHNICAL UNIVERSITY of ATHENS
DEPARTMENT of ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMPUTER SCIENCE

An Explainable AI Model for ICU admission prediction of COVID-19 patients

DIPLOMA THESIS

Eleni Dazea

Supervisor: Petros Stefaneas
Assistant Professor N.T.U.A.

Athens, July 2021

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF COMPUTER SCIENCE

An Explainable AI Model for ICU admission prediction of COVID-19 patients

DIPLOMA THESIS

Eleni Dazea

Supervisor: Petros Stefaneas
Assistant Professor N.T.U.A.

Approved by the examining committee on July 9th, 2021

....	....	....
P.Stefaneas	N.Papaspyrou	A.Pagourtzis
Assistant Professor N.T.U.A.	Professor N.T.U.A.	Professor N.T.U.A.

Athens, July 2021

....
Eleni Dazea
Electrical and Computer Engineer

Περίληψη

Η μηχανική μάθηση είναι ένα πεδίο που συναντάται πλέον σε όλους τους τομείς της ανθρώπινης ζωής. Παρόλα όμως τα οφέλη που παρέχει, είναι πολύ δύσκολο τα μοντέλα μηχανικής μάθησης να γίνουν κατανοητά από έναν άνθρωπο. Ειδικότερα σε ιατρικές εφαρμογές και στα αυτοκίνητα χωρίς οδηγό, είναι πολύ σημαντικό να μπορεί ο χρήστης να καταλάβει ποια διαδικασία ακολούθησε ακριβώς το μοντέλο για να καταλήξει σε ένα συμπέρασμα και τη σημασία των χαρακτηριστικών του ίδιου του μοντέλου, έτσι ώστε να αποφεύγονται λάθη και να βελτιώνεται η χρηστικότητά του.

Στην παρούσα διπλωματική, θα ασχοληθούμε με την ανάπτυξη ενός explainable AI μοντέλου, για να προβλέψουμε την εισαγωγή ασθενών COVID-19 σε ΜΕΘ με βάση τα συμπτώματά τους. Βασικό χαρακτηριστικό του μοντέλου μας είναι η δυνατότητα ερμηνείας και αιτιολόγησης της πρόβλεψης με βάση τα συμπτώματα του κάθε ασθενή. Έτσι, ένας γιατρός που το χρησιμοποιεί θα γνωρίζει ποιοι ασθενείς έχουν τη μεγαλύτερη πιθανότητα να εισαχθούν σε ΜΕΘ, για να τους έχει υπό παρακολούθηση, και θα μπορεί ταυτόχρονα να ελέγξει και την εγκυρότητα της πρόβλεψης.

Για την παραγωγή του τελικού προγράμματος, ξεκινήσαμε με την εκπαίδευση πολλών μοντέλων με διαφορετικούς αλγόριθμους σε Python, χρησιμοποιώντας δεδομένα από ασθενείς COVID-19 του νοσοκομείου Sirio Libanes στο Σάο Πάολο της Βραζιλίας. Έπειτα, επιλέξαμε αυτό με την καλύτερη ακρίβεια, στη δική μας περίπτωση ήταν ένα μοντέλο adaboost με random forest weak learner, και το μεταφέραμε στην R, όπου χρησιμοποιήσαμε τη βιβλιοθήκη InTrees για να παράξουμε ένα σύνολο από τους πιο σημαντικούς κανόνες, με βάση τους οποίους γίνεται ο διαχωρισμός των ασθενών σε αυτούς που θα εισαχθούν σε ΜΕΘ και εκείνους που θα αναρρώσουν. Τέλος, με τους παραπάνω κανόνες, δημιουργήσαμε ένα πρόγραμμα σε Prolog, χρησιμοποιώντας τον Γοργία, το οποίο δέχεται ως όρισμα κάποια βασικά αποτελέσματα κλινικών εξετάσεων για κάθε ασθενή και επιστρέφει την πρόβλεψη για την πορεία της υγείας του και τα βασικά συμπτώματα με βάση τα οποία πήρε αυτή την απόφαση. Για την επικοινωνία του χρήστη με τον Γοργία, δημιουργήσαμε ένα πρόγραμμα σε Java.

Λέξεις-Κλειδιά: Επεξηγήσιμη τεχνητή νοημοσύνη, Μηχανική Μάθηση, Λογικός προγραμματισμός χώρις άρνηση ως αποτυχία, Γοργίας, COVID-19

Abstract

Machine Learning is a field that is widely used in all aspects of our lives. However, despite of the benefits of its use, its function and the process that every model follows to produce a result can not be easily comprehended by a human. Especially in medical applications and self driving cars, it is very important for the user to understand the steps the model takes to reach the solution and the importance of the features of the model, in order to avoid mistakes and improve the model's functionality.

In this thesis, we created an explainable AI model that can predict the ICU admission of COVID-19 patients, based on their symptoms and lab results. An important feature of our model is the interpretation and justification of the prediction to the user. This would allow for example a doctor that uses the program to know in advance which patients are more likely to be admitted to the ICU and monitor them and also assess the validity of such prediction.

For the creation of the program, we first trained models with a variety of different algorithms in Python, using COVID-19 patients' data from the Sirio Libanes hospital in Sao Paolo, Brazil. Then, we took the model with the highest accuracy, which in this case was used an adaboost algorithm with a random forest weak learner and transferred it in the R language, where we used the InTrees library to create a sum of the most important rules, which we can use to get a good ICU admission prediction. Finally, with the above rules, we created a Prolog program, using the Gorgias framework, which takes as an input some important patient lab results and returns a prediction on whether the patient will be admitted and the key symptoms based on which the program produced that result. The framework for the user's communication with Gorgias was written in Java.

Keywords: Explainable AI, Machine Learning, Logic Programming without Negation as Failure, Gorgias, COVID-19

Ευχαριστίες

Η παρούσα διπλωματική πραγματοποιήθηκε στη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Αρχικά, θα ήθελα να ευχαριστήσω τον Επιβλέποντα Καθηγητή μου κ. Πέτρο Στεφανέα για την πολύτιμη βοήθεια και καθοδήγηση του καθ' όλη τη διάρκεια εκπόνησης αυτής της διπλωματικής. Επιπλεόν, θα ήθελα να ευχαριστήσω τους καθηγητές κ. Ν. Παπασπύρου και κ. Α. Παγουρτζή για τη διδασκαλία τους και το ενδιαφέρον που μου καλλιέργησαν για την Επιστήμη των Υπολογιστών.

Ευχαριστώ επίσης και όλους τους φίλους που απέκτησα κατά τη διάρκεια των φοιτητικών μου χρόνων που έκαναν την καθημερινότητα μου στη σχολή πιο ευχάριστη. Ιδιαίτερα, ευχαριστώ τον Παναγίωτη για την υπομονή και τη συνεχή στήριξή του.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξη που μου παρείχε σε όλη τη διάρκεια των σπουδών μου.

Ελένη Δαζέα,
Ιούλιος 2021

Contents

List of Figures

Chapter 1

Εκτεταμένη Περίληψη

1.1 Εισαγωγή

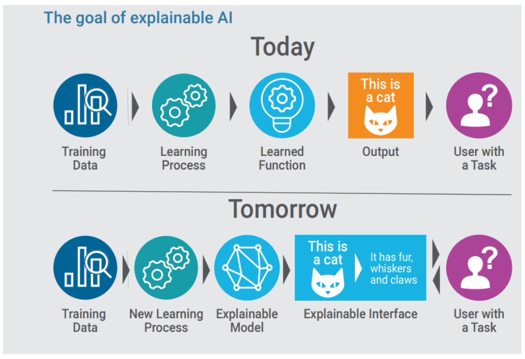
1.1.1 Επεξηγήσιμη τεχνητή νοημοσύνη

Η επεξηγήσιμη τεχνητή νοημοσύνη αποτελείται από ένα σύνολο μεθόδων που εξηγούν την διαδικασία της ανάλυσης δεδομένων και τα αποτελέσματα διαφόρων αλγορίθμων τεχνητής νοημοσύνης. Οι αλγόριθμοι μηχανικής μάθησης μπορούν να χωριστούν σε δύο κατηγορίες: λευκού κουτιού και μαύρου κουτιού, ανάλογα με την ευκολία κατανόησης της λογικής πίσω από κάθε αποτέλεσμα που δίνουν. Η λογική του λευκού κουτιού, μας δίνει διαφάνεια και ερμηνευσιμότητα στις αποφάσεις του μοντέλου για το εκάστοτε πρόβλημα. Αντίθετα, οι αλγόριθμοι μαύρου κουτιού δεν μπορούν να γίνουν κατανοητοί από έναν άνθρωπο. Με βάση όμως το κοινωνικό δικαίωμα στην κατανόηση των αποφάσεων ενός αλγορίθμου από τους χρήστες, όταν αυτός τους επηρεάζει σε κάποια έκφανση της ζωής τους, είναι πολύ σημαντικό να είναι κατανοητοί και οι αλγόριθμοι αυτού του είδους, τόσο λόγω νομοθεσίας, όσο και για να αποφεύγονται λάθη και να δημιουργείται ένα κλίμα εμπιστοσύνης των χρηστών στα μοντέλα τεχνητής νοημοσύνης. Ο ρόλος της επεξηγήσιμης τεχνητής νοημοσύνης είναι ακριβώς αυτός, να μετατρέπει δηλαδή τα μοντέλα μαύρου κουτιού σε μοντέλα κατανοητά για τον χρήστη.

1.1.2 COVID-19

Το 2019 ξέσπασε η πανδημία κορωνοϊού, η οποία ξεκίνησε το Δεκέμβρη του 2019 από την Wuhan της Κίνας. Μέχρι τον Φεβρουάριο του 2021, υπήρχαν 106 εκατομμύρια κρούσματα και 2.34 εκατομμύρια θάνατοι από τον ιό. Η μόλυνση διαρκεί γύρω στις δύο εβδομάδες και η θνησιμότητα είναι κατά μέσο όρο γύρω στο 2.2%. Οι ασθενείς που βρίσκονται σε μεγαλύτερο κίνδυνο είναι όσοι έχουν καρδιαγγειακά προβλήματα, παχυσαρκία, προβλήματα στους πνεύμονες, βρίσκονται σε ανοσοκαταστολή ή είναι άνω των 65 ετών.

Ο COVID-19 έχει καθηλώσει τα συστήματα υγείας πολλών χωρών, καθώς οι σοβαρά



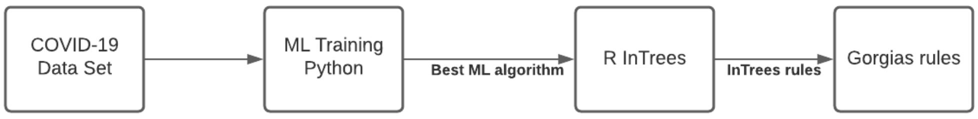
Σχήμα 1.1: Επεξηγήσιμη τεχνητή νοημοσύνη

νοσούντες εισάγονται σε ΜΕΘ για να επιβιώσουν. Αυτό συνέβη και στην Ελλάδα, όπου σε κάποιες περιοχές, οι ασθενείς που χρειάζονταν διασωλήνωση ήταν περισσότεροι από της διαθέσιμες ΜΕΘ.

1.1.3 Σκοπός

Στην παρούσα διπλωματική, εκπαιδεύσαμε πολλά διαφορετικά μοντέλα μηχανικής μάθησης για να προβλέψουν ποιοι ασθενείς θα χρειαστεί να εισαχθούν σε ΜΕΘ, με βάση τα συμπτώματα και τα αποτελέσματα των εξετάσεών τους. Στη συνέχεια, επιλέξαμε το μοντέλο με την καλύτερη ακρίβεια και με βάση αυτό δημιουργήσαμε ένα μοντέλο επεξηγήσιμης τεχνητής νοημοσύνης, το οποίο λαμβάνει ως είσοδο συγκεκριμένα αποτελέσματα εξετάσεων ενός ασθενή και επιστρέφει την πρόβλεψη για το αν ο ασθενής θα εισαχθεί σε ΜΕΘ καθώς και τα συμπτώματα με βάση τα οποία κατέληξε σε αυτή την κατηγοριοποίηση για τον συγκεκριμένο ασθενή.

Σκοπός είναι το σύστημα αυτό να μπορεί να χρησιμοποιηθεί από το ιατρικό προσωπικό στα νοσοκομεία, ώστε να προβλέπουν ποιοι ασθενείς θα νοσήσουν σοβαρά για να έχουν το χρόνο να διαχειριστούν τα κρεβάτια ΜΕΘ ή να στέλνουν τους ασθενείς σε νοσοκομεία άλλων περιοχών που έχουν περισσευούμενες κλίνες. Παράλληλα, θα μπορέσουμε να κατανοήσουμε και με βάση ποια συμπτώματα γίνεται η κατηγοριοποίηση των ασθενών σε σοβαρά νοσούντες και μη, ώστε οι γιατροί να μπορούν να παρακολουθούν αυτά τα συμπτώματα.



Σχήμα 1.2: Διαδικασία δημιουργίας τελικού μοντέλου

1.2 Λογικός Προγραμματισμός χωρίς Άρνηση ως Αποτυχία

Ο λογικός προγραμματισμός χωρίς Άρνηση ως Αποτυχία είναι μία επέκταση του λογικού προγραμματισμού που περιλαμβάνει την άρνηση αλλά όχι την άρνηση ως αποτυχία. Ορίστηκε στη δημοσίευση των Γιάννης Δημόπουλος και Αντώνης Κάκας [8].

Παρακάτω, θα περιγράψουμε συνοπτικά την διαδικασία απόδειξης με βάση αυτό το είδος λογικού προγραμματισμού. Ο προγραμματισμός αυτός περιλαμβάνει δύο ειδών παραγωγές, την Α και την Β. Ξεκινάμε με την Α η οποία προσπαθεί να αποδείξει τον αρχικό στόχο, κατασκευάζοντας ένα κομμάτι θεωρίας. Από την Α προκύπτει μία παραγωγή Β, η οποία προσπαθεί να επιτεθεί στην Α και να αποδείξει την άρνηση μέρους της θεωρίας που χρησιμοποιήθηκε από την Α. Αντίστοιχα, μέσα από την παραγωγή τύπου Β, παράγεται μια άλλη παραγωγή τύπου Α που πάλι προσπαθεί να επιτεθεί στην προηγούμενη. Αν καταφέρουμε να αποδείξουμε τον αρχικό στόχο από μία παραγωγή Α αλλά καμία παραγωγή Β δεν πετύχει τον στόχο της τότε το σύνολο των κανόνων των παραγωγών Α αποτελεί υποσύνολο κανόνων της θεωρίας και ο αρχικός στόχος είναι ασθενές συμπέρασμα αυτής. Αν παράλληλα αποτύχουμε να αποδείξουμε και την άρνησή του με την ίδια διαδικασία, τότε αποτελεί και ισχυρό συμπέρασμα.

1.3 Γοργίας

Ο Γοργίας αποτελεί μία δομή επιχειρηματολογίας, η οποία στηρίζεται στην Prolog και συνδυάζει την λογική της απαγωγής και τον συλλογισμό με προτιμήσεις [5]. Μπορεί να παράξει επιχειρηματολογία σε δυναμικά και εξελισσόμενα περιβάλλοντα με ελλιπείς πληροφορίες. Παράδειγμα τέτοιων εφαρμογών είναι νομικοί συλλογισμοί που βασίζονται σε αντικρουόμενους νόμους μερικοί από τους οποίους είναι υψηλότερης τάξης από άλλους.

Ορισμένα κατηγορήματα του Γοργία είναι:

- Ο Γοργίας χρησιμοποιεί το κατηγόρημα rule/3 για να αναπαραστήσει έναν κανόνα της θεωρίας.
- Το κατηγόρημα prefer/2 χρησιμοποιείται για να δείξει την σχετική δύναμη ανάμεσα σε δύο κανόνες της θεωρίας.
- ’Ενα abducible literal αναπαρίσταται με το abducible/2
- Με το conflict/2 δηλώνουμε ότι δύο κανόνες συγκρούονται

Για να υποβάλλουμε ερωτήματα στο σύστημα του Γοργία χρησιμοποιούμε την εντολή prove(Goal, Delta), με Goal το ερώτημα που θέλουμε να αποδείξουμε και Delta το αποδεκτό επιχείρημα που θα στηρίζει το ερώτημά μας.

1.4 Υλοποίηση

1.4.1 Δεδομένα

Τα δεδομένα για την εκπαίδευση των μοντέλων τα πήραμε από την ιστοσελίδα Kaggle (https://www.kaggle.com), η οποία περιείχε δεδομένα COVID-19 ασθενών από το νοσοκομείο Sirio-Libanes στο Σάο Πάολο της Βραζιλίας [1]. Τα δεδομένα αυτά είχαν κανονικοποιηθεί με έναν Min-Max Scaler, ώστε να παίρνουν τιμές από -1 εώς 1 και περιείχαν τις δημογραφικές πληροφορίες των ασθενών, υποκείμενα νοσήματα, αποτελέσματα αιματολογικών εξετάσεων, καθώς και κλινικές μετρήσεις. Συνολικά, είχαμε 54 δεδομένα για κάθε έναν από τους 385 ασθενείς. Για τις αιματολογικές και τις κλινικές μετρήσεις, τα δεδομένα είχαν υποστεί περαιτέρω επεξεργασία, καθώς αντί για την τιμή της κάθε μέτρησης, είχαμε mean, median, minimum, maximum και relative diff, το οποίο είναι η διαφορά των minimum και maximum διαιρεμένη με το median. Επίσης, για κάθε ασθενή, είχαμε δεδομένα για τα εξής χρονικά παράθυρα παραμονής τους στο νοσοκομείο: 0-2 ώρες, 2-4 ώρες, 4-6 ώρες, 6-12 ώρες και 12 και πάνω, εκτός από την περίπτωση που ο ασθενής εισήχθη σε ΜΕΘ πριν από τις 12 ώρες, όπου έχουμε δεδομένα μέχρι το χρονικό παράθυρο στο οποίο έγινε η εισαγωγή. Με βάση τα παραπάνω, έχουμε 231 στήλες και 1925 γραμμές στον πίνακα δεδομένων.

Για την εκπαίδευση των μοντέλων χρησιμοποιήσαμε 80% των δεδομένων και για την αξιολόγησή τους το υπόλοιπο 20%.

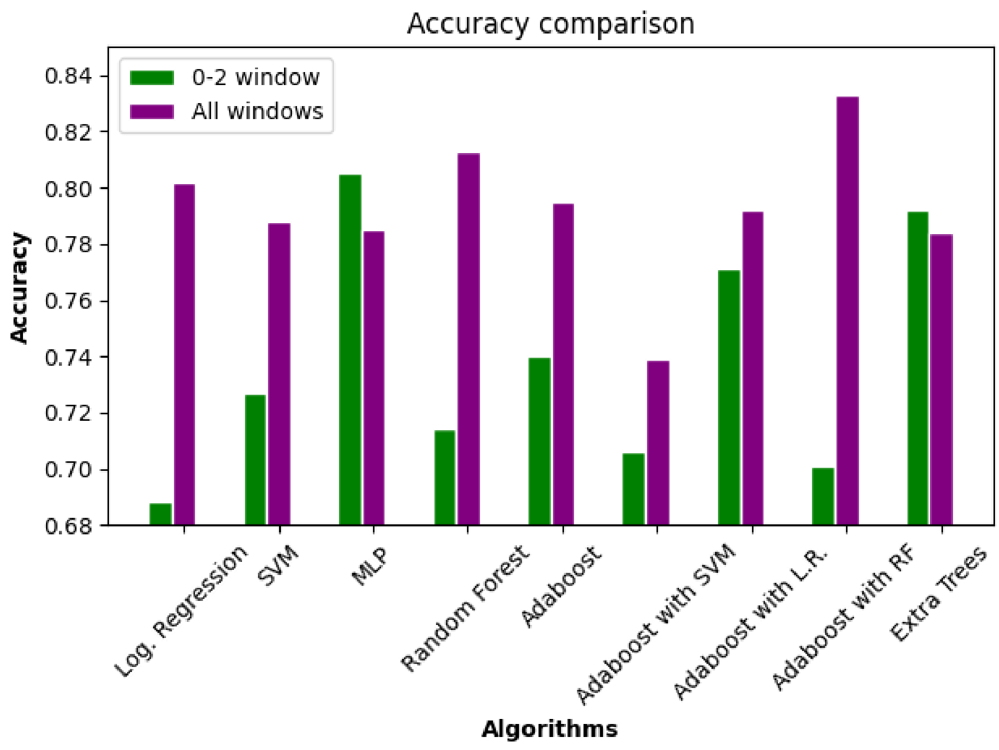
1.4.2 Εκπαίδευση

Αρχικά, δοκιμάσαμε να εκπαιδεύσουμε τα μοντέλα μόνο με δεδομένα από το πρώτο χρονικό παράθυρο και στη συνέχεια επαναλάβαμε την ίδια διαδικασία, χρησιμοποιώντας όλα τα χρονικά παράθυρα.

Οι αλγόριθμοι που χρησιμοποιήσαμε είναι:

- Logistic Regression
- SVM
- MLP
- Random Forest
- Adaboost
- Adaboost with an SVM weak learner
- Adaboost with a Logistic Regression weak learner
- Adaboost with a Random Forest weak learner
- Extra Trees

Από τη σύγκριση της ακρίβειας όλων των μοντέλων, καταλήξαμε στην επιλογή της Adaboost με Random Forest weak learner που εκπαιδεύτηκε στο σύνολο των δεδομένων.



Σχήμα 1.3: Σύγκριση όλων των μοντέλων

1.4.3 Intrees

Στη συνέχεια, εφαρμόσαμε στο παραπάνω μοντέλο τη βιβλιοθήκη Intrees της R, η οποία έχει τις παρακάτω δυνατότητες για τα μοντέλα που χρησιμοποιούν δέντρα:

- Εύρεση συχνότητας κανόνων (Measuring)
- Pruning
- Εύρεση σχετικών και μη επαναλαμβανόμενων κανόνων (Select rules)

Μετά την εφαρμογή των παραπάνω συναρτήσεων, παράχθηκε ένα μοντέλο με μόνο 10 κανόνες, οι οποίοι έχουν την εξής σειρά προτεραιότητας:

1. Αν η μέση συγκέντρωση οξυγόνου στις φλέβες > 0.91 και το ελάχιστο της συγκέντρωσης νατρίου <= 0.32 → Όχι ΜΕΘ
2. Αν μέση συγκέντρωση οξυγόνου στις αρτηρίες <= 0.92 → ΜΕΘ
3. Αν το μέγιστο γαλακτικό οξύ > 0.36 και η μέση συστολική αρτηριακή πίεση > 0.04 → Όχι ΜΕΘ

4. Αν οι μέσοι καρδιακοί παλμοί > 0.14 $\rightarrow$ ΜΕΘ

5. Αν το μέσο του pH στις φλέβες <= 0.38 και το μέγιστο του pH στις φλέβες > 0.35 και η μέση συγκέντρωση οξυγόνου στις φλέβες > 0.25 $\rightarrow$ Όχι ΜΕΘ

6. Αν μέση συγκέντρωση γαλακτικού οξέος <= 0.36 και η μέση συγκέντρωση του ασβεστίου <= 0.34 $\rightarrow$ ΜΕΘ

7. Αν δεν ισχύει τίποτα από τα παραπάνω $\rightarrow$ ΜΕΘ

Το συγκεκριμένο μοντέλο, έχει τα παρακάτω χαρακτηριστικά με βάση τα δεδομένα μας:

$$accuracy = 78.7\%$$
$$precision = 75\%$$
$$recall = 61.2\%$$
$$f1score = 67.4\%$$

1.4.4 Επεξηγήσιμο Μοντέλο

Χρησιμοποιώντας τα παραπάνω, μπορούμε τώρα να κατασκευάσουμε ένα επεξηγήσιμο μοντέλο στον Γοργία.

Αρχικά, ορίζουμε τα δύο πιθανά αποτελέσματα του μοντέλου μας, ο ασθενής να μπει ή να μην μπει σε ΜΕΘ

$$icu(Patient), noticu(Patient)$$

Έπειτα, ορίζουμε ότι τα δύο αυτά αποτελέσματα είναι αντιφατικά

```
complement(icu(Patient),noticu(Patient)).
complement(noticu(Patient),icu(Patient)).
```

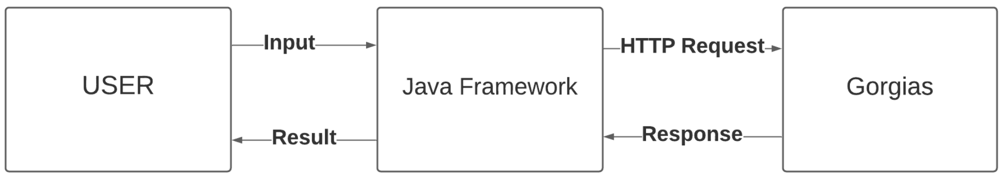
Γράφουμε τους 7 κανόνες

```
rule(r1(Patient),noticu(Patient),[]):- satvenusmean(Patient,Satvenmean), calciummin
    (Patient,Calmin), Satvenmean > 0.91, Calmin =< 0.32 .
...
rule(r7(Patient),icu(Patient),[]):-  satvenusmean(Patient,Satvenmean), Satvenmean <
     10.
```

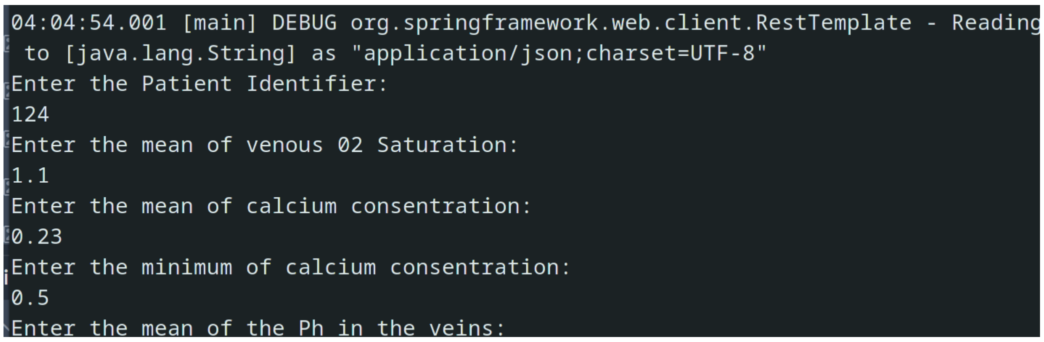
Τέλος, ορίζουμε την προτεραιότητα μεταξύ των κανόνων με το κατηγόρημα prefer

```
rule(pr1(Patient), prefer(r1(Patient), r2(Patient)),[]).
```

Για την επικοινωνία με το πρόγραμμα στον Γοργία, χρησιμοποιήσαμε το API του Gorgias Cloud και δημιουργήσαμε ένα πρόγραμμα σε Java, το οποίο δέχεται από τον χρήστη τα βασικά στοιχεία για τον κάθε ασθενή, στη συνέχεια τα στέλνει στον Γοργία, αυτός με τη σειρά του τα επεξεργάζεται και επιστρέφει την πρόβλεψη και τον κανόνα με βάση τον οποίο κατέληξε σε αυτή την πρόβλεψη. Το πρόγραμμα σε Java τα επεξεργάζεται και τέλος τα παρέχει στον χρήστη αναλυτικά.

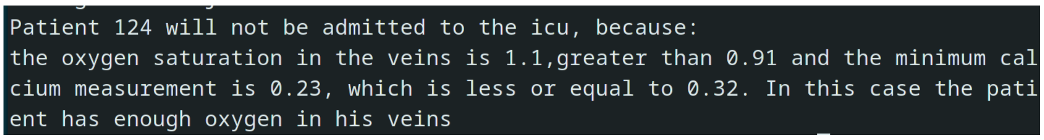


Σχήμα 1.4: Επικοινωνία χρήστη με τον Γοργία μέσω Java framework



```
04:04:54.001 [main] DEBUG org.springframework.web.client.RestTemplate - Reading
 to [java.lang.String] as "application/json;charset=UTF-8"
Enter the Patient Identifier:
124
Enter the mean of venous 02 Saturation:
1.1
Enter the mean of calcium consentration:
0.23
Enter the minimum of calcium consentration:
0.5
Enter the mean of the Ph in the veins:
```

Σχήμα 1.5: Είσοδος επεξηγήσιμου προγράμματος



```
Patient 124 will not be admitted to the icu, because:
the oxygen saturation in the veins is 1.1,greater than 0.91 and the minimum cal
cium measurement is 0.23, which is less or equal to 0.32. In this case the pati
ent has enough oxygen in his veins
```

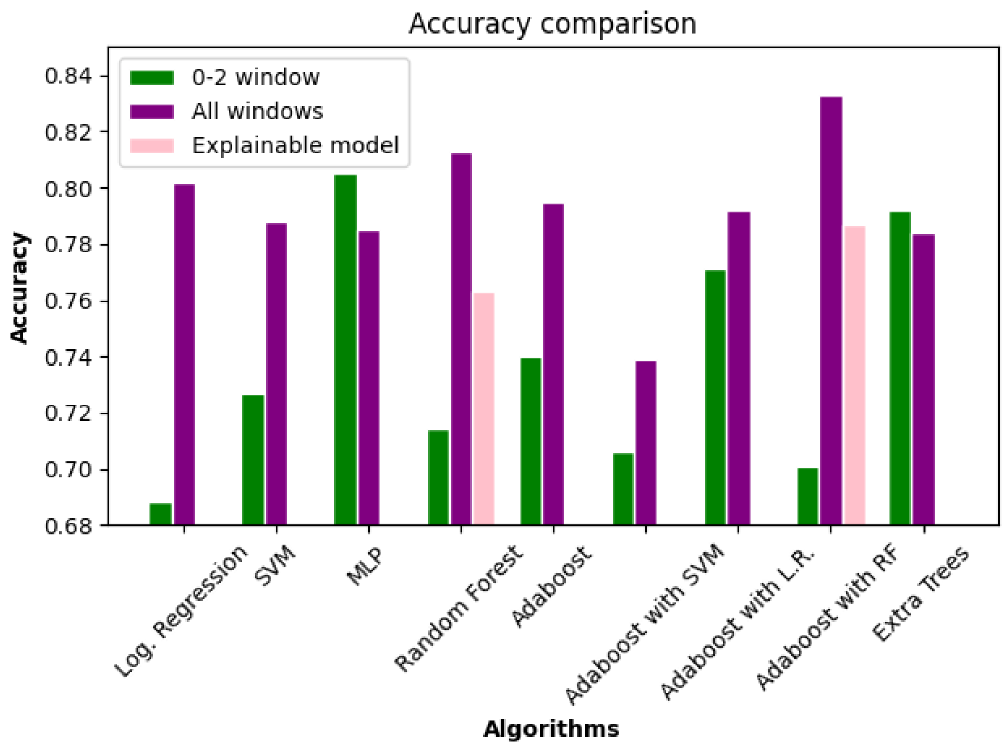
Σχήμα 1.6: Αποτελέσματα επεξηγήσιμου προγράμματος

1.4.5 Αποτελέσματα

Το τελικό αποτέλεσμα είχε ακρίβεια 5% χαμηλότερη από το αρχικό, όμως ήταν συγκρίσιμη με όλα τα υπόλοιπα μοντέλα που εκπαιδεύσαμε. Συγκεκριμένα, είχε παρόμοια αποτελέσματα με τα Extra Trees, Adaboost, MLP και SVM, ενώ είχε και τη δυνατότητα της αναλυτικής επεξήγησης των αποτελεσμάτων που επιστρέφει. Παράλληλα, μας επέστρεψε 7 κανόνες σύμφωνα με τους οποίους γίνεται η πρόβλεψη, δίνοντας έτσι τη δυνατότητα να συλλέξουμε τα βασικά συμπτώματα με βάση τα οποία μπορούμε να προβλέψουμε αν κάποιος ασθενής θα νοσήσει σοβαρά. Τα συμπτώματα αυτά είναι:

- Χαμηλή συγκέντρωση οξυγόνου
- Χαμηλή συγκέντρωση ασβεστίου
- Χαμηλή συγκέντρωση γαλακτικού οξέος
- Χαμηλή πίεση
- Ταχυπαλμία
- Χαμηλό pH στο αίμα

Όσον αφορά τη χαμηλή συγκέντρωση του γαλακτικού οξέος, πιστεύουμε πως είναι στατιστική ανωμαλία λόγω έλλειψης δεδομένων κατά την διαδικασία της εκπαίδευσης των μοντέλων. Όλα τα υπόλοιπα αποτελέσματα έχουν συνδεθεί με σοβαρή ασθένεια από διάφορες μελέτες ή είναι γνωστά συμπτώματα του COVID-19 [14] [11] [9].



Σχήμα 1.7: Σύγκριση όλων των μοντέλων που δημιουργήσαμε

1.5 Συμπεράσματα και Μελλοντικές Επεκτάσεις

Στην παρούσα διπλωματική, δημιουργήσαμε ένα επεξηγήσιμο μοντέλο που προβλέπει την εξέλιξη της ασθένειας ανθρώπων με COVID-19 και καταλήξαμε σε κάποια συμπεράσματα για τα βασικά συμπτώματα με βάση τα οποία μπορούμε να προβλέψουμε σοβαρή νόσο. Τα αποτελέσματα αυτά βέβαια θα πρέπει να αντιμετωπιστούν με προσοχή, κυρίως λόγω των ελάχιστων δεδομένων που είχαμε στη διάθεσή μας για την εκπαίδευση των μοντέλων (385 ασθενείς). Παραθέτουμε κάποιες προτάσεις για την επέκταση αυτής της διπλωματικής:

- Εμπλουτισμός του data set με περισσότερα δεδομένα ασθενών με COVID-19, ώστε να έχουμε μεγαλύτερη ακρίβεια και να αποφύγουμε τυχόν στατιστικές ανωμαλίες.
- Χρήση στρατηγικής μείωσης διαστάσεων (dimensionality reduction strategy) για να μειωθεί ο αριθμός των features των δεδομένων, ώστε να εκπαιδεύσουμε τα μοντέλα μόνο στα σημαντικά δεδομένα. Αυτό μπορεί να αυξήσει την ακρίβεια του μοντέλου αλλά οι στατιστικές ανωμαλίες και οι ελλείψεις του συγκεκριμένου πίνακα δεδομένων θα παραμείνουν.
- Εφαρμογή της ίδιας διαδικασίας για την πρόβλεψη της πορείας άλλων ασθενειών,

όπως π.χ. εμφράγματα.

- Εφαρμογή της ίδιας διαδικασίας με τη χρήση Intrees και Γοργία για τη μετατροπή άλλων μοντέλων από Adaboost, Random Forest, Extra Trees ή οτιδήποτε χρησιμοποιεί Decision Trees σε επεξηγήσιμα.

Chapter 2

Introduction

Nowadays, Machine Learning applications are becoming more and more integrated in all aspects of our everyday lives, from recommendation algorithms for movies to self driving cars. While Machine Learning as a concept has existed since the 1940s, it only started to become widely used in the recent years. This was mostly due to the advancements in Graphics Processing Units (GPUs) and CPUs, which resulted in an increase of computers' processing power. ML algorithms are usually very complex and computationally heavy. Deep Neural Networks (DNNs) for instance, consist of multiple layers and require hundreds of iterations to converge. This makes them very difficult to understand, because, in order to find out why an algorithm gives a specific label to an input, we need to follow the entire process of the algorithm's training and be able to read the layers the input goes through to get a specific result. Distinguishing possible errors in a trained ML model that seems to perform well on the training a test set is not an easy task.
With the introduction of ML in various aspects of life, it also begun to be used in the medical field, mostly with DNNs in medical imaging. However, in fields like medicine and self-driving cars, there is a need for transparency in the decision process. For example, it's not enough to know that a patient might have a stroke, we need to understand which symptoms the ML model considered important for that decision and the process that it followed to reach that conclusion. The reason for this, is that in the above example, making a wrong classification can cause a patient to be overlooked and result in a life threatening situation. The initiative of trying to explain how a ML model works to the user is called Explainable AI (XAI).

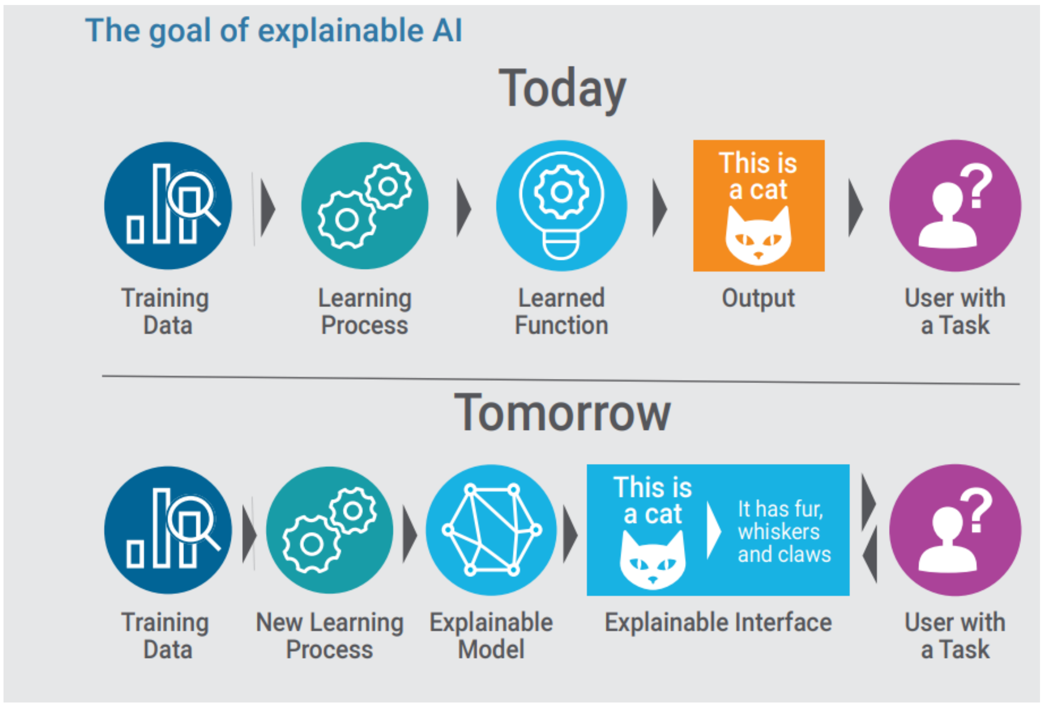


Figure 2.1: Explainable AI

2.1 Covid-19

The coronavirus 2019 pandemic was caused by the severe acute respiratory syndrome coronavirus 2(SARS-CoV-2) and was first identified in December 2019 in Wuhan, China. The World Health Organization(WHO) declared the outbreak a pandemic in March 2020. As of February 2021, there have been around 106 million confirmed cases and 2.34 million deaths from Covid-19.

The estimate of the basic reproduction number (R_0) is around 5.7, which renders it extremely infectious [13]. R_0 is the expected number of infections generated by one person with the virus. The disease lasts typically around two weeks. According to the above numbers, the death to case ratio is around 2.2%, although it varies by region. Those who tend to be at greater risk from serious complications are people with underlying medical conditions, such as serious heart and lung problems, obesity, those who are immunocompromised and elderly people (over 65 years old)[2].

Most infected people will develop mild to moderate symptoms at around 81%, 14% will develop serious symptoms, such as dyspnea and hypoxia and 5% will get critical symptoms (respitory failure, multiorgan dysfunction).Around 95% of people who contract COVID-19 recover. One third of the infected people will show no symptoms of the disease, but they will still be able to pass it on[3] .

The infection-fatality ratio (IFR) differs greatly between age groups. The CDC estimates for each age group as of September 2020 are the following[6]:

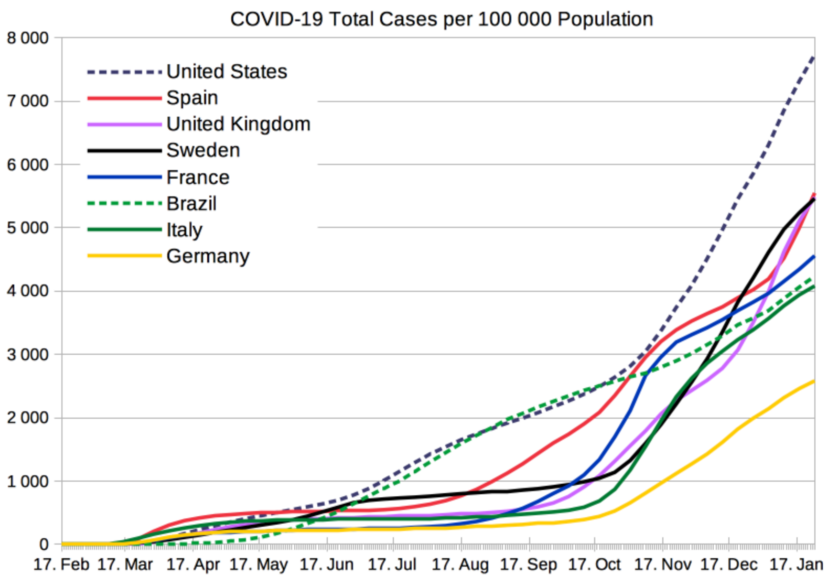


Figure 2.2: Cases per 100,000 population of most affected countries by ECDC

Age group	IFR
0-19	0.002%-0.01%
20-49	0.007%-0.03%
50-69	0.25%-1%
70+	2.8%-9.3%

One of the biggest issues during the pandemic for all countries affected was the capacity of the healthcare system. People who show critical symptoms of the disease, which can be life threatening, need to be treated at the hospital. As a rough estimate, only 5% of the people that contract COVID-19 exhibit critical symptoms and get seriously ill. However, when the total number of cases in a country is too high, the above percentage results in a very large number of patients needing hospitalization, which in turn causes the healthcare system to be overwhelmed. This has occurred in various countries, including Greece, where the number of patients requiring an ICU bed was larger than the number of existing ICU beds. To counter this phenomenon, one goal is to reduce the spread of the disease, in order for countries to prepare and increase the number of ICU beds and ventilators and also have time for the distribution of vaccines.

One thing that could prove to be helpful in cases of countries where the overwhelmed hospitals are on specific regions, is a good prediction of how many patients of a specific hospital will require intubation in the near future. With this tool, if that hospital has no more ICU beds left, they would have time to transfer patients efficiently in nearby regions, where the hospitals may still have some empty ICU beds.

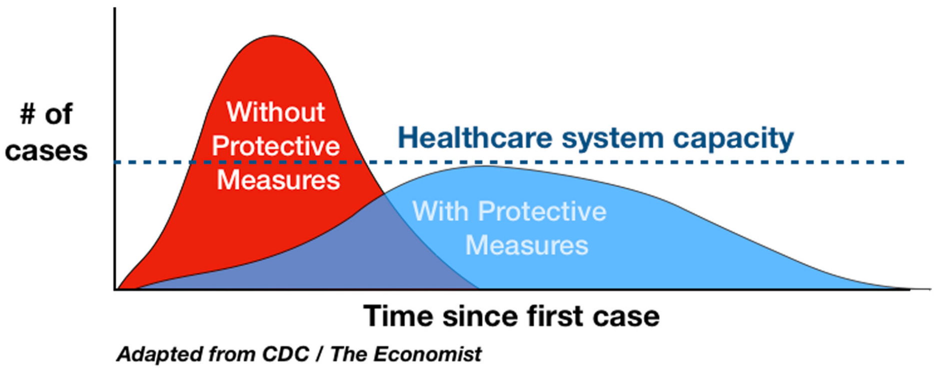


Figure 2.3: Number of cases and healthcare system capacity

2.2 Contribution of this Thesis

In this thesis, we train several Machine Learning algorithms, that can predict which COVID-19 hospital patients will end up in the ICU. We compare said algorithms and choose the one with the highest accuracy. We then take that model and create an Explainable AI one based on it, that can also predict severe disease in COVID patients. The model will be able to explain to a user based on which symptoms it has decided on a particular classification and which signs the medical staff should look for, when they themselves are trying to predict the course of the disease for each patient.

Chapter 3

Background

3.1 Machine Learning

Machine learning is a type of Artificial Intelligence(AI). It also intersects with Computer Science, Statistics and Information Theory. Its aim is to allow software applications to become more accurate at predicting outcomes by learning from existing data, without being specifically programmed to do so. This is usually done by labeling the correct answers as valid, which are then used as training data for the computer to improve its precision.

The main categories of ML are supervised, unsupervised, semi-supervised and reinforcement learning.

1. In supervised learning, the program is given the inputs and the correct labels for these inputs (train data set), so that it can learn to predict the output of the given inputs.

2. In unsupervised learning, the program is given no correct output to the input, so that it can find patterns and similarities in parts of the data and organize those parts into clusters.

3. Semi-supervised learning is between the two above categories. It can use both labeled and unlabeled data, though usually more unlabeled than labeled, even though when taking more labeled data, the accuracy is improved. This technique is normally used when we have a lot of unlabeled data and labeling all of them would require too much time.

4. In reinforcement learning, the program interacts with an environment which changes dynamically, in order to achieve a goal. As it navigates, it makes decisions which are either rewarded or penalized during feedback and the program's goal is to maximize the reward. A good example of this would be a self-driving car.

3.2 Supervised Learning

A supervised learning algorithm analyzes the training data and creates an inferred function in order to map new data. An efficient algorithm will be able to categorize unseen instances, so it needs to be able to generalize from the given training data set.

3.2.1 How it works:

Given a set of N training examples $\{(x_1, y_1), ...(x_N, y_N)\}$ where x_i is the feature vector of the i-th example and y_i the label of said example. The learning algorithm seeks a function $g : X \to Y$, with X the input and Y the output. We usually represent g using $f : X \times Y \to \mathbb{R}$ so that we define g as the function we get when the y value returns the highest score :

$$g(x) = arg_y max\ f(x, y) \tag{3.1}$$

To measure how well g maps output to input, we define a loss function $L : X \times Y \to \mathbb{R}^{\geq 0}$. For (x_i, y_i) the loss of y' is $L(y_i, y')$.
We also define a risk function ($R(g)$), which returns the expected loss of function g :

$$R_{emp}(g) \ = \frac{1}{N} \sum_i L(y_i, g(x_i)). \tag{3.2}$$

What we want is to find the g function that minimizes the risk function $R_{emp}(g)$

An example of what this function might look like is in the picture below:

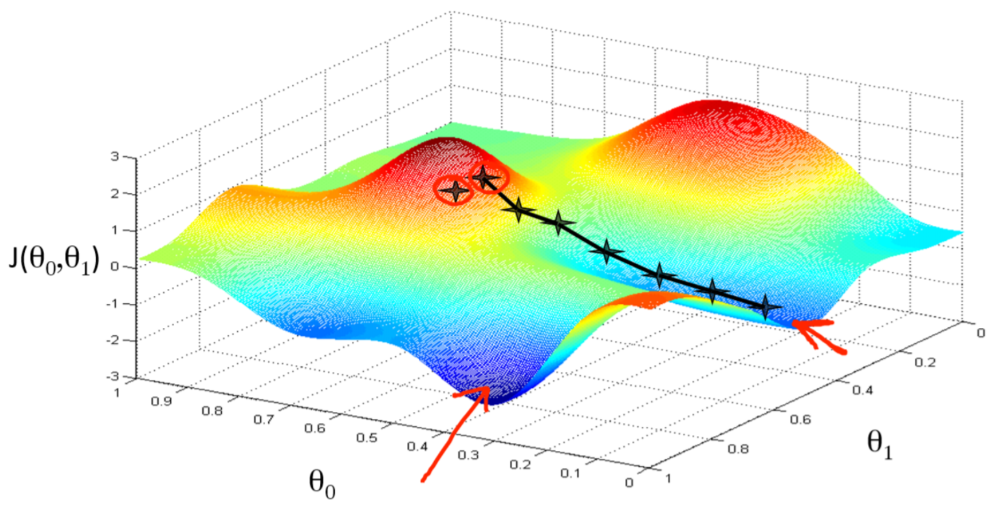


Figure 3.1: R(g) and local minima

When designing a supervised learning algorithm there are four issues to consider:

1. The bias-variance tradeoff. Bias is the accuracy of the predictions. High bias means that the algorithm is inaccurate, it is usually a sign of underfitting (the algorithm is making assumptions without taking into account all the data, or the data is too few). Variance is the sensitivity to small changes in the input. It causes noise and is a sign of overfitting (the algorithm is too complicated, so it has taken into account many features from the data which should not correlated with the result and fails to generalize for new data). We must find a balace between the two, as usually lowering the one increases the other.

2. Function complexity and amount of training data. If a function is simple (the function needs only few features to give a correct result), then a high-bias low-variance program will be enough. However, if the important features are too many and the function is complex, we need many data for the training algorithm, which will be low-bias and high-variance.

3. Dimensionality of input. If the input vectors have too many features, the program will not find the function easily. In this case, we usually remove irrelevant features, using the dimensionality reduction strategies.

4. Noise in output. If a lot of data have an incorrect output, then the algorithm should not try to be very precise on its guesses, as that can lead to overfitting. Early stopping and detecting are two of the approaches most commonly used to counter this issue.

The most common supervised learning algorithms are:
Support Vector Machines(SVNs), linear regression, logistic regression, naive Bayes, linear discriminant analysis, decision trees, k-nearest neighbor algorithm, Neural Networks (multilayer perceptron) and similarity learning.

3.2.2 Types of supervised learning problems

Supervised learning problems can be categorized in two types: regression and classification.

Regression

In regression we are trying to find a mapping function f for input x to output y, where y is a continuous variable. These problems are usually about quantities or sizes. One example of a regression problem would be to try and find the price of houses according to its size, number of rooms and location.

Common regression algorithms are linear regression, Support Vector Regression (SVR) and regression trees.

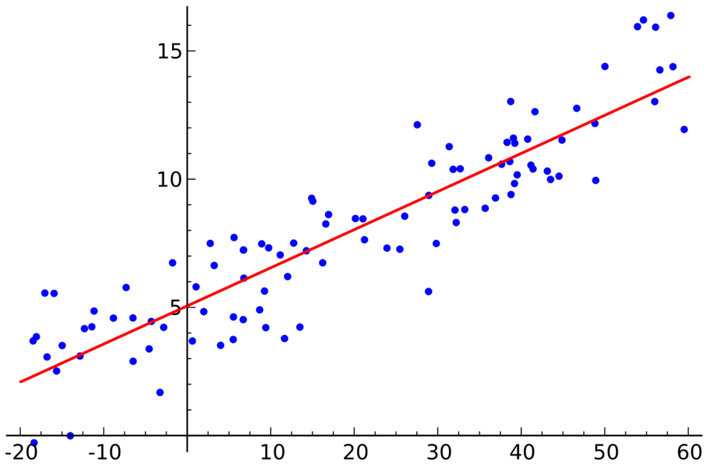


Figure 3.2: A linear regression example function

Classification

In classification algorithms, we are trying to find a mapping function f for input x and output y, where y values are discrete. In this case, we need to sort inputs into categories. An example of such a problem would be to deduct from given pictures of animals whether they are cats or not. Common classification algorithms include logistic regression, naive Bayes, Decision Trees, Neural Networks and K-nearest-neighbors.

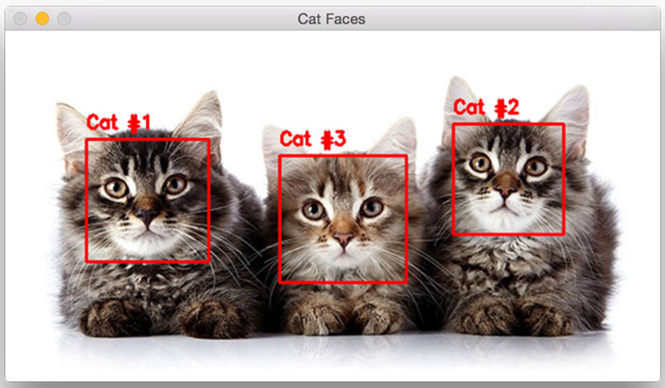


Figure 3.3: Example of image classification

3.3 Supervised Machine Learning Algorithms

In this section, we will analyze the various ML algorithms used in this thesis.

3.3.1 Logistic Regression

Logistic regression is a type of binary regression, in the sense that it has only two values, 0 and 1. It uses a sigmoid function called the logistic function, which takes an input and returns an output between 0 and 1. The standard one is $\sigma : \mathbb{R} \to (0, 1)$:

$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{3.3}$$

where t is a linear function of the input x, such as $t = a_0 + a_1 x$. This function describes the probability that $P(Y) = 1$, with Y a Bernoulli response variable, that this x input will give a 1 output. The algorithm then tries to find the variables a_n, so that the model can have good accuracy.

The cost function is defined as:

$$cost(h_\theta(x), y) = -y\, log(h_\theta(x)) - (1 - y)\, log(1 - h_\theta(x)) \tag{3.4}$$

and the goal is to minimize said function.

We can choose the decision boundary, according to which the algorithm decides in what class the input belongs. It is usually set in 0.5, so that if the model returns a number smaller that 0.5 then it is classified as a 0 and if it is higher than 0.5 it is classified as 1.

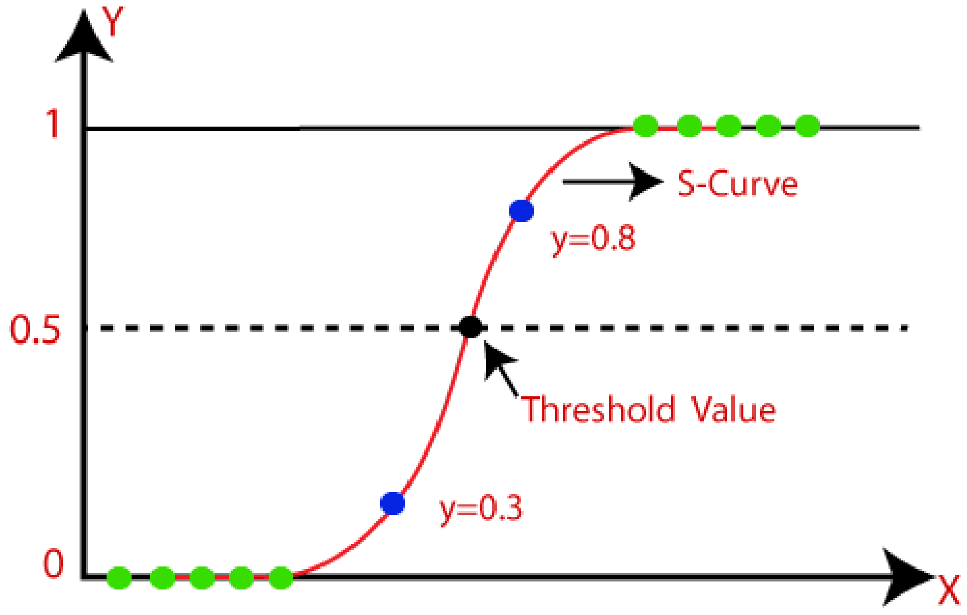


Figure 3.4: Logistic Regression with 0.5 decision boundary

3.3.2 SVM

Support Vector Machines (SVMs) work by assuming an n-dimensional plane, where n is the number of dimensions of the x input, and trying to separate all the inputs with an (n-1) dimensional hyperplane. A hyperplane in an n-dimensional Eulclidean space is a flat n-1 dimensional subset of said space, which divides it into two parts. We choose the hyperplane by maximizing the distance from the nearest datapoint of each side. This is called the maximum-margin hyperplane.

An example for this in a 3D plane with a 2D maximum-margin hyperplane is in the picture below.

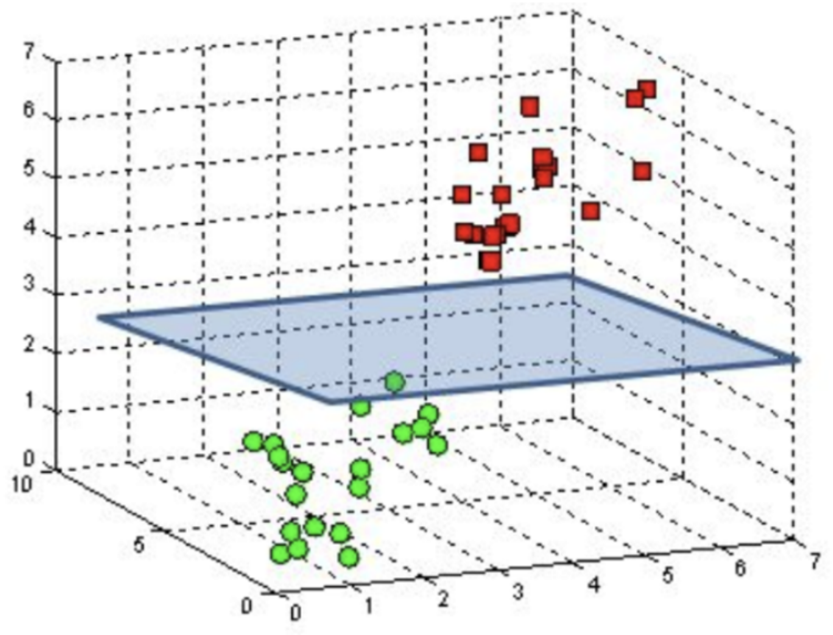


Figure 3.5: SVM in a 3D plane

If we assume training examples: $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ with y_i either 1 or -1 and x_i an m-dimensional vector, then the maximum-margin hyperplane consists of all x points which satisfy:

$$w^T x - b = 0 \tag{3.5}$$

where $||w||$ is the maximum margin.

Using the above, we can then define the cost function which SVM tries to minimize as:

$$[\,\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(w^T x_i - b))\,] + \lambda||w||^2 \tag{3.6}$$

Some parameters we can change in this algorithm are gamma, which defines how much one training example can influence the algorithm, and C, which chooses between a smooth hyperplane and classifying all inputs correctly.

3.3.3 MLP

A multilayer perceptron (MLP) is a type of artificial neural network. Neural Networks (NN) are created to loosely model the neurons of a biological brain and are most commonly used in classification problems. The neurons are arranged in a directed weighted graph, so that each neuron receives an input and then passes its output on to other neurons. Every neuron is a node and is linked to other nodes with weighted links, which determine the links' importance.

Every neuron has multiple inputs and one output, which may then be passed to other neurons multiplied by a weight. This output is produced by the weighted sum of every input to the node:

$$\upsilon_k = \sum_{j=0}^{m} w_{kj} x_j \tag{3.7}$$

The weighted sum is then passed from an activation function $\phi()$, whose purpose is to introduce non-linearity to the output.

$$y_k = \phi(\upsilon_k) \tag{3.8}$$

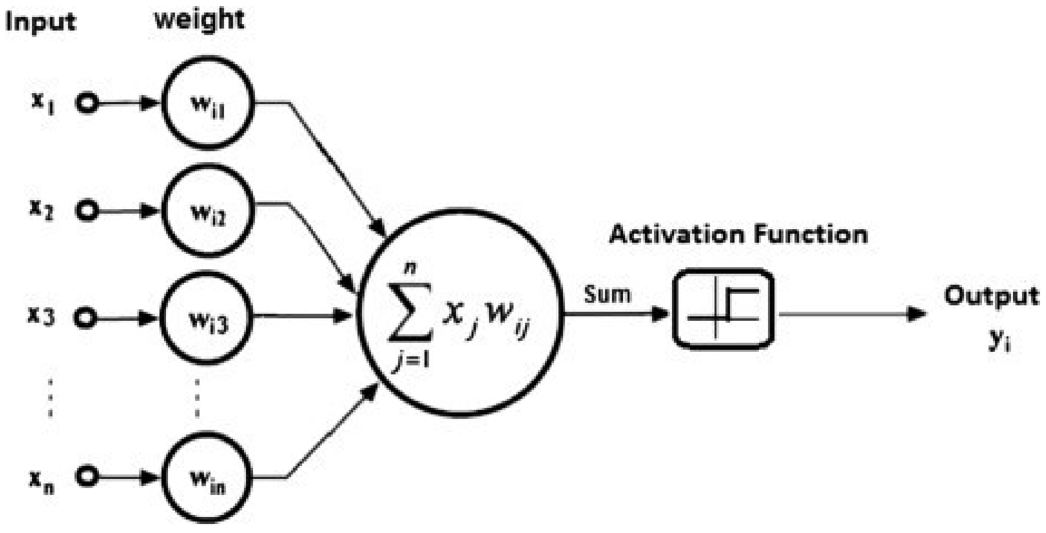


Figure 3.6: Weighted sum and activation function

During the learning process, the weights are adjusted so that the cost function is minimized. If the output is different from the label, then the error is transferred to the previous nodes and the weights are adjusted depending on their contribution to the error. As the cost function result keeps declining, the process continues. The method we use to reevaluate the weights is called back propagation and it calculates the gradient of the cost function and updates using gradient descent.

Multilayer perceptrons are one of the most common neural network arrangements. They consist of at least three layers of neurons, in which the nodes of every layer are all connected to every node of the previous and the next layer. For that reason, they are called fully connected. The first layer is the input layer, the last is the output layer and every other one in the middle is a hidden layer. The hidden layers are the ones responsible for providing the correct output after training.

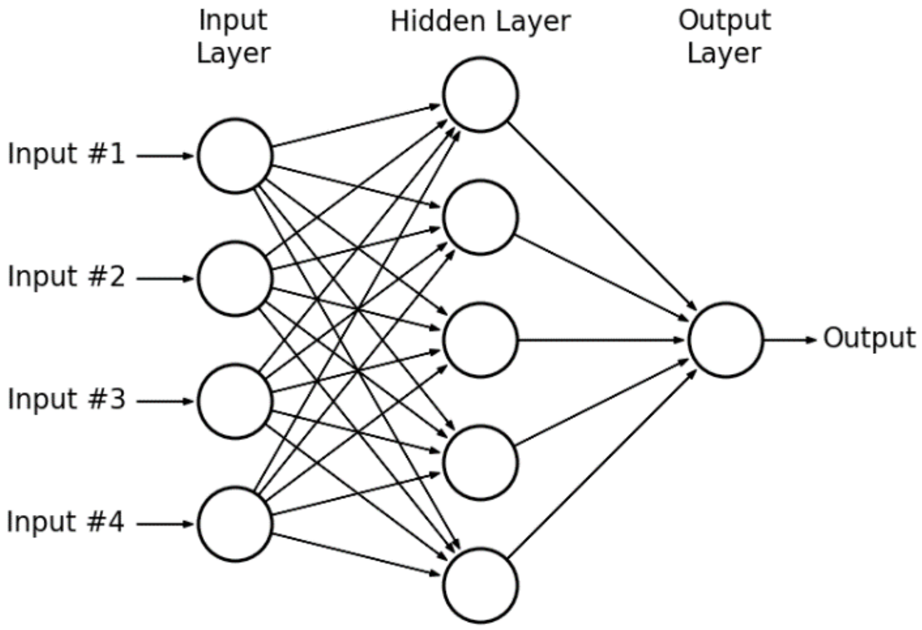


Figure 3.7: A multilayer perceptron with one hidden layer

3.3.4 Decision Trees

Decision Trees are used in supervised machine learning for both regression and classification problems. It starts with the root and splits into branches recursively. The branch ends when we have reached a decision and we can't split any more. The decisions are the leaves. Classification trees end in one of the options for the y value and regression trees end in continuous values.

The procedure is the following. At the root, all features are considered and we calculate how much accuracy a split on each feature will cost. We then choose the one that costs the least. This continues recursively. When the branch reaches a conclusion the process ends. The maximum depth of the tree is the length of its longest branch. We usually want to contain the maximum depth to avoid overfitting.

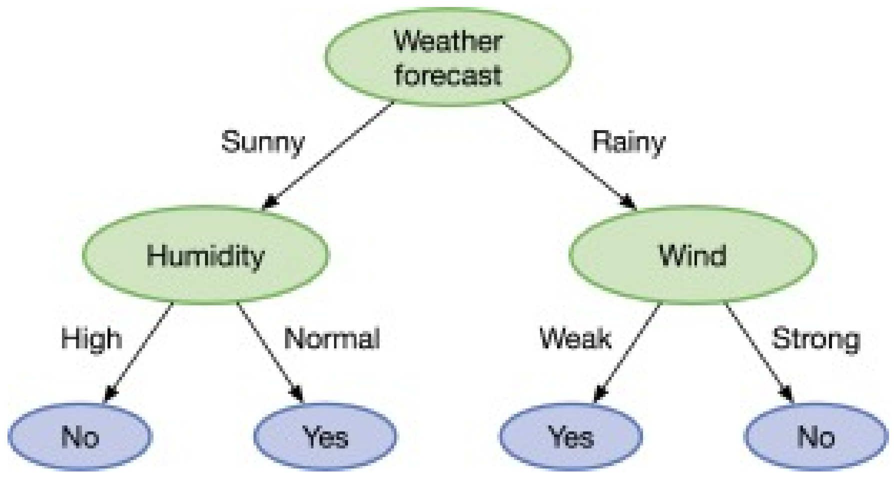


Figure 3.8: Decision Tree example

Random Forest

The random forest algorithm consists of a large number of individual decision trees. Each decision tree gives a prediction for an input, and the decision with the most votes of the trees in the forest is the model's prediction. In order for random forests to work well, the individual trees need to not be correlated with each other. This is achieved through bagging.

Bagging works by selecting random features of the input for the split of each tree, so that each decision tree works with different data than the rest.

Also, unlike decision trees, the trees of the forest split their nodes by picking from a random set of features without trying to find the one which provides the lowest cost. This is done to force even more variation between the trees of the forest.

Usually in classification problems, when we have p features, each tree uses about $\sqrt{p}$ in each split.

The main drawback of random forests in contrast to decision trees is that, even though they return better results and solve the overfitting issue, they sacrifice the interpretablity of decision trees, as the latter are one of the easiest to understand compared to most supervised ML models.

Extra Trees

Unlike random forest, extra trees fits the entire training dataset to every decision tree. It randomly selects the feature for the split point and also randomly chooses the point. The hyperparameters are the number of decision trees, the number of features for the random split selection and the minimum number of samples for the creation of a split point.

Because of the random selections, extra trees algorithms have higher variance, which can be reduced by increasing the number of trees in the forest.

Adaboost

Adaboost is a meta-algorithm. It can be used on its own or with another ML algorithm as a 'weak learner'. For that, it more often uses decision trees.

In order to explain how Adaboost works, we will use a special type of decision trees, called decision stumps, as a weak learner. Stumps are trees with only one node and two leaves.

How it works

First, we create another array next to the features which indicates how important it is for each one to be classified. In the beginning the all have equal weights, $\frac{1}{N}$ where N is the number of features. Then, we create a stump for each feature and check how accurately it classifies all the input data, based on the weighted features. After this step, we assign more weight to the stumps that correctly classified more data and also more weight to the features that were not classified correctly, so that this doesn't happen in the next iteration. When all data are correctly classified or we have reached the maximum iteration limit, this process ends.

Let's assume training data $\{(x_i, y_i)\}_{i=1}^{N}$ with $x_i \in \mathbb{R}^{\mathrm{M}}$ and $y_i \in \{-1, 1\}$. We then define the loss function as:

$$I(f_m(x), y) = \begin{cases} 0 & f_m(x_i) = y_i \\ 1 & f_m(x_i) \neq y_i \end{cases} \tag{3.9}$$

The adaboost pseudocode [4] is the following:

Algorithm 1 Adaboost psedocode

```
1: for $i = 1$ to $N$ do
2:     $w_i^{(1)} = 1$
3: end for
4: for $m = 1$ to $M$ do
5:     $\epsilon_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(f_m(x_i)_i)}{\sum_i w_i^{(m)}}$        ▷ Fit weak classifier m to minimize function
6:     $a_m = ln\frac{1 - \epsilon_m}{\epsilon_m}$        ▷ Update weight for m-th classifier
7:     for $i = 1$ to $N$ do
8:         $w_i^{(m+1)} = w_i^{(m)} e^{a_m I(f_m(x_i) \neq y_i)}$        ▷ Update feature weights
9:     end for
10: end for
```

The final classifier is the linear combination of all weak classifiers:

$$g(x) = sign(\sum_{m=1}^{M} a_m f_m(x)) \tag{3.10}$$

Like random forest, adaboost is also not prone to overfitting, it is however sensitive to noise in the data.

Decision Tree Pruning

Pruning is a data compression technique used in machine learning, which reduces complexity and removes redundant parts of a classifier. The goal is to reduce the size of the classifier without reducing the accuracy of the model. There is a top-down and a bottom-up approach.
Bottom-up: We start from the leaves and work recursively to the top as we determine the relevance of every node. If they are not relevant, they are dropped or are replaced by a leaf.
Top-down: We now start at the root of the tree and work recursively to the leaves. In every node, we calculate whether it is relevant for all n items. If we prune a middle node, we might drop an entire sub-tree despite it being relevant or not.
Some of the most commonly used pruning algorithms are reduced error pruning, cost complexity pruning and statistic-based pruning.

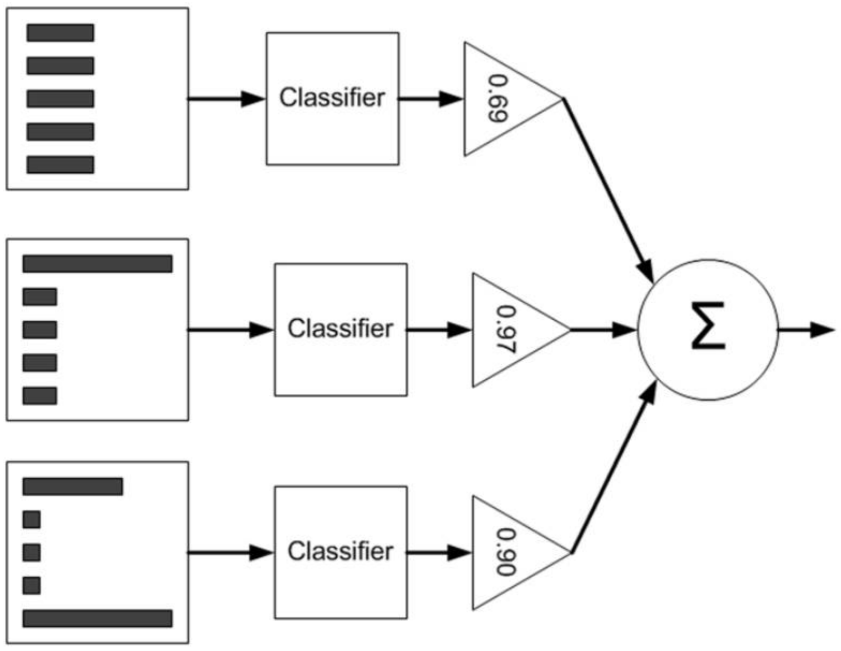


Figure 3.9: Adaboost example

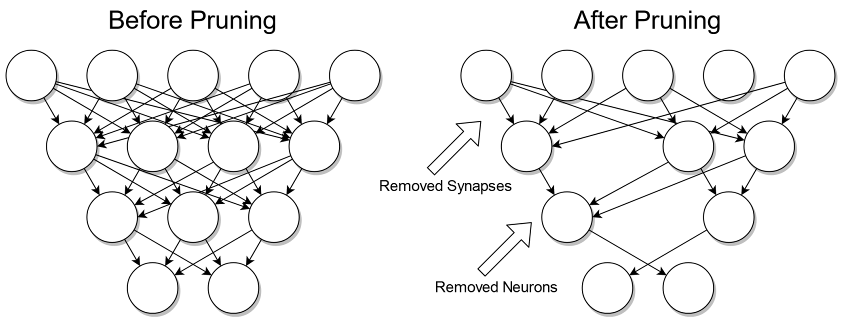


Figure 3.10: Pruning example

3.4 Precision, Recall and F1 score

The way we usually judge if a binary classification machine learning algorithm is good at predicting the correct labels, is by using the accuracy metric, which is the number of correct guesses divided by the input size. It is however a flawed metric when the dataset is imbalanced. For example, if we consider a dataset where 95% of the data have a negative label and the rest a positive one and we create a program that always predicts negative, we will get a 95% accuracy. To mitigate this issue, we also use two other metrics, precision and recall.

Precision describes how many of the positive results that we got from the algorithm are relevant and recall how many of the relevant cases we managed to pick. In probabilities, precision is the probability of randomly selecting a positive test case from all the cases that were categorized as positive and recall is the probability of randomly selecting a case that was classified as positive from all positive cases. If we divide the results into four categories: true positive (tp, the positive results that where guessed as positive by the algorithm), true negative (tn, the negative results that were guessed negative), false positive (fp, the negative results that were guessed positive) and false negative (fn, the positive results that were guessed negative), then precision and recall are defined as:

$$precision = \frac{tp}{tp + fp} \tag{3.11}$$

$$recall = \frac{tp}{tp + fn} \tag{3.12}$$

There is also a measure which combines them and is the harmonic mean of the two, called F1 score:

$$F1\ score = 2\,\frac{precision * recall}{precision + recall} \tag{3.13}$$

Like with accuracy, the highest possible value for the above metrics is 1.0 and the lowest 0.0.

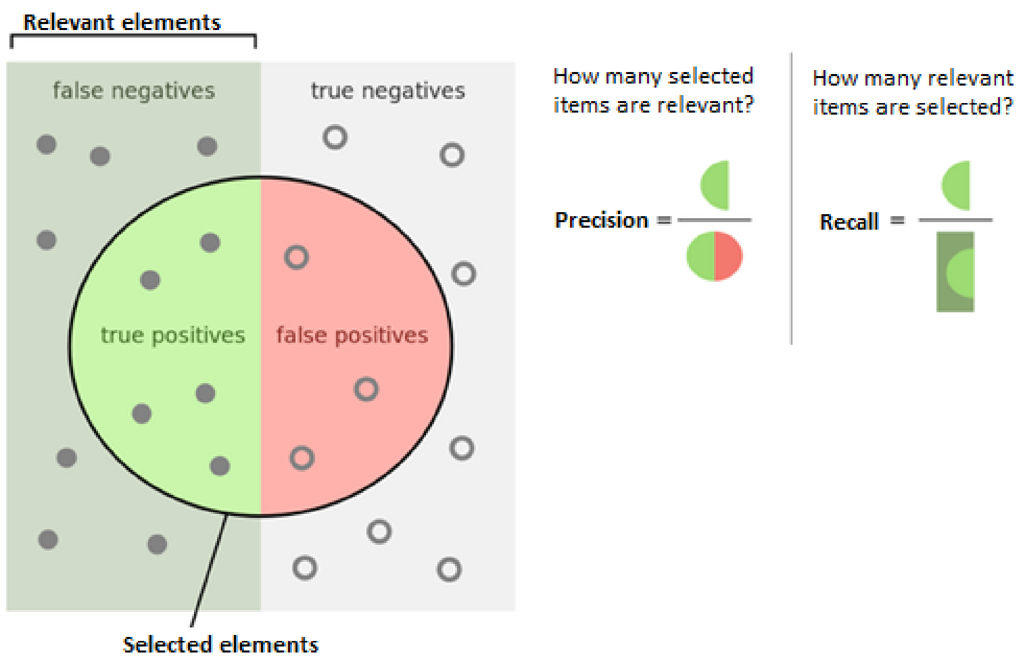


Figure 3.11: Precision and Recall

3.5 Explainable AI

Explainable AI consists of methods that try to explain the process and results of Artificial Intelligence algorithms and applications. Machine learning algorithms can be either black-box or white-box. White-box algorithms are already easy to interpret, while black-box are hard to explain and difficult to understand even by their own developers. A good example for black-box models are deep neural networks, random forest algorithms etc.

XAI models are algorithms that try to turn all black-box algorithms into white-box, and as such they follow the principles of transparency, interpretability and explainability.

- We have transparency when the designer of the algorithm can describe the model parameters and labels of the program.
- Interpretability is when the process of solving the problem and the reasoning behind the output can be understood by a human.
- Explainability is a sum of interpretable features that can be combined to produce a decision based on an example.

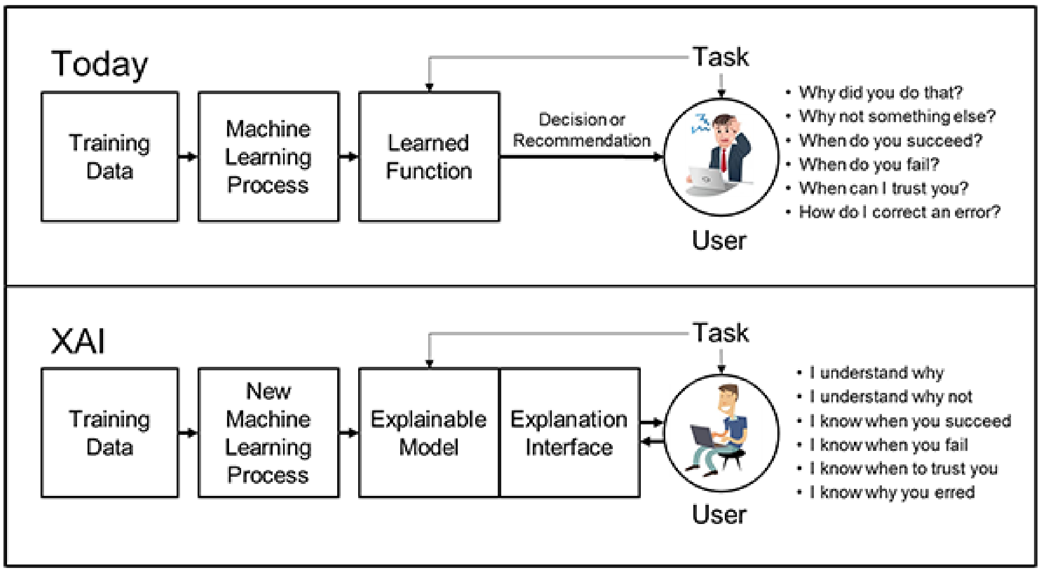


Figure 3.12: XAI importance

The main reasons behind XAI development are the social right to explanation, which is the right of an individual to understand and be given an explanation for the output of an algorithm that they use or affects them individually, legally or financially, and also to improve user experience and provide trust to the users for the validity of the results.

An even more important reason for the use of XAI is to predict how well a model will generalize. Because ML's operation is very complicated, models might start learning based on features and patterns which are accidental or useless to us. A good example of this is bias. If for example we train a ML model for loan approvals based on previous data of people doing the same job, the algorithm will incorporate the bias of humans in its programming. If we can then interpret the model, we will notice the bias and fix the data set.

There are specific industries where XAI is becoming more important. In medical applications, doctors that use a model need to understand how a conclusion was reached and in some cases come to a different one, because there are cases when human interpretation is needed. Another example is autonomous vehicles. The programmers need to be able to understand why the model makes specific predictions, in order to find mistakes in the program's reasoning.

The European Union has introduced a right to explanation in GDPR for the above issues, even though it covers only a small aspect of interpretability.

3.6 Related Work

In the paper published in the 19th International Conference on Bioinformatics and Bioengineering ”*Intergrating Machine Learning with symbolic reasoning to build an explainable AI model for stroke prediction*”[12], Gorgias and an R library called Intrees were deployed in order to build an explainable AI model which predicts if a patient will have a stroke and also explains why according to their medical history.

This is done by taking a random forest algorithm in R, modifing the forest using the package inTrees, in order to keep only the most important branches of the forest, and then using those branches to build Prolog rules in Gorgias that return the same result but also have the ability to explain the proof process.

In said paper, they managed to take a model with a 78% accuracy and turn it into an explainable AI model with 77% accuracy.

Chapter 4

Logic Programming without Negation as Failure

Logic Programming without Negation as Failure,(LPwNF) is a way to extend logic programming to use explicit negation but not contain negation as failure (NAF). The latter is embodied later, in the argumentation based semantics. The following definitions and proofs are in Logic Programming without Negation as Failure - Yannis Dimopoulos and Antonis Kakas [8] .

4.1 The framework

In LPwNF, logic programs are non-monotonic theories, in which every program is considered a collection of default sentences, from which we must choose an appropriate subset, called extension, to reason with. Those sentences are written in the usual logic programming language with the exception of the use of the explicit negation instead of negation as failure. The following program is an example of this particular language:

$$\begin{aligned} fly(x) &\leftarrow bird(x) \\ \neg fly(x) &\leftarrow penguin(x) \\ bird(x) &\leftarrow penguin(x) \\ bird(Tweety)& \end{aligned}$$

with a priority relationship between the rules according to which the second rule is higher than the first. Now we can derive that Tweety can fly, because we can derive $fly(Tweety)$, but not $\neg fly(Tweety)$.

If we add $penguin(Tweety)$, then we will derive that $\neg fly(Tweety)$, as both fly and $\neg fly$ can be derived, but as we said before the second rule is higher than the first and overrides it.

Below are the basic definitions for LPwNF:

Definition 4.1.1 (Logic Program or Non-Monotonic Theory) *A program (K,<) is a set of rules K along with a priority relation < on those rules K.*

Definition 4.1.2 (Attacks) *Let's assume program (K,<) and T,T'$\subseteq$ K.Then, T' attacks T iff there exist L, $T_1 \subseteq$T' and $T_2 \subseteq$T such that:*

(i) *$T_1 \vdash_{min}$ L and $T_2 \vdash_{min} \neg$ L.*

(ii) *($\exists r' \in T_1$, r $\in T_2$ s.t. r' < r) $\Rightarrow$ ($\exists r' \in T_1$, r $\in T_2$ s.t. r < r').*

where T $\vdash_{min}$ L means that L can be derived from T but no proper subset of T.

Definition 4.1.3 (Consistency) *Let K a set of rules. K is consistent iff for any ground literal k s.t k $\vdash$ K,then k $\nvdash \neg K$.*

Definition 4.1.4 (Admissibility) *Let ($\mathcal{K}$,<) a program and K a closed subset of $\mathcal{K}$. Then K is admissible iff:*

(i) *K is consistent*

(ii) *for any K' $\subseteq \mathcal{K}$ if K' attacks K then K' attacks K'.*

Definition 4.1.5 (Non-monotonic credulous consequence) *Let (K,<) be a program and L a ground literal. Then L is a non-monotonic credulous consequence of the program iff L holds in a maximal admissible set of K.*

Definition 4.1.6 (Non-monotonic sceptical consequence) *Let (K,<) be a program and L a ground literal. Then L is a non-monotonic sceptical consequence of the program iff L holds in every maximal admissible set of K.*

Now let's consider a program P in NAF:

$$p \leftarrow\ q, not\ r.$$

Not r is interpreted as unless r, so this rule will be transformed to:

$$p \leftarrow q$$
$$\neg p \leftarrow r$$

with the second rule higher than the first. This is like the above tweety example, so if both q and r are true, then, since the second rule is higher, $\neg p$ is true.

4.2 Proving LPwNF

After presenting the basic definitions we can describe the proof procedure of the LP-wNF framework. It consists of two kinds of derivations, called *Type A* and *Type B*, which are then used to construct an admissible set for any given goal.

Type A derivations generate part of the needed theory that can derive an initial goal, whilst type B derivations provide sentences to counterattack (defend) the attacks against the theory. When a type B derivation identifies a new attack, a new type A derivation is created to find a counterattack to it.

Type A derivations are SLD-resolutions that collect rules. Type B derivations are rules *r* with body *k*. These rules have been used before in a type A derivation. In order for a type B derivation to begin, there needs to exist an *r'* rule with a ¬ *k* body, which is higher than rule *r*. This type B derivation then tries to prove this rule and, by doing that, attack the type A derivation from which it started.

Subsequently, a type B derivation can create type A derivations, which will then try to defend the rules from derivation B attacks. Those begin with a rule *s* with head *l*, that has been previously used in a type B derivation to prove a rule *s'* with head ¬ *l*. As in type B derivations, the *s* rule must be higher than *s'*.

If the first type A derivation manages to prove the initial goal, while no type B derivation has achieved to prove a goal, then the initial goal has been proven and it is a non-monotonic credulous consequence of the theory. The rules used by the type A derivation constitute a maximal admissible set of the theory.

In order for the goal to be a non-monotonic sceptical consequence, it should be both a non-monotonic credulous consequence, meaning that the above process should be completed, and at the same time the calculation of its negation must fail.

We must note that rules which have been proven in previous stages of the process don't need to be proved again. For example, if in a type A derivation we must prove a rule *r* that has been proven in a previous type A derivation, we can accept that it applies and it doesn't have to be proved again.

The following calculation is correct, it manages to return the acceptable subset of the theory every time and the initial goal is a non-monotonic sceptical consequence of the theory.

4.2.1 Example of proof sequence

Let's consider the following logic program:

$r_1 : fly(x) \leftarrow bird(x).$

$r_2 : \neg fly(x) \leftarrow penguin(x).$

$r_3 : penguin(x) \leftarrow walkslikepeng(x).$

$r_4 : \neg penguin(x) \leftarrow \neg flatfeet(x).$

$r_5 : bird(x) \leftarrow penguin(x).$

$r_6 : bird(T).$

$r_7 : walkslikepeng(T).$

$r_8 : \neg flatfeet(T)$.
with priorities: r$_2 > r_1, r_4 > r_3$.
If we want to prove that $fly(T)$ the proof is below in the next figure. The square is a type B derivation, while the other two are type A derivations.

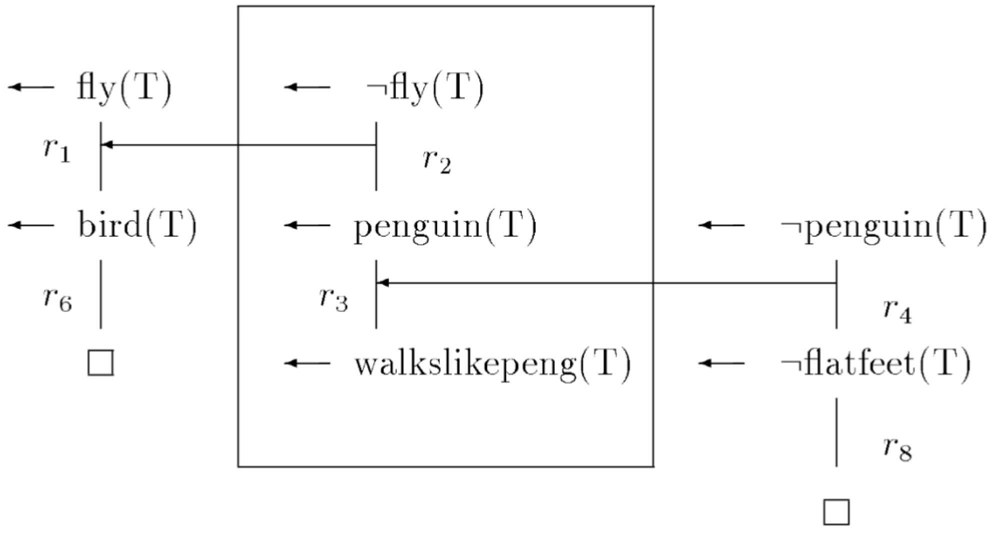


Figure 4.1: Proof for fly(T), long arrows symbolize attacks

According to the above proof process, we start with the type A derivation trying to prove $fly(T)$, which is r_1. However, there is also r_2, which attacks r_1, as it has the negation of r_1's head as a head and is also higher. Then, a type B derivation starts, trying to prove that $\neg fly(T)$. For this we use rules r_2 and r_3, but because r_4 exists, a type A derivation starts to defend from above attack. This type A derivation manages to prove $\neg penguin(T)$, finds a defence to the type B derivation's attack and thereby proves the initial goal of $fly(T)$.

According to the above, $fly(T)$ is a non-monotonic credulous consequence of the initial theory and the admissible set is: $\{r_1, r_4, r_6, r_8\}$.

If we now consider a modifies example where $r_4 > r_3$ didn't exist. Now, if we replace r_4 with $r_4' : \neg penguin(x) \leftarrow \neg singslikepeng(x)$, then r_3 and r_4' don't need to be ordered. If we also replace $r_8 : singslikepeng(T)$, the proof is still valid, since, when we generate a type A derivation from a B one, the top rule of A does't have to be higher than that of B, only not lower. With this change, we can also prove that $\neg fly(T)$ exists too.

Chapter 5

Gorgias

Gorgias is a general argumentation framework that uses the ideas of preference reasoning and abduction while maintaining the benefits of both. It can form the basis for reasoning about adaptable preference policies in the face of incomplete information from dynamic and evolving environments. For instance, in house acquisition, one may prefer certain features over others; in scheduling, meeting some deadlines may be more important than meeting others; in legal reasoning, laws are subject to higher principles, like lex superior or lex posterior, which are themselves subject to ”higher order” principles. All information used are from Gorgias' tutorial [5].

5.1 Semantics

Gorgias is a Prolog framework. Its predicates can be divided in three categories:

1. abducibles
2. defeasible
3. backround

Gorgias uses prolog symbols followed by predicate symbols to denote rules, conflicts and preference between rules. The usual syntax is:

```
rule(Label, Head, Body)
```

Head is a literal, *Body* is a list of literals and *Label* is a compound term composed of a rule name and selected variables from the Head and Body. We then use *prefer* to describe priority relations: prefer(Label1, Label2). In this predicate Label1 is preferred to Label2 in the case both literals hold. This way, we can encode the relative strength of rules between contradictory rules. An abducible literal L is specified with the predicate *abducible/2*, i.e.:

```
abducible(regular_customer(_), []).
```

Finally, the statement *conflict(Label1,Label2).* indicates that the two labels are conflicting. In most cases it will be true iff Label1 and Label2 are contrary literals.

5.2 Use of Gorgias

Gorgias uses SWI-Prolog. At the beginning of the file we always put the following two lines of code:

```
:- compile('../lib/gorgias.pl').
:- compile('../ext/lpwnf.pl').
```

The first line loads the system, the second one is a collection of rules that define a qualification relation between arguments, which is used by the attacking relation to encode the relative strength of the arguments.

5.3 Knowledge Depiction with Gorgias

We use prolog rules and Gorgias' predicates as mentioned before for an example in which a bird can fly but a penguin cannot. This would be depicted as:

```
rule(r1(X),fly(X),[bird(X)]).
rule(r2(X), neg(fly(X)),[penguin(X)]).
```

The above two rules denote that if something flies, it's a bird. Then we say that Tweety is both a penguin and a bird:

```
rule(f1, bird(tweety), []).
rule(f2, penguin(tweety), []).
```

Then we need to solve the conflict:

```
rule(pr1(X), prefer(r2(X), r1(X)), []).
```

The above statement means that if X is both a bird and a penguin it doesn't fly, because r2 is preferred to r1.

5.4 Answering Queries

In general, in order to construct a solution for a specific query, we begin from an initial argument and we add to it a suitable defence. We will concentrate on the admissible arguments. An argument is admissible if it can stand to any possible attack. Obviously, an argument that attacks itself is not admissible, as you cannot attack an empty set.

The process of computing whether an argument is admissible can be done in two phases. In the first phase, a goal is reduced to a closed set that proves it. Then, the initial argument is expanded with suitable defences from any attack in that set. This expanse then may result in new conflicts and therefore the systems repeats this process until there are no more conflicts (meaning that the argument is admissible) or no more defences (and therefore we couldn't find an admissible set of arguments).

Queries are submitted in the following format:

```
prove(Goals, Delta).
```

where *Goals* is a list of literals and *Delta* is an admissible argument for the given query.

Subsequently, in the above example, if we give Gorgias the following query:

```
prove([neg(fly(tweety))],Delta).
```

we will get the result:

```
Delta = [f2, r2(tweety)].
```

which proves that, if Tweety can't fly, then it is a penguin. However, the following query:

```
prove([fly(tweety)],Delta)
```

has no solution, because r2 attacks r1 and r2 is preferred. However if we comment out the preference rule, Gorgias will generate two answers, one for r1 and one for r2.

5.5 Dynamic Preferences

5.5.1 Inheritance with exceptions

In the Tweety example, the preference rule was static (The Body of the rule was an empty List). Now we will get into dynamic preferences, the ones that have exceptions.

Now, let's say that we have an inheritance hierarchy based on the figure bellow:

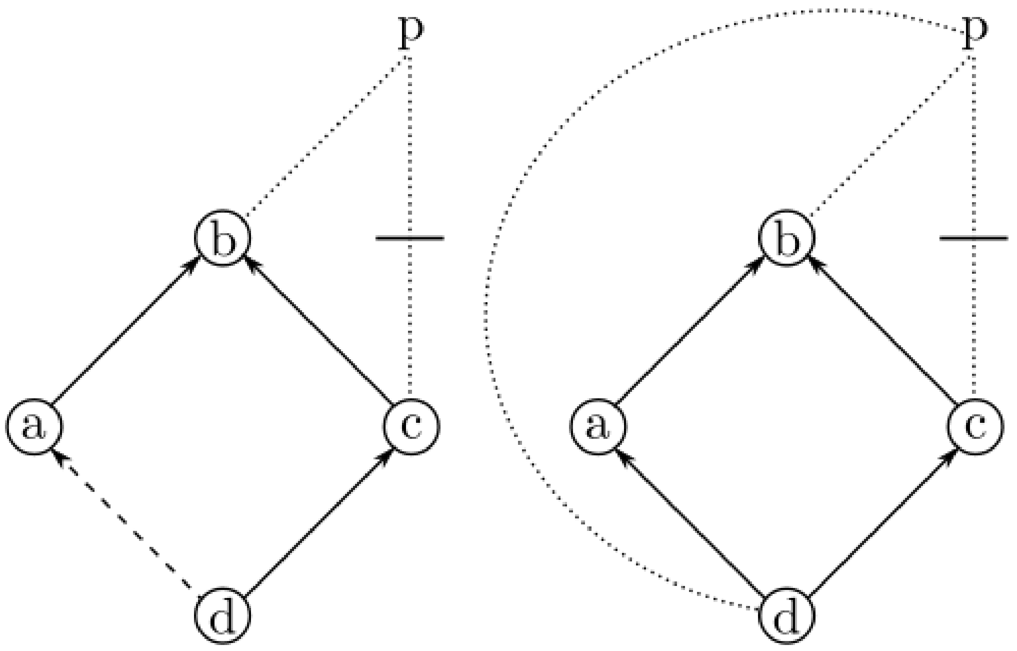


Figure 5.1: Inheritance Hierarchy

The arrows represent the relation between the classes and subclasses and the dotted lines the properties of objects that belong to said classes. The dotted arrow denotes the possible subclass relation between *a* and *d*. In order to describe this hierarchy in Gorgias we use the following rules:

```
rule(f1, subclass(a,b), []).
rule(f2, subclass(c,b), []).
rule(f3, subclass(d,c), []).
rule(f4, is_in(x1,a),   []).
rule(f5, is_in(x2,c),   []).
rule(f6, is_in(x3,d),   []).

rule(d1(X), has(X,p), [is_in(X,b)]).
rule(d2(X), neg(has(X,p)), [is_in(X, c)]).

rule(pr1, prefer(d2(X), d1(X)), []).
```

According to the above rules, x1 belongs to class a, x2 to c and x3 to d. *has(X,P)* denotes that X has property P. The following two rules represent the general properties of relations *subclass* and *is_in* and the last two, the closed world assumptions for simple hierarchies:

```
rule(r1(C0,C2), subclass(C0,C2), [C0 \= C1, C1 \= C2, C0 \= C2, subclass(C0,C1),
    subclass(C1,C2)]).
rule(r2(X,C1), is_in(X,C1),      [subclass(C0,C1), is_in(X,C0)]).
rule(d3(X,C), neg(is_in(X, C)),   []).
rule(d4(A,B), neg(subclass(A,B)), []).
```

5.5.2 Higher-Order Preferences

We will now extend the previous program with the rule which states that d is a subclass of a:

```
rule(f7, subclass(d,a), []).
```

and then that d1 is preferred to r2:

```
rule(d4, prefer(d1(X), d2(X)), [is\_in(X,a)]).
```

Now we can prove both d1(X) < d2(X) and d2(X) < d1(X) for x3 and there is a conflict, because we can prove that has(x3,p) and neg(has(x3,p)) are true. In order to resolve the above conflict, we need to rewrite d3 and d4 rules as below:

```
rule(d3_1(X), prefer(d2(X), d1(X)), [is_in(X,c), neg(is_in(X,a))]).
rule(d4_1(X), prefer(d1(X), d2(X)), [is_in(X,a), neg(is_in(X,c))]).
```

However, this way of writing rules, increases the possibility of a wrong answer and degrades the expressiveness of the language, because it increases the number of literals in the body of priority rules. A better way to write the above is with higher order priorities as below:

```
rule(d5(X), prefer(d4(X), d3(X)), [is_in(X,a)]).
```

According to this, d has p even though d is also a subclass of c.

The above ability can prove useful in the next legal reasoning scenario:

There are two legal principles that apply to one case concerning a security interest of a certain ship. Those principles are conflicting. The one law is newer than the other but the second is a federal law. ”Lex Posterior” gives precedence to the newer law but ”Lex Superior” to the one supported by the higher authority, which is the federal law.
The above can be solved in Gorgias by the following:

```
rule(lex_posterior(X,Y), prefer(X,Y), [newer(X,Y)]).
rule(lex_superior(X,Y),  prefer(Y,X), [state_law(X),federal_law(Y)]).
rule(prpr, prefer(lex_superior(X,Y),lex_posterior(X,Y)), []).
```

We use again the higher order priority, in this case lex posterior is inferior to lex superior, so the federal law applies.

5.5.3 Abduction

Up until this point, we have seen that Gorgias can return a result when there are conflicting arguments with different priorities. Sometimes, an argument depends on whether or not we have specific information about the case we are studying. However, there are cases in which some information may be missing and therefore we need to make assumptions in order to build an admissible argument. This information may also be dynamic. To solve this issue, we use abductive reasoning.

In abduction, we separate a set of predicates, the abducible predicates, which express the incomplete information. Then, when we send a query to Gorgias, the system extends the theory with ground abducibles in order to satisfy the given goal.

An example where abducibles can be used is a knowledge base that describes when a specific user has permission to write in a given file of a UNIX system.

```
rule(owner_changes_permissions(U,F),
can_change_permissions(U,F),
[is_file_owner(U,F)]).
rule(root_changes_permissions(U,F),
can_change_permissions(U,F),
[is_root(U)]).
rule(change_permissions(U,F),
has_write_permission(U,F),
[can_change_permissions(U,F)]).
rule(write_file(U,F),
can_write_file(U,F),
[has_write_permission(U,F)]).
```

Then, we need to define the abducibles:

```
abducible(is_root(_),[]).
abducible(is_file_owner(_,_),[]).
abducible(has_write_permission(_,_),[]).
```

When we give the following query to Gorgias we will get the result:

```
?-prove([can_write_file(user,file)],Delta).

Delta = [ass(is_file_owner(user, file)),owner_changes_permissions(user, file),
    change_permissions(user, file),write_file(user, file)] ;
Delta = [ass(is_root(user)),root_changes_permissions(user, file),
    change_permissions(user, file),write_file(user, file)] ;
```

```
Delta = [ass(has_write_permission(user, file)),write_file(user, file)] ;
false.
```

In this process, Gorgias extends the predicates with the abducibles and then it constructs the admissible arguments which show what a user must do to gain access to the file.

Chapter 6

Implementation

6.1 Purpose

One of the biggest problems that hospitals have during the pandemic is a lack of sufficient ICU beds. We wanted to implement an explainable AI algorithm that takes COVID-19 patient data and predicts whether they will be admitted to the ICU. Such an algorithm would show medical personnel the most important symptoms to look for, which patients that should be under surveillance for severe complications. This process should also hopefully give enough time for the stuff to manage ICU beds or if they no longer have any, to contact other hospitals that may still do.

6.2 Dataset

The data set was chosen from the webpage Kaggle (`https://www.kaggle.com`). It contains anonymized data from Hospital Sírio-Libanês in São Paulo, Brazil [1]. Brazil was one of the most affected countries by COVID-19. The data was scaled by column with Min Max Scaler to fit between -1 and 1. They consist of the patients demographic information (3 data), their previous diseases (9 data), blood results (36 data) and vital signs (6 data). They all add to 54 features, which are then expanded to mean, median, maximum, minimum, difference and relative difference. Difference is the one between minimum and maximum : $diff = max - min$, while relative difference is difference divided by the median :$relative\ diff = diff/median$.

For every patient, there can be more than one row of data, as they are grouped chronologically in time from admission windows: 0-2 hours, 2-4, 4-6, 6-12, 12 and above. Not all patients reach the 12 hour mark, as many were admitted to the ICU before. The last column is 0 for non-admission to the ICU during that specific window and 1 for admission.

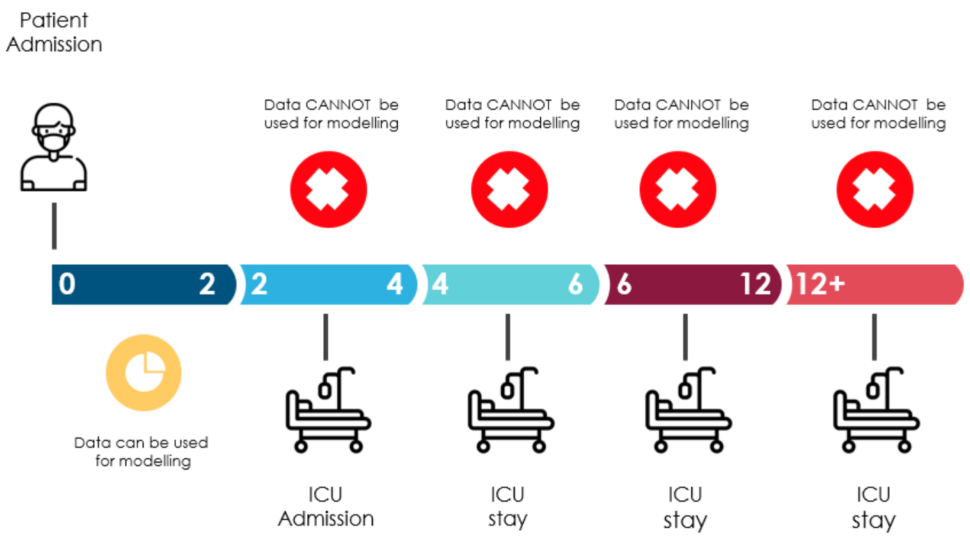

Patient Admission
Data CANNOT be used for modelling
Data CANNOT be used for modelling
Data CANNOT be used for modelling
Data CANNOT be used for modelling
0
2
2
4
4
6
6
12
12+
Data can be used for modelling
ICU Admission
ICU stay
ICU stay
ICU stay

Figure 6.1: Data in time windows

There are a lot of missing data in the data set, as some tests were done more frequently than others. The ones that are missing are similar to the ones from neighboring windows. Overall, there are 231 columns in the data set, which are used as features, and 1925 rows, which however don't imply that there are as many patients, because we have multiple data for almost every one of them. The number of patients in this data set is 385.

In order to train our algorithms with this data set, first we had to fill the blank rows with the previous data, then we need to get rid of the rows in which the patient is already in the ICU, as those should not be part of the training process.

After preparing the data set, we use the Python function $GroupShuffleSplit$, in order to separate it into training and test data, (80% and 20% of the original data set respectively), while being grouped by the Patient Identifier, as one patient cannot be both on the training and the test data.

GroupShuffleSplit function

```
from sklearn.model_selection import GroupShuffleSplit
def train_test_with_id(test_size,dt_df):
train_inds, test_inds = next(GroupShuffleSplit(test_size=test_size,
 n_splits=2, random_state = 7).split(dt_df, groups=dt_df['
 PATIENT_VISIT_IDENTIFIER']))

train = dt_df.iloc[train_inds]
test = dt_df.iloc[test_inds]
return train, test
```

The final step is to drop the Patient Identifier column, as the algorithm should not make decisions based on finding individuals, and the ICU column, because it is the output.

After the above process is finished, we can begin with the training.

6.3 Training

For the training we first tried to use only the 0-2 hours window to see if we could produce a good result that way.

6.3.1 Fewer Data

We used a variety of ML algorithms:

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

logisticRegr = LogisticRegression(random_state=0)
logisticRegr.fit(X_train,Y_train)
```

2. SVM

```
from sklearn import svm

clf = svm.SVC(kernel='linear')
clf.fit(X_train, Y_train)
```

   In this case the linear kernel worked the best.

3. MLP

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='adam', alpha=1e-5,hidden_layer_sizes=(29,2),
    random_state=1,max_iter=400)
clf.fit(X_train, Y_train)
```

   For the MLP classifier we used the adam optimization algorithm, which is an extension of the stochastic gradient descent algorithm. We also used 2 hidden layers, the first with 29 and the other with 2 neurons respectively. Finally, we set the maximum iterations to 400.

4. Random Forest

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=900, max_depth=None,
    min_samples_split=2, random_state=0)
clf.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

In the random forest classifier we set maximum iterations to 900 and we allowed the maximum depth of each individual decision tree to be as big as it can.

5. Adaboost

```
from sklearn.ensemble import AdaBoostClassifier

classifier = AdaBoostClassifier(None,130,0.1,'SAMME.R',10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

For the simple Adaboost classifier we use decision trees as the base estimator, the number of estimators is set to 130 and the algorithm used is SAMME.R. SAMME.R works similar to SAMME but it converges faster and achieves a lower test error. It also gives all models an equal weight of one and outputs the probability of an item belonging to a class.

6. Adaboost with SVM as weak learner

```
clf = svm.SVC(kernel='linear')
classifier = AdaBoostClassifier(clf,200,0.1,'SAMME',10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

Here we used Adaboost with SVM as a weak learner. The SVM algorithm had the same parameters as the one used before, but Adaboost here used SAMME and not SAMME.R, because in order to use SAMME.R, the weak learner must support the calculation of class probabilities, which isn't possible with SVM. The number of estimators was set to 200.

7. Adaboost with Logistic Regression as a weak learner

```
clf = LogisticRegression(random_state=0)
classifier = AdaBoostClassifier(clf,400,0.1,'SAMME.R',10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

In this instance, we used Adaboost with logistic regression as a weak learner. Logistic regression had the default parameters and Adaboost used the SAMME.R algorithm and 400 estimators.

8. Adaboost with Random Forest as a weak learner

```
rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None
    ,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
classifier = AdaBoostClassifier(rf,50,0.01,'SAMME.R',10)
```

Here, Adaboost uses random forest as a weak learner. Random forest uses the gini criterion to measure the quality of the split, which is the default, a maximum depth of 10 for the decision trees, a minimum of 2 samples required to be at a leaf node and 100 estimators. Adaboost uses 50 estimators, which is the default, a learning rate of 0.01, so that each classifier contributes less to the result, and the SAMME.R algorithm.

9. ExtraTrees Classifier

```
from sklearn.ensemble import ExtraTreesClassifier

classifier = ExtraTreesClassifier()
classifier.fit(X_train,Y_train.values.ravel())
```

After training, we tested all those algorithms with the test set, with the results in Figure 6.2.

From those results, we notice that the highest accuracy belongs to the MLP algorithm, 80.5%, even though it was significantly slower than all the other ones. The problem was complicated enough for the complexity of a neural network to be an asset. The next was Extra Trees with a maximum accuracy of 79.2%, although it varied, as, because of the random selection, the result was different every time the algorithm ran. Adaboost didn't perform as well and, with this data set, the best weak learner was Logistic Regression, which gave an accuracy of 77.1%. The worst result was that of logistic regression with an accuracy of 68.8%, which was to be expected, as the problem is too complex and has too many features and too little data for the logistic regression algorithm. Adaboost with random forest and Adaboost with SVM also had a relatively low accuracy (70% and 72.7% respectively). Finally, the simple SVM, Adaboost and Random Forest algorithms were

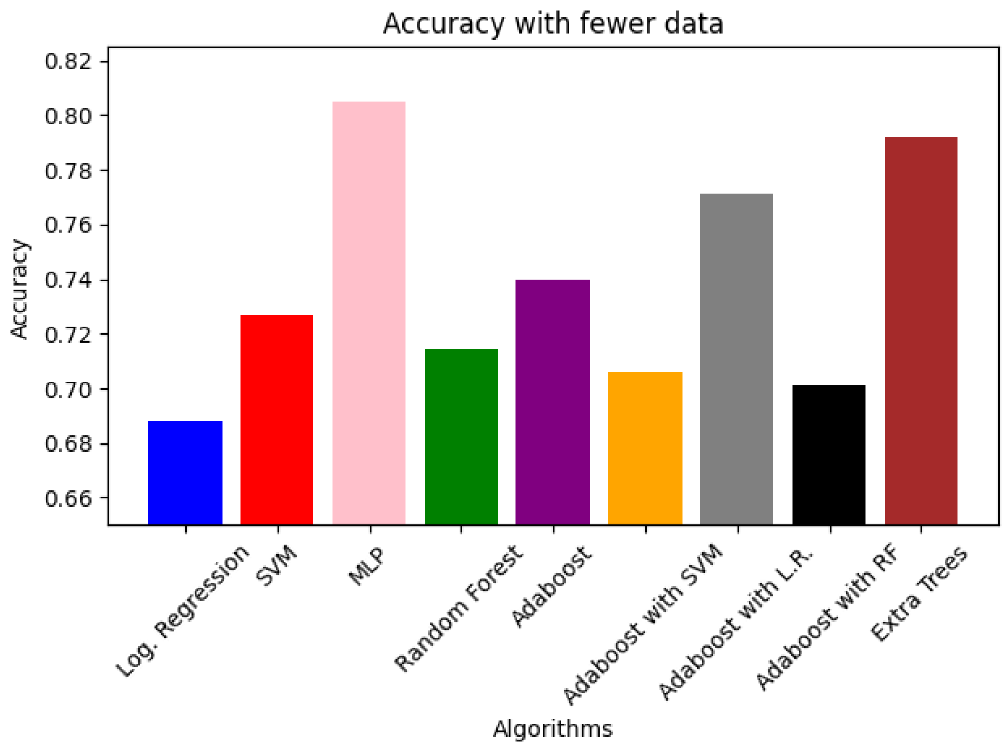


Figure 6.2: Training results with data from the first window

also not as successful as the first three algorithms.

In order to achieve higher accuracy, we then used data from all five time windows.

6.3.2 All Data

For this training, we used all windows in which the patients were not yet admitted to the ICU.
We used again the same ML algorithms as previously, but some of them with different parameters:

1. MLP

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='adam', alpha=1e-5,hidden_layer_sizes=(31,2),
    random_state=1,max_iter=320)
clf.fit(X_train, Y_train)
```

In this instance, we used the same SVM algorithm as before but with 31 neurons on the first layer and 2 on the second as previously.

2. Adaboost with Random Forest as weak learner

```
rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None
    ,
                       criterion='gini', max_depth=None, max_features='auto'
    ,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
classifier = AdaBoostClassifier(rf,50,0.01,'SAMME.R',10)
```

This algorithm is the same as before but with 200 estimators instead of 100. Adaboost uses again 50 estimators and a 0.01 learning rate.

3. Random Forest

```
clf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=
    None,
                       criterion='gini', max_depth=None, max_features='auto'
    ,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=250,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

clf.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

This random forest algorithm uses again the default gini criterion, a minimum of 2 samples for a leaf node, a minimum of 2 samples for a split and 250 estimators.

4. Adaboost

```
classifier = AdaBoostClassifier(None,140,1,'SAMME.R',10)
classifier.fit(X_train,np.array(Y_train).reshape(Y_train.shape[0],1))
```

In this Adaboost algorithm, we have the SAMME.R algorithm, 140 estimators and the default learning rate of 1.

5. SVM

```
from sklearn import svm

clf = svm.SVC(kernel='linear',gamma='auto',class_weight='balanced',max_iter
    =-1)
clf.fit(X_train, Y_train)
```

For the SVM algorithm, we used again the linear kernel, the class weight is balanced, which means that the weights are automatically adjusted inversely proportional to the class frequencies of the input data, and the maximum iterations are disabled, so that the algorithm finishes only when it has converged.

The results of the next training cycle were the following:

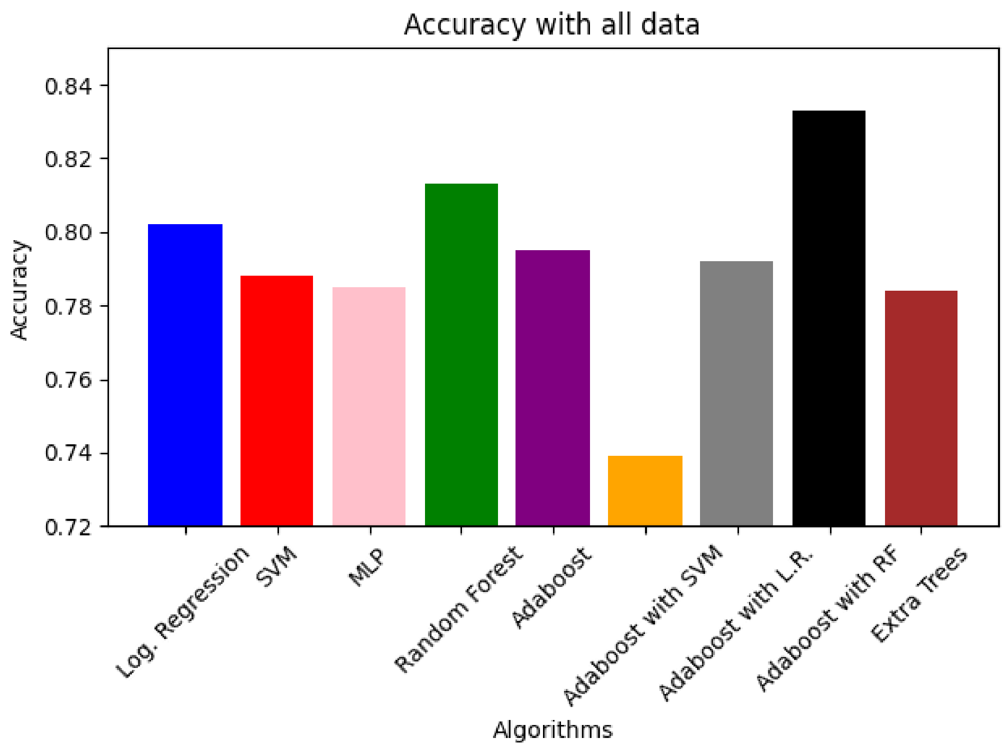


Figure 6.3: Training results with data from all windows

After this training process, the algorithm that exhibited the best accuracy was Adaboost with Random Forest as a weak learner (83.3%) and the second best was the simple Random Forest (81.3%). From that, we can conclude that the Adaboost and Random Forest algorithms achieved a better result when using more data. All other algorithms had very similar accuracy around 78-80% except from Adaboost with SVM as a weak learner, which had the worst predictions, with an accuracy of 74%.

Finally, we have the comparison for each algorithm between the 0-2 window data and all windows data from the two training cycles:

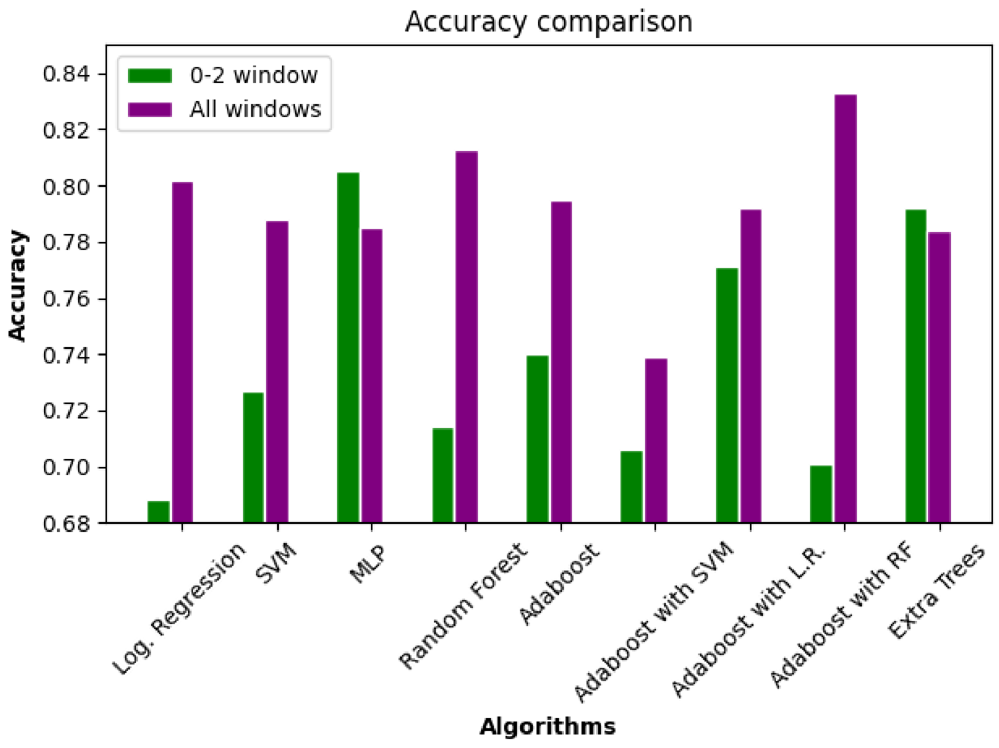


Figure 6.4: Final algorithm and dataset comparison

We notice an increased accuracy in almost all algorithms, except MLP and Extra Trees, when we increase the training data set size. MLP had a decrease in correct classifications, although it was not very significant, and it may have been due to the very big number of features, as the algorithm failed to converge. Extra Trees also had a small decrease, but due to the algorithm's random data selection, the difference is not relevant. For all other algorithms, and especially Adaboost with Random Forest, logistic regression and Random Forest, the data set with just the first time window did not contain enough data to train them. It is safe to assume that, if the data set contained data from more patients, the result for almost all algorithms would have been significantly better.

From the above graph, we conclude that the best ML algorithm in both cycles was Adaboost with Random Forest as a weak learner using the entirety of the data set, with an accuracy of 83.3% on the test set and the second best was Random Forest with 81.3%, and therefore we will try to use these two models to create the explainable AI algorithm for this thesis.

6.4 Intrees

Intrees is an R framework that prunes, measures and selects rules from a tree ensemble, such as random forests and boosted trees, and calculates the variable interactions [7].

6.4.1 The framework

Extract Rules

The InTrees framework expresses rules as: $\{C \Rightarrow T\}$, where C is the condition and T is the outcome. It starts from the root of the tree and works its way down until it reaches the leaves. However, the most informative splits are usually done at the top of the tree, so the algorithm may stop when a maximum depth is reached. Then it can assign the outcome based on the training data. This is more computationally efficient than working all the way down to every tree.

Algorithm 2 ruleExtract psedocode

```
1: procedure ruleExtract(ruleSet,node,C)
2: if leafnode = True then
3:  $currentRule \leftarrow \{C \Rightarrow pred_{node}\}$
4:  $ruleSet \leftarrow \{ruleSet, currentRule\}$
5: end if
6: for $child_i$ = every child of node do
7:  $C \leftarrow C \wedge C_{node}$
8:  $ruleSet \leftarrow ruleExtract(ruleSet, child_i, C)$
9: end for
10: end procedure
```

Measure Rules

Here, the InTrees algorithm measures the frequency of a rule, which is the proportion of the data that satisfy each one, the error (the number of incorrectly classified instances by the rule divided by the instances that satisfy the condition) and the complexity, which is the length of the value-variable pairs condition.

Prune Rules

Pruning strives to reduce the variable-value pairs of every condition, by getting rid of the ones deemed irrelevant.

Let's assume that E is the Error of a rule and E_0 the Error of the original rule $\{C \Rightarrow T\}$ and E_{-i} the E of the rule if we take out the i-th pair. We can then measure the effectiveness

Algorithm 3 condExtract psedocode

```
1: procedure condExtract(conSet, node, C, maxDepth, currentDepth) currentDepth = currentDepth + 1
2: if leafNode = True or currentDepth = maxDepth then
3: $condSet \leftarrow \{condSet, currentCond\}$
4: $return\ condSet$
5: end if
6: for $child_i$ = every child of node do
7: $C \leftarrow C \wedge C_{node}$
8: $condSet \leftarrow condExtract(condSet, child_i, C, maxDepth, currentDepth)$
9: end for
10: end procedure
```

of the reduction by using the $decay$ parameter:

$$decay_i = \frac{E_{-i} - E_0}{max(E_0, s)} \tag{6.1}$$

where s is a very small number, so that we don't divide with 0 if $E_0 = 0$. Another way we can define $decay_i$ is:

$$decay_i = E_{-i} - E_0 \tag{6.2}$$

If decay is smaller than a given number which is the threshold, the i-th pair can be left out.

Select Rules

The select rules function tries to find the top rules which are relevant and non-redundant. One way to achieve that is by applying feature selection to the conditions by creating a new data set (I):
Let's assume $\{c_1, c_2, ...c_J\}$ the conditions of a rule set, then I_{ij} denotes whether a condition c_j is satisfied by the i-th instance:

$$I_{ij} = \begin{cases} 1 & c_j issatisfiedbythei - thinstance \\ 0 & otherwise \end{cases} \tag{6.3}$$

The new data set is : $\{[I_{i1}, I_{i2}, ..., I_{ij}, t_i], i = 1, ..., j\}$ with t_i the target value of the i-th instance. Then we can apply feature selection on this data set to find the relevant and non-redundant rules.

Another way to achieve rule selection is by taking into account the rule complexity (length of the rule). We can use Regularized Random Forests (RRF) and their information gain:

$$Gain_R(X_i) = \begin{cases} \lambda_i \times Gain(X_i) & X_i \notin F \\ Gain(X_i) & X_i \in F \end{cases} \tag{6.4}$$

where F is a set of indices used to split previous nodes, it starts as empty at the root, $Gain$ represents the information gain for a specific metric and $\lambda_i \in (0, 1]$ is the penalty coefficient, the number with which we penalize a metric if a variable was not used in previous nodes. We then add the i-th variable to F if it provides new information to the ones that already exist. All $_i$ are the same number in RRF, the smaller it is, the bigger the penalty. It also depends on the importance score that is calculated by the Random Forest algorithm, so with higher importance we get a higher λ_i. Here, the λ variable is calculated without the importance score:

$$\lambda_i = \lambda_0 * (1 - \gamma * \frac{l_i}{l*}) \tag{6.5}$$

with l_i the length of the condition and l^* the maximum length of all conditions in the set. γ controls the weight and λ_0 is the base coefficient. Here is the λ_i equation if we include the importance score:

$$\lambda_i = \lambda_0 * (1 - \gamma * \frac{l_i}{l*} + \beta * imp_i) \tag{6.6}$$

where imp_i the importance of the specific condition.

6.4.2 Use of InTrees

As stated before, the InTrees framework is written in R, so we had to take the trained model from Python and turn it into a format that R can process.

In R, array indentation starts from 1, whereas in Python with 0. Also, array format in R looks like $Array[, 5]$ as opposed to python's $Array[4]$.

On this specific model, the random forests were 2 and each one had 400 decision trees. Because of the SAMME.R algorithm, both random forests have the same weight of 1. The following Python command gives us the first decision tree of the random forest:

Decision Tree

```
decision_tree = rand_forest.estimators_[0]
```

and the next one returns the text representation of said decision tree followed by its format:

Text Representation

```
text_representation = tree.export_text(decision_tree,max_depth=100)
print(text_representation)


|--- feature_201 >  -0.33
|   |--- feature_205 <= 0.08
|   |   |--- feature_139 <= 0.38
|   |   |   |--- feature_147 <= -0.46
|   |   |   |   |--- feature_114 <= -0.76
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_114 >  -0.76
|   |   |   |   |   |--- feature_129 <= -0.63
|   |   |   |   |   |   |--- feature_85 <= -0.95
|   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |   |--- feature_85 >  -0.95
|   |   |   |   |   |   |   |--- class: 0.0
|   |   |   |   |   |--- feature_129 >  -0.63
|   |   |   |   |   |   |--- class: 1.0
|   |   |   |--- feature_147 >  -0.46
|   |   |   |   |--- feature_122 <= -0.65
|   |   |   |   |   |--- class: 1.0
|   |   |   |   |--- feature_122 >  -0.65
|   |   |   |   |   |--- class: 0.0
|   |   |--- feature_139 >  0.38
|   |   |   |--- feature_162 <= 0.10
|   |   |   |   |--- class: 1.0
|   |   |   |--- feature_162 >  0.10
|   |   |   |   |--- class: 0.0
|   |--- feature_205 >  0.08
|   |   |--- class: 1.0
```

The number of ”|” on the left of every node represents the depth of that node in the tree. The nodes above that which have less ”|”, are its parents and the first node is the root. We now need to take it and turn it into a representation such as the one bellow, which is the one that R uses to represent decision trees:

Decision Tree in R

```
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]<=0.46 & X[,115]<=0.76
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]<=0.46 & X[,115]>0.76
    & X[,130]<=0.63 & X[,86]<=0.95
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]<=0.46 & X[,115]>0.76
    & X[,130]<=0.63 & X[,86]>0.95
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]<=0.46 & X[,115]>0.76
    & X[,130]>0.63
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]>0.46 & X[,123]<=0.65
X[,202]>0.33 & X[,206]<=0.08 & X[,140]<=0.38 & X[,148]>0.46 & X[,123]>0.65
X[,202]>0.33 & X[,206]<=0.08 & X[,140]>0.38 & X[,163]<=0.10
X[,202]>0.33 & X[,206]<=0.08 & X[,140]>0.38 & X[,163]>0.10
```

```
X[,202]>0.33 & X[,206]>0.08
```

This is every branch on the right side of the decision tree shown above. In order to do that, we create a string of all the parents of its leaf using the required format (instead of $feature\_number$ we put $X[, number + 1]$) and put a "&" between them.

Format Conversion

```
node = ("X[,{}]{}{}".format(feature_int, tempstr[1],tempstr[2] ))
tree = tree + node + " & "
```

After this step, the R language can use the above as a one random forest and process it accordingly.

We use the same train and test sets as before and the list created above, as the extracted rules of the random forest algorithm. We can use it as metrics with the following command

Rule Metric

```
ruleMetric <- getRuleMetric(mylist,X_train,Y_train)
```

then prune those metrics:

Prune Rules

```
ruleMetric <- pruneRule(ruleMetric,X_train,Y_train)
```

so that we are left with much fewer paths. Now, we can use the select rules function to find the top relevant and non-redundant rules:

Select Rule Metric

```
ruleMetric <- selectRuleRRF(ruleMetric,X_train,Y_train)
```

Finally, we need to apply those rules on the test set

Apply learner

```
learner <- buildLearner(ruleMetric,X_train,Y_train)
pred <- applyLearner(learner,X_test)
```

and calculate accuracy, precision, recall and F1-score.

Accuracy

```
precision <- length(which((Y_test=="ICU")&(pred=="ICU")))/length(which(pred
    =="ICU"))
recall <- length(which((Y_test=="ICU")&(pred=="ICU")))/length(which(Y_test==
    "ICU"))
f1score <- 2*precision*recall/(precision+recall)
accuracy <- length(which(Y_test==pred))/length(Y_test)
```

6.4.3 Results

Adaboost with Random Forest as weak learner

We first trained the adaboost with random forests algorithm. The selectRuleMetric command returned the 7 most relevant rules, which were:

metric	len	freq	err
1	2	0.0116906474820144	0
2	1	0.0386690647482014	0.0930232558139535
3	2	0.295863309352518	0.142857142857143
4	1	0.0170863309352518	0.0526315789473685
5	3	0.448741007194245	0.232464929859719
6	2	0.0638489208633094	0.183098591549296
7	1	0.12410071942446	0.463768115942029

metric	condition	pred
1	X[,159]>0.91 & X[,50]<=0.32	Not ICU
2	X[,154]<=0.92	ICU
3	X[,91]>0.36 & X[,194]>0.04	Not ICU
4	X[,195]>0.14	ICU
5	X[,139]<=0.38 & X[,141]>0.35 & X[,159]>0.25	Not ICU
6	X[,89]<=0.36 & X[,49]<=0.34	ICU
7	X[,1]==X[,1]	ICU

”len” is the length of the path, ”freq” represents how many input instances are satisfied by the condition , ”err” is the number of incorrectly classified instances and ”pred” the prediction of whether the patient will be admitted to the ICU or not.

We can now look at the initial data set to see which symptoms correspond to the given conditions:

1st

```
If  Sat02_venous_mean > 0.91 and calcium_min <= 0.32 then "Not ICU"
```

If the saturation of oxygen in the veins is enough and the minimum of calcium concentration is not too high, then the patient is probably not going to get admitted. The oxygen saturation was a metric that was expected to show up in the results, as COVID-19 is a disease that affects the lungs and it has also been shown in various studies that low oxygen saturation in the veins can lead to hospitalization [11]. The minimum of calcium is probably due to the small sample of the data set, as it doesn't appear to have medical meaning.

2nd

```
If Sat02_arterial_mean <=0.92 then "ICU"
```

If the oxygen saturation in the arteries is low, then the patient is not getting enough oxygen and will probably be admitted to the ICU.

3rd

```
If lactate_max > 0.36 and bloodpressure_sistolic_mean > 0.04 then "Not ICU"
```

If the lactate maximum is above average and mean of the systolic blood pressure is also at least above average, then the patient will not be admitted.

4th

```
If heartrate_mean > 0.14 then "ICU"
```

If the mean of the heart rate is much higher than average, then the patient will probably be admitted to the ICU. This was also to be expected, as COVID-19 patients have been reported to have a much higher heart rate than usual and also patients with pre-existing heart conditions have a higher chance of becoming critically ill.

5th

```
If ph_venus_mean <= 0.38 and ph_venus_max > 0.35 and sat02_venous_mean >
    0.25 then "Not ICU"
```

If the mean of the pH of the veins is not too high, the max high and the mean of oxygen saturation is higher than average, then the patient will no be admitted. Lower pH levels have been associated with COVID-19 and some have linked lower pH with lower survival rates [9].

6th

```
If lactate_mean <= 0.36 and calcium_mean <= 0.34 then "ICU"
```

If the mean of the lactate is relatively low and the mean of calcium is also a bit low, then the patient might be admitted to the ICU. Low serum calcium has been linked with severe disease in various studies [14].

7th

```
If none of the above are true then "ICU"
```

In the above metrics there was no mention of age, sex or preexisting conditions of patients. All the metrics were about blood results and vitals which were also tho ones that changed the most from patient to patient or even for the same patient in different time windows. Some results, such as lactate levels were the opposite of what would be expected, as higher lactate levels have been linked to severe symptoms while in the model we saw that lower lactate levels meant increased chance of ICU admissions. It is important to note that we are not a medical professionals and therefore not qualified to make assumptions on medical data, however most deviations are due to the small sample size of the data set. 385 patients, all from the same hospital during the same period of time, are not enough to draw conclusions for the symptoms and the severity of a disease.

The accuracy, precision, recall and f1 score of this model were:

$$\begin{aligned} accuracy &= 78.7\% \\ precision &= 75\% \\ recall &= 61.2\% \\ f1\ score &= 67.4\% \end{aligned}$$

For the selection of the model, more emphasis was put on finding the one with the higher possible recall, because, for the selection of possible ICU patients, it's more important not to miss a potential critically ill patient, than to not have many false positives or have a good overall accuracy.

Random Forest

Then we tried the same but using the simple random forest algorithm. The selectRuleMetric command returned the following 10 rules:

metric	len	freq	err
1	2	0.0116906474820144	0
2	2	0.0215827338129496	0.0416666666666666
3	2	0.0539568345323741	0.0666666666666667
4	2	0.138489208633094	0.123376623376623
5	2	0.0251798561151079	0.178571428571429
6	1	0.0260791366906475	0.172413793103448
7	2	0.58273381294964"	0.195987654320988
8	1	0.0188848920863309	0.19047619047619
9	2	0.0305755395683453	0.352941176470588
10	1	0.0908273381294964	0.465346534653465

metric	condition	pred
1	X[,160]>0.91 & X[,139]>0.41	Not ICU
2	X[,166]<=0.05 & X[,207]>0.31	ICU
3	X[,136]>0.26 & X[,140]<=0.53	ICU
4	X[,200]>0.06 & X[,165]>0.05	Not ICU
5	X[,89]<=0.16 & X[,11]>0.50	ICU
6	X[,141]<=0.32	ICU
7	X[,159]>0.34 & X[,160]<=0.47	Not ICU
8	X[,164]>0.08	Not ICU
9	X[,89]<=0.36 & X[,51]<=0.35	ICU
10	X[,1]==X[,1]	ICU

Now, let's find out which symptoms are used on the rules above.

1st

```
If Sat02_venous_mean > 0.91 and ph_venous_mean > 0.41 then "Not ICU"
```

If the oxygen saturation in the veins is high and the pH is also high, then the patient will not be admitted. Just like the first rule of the adaboost algorithm, the most important feature is the oxygen saturation in the blood and here specifically in the veins. If there is enough oxygen then the patient's lungs work well and they do not have severe symptoms. Also, a low pH in the veins (acidic), as we have mentioned before, has been linked with severe illness, so here the high pH value is a good sign.

2nd

```
If sodium_max <= 0.05 and heart_rate_min > 0.31 then "ICU"
```

If the sodium maximum is low and the heart rate high, then the patient will need to go to the ICU. Low sodium has been linked to severe disease in COVID-19 patients [10]. An elevated heart rate is a common symptom of fever and infection.

3rd

```
If ph_arterial_max > 0.26 and ph_venous_mean <= 0.53 then "ICU"
```

If the pH of the blood in the arteries is high and in the veins not as high, the blood gets more acidic when in the veins, then the patient will probably be admitted to the ICU. As we stated before, this has been observed in studies.

4th

```
If bloodpressure_sistolic_median > 0.06 and sodium_min > 0.05 then "Not ICU"
```

If the blood pressure is above average and the minimum of the sodium measurements is also above average, the patient is not in danger. As stated before, low blood pressure and low sodium are usual predictors of serious symptoms.

5th

```
If lactate_mean <= 0.16 and immunocompromised then "ICU"
```

If lactate is low and the patient is immunocompromised (the patient's immune defences are low and can't protect them from any illness or infection) then they will probably get severe symptoms. Lactate here is again not a good indicator for other data sets, but immunocompromised patients are proven to be high risk in becoming very sick, because their immune systems can't fight the virus.

6th

```
If ph_venous_max <= 0.32 then "ICU"
```

If the pH of the blood in the veins is low (the blood is acidic), the patient risks going to the ICU. Again, as we mentioned before, acidic blood is an indicator of severe disease.

7th

```
If sat02_venous_mean > 0.34 and sat02_venous_min <= 0.47 then "Not ICU"
```

Low oxygen levels in the blood indicate that the patient already has serious symptoms, therefore high values for oxygen saturation in the veins is again a good sign.

8th

```
If sodium_mean > 0.08 then "Not ICU"
```

If sodium on the bloodstream is above average, then the patient is not in any danger.

9th

```
If lactate_mean <= 0.36 and calcium_max <= 0.35 then "ICU"
```

If the patient has low lactate and low calcium then they will probably be admitted. As we stated before, low calcium is an indicator for severe disease [14] but low lactate is not, so this is probably an error because of the small size of the data set, where some patients happened to have low lactate while also becoming severely ill.

10th

```
If none of the above then "Not ICU"
```

In the above metrics, we used very similar features as in the adaboost with random forest as a weak learner algorithm. Again, all metrics were about blood results and vitals and we also found a correlation between low lactate levels and ICU admission, which indicates that this is indeed an issue of the data set. The rest of the rules were mostly about oxygen saturation, as in the previous ones, pH, heart rate and calcium, all of which were also present in the previous algorithm's rules. New additions were sodium and immunocompromised patients, which were all in accordance with studies that agree these symptoms are important in predicting if a patient will get severely ill. The following are the accuracy, precision, recall and f1 score of the above rules for the test set.

$$\begin{aligned} accuracy &= 76.3\% \\ precision &= 68\% \\ recall &= 64.1\% \\ f1\ score &= 66\% \end{aligned}$$

Comparison

The two sets of rules were in a lot of cases similar, the algorithms seemed to agree on which the important symptoms are. The rules from the adaboost algorithm were a bit better on the accuracy score (78.8%) than the random forest (76.3%) with a better f1 score as well. Only recall was a bit better on the random forest algorithm's rules. It is also interesting to note that the adaboost algorithm in Python also had a better accuracy than random forest by about 2%.

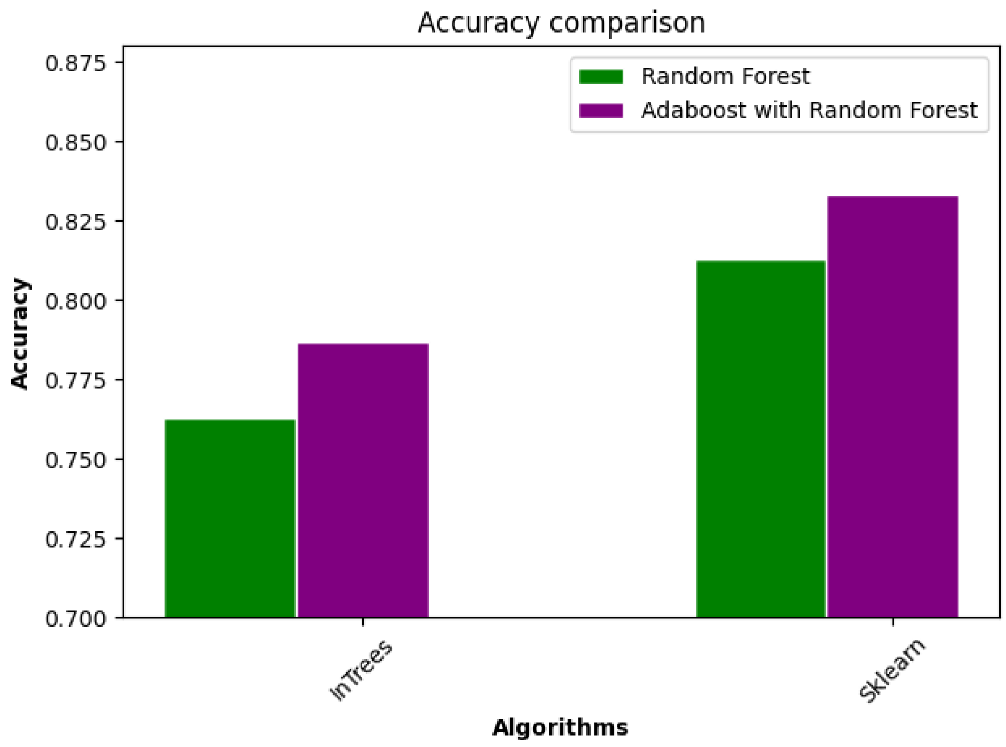


Figure 6.5: Comparison of Adaboost and Random Forest algorithms before and after intrees

Reducing the rules to only 7 and 10 respectively in turn reduces the accuracy a bit on both algorithms. However, when considering how many rules are discarded, a 5% reduction is not that impactful.

From the two intrees algorithms, we will continue the explainable AI process with the adaboost one, as it has both higher accuracy and a higher f1 score.

6.5 Gorgias

For the next step, we used Gorgias to depict the 7 rules from the InTrees framework to an argumentation framework.

Since we have only two options for the result, admission to the ICU and non admission to the icu, for every patient we have the options below:

$$icu(Patient), noticu(Patient)$$

The process of constructing the Prolog code is the following:

1 Define all arguments from the 7 rules

```
    :- dynamic  satvenusmean/2, calciummean/2, calciummin/2,  phvenousmean
    /2, phvenousmax/2, satarterialmean/2, heartratemean/2,
    bloodpressuresismean/2,  lactatemean/2, lactatemax/2.
```

2 Define the complementary nature of the two result options

```
complement(icu(Patient),noticu(Patient)).
complement(noticu(Patient),icu(Patient)).
```

As stated before, the *complement* function states that only one of the two results can be returned as true every time.

3 Define all rules based on the 7 rules we got from InTrees

```
rule(r1(Patient),noticu(Patient),[]):- satvenusmean(Patient,Satvenmean),
    calciummin(Patient,Calmin), Satvenmean > 0.91, Calmin =< 0.32 .
rule(r2(Patient),icu(Patient),[]):-satarterialmean(Patient,Satartmean),
    Satartmean =< 0.92.
rule(r7(Patient),icu(Patient),[]):-  satvenusmean(Patient,Satvenmean),
    Satvenmean < 10.
```

On the right part of the command we put the arguments that have to be satisfied, on the left, the head is the result we get if all arguments on the right are true, the body is blank and the name of the rule is r_{number}.

4 Resolve conflicts, so that we can respect the order of the rules based on their importance

```
rule(pr1(Patient), prefer(r1(Patient), r2(Patient)),[]).
rule(pr7(Patient), prefer(r2(Patient), r3(Patient)),[]).
```

In order to resolve conflicts, we use the $prefer/2$ argument, which states that from the two rules in the input, if both are satisfied, the first is preferred. In this case, we use it to keep the hierarchy of rules, as stated in the output of InTrees.

The argumentation process

For one patient, we get an input of all the vitals and blood results that we need, for example:

```
assert(satvenusmean(patient_number,0.5)).
```

Then, we get one request to prove that the patient will end up in the icu and one that they will not.

```
prove([icu(patient_number)],Delta).
prove([noticu(patient_number)],Delta).
```

The program returns two answers, one for the correct prediction, with the rule that is the first to be satisfied from the 7 rules, and the other with $False$.

```
ICU:
Delta=[r2,pr5(r2,r3)].
Not ICU:
False.
```

After verifying that the program responds as expected, we tested it with the entire test set, to find out if ICU and NotICU were really complementary (that we can't have $icu(Patient)$ and $noticu(Patient)$ be both true or both false). There was no conflict on the test set or the train set.

We also tested to see if the result for every patient was the same in both the R program and the Gorgias one.

We tested this by putting the Gorgias results in a csv file, the R results in another csv file and then we used the linux command $diff$ to find any differences:

```
diff gorgias_results.csv r_results.csv
```

There weren't any so it turns out we have again

$$accuracy = 78.7\%$$
$$precision = 75\%$$
$$recall = 61.2\%$$
$$f1score = 67.4\%$$

Both programs work the same way.

6.6 Java Interface

Gorgias is a cloud application, so, in order to use it locally from a computer, we need an API to communicate with it. The Gorgias team has a Java code with the API available, so for the final part of the project we will use the Java programming language to send the data to Gorgias and receive the results.

We first had to import the API code, initialize the environment, connect to our profile in the Gorgias cloud and consult the ICU prediction document we created there.

```java
import gr.tuc.gorgiasCloud.client.GorgiasCloud;

GorgiasCloud gorgias =  new GorgiasCloud();
gorgias.login("username", "password");
gorgias.consult("ICU_prediction.pl","Corona"); //file under Corona/ICU_prediction.
    pl
```

Now we have the *gorgias* class, which has all the necessary functions.

After that, we need to send the *assert* commands of our input data

```java
gorgias.asserta(command);
```

The assert command will send the command we want to the Gorgias cloud and return an HTTP 200 OK or an error message if something goes wrong. And finally we will need to send a *proof* command

```java
_prove = "[noticu("+number+")],Delta";
it = gorgias.prove(_prove, 10).iterator();
```

The gorgias prove function sends the prove command and returns either an error if something unexpected happens or a list of Deltas of the rules that satisfy the command we just sent or the word *False* of the command can't be proven.

When we have finally sent everything we want, we finish by

```java
gorgias.unload("ICU_prediction.pl","Corona");
gorgias.logout();
```

The *unload* command unloads the prolog file, and logout closes the session.

Model verification

We can now check our Gorgias model for its errors and accuracy.
First, we start with the test set, which is in a csv file.

```
List<String[]> test_data = new ArrayList<>();
try(BufferedReader br = new BufferedReader(new FileReader("X_test_corona.csv")))
{
  String line = "";
  while ((line = br.readLine()) != null) {
    test_data.add(line.split(","));
  }
} catch (FileNotFoundException e) {
      Some_error_logging
}
```

and then we do the same for the classifications of the test set inputs.

Then, according to the 7 rules, we choose the specific symptoms from each input array to create the rules necessary, and then we assert each rule using the Gorgias API, as described before.

```
String[] Symptoms = {" satvenusmean","calciummean","calciummin","phvenousmean","
    phvenousmax","satarterialmean","heartratemean","bloodpressuresismean","
    lactatemean","lactatemax"};
    int[] Values = {158,48,49,138,140,153,194,193,88,90};
    String command;
    for(int i=0; i<Xarray.length; i++) {
        for(int j=0; j<Values.length; j++){
          command = (Symptoms[j]+"("+i+","+Xarray[i][Values[j]]+")");
          gorgias.asserta(command);
        }
      }
```

After we have asserted all rules for a particular patient, we can now request a proof from Gorgias on whether the patient will not be admitted

```
_prove = "[noticu("Patient_number")],Delta";
it = gorgias.prove(_prove, 10).iterator();
while(it.hasNext()) {
  temp = it.next();
  System.out.println("proof not icu \n"+temp);
}
```

and then the same for the admission with

```
_prove = "[icu("Patient_number")],Delta";
```

When all results of a patient are sent back from the Gorgias API, we can check for errors by inspecting if we have received a classification for both ICU admission and non admission or if we didn't receive a classification for either. We didn't spot any errors. Then, we compare the Gorgias classification to the correct classification in order to find the accuracy, precision, recall and f1 score and they were the same as in R, as we mentioned above.

```
if (icures == noticures) {
  errors++;
} else {
  if ((((y_val) == 0) && noticures) || (((y_val == 1)) && icures)) {
    correct_assumpt++;
  }
}
accuracy = correct_assumpt/test_set_number;
```

Explainable model

For the explainable java program, we use the same parts to communicate with the Gorgias API as in the previous one. For the rest, we ask the user from the terminal for the patient's blood results we need, according to the 7 rules:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the Patient Identifier:");
patient_id = Integer.parseInt(br.readLine());
System.out.println("Enter the mean of venous O2 Saturation:");
temp = Float.parseFloat(br.readLine());
symptoms.add(temp);
System.out.println("Enter the mean of calcium consentration:");
temp = Float.parseFloat(br.readLine());
symptoms.add(temp);
```

After getting all the necessary data, we create the gorgias' arguments:

```
String[] Names = {" satvenusmean","calciummean","calciummin","phvenousmean","
    phvenousmax","satarterialmean","heartratemean","bloodpressuresismean","
    lactatemean","lactatemax"};
for (int i=0; i<10; i++){
    command = (Names[i]+"("+patient_id+","+symptoms.get(i)+")");
```

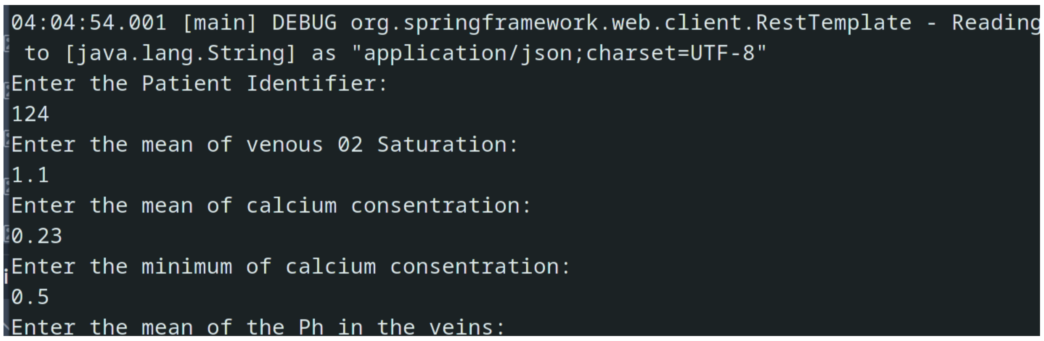
Then, using the same process as previously we send all arguments to Gorgias an get a result for $icu(Patient)$ and $noticu(Patient)$.

If one of them is $false$ and the other has returned a result, we proceed.

For the final step, we return the result to the user, explaining the reasoning behind the classification:

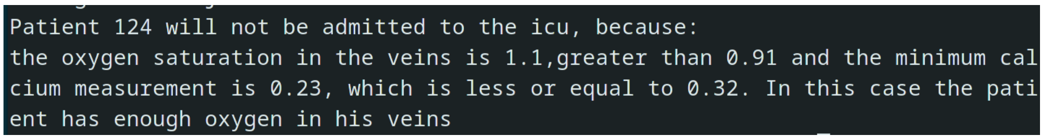
```
if (first_icu.equals("false")){
        result = first_noicu;
        System.out.println("Patient "+patient_id+" will not be admitted to the icu
    , because:");

      }else {
        result = first_icu;
        System.out.println("Patient "+patient_id+" will be admitted to the icu,
    because:");
      }
      if (result.charAt(2)=='1') {
        System.out.println("the oxygen saturation in the veins is "+symptoms.get
    (0)+",greater than 0.91 and the minimum calcium measurement is "+symptoms.get
    (1)+", which is less or equal to 0.32. In this case the patient has enough
    oxygen in his veins ");
      } else if (result.charAt(2)=='2'){
        System.out.println("the oxygen saturation in the arteries is "+symptoms.
    get(5)+",less or equal to 0.92, which suggests that the patient is not getting
     enough oxygen and might need to be intubated.");
            ....
```

The following is an example of how the program looks from a user's perspective.



```
04:04:54.001 [main] DEBUG org.springframework.web.client.RestTemplate - Reading
 to [java.lang.String] as "application/json;charset=UTF-8"
Enter the Patient Identifier:
124
Enter the mean of venous 02 Saturation:
1.1
Enter the mean of calcium consentration:
0.23
Enter the minimum of calcium consentration:
0.5
Enter the mean of the Ph in the veins:
```

Figure 6.6: Input on the explainable program



```
Patient 124 will not be admitted to the icu, because:
the oxygen saturation in the veins is 1.1,greater than 0.91 and the minimum cal
cium measurement is 0.23, which is less or equal to 0.32. In this case the pati
ent has enough oxygen in his veins
```

Figure 6.7: Result of the explainable program

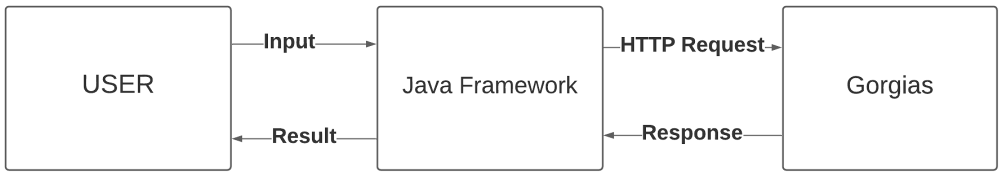


Figure 6.8: User communication with Gorgias through the Java framework

6.7 Results

To summarise the above process, we managed to take an adaboost with random forest as a weak learner algorithm and move it through the R language, so that we could use the inTrees framework and be left with only a few rules. Then, we created a Gorgias program based on said rules that can decide if a COVID-19 patient with the given blood results and vital signs will end up going to the ICU.

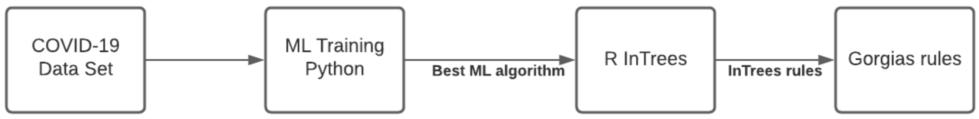


Figure 6.9: Model creation process

The final explainable model that we used had the following accuracy, f1 score, precision and recall:

$$accuracy = 78.7\%$$
$$precision = 75\%$$
$$recall = 61.2\%$$
$$f1score = 67.4\%$$

The model had about a 5% less accuracy than the normal adaboost one. However, it was comparable to the other trained models that we tried. Specifically, it had similar accuracy to Extra Trees,Adaboost with Logistic Regression as a weak learner, Adaboost, MLP and SVM. Also, by losing some accuracy we gained in explainability. The final model consists of only 7 rules, as opposed to the standard Adaboost we used which had about 800 decision trees. We also tested an explainable model on the Random Forest algorithm, which came back with 76.3% accuracy which is a bit lower that the other models.

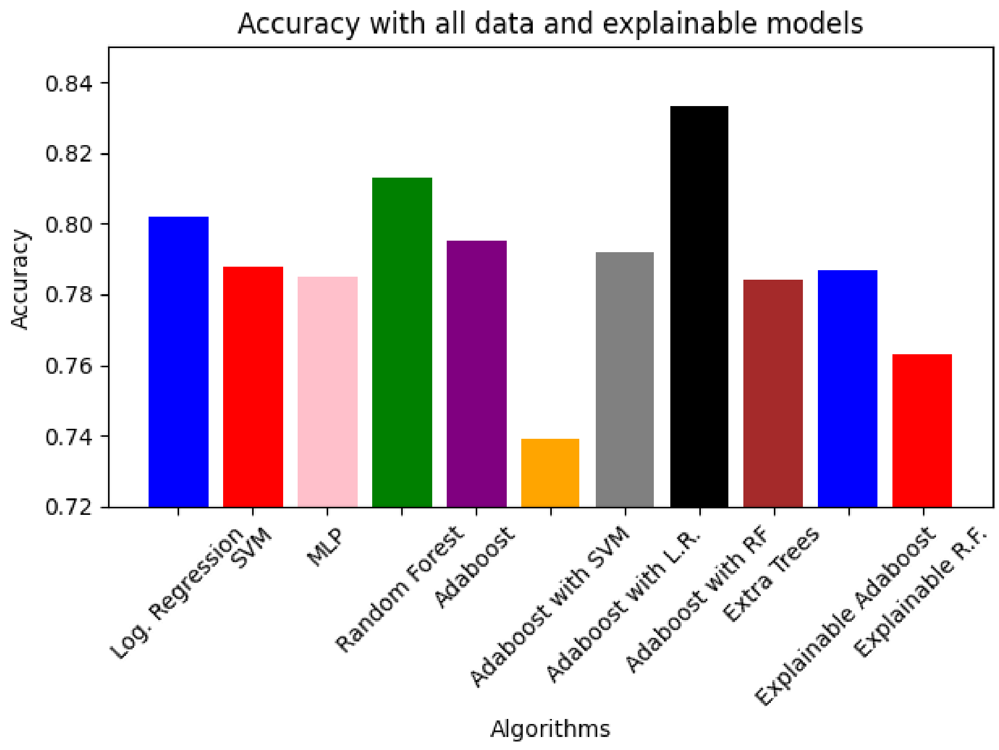


Figure 6.10: Comparison of all ml algorithms and explainable models

It is also evident that the explainable adaboost model had a better accuracy than the models we trained on only the first 0-2 window data from the data set. Now, when comparing with those models, the random forest explainable model is also comparable to some, e.g. Adaboost with SVM trained in the 0-2 window and performed better that others that probably didn't have enough data to converge.

In the explainable model, the important blood results and vitals that indicate that a patient will get severely ill are:

- Low oxygen saturation
- Low calcium concentration
- Low lactate
- Low blood pressure
- High heart rate
- Low pH in the blood

As we stated before, low lactate is probably a statistical anomaly because of the small data set. The rest, have all been linked in studies with severe illness.

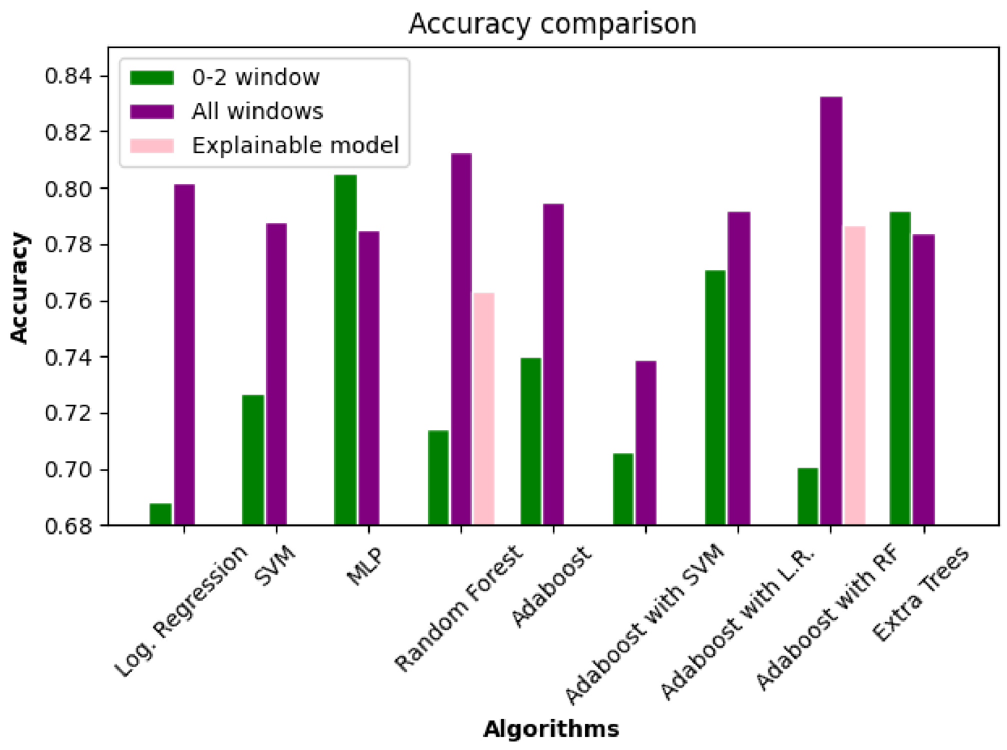


Figure 6.11: Comparison of all ml algorithms, explainable models and data sets

Chapter 7

Conclusion and Future Work

In this Thesis, we first created a ML model that can predict if a COVID-19 patient will be intubated within a short period of time, based on their medical history, blood results and vital signs. We used a data set from real COVID patients from Sirio-Libanes Hospital in Sao Paolo Brazil and tried several ML algorithms, such as Logistic Regression, SVM, MLP, Random Forest etc. We then compared them and the one with the best accuracy of 83.3%, was Adaboost with a Random Forest weak learner. This model can give an accurate prediction, but could not return to the user why a patient is classified as a potential ICU patient or the reasoning behind the prediction, as the Adaboost is a very complex black-box machine learning algorithm.

In order to solve this issue, we decided to create an explainable model, which could list the most important symptoms, based on which it classifies patients. For the first part of the process, we used the InTrees framework in the R language, which has functions that can prune rules from trees and select the top and non redundant rules from a random forest model or an Adaboost model with a random Forest weak learner. After using InTrees on our model, we were left with only 7 rules out of the initial 800, based on which we could classify the patients with a 78.7% accuracy.

For the next part of the process, we had to create a program that would allow the user to enter the patient information necessary to make a prediction and would then return the classification and the reasoning behind it. For this, we used the Gorgias framework, which is a Prolog based general argumentation framework, that can get rules as an input and compute if an argument is admissible. In this case the argument would be $icu(Patient)$ and $noticu(Patient)$ (whether a patient will be admitted to the ICU or not). This framework also allows higher-order preferences, which is important in our case, as the 7 rules of the model are ordered, if the first one is not satisfied, we move to the second etc. Consequently, we rewrote the 7 rules in Prolog in the Gorgias framework and tested it with the test set to see if the accuracy remained the same as with the R Intrees model. It turns out that the two programs have the exact same results and consequently the same accuracy.

Finally, we wrote a Java program to allow the user to communicate with the Gorgias framework. This program requires the user to enter the patient id number and only a few blood results and vitals of the patient, then it sends them to Gorgias and returns to the user the classification for the patient (icu or no icu admission) and the reasoning based on which Gorgias deduced the result, in this case the exact symptoms based on which the classification was made.

In conclusion, we have created a process that can give medical professionals information on which patients might require intubation and which symptoms they should look out for to predict serious illness in COVID-19 patients.

Our work can be extended in various directions. First of all, the same process, from start to finish, can be applied on many other diseases, such as strokes, heart attacks etc. as long as there is a good data set with blood results, vitals and medical information. We could train models that are based on Decision Trees, use the InTrees framework to reduce the rules to only a few and then create an explainable model that accurately predicts the course of each disease for every patient and explains its results to the user using Gorgias or any language of our choice.

Secondly, the explainable part from the R program and forward, can be used to create an explainable model of any Python trained classification model that came from Decision Trees, Random Forest, Extra Trees and Adaboost, regardless of what it classifies. We can take any such model and turn it into an explainable AI one, by changing very little things in the process we described on this thesis.

Lastly, it would be interesting to reiterate the whole process with a bigger data set from COVID-19 Patients, in order to find all statistically significant symptoms that prognosticate which patients will get severely ill and also have a model with a much higher accuracy and recall. In this case however, it is possible that another algorithm, that is not based on decision trees, could result in the highest accuracy, so we would have to use different methods of turning the model to a white box one. In the case that a bigger data set is hard to find, we could repeat the process after applying a dimensionality reduction strategy to decrease the number of features of the model to the ones that appear to be important, and then start training with various ML algorithms. This would probably achieve a higher accuracy, but it would still have some statistical anomalies because of the small size of the data set.

The Explainable AI techniques are a relatively recent addition to Artificial Intelligence, so there is still a lot to experiment with by using any of the existing black-box machine learning algorithms.

Bibliography

[1] https://www.kaggle.com/S%C3%ADrio-Libanes/covid19.

[2] https://www.nhs.uk/conditions/coronavirus-covid-19/people-at-higher-risk/whos-at-higher-risk-from-coronavirus/.

[3] https://www.cdc.gov/coronavirus/2019-ncov/hcp/clinical-guidance-management-patients.html.

[4] https://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf.

[5] *Gorgias Tutorial.* https://www.cs.ucy.ac.cy/~nkd/gorgias/tutorial.html.

[6] CDC. https://web.archive.org/web/20200912034420/https://www.cdc.gov/coronavirus/2019-ncov/hcp/planning-scenarios.html.

[7] H. Deng, *Interpreting tree ensembles with intrees*, (2014). https://arxiv.org/pdf/1408.5456v1.pdf.

[8] Y. Dimopoulos and A. Kakas, *Logic programming without negation as failure*. Available at: https://www.cs.ucy.ac.cy/~nkd/gorgias/docs/ISLP95-camera.pdf.

[9] D. Elezagic, W. Johannis, V. Burst, F. Klein, and T. Streichert, *Venous blood gas analysis in patients with covid-19 symptoms in the early assessment of virus positivity*, (2020). https://www.degruyter.com/document/doi/10.1515/labmed-2020-0126/html#j_labmed-2020-0126_ref_013_w2aab3b7c24b1b6b1ab2b2c13Aa.

[10] Y. Luo, Y. Li, and J. Dai, *Low blood sodium increases risk and severity of covid-19: a systematic review, meta-analysis and retrospective cohort study*, (2020). https://www.medrxiv.org/content/10.1101/2020.05.18.20102509v1.

[11] O. V. Oliynyk, M. Rorat, and W. Barg, *Oxygen metabolism markers as predictors of mortality in severe covid-19*. https://www.ijidonline.com/article/S1201-9712(20)32535-2/pdf.

[12] N. Prentzas, A. Nicolaides, E. Kyriacou, A. Kakas, and C. Pattichis, *Integrating machine learning with symbolic reasoning to build an explainable ai model for stroke prediction*, in 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE), 2019, pp. 817–821.

[13] S. Sanche, Y. T. Lin, C. Xu, E. Romero-Severson, N. Hengartner, and R. Ke, *High contagiousness and rapid spread of severe acute respiratory syndrome coronavirus 2*, (2020). `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7323562/`.

[14] X. Zhou, D. Chen, L. Wang, Y. Zhao, L. Wei, Z. Chen, and B. Yang, *Low serum calcium: a new, important indicator of covid-19 patients from mild/moderate to severe/critical*. `https://pubmed.ncbi.nlm.nih.gov/33252122/`.