ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

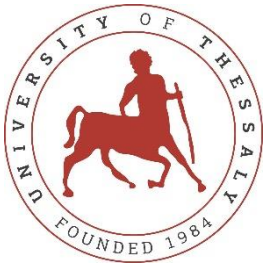
Μελέτη και σύγκριση μεθόδων μηχανικής μάθησης σε ιατρικά δεδομένα

Διπλωματική Εργασία

Λέφογλου Πολυδεύκης

Επιβλέπουσα: Τσομπανοπούλου Παναγιώτα

Σεπτέμβριος 2025



UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Study and comparison of machine learning methods on medical data

Diploma Thesis

Lefoglou Polydefkis

Supervisor: Tsompanopoulou Panagiota

September 2025

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπουσα	**Τσομπανοπούλου Παναγιώτα** Αναπληρώτρια Καθηγήτρια, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Βασιλακόπουλος Μιχαήλ** Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Φεύγας Αθανάσιος** Ε.ΔΙ.Π., Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστήμιο Θεσσαλίας

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διπλωματική εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο/Η Δηλών/ούσα

Λέφογλου Πολυδεύκης

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Lefoglou Polydefkis

Διπλωματική Εργασία

Μελέτη και σύγκριση μεθόδων μηχανικής μάθησης σε ιατρικά δεδομένα

Λέφογλου Πολυδεύκης

Περίληψη

Αυτή η διπλωματική εργασία εξερευνά διάφορες μεθόδους μηχανικής μάθησης και δίνει έμφαση στις εφαρμογές τους σε ιατρικά δεδομένα. Συγκεκριμένα, καλύπτεται μία ποικιλία αλγόριθμων κατηγοριοποίησης και μεθόδων μείωσης διαστάσεων, και 2 είδη νευρωνικών δικτύων, τα συνελικτικά νευρωνικά δίκτυα και τα επαναλαμβανόμενα νευρωνικά δίκτυα. Μετά την επεξήγηση αυτών των μεθόδων, πειραματιστήκαμε με αυτές πάνω σε διάφορα είδη ιατρικών δεδομένων. Συνολικά, έγιναν 3 πειράματα. Στο πρώτο πείραμα, εφαρμόσαμε μεθόδους μείωσης διαστάσεων σε συνδυασμό με αλγόριθμους κατηγοριοποίησης για την διάγνωση του καρκίνου χρησιμοποιώντας δεδομένα γονιδιακής έκφρασης, τονίζοντας τα οφέλη των μεθόδων μείωσης διαστάσεων στην διαχείριση μεγάλων δεδομένων. Στο δεύτερο πείραμα εστιάζουμε στην χρήση συνελικτικών νευρωνικών δικτύων πάνω σε ακτινογραφίες στήθους για την ανίχνευση της πνευμονίας, επιδεικνύοντας την Προσαρμοστική εξίσωση ιστογράμματος περιορισμένης αντίθεσης, μία πολύ ισχυρή τεχνική προεπεξεργασίας εικόνων, και την χρήση καμπυλών ROC και PR για την επιλογή του καλύτερου ορίου απόφασης σε προβλήματα δυαδικής κατηγοριοποίησης. Στο τελευταίο πείραμα, επιδεικνύουμε την ικανότητα των επαναλαμβανόμενων νευρωνικών δικτύων στην διαχείριση διαδοχικών δεδομένων, που στην περίπτωσή μας είναι κλινικές σημειώσεις, εξερευνώντας ταυτόχρονα την χρήση της μάθησης με μεταφορά. Όλα τα αποτελέσματα των πειραμάτων αποσκοπούν στο να δείξουν πως η μηχανική μάθηση παρέχει ακριβείς και αποδοτικές λύσεις στον τομέα της ιατρικής.

Λέξεις-Κλειδιά:

Μηχανική μάθηση; ιατρικά δεδομένα; κατηγοριοποίηση; μείωση διαστάσεων; νευρωνικά δίκτυα; συνελικτικά νευρωνικά δίκτυα; επαναλαμβανόμενα νευρωνικά δίκτυα; προσαρμοστική εξίσωση ιστογράμματος περιορισμένης αντίθεσης; καμπύλη ROC; καμπύλη PR; μάθηση με μεταφορά

Diploma Thesis

Study and comparison of machine learning methods on medical data

Lefoglou Polydefkis

Abstract

This diploma thesis explores several machine learning methods and emphasizes on their application on medical data. Specifically, we cover a variety of classification algorithms and dimensionality reduction methods, and two types of neural networks, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). After explaining those methods, we experimented with them across different types of medical data. In total, 3 experiments were conducted. In the first experiment, we applied dimensionality reduction methods in combination with classification algorithms for cancer diagnosis using gene expression data, highlighting the benefits of dimensionality reduction methods when handling large datasets. The second one focuses on the use of CNNs on chest X-rays to detect pneumonia, while showcasing Contrast Limited Adaptive Histogram Equalization (CLAHE), a very potent image preprocessing technique, as well as the use of ROC and PR curves to choose the optimal decision threshold in binary classification tasks. In the final experiment, we demonstrate RNNs and their suitability for handling sequential data, which in our case were clinical notes, while also exploring the use of transfer learning. Overall, the results of the experiments emphasize the role of machine learning in providing accurate and efficient solutions in the medical field.

Keywords:

Machine learning; medical data; classification; dimensionality reduction; neural networks; CNN; RNN; CLAHE; ROC curve; PR curve; transfer learning

Table of Contents

List of Figures

List of Tables

Abbreviations

TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
ROC	Receiver Operating Characteristic
PR	Precision-Recall
AUC	Area Under Curve
SVM	Support Vector Machine
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
LDA	Linear Discriminant Analysis
ICA	Independent Component Analysis
NMF	Non-Negative Matrix Decomposition
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CLAHE	Contrast Limited Adaptive Histogram Equalization
LSTM	Long Short-Term Memory

1. Introduction

Machine Learning is a branch of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed. It involves using algorithms to analyse data, find patterns and make predictions or decisions based on it. Most machine learning techniques fall into 2 categories, supervised and unsupervised learning. Supervised learning techniques use labelled datasets to train algorithms for prediction or classification. Unsupervised learning techniques analyse unlabelled data to identify patterns and relationships within them. For techniques that combine labelled and unlabelled data for training, the term semi-supervised learning is used.

The history of machine learning roots back to decades of human desire and effort to study human cognitive processes and subsequently, the desire to develop algorithms that mimic human thought processes. Throughout the years, many advancements in machine learning methodologies have been made. Those advancements combined with the exponential growth in computing power and data availability have led to the rise of machine learning over the years. In modern days, machine learning's applications have expanded across nearly every sector and it has transformed industries by enabling data-driven decision-making and automation.

1.1 Subject of diploma thesis

In this diploma thesis we cover various machine learning techniques. Aside from explaining them, we specifically focus on their application on medical data. The medical field has always been a strong supporter of cutting-edge technologies, so it is not a surprise that machine learning has found several applications in this industry. Medical data come in many forms including imaging, laboratory results and other information that can be collected during patient care. Regardless of type, finding patterns and insights manually can be difficult sometimes, especially when it comes to electronic health records which contain massive amounts of data. The huge variety of machine learning methods means that they can help us analyse any kind of medical data, as long as the appropriate method is chosen, and use them for diagnostic or prognostic purposes. Furthermore, predictive analytics based on machine learning can assist to save time and money invested in clinical trials while also delivering accurate results. All in all, machine learning has revolutionised the medical field by enabling more efficient, more accurate and personalised healthcare.

1.1.1 Contribution

1. We covered classification in machine learning and explained various evaluation metrics

2. We covered four classification algorithms: Support Vector Machine, Random Forest, Logistic Regression and K-Nearest Neighbors.

3. We covered Gradient Descent as a part of many machine learning methods including some classification algorithms.

4. We explained what dimensionality reduction is and its importance in machine learning

5. We covered five dimensionality reduction methods: Principal Component Analysis, Singular Value Decomposition, Linear Discriminant analysis, Non-Negative Matrix Factorization and Independent Component Analysis.

6. We explain eigenvalues and eigenvectors, which are a core part of most dimensionality reduction methods.

7. We covered neural networks in general

8. We expanded on CNNs and RNNs, and their applications on medical data

9. We experimented with the aforementioned machine learning methods on medical data

1.2 Organization of the volume

Chapter 2 focuses on classification and a model's evaluation metrics.

Chapter 3 covers the classification algorithms and Gradient Descent.

Chapter 4 covers the dimensionality reduction methods, as well as eigenvalues and eigenvectors.

Chapter 5 contains all information related to neural networks.

Finally, chapter 6 covers the experiments that were conducted.

2 Classification

2.1 Introduction

As mentioned before, machine learning tasks fall under two categories, regression and classification. Most machine learning tasks on medical data belong predominantly in the classification category, since very often the goal is to make a diagnosis.

In classification tasks, aside from accuracy, which is the most basic metric, a model's performance can be measured by some additional metrics. The right choice depends on whether the problem is binary or multi-class and whether the dataset is imbalanced (different number of samples for each class) or not.

2.2 Accuracy

Accuracy is the easiest metric to understand and, while it is often useful, it can be misleading for imbalanced datasets.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ predictions}$$

2.3 Confusion matrix and derived metrics

2.3.1 Confusion matrix

A very useful tool for classification tasks is the Confusion Matrix. It summarizes the number of correct and incorrect predictions made by the model for each class. In essence, it helps understand where the model is getting "confused" by showing how often instances of one class are misclassified as another.

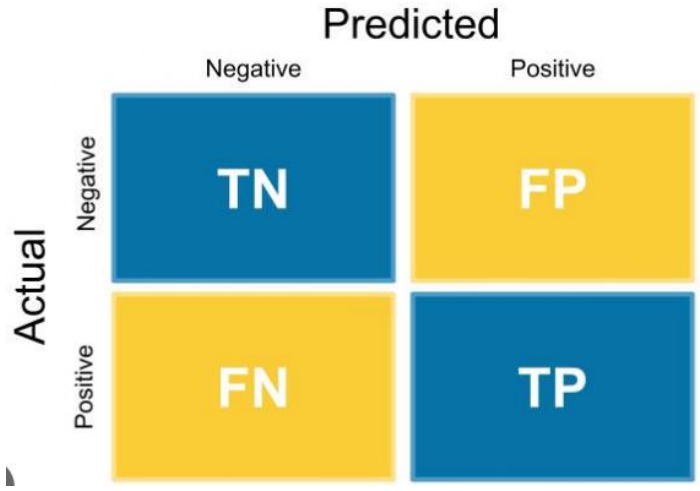


Figure 2.1: Example of a confusion matrix.[1]

In binary classification, there are four possible scenarios. A prediction can be: True positive (TP), False positive (FP), True negative (TN) and False negative (FN). Later on, we will explain the metrics that can be derived from the confusion matrix.

2.3.2 Precision

[1] Precision measures the accuracy of positive predictions. A high precision score indicates that when the model predicts a positive, it is likely to be correct. It is particularly important when predicting a false positive comes at a high cost.

$$Precision = \frac{TP}{TP + FP}$$

2.3.3 Recall

Recall (or sensitivity) measures the ability of the model to find all the relevant instances. A high recall score indicates that the model is good at identifying most of the actual positive cases. It is especially important when the cost of a false negative is high. In medical diagnosis, recall is an exceptionally important metric, because a false negative essentially means missing a disease, an outcome that can have serious ramifications.

[1] https://www.jcchouinard.com/confusion-matrix-in-scikit-learn/

$$Recall = \frac{TP}{TP + FN}$$

2.3.4 F1-score

The F1-score is the harmonic mean of precision and recall and provides a balance between the two metrics.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.4 Probability-based metrics

Aside from the metrics derived from the confusion matrix, there are some probability-based metrics that can be used to measure a model's performance.

2.4.1 Receiver Operating Characteristic Curve (ROC Curve)

[2] The ROC curve is a graph used to check how well a binary classification model works. It plots the True Positive Rate (Recall) vs the False Positive Rate at different decision thresholds for turning probabilities into class predictions. The metric that is derived from that graph is the AUC, which essentially represents the probability that the model, if given a randomly chosen positive and negative example, will rank the positive higher than the negative. The AUC ranges from 0 to 1. An AUC close to 1 means that the model effectively distinguishes between the two classes, while an AUC close to 0 means that the model struggles to differentiate between the two classes. A model that guesses randomly would score an AUC of 0.5. AUC-ROC is effective when the dataset is balanced, and false positives and false negatives are of similar importance.

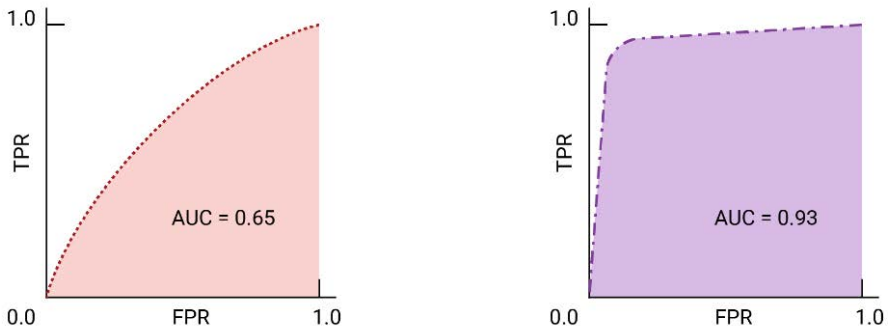


Figure 2.2: Examples of ROC curves.[2]

2.4.2 Precision-Recall (PR) Curve:

A PR curve is also a graph that illustrates the performance of a binary classification model. It plots the recall vs the precision, at different decision thresholds. Similarly to ROC curves, a model's performance is evaluated by calculating the AUC. The key difference between the two curves is that the PR curve is affected less by the dominance of the majority class. For that reason, PR curves are preferred over ROC curves when handling imbalanced datasets.

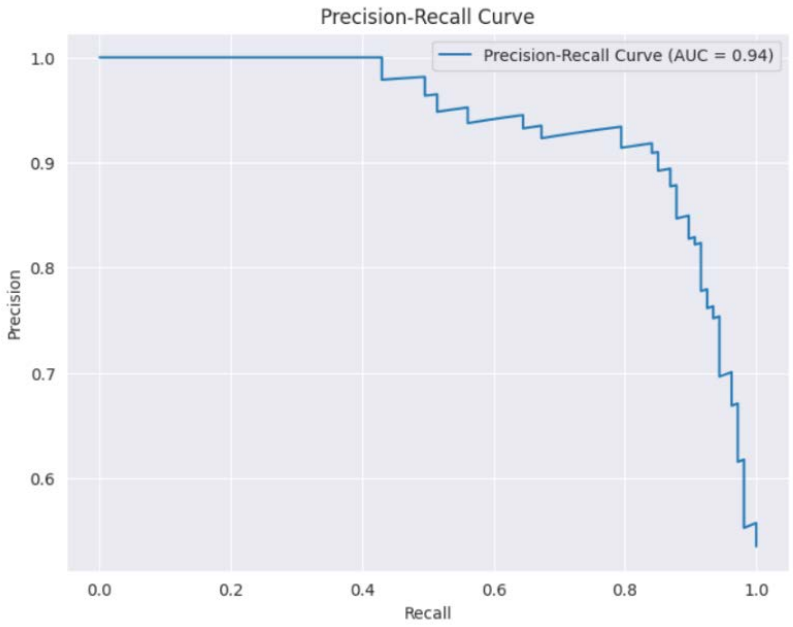


Figure 2.3: Example of a PR curve.[3]

[2] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
[3] https://www.geeksforgeeks.org/machine-learning/precision-recall-curve-ml/

2.4.3 Decision thresholds

Since ROC curves and PR curves evaluate a model at multiple decision thresholds, they can both be used to find the best one. While the default 0.5 is a common choice, it may not be optimal, especially when handling imbalanced datasets.

Using the ROC curve: To find the threshold that balances sensitivity and specificity the best, we use Youden's J statistic.

$$J = TPR - FPR$$

The threshold that is chosen is the one that maximizes *J.*

Using the PR curve: The most common way to choose a decision threshold using the PR curve, is to choose the one that maximizes the F1-score, to achieve a good balance between precision and recall. Alternatively, we can optimize for high recall or high precision, depending on our needs.

2.5 Multi-class classification

For multi-class classification problems, the way that the metrics are calculated is a bit different (except for accuracy).

2.5.1 Confusion matrix and the derived metrics

Instead of a 2x2, the confusion matrix is an MxM (M is the number of classes). The correct predictions are in the diagonal, and the rest are the mistakes.

The derived metrics (Precision, Recall, F1-score) are first calculated per-class. To do that, we need to compute, for each class, the correctly predicted instances of the class (true positives), the incorrectly predicted instances of the class (false positives) and the missed instances of the class (false negatives). After calculating all of them the results are combined by using one of the three averaging strategies: Macro averaging, micro averaging and weighted averaging.

Macro averaging computes the metric for each class independently and then takes their mean. This strategy is best when we care about each class equally.

$$Precision(macro) = \frac{1}{N}\sum_{i=1}^{N}\frac{TP_i}{TP_i + FP_i}$$

Micro averaging first aggregates TP, FP and FN across all classes and then calculates the metric. In imbalanced datasets, the metric is dominated by the majority class.

$$Precision(micro) = \frac{\sum_{i=1}^{N} TP_i}{\sum_{i=1}^{N}(TP_i + FP_i)}$$

Weighted averaging computes the metric for each class, just like macro averaging, but it adds a weight depending on the number of instances in each class. It is a balanced approach that takes class prevalence into account.

$$Precision(weighted) = \sum_{i=1}^{N}\frac{n_i}{n} * \frac{TP_i}{TP_i + FP_i}$$

Where n_i is the number of instances in class i and n is the total number of instances.

2.5.2 ROC and PR curves

[3] Similarly to the metrics derived from the confusion matrix, the ROC and PR curves also require the use averaging strategies to be calculated.

1) One-vs-Rest

In One-vs-Rest each class is treated as ''positive'' and all other as ''negative''. Then the macro average AUC (average AUC across all classes) or the micro average AUC (global AUC) is calculated to get the ROC/PR curves per class.

2) One-vs-One

In One-vs-One the ROC/PR curve for each combination of classes is plotted. The AUC is calculated in the same ways as in One-vs-Rest.

The One-vs-Rest strategy is usually preferred due to the complexity of the One-vs-One strategy.

3 Classification algorithms

3.1 K-Nearest Neighbours

3.1.1 Introduction

[4] The k-nearest neighbours (k-NN) algorithm is a classic non-parametric supervised learning method originally introduced by Evelyn Fix and Joseph Hodges in 1951 and later formalized by Thomas Cover. While it is most commonly applied to classification problems, the algorithm can also be extended to handle regression tasks.

The underlying principle of k-NN is straightforward: for a given target data point, the algorithm identifies the “k” closest points in the training dataset, typically using a distance metric such as Euclidean, Manhattan, or Minkowski distance. Once the neighbours are identified, predictions are made either by majority voting (in classification) or by averaging the numerical values of the neighbours (in regression). This intuitive approach allows k-NN to capture local structure in the data, making it useful in a wide variety of tasks.

Unlike many machine learning models, k-NN does not involve a formal training phase. Instead, it stores the entire dataset and defers computation until prediction time. For this reason, it is often described as a “lazy learner” algorithm. This simplicity makes k-NN very easy to implement and understand, but it also introduces scalability challenges. As the dataset grows larger, both storage requirements and computation time increase significantly, since the algorithm must calculate distances to potentially every point in the dataset for each new prediction.

Despite these limitations, k-NN remains a popular and widely used technique due to its interpretability, flexibility, and ability to perform well in low-dimensional problems. It has found applications in pattern recognition, recommendation systems, medical diagnosis, and other domains where the relationships between data points are highly local. Recent adaptations, such as weighted k-NN and approximate nearest neighbour search, aim to address some of the algorithm’s weaknesses while preserving its simplicity.

3.1.2 Algorithm explanation

i) Select a value for K.

K represents the number of nearest neighbours that will be taken into account to make a prediction. The optimal value for K largely depends on the input data. For data with more outliers or noise, the algorithm will likely perform better with higher value for K. Specifically for classification, it is also recommended to use an odd number for K, to avoid ties during the voting phase (step iv).

ii) Calculating the distance.

To measure the similarity between target and training data points, we calculate the distance between them. Euclidean distance is usually used for that task.

$$Euclidean\ distance(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

iii) Finding the nearest neighbours.

The nearest neighbours are the k data points with the smallest distances to the target point.

iv) Making a prediction.

For classification: Classify the target data point into the category that the majority of its nearest neighbours belong to.

For regression: The predicted value of the target data point is the average of the values of its nearest neighbours.

3.2 Random Forest

3.2.1 Introduction

[5] Random Forest is a widely used machine learning algorithm introduced and trademarked by Leo Breiman and Adele Cutler. It is versatile, capable of handling both classification and regression tasks, and has become one of the most popular ensemble learning techniques in modern data science.

The method builds upon decision trees, which are simple yet powerful models that split data into branches based on feature values to make predictions. While decision trees are intuitive and easy to interpret, they are also prone to certain drawbacks, particularly high variance and a tendency to overfit training data. Random Forest addresses these issues by constructing a collection, or "forest," of decision trees and then aggregating their results to produce a final prediction. For classification tasks, the algorithm typically takes the majority vote across trees, while in regression tasks, it averages their outputs.

Random Forest belongs to the broader category of ensemble methods, which rely on combining multiple models to improve predictive performance and robustness. In the case of Random Forest, two main techniques are used to increase diversity among the trees: bootstrap aggregating (bagging), where each tree is trained on a random sample of the data with replacement, and feature randomness, where each split considers only a random subset of features rather than the entire set. These strategies reduce correlation between trees and contribute to the model's strong generalization ability.

One of the key advantages of Random Forest is its resilience against overfitting and noise compared to single decision trees, as well as its ability to handle large datasets and high-dimensional feature spaces. It can also provide estimates of feature importance, making it valuable for exploratory data analysis and interpretability in some contexts. However, the algorithm can be computationally expensive for very large datasets and may lose some of the transparency of individual decision trees due to its ensemble nature.

Random Forest has been applied across diverse fields, including bioinformatics, medical diagnosis, finance, and image recognition, due to its balance of accuracy, robustness, and ease of use. Its success has also inspired further developments in ensemble learning, such as Gradient Boosting and Extreme Gradient Boosting (XGBoost).

3.2.2 Explanation

Since the Random Forest Classifier is an ensemble of Decision Tree Classifiers, firstly we need to understand how a Decision Tree Classifier works. A decision tree is built by choosing a condition that splits our training data into two subsets. We continue splitting the data until our decision tree consists of leaf nodes that contain data from only one class (pure leaf nodes).

The most used metric to choose the best condition to split a node is Information Gain. (Metrics such as Gini impurity and mean square error can also be used to evaluate the quality of the split)

$$Information\ Gain = E(parent) - w_1 E(child\ 1) - w_2 E(child\ 2)$$

The weights w_1, w_2 are equal to the number of data points in a child node divided by the number of data points in the parent node and E represents the entropy (measure of data impurity) of each node.

$$Entropy = -\sum_{i=1}^{N} p_i \log_2 p_i \qquad (p_i\ is\ the\ probability\ of\ class\ i)$$

The algorithm of Random Forest Classifier is as follows:

i) Bootstrap sampling

Random samples are picked (with replacement) to train each tree. This is the most well-known ensemble method known as bagging or bootstrap aggregation which was introduced by Leo Breiman in 1996.

ii) Random feature selection

A random subset of the features is chosen to train each tree.

iii) Building the decision trees

All the decision trees are built by following the 2 previous steps. Selecting the number of trees can be challenging. It is important to use a lot of trees to get the best prediction accuracy but using too many can hurt the computational efficiency of the algorithm.

iv) Majority voting

Each tree makes its own prediction. We classify the target data point into the category that the majority of the decision trees predict.

Note: As mentioned above, Random Forest can also be used for regression problems. A similar procedure is followed, with the main difference being that the final result is the average of the results of the decision trees.

3.3 Logistic Regression

3.3.1 Introduction

[6] Linear regression is one of the most fundamental supervised learning algorithms and one of the earliest statistical methods applied to predictive modeling. The technique aims to identify the linear combination of independent variables (features) that best explains the relationship with a dependent variable (target). By fitting a straight line, or more generally, a hyperplane in higher dimensions, linear regression models can be used to make predictions on new, unseen data.

The central assumption behind linear regression is that the relationship between predictors and the target variable is linear. This makes the algorithm simple, interpretable, and effective in many practical scenarios. Linear regression is particularly well-suited for tasks where the target variable is continuous, such as predicting house prices, stock values, or patient vital signs. However, when the target variable is categorical, a linear function is not the most appropriate model, as it cannot naturally constrain predictions to discrete class memberships.

To address this limitation, Logistic Regression was developed. Instead of fitting a straight line, it applies the logistic (or sigmoid) function, which outputs values between 0

and 1, representing probabilities. This S-shaped curve allows the model to estimate the likelihood that a given data point belongs to a particular class. A threshold value, typically set at 0.5, is then used to assign class labels. For example, if the probability of a patient having a disease is predicted to be greater than 0.5, the model classifies the patient as "positive," otherwise as "negative."

The strength of Logistic Regression lies in its simplicity, interpretability, and ability to provide probabilistic outputs, which are particularly valuable in risk assessment and medical decision-making. It also extends naturally to multi-class problems through techniques such as one-vs-rest (OvR) or softmax regression. However, Logistic Regression assumes linear separability in the feature space, which limits its performance in more complex, non-linear problems—cases where methods like Support Vector Machines (SVM) or Neural Networks may be more effective.

Both Linear and Logistic Regression remain essential building blocks in machine learning, serving not only as standalone methods but also as conceptual foundations for more advanced models. Their interpretability and efficiency ensure their continued use in diverse fields, from economics and healthcare to natural language processing and bioinformatics.

3.3.2 Mathematical explanation

Sigmoid function:

$$s(z) = \frac{1}{1 + e^{-z}}$$

In logistic regression $z = w^T x + b$

Where: x is the feature matrix, w^T are the weights/coefficients and b is the bias.

The sigmoid function represents the predicted probability of a data point belonging to the class labelled as 1. The goal is to find w and b that minimizes the following loss function:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log_2(p_i) + (1 - y_i)\log_2(1 - p_i)$$

Where y_i is the class label (0 or 1) and p_i the predicted probability.

After initializing w and b, we use gradient descent to find their optimal values and by extension the sigmoid function that fits our data the best.

3.4 Support Vector Machine (SVM)

3.4.1 Introduction

[7] Support Vector Machines (SVMs) are a relatively modern learning method primarily used for binary classification tasks. The fundamental idea is to identify a hyperplane that best separates data points in a *d*-dimensional space into two categories. In practice, data is often not linearly separable, so SVMs make use of kernel functions to project the data into a higher-dimensional feature space where separation becomes feasible.

Ordinarily, working directly in such high-dimensional spaces would typically lead to excessive computation and risk of overfitting. However, SVMs employ the "kernel trick," which allows computations to rely only on dot products in the transformed space, avoiding the explicit construction of the higher-dimensional representation. This makes the method both efficient and practical.

Another strength of SVMs lies in their strong theoretical grounding. Unlike neural networks, SVMs have a calculable Vapnik-Chervonenkis (VC) dimension, which provides a measure of their ability to generalize to unseen data. As a result, SVMs are both conceptually simple and robust in application.

Beyond classification, SVMs have also been extended to regression problems, where the goal is to predict continuous values rather than categorical outcomes.

3.4.2 Mathematical explanation

The objective is to minimize $||w||^2$ (weight vector) subject to:

$$y_i(w * x_i + b) \geq 1$$

Where b is the bias, x_i the training data and y_i the class labels (-1,1).

The minimization of $||w||^2$ is achieved by using Gradient Descent. For largest datasets (like the one used in the experiments), Stochastic Gradient Descent (SGD) is suggested to speed

up the process. Unlike Gradient Descent which uses the entire dataset to calculate the gradient, SGD uses a small batch of data points to calculate the gradient.

3.5 Important method used by classifiers: Gradient Descent

3.5.1 Introduction

While gradient descent is not a classification algorithm, as mentioned above, is it part of both Logistic regression and SVM. So, to fully understand them, we also have to be familiar we gradient descent.

Gradient descent is one of the most widely used optimization algorithms in machine learning and plays a central role in training modern models, particularly neural networks. At its core, gradient descent is an iterative method for minimizing an objective (or cost) function, denoted as $J(\theta)$, where θ represents the model's parameters. The algorithm works by calculating the gradient of the cost function with respect to the parameters, $\nabla J(\theta)$, and then updating the parameters in the opposite direction of this gradient. This process gradually reduces the value of the cost function, moving the parameters closer to a local (or sometimes global) minimum.

The size of each update step is controlled by a hyperparameter called the learning rate, η. Choosing the right learning rate is critical: if it is too large, the algorithm may overshoot the minimum and fail to converge; if it is too small, convergence will be very slow, requiring many iterations.

The origins of gradient descent can be traced back to the 19th century. Augustin-Louis Cauchy first proposed the method in 1847, followed independently by Jacques Hadamard in 1907. The convergence properties of gradient-based methods for non-linear optimization were later studied by Haskell Curry in 1944. Since then, gradient descent has become a cornerstone in numerical optimization, growing in importance with the rise of computational methods and machine learning.

Today, gradient descent is by far the most common optimization technique for training neural networks and other machine learning models. It forms the basis for a variety of advanced optimization strategies, such as Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, and adaptive methods like Adam, RMSProp, and Adagrad. These variants improve the basic algorithm by introducing randomness, adaptive learning rates, or momentum to accelerate convergence and improve performance on large, high-dimensional datasets.

The enduring appeal of gradient descent lies in its simplicity, scalability, and effectiveness. Whether in classical regression problems or in deep learning frameworks powering image recognition, natural language processing, and medical diagnosis, gradient descent continues to be a key driver of modern machine learning success.

Gradient descent update rule:

$$\theta = \theta - \eta * \nabla J(\theta)$$

3.5.2 Variations

Different implementations of gradient descent are used depending on the size of the dataset and computational requirements:

1) Batch gradient descent

Batch gradient descent uses the entire training dataset to compute the gradient of the cost function at each iteration. It guarantees a stable convergence path but can be very slow and computationally expensive for large datasets. That means that it is best suited for smaller datasets or when computational resources are not a limiting factor.

2) Stochastic gradient descent

Stochastic gradient descent (SGD) updates parameters using the gradient from a single randomly chosen data point at each iteration. It introduces randomness into the optimization process, which allows the algorithm to escape shallow local minima but also makes convergence noisier. Its main benefit is that it is computationally efficient, making it ideal for very large datasets.

3) Mini-Batch gradient descent

The main idea behind mini-batch gradient descent is a compromise between batch and stochastic methods. The gradient is computed on small subsets (mini-batches) of the dataset at each iteration. It balances computational efficiency with convergence stability and it is the most common approach in deep learning frameworks, as it leverages vectorized operations and parallel processing on GPUs.

These variants can also be combined with optimization enhancements such as momentum (to smooth noisy updates), learning rate scheduling (to adjust step size over time), and adaptive optimizers (e.g., Adam, RMSProp), which automatically tune learning rates for each parameter.

4 Dimensionality Reduction Methods

4.1 Introduction

Dimensionality reduction refers to representing a given dataset using a lower number of features (dimensions) while still capturing the original data's meaningful properties. Dimensionality reduction is achieved by either feature extraction or feature selection. Feature extraction methods transform the original features into a new set of features, while feature selection methods choose a subset of the original features. Although dimensionality reduction methods differ in operation, they mostly rely on matrix factorization which is the process of decomposing a matrix into a product of lower-dimensional matrices with the intend to reveal hidden patterns and relationships within the data.

In machine learning, dimensionality reduction methods are used during the preprocessing of the dataset, as it plays a major role in lifting the "Curse of Dimensionality". The Curse of Dimensionality describes the trade-off between increasing the number of model dimensions and losing generalizability. As more input variables are added, the dimensional space of the model grows. If the number of data points does not increase proportionally, the dataset becomes sparse. With greater sparsity, data points appear less alike, making it harder for predictive models to detect meaningful patterns.

To compensate, models may end up overfitting the training data, which reduces their ability to generalize to new data. High dimensionality also complicates model interpretation, often introducing multicollinearity—where some variables are highly correlated or redundant.

One way to mitigate sparsity is by collecting more data, but the amount of data required grows exponentially with each added dimension, which is often impractical. For this reason, dimensionality reduction techniques are used to simplify datasets, enhance interpretability, and improve model performance.

4.2 Important linear algebra background: Eigenvalues and eigenvectors

Eigenvalues and eigenvectors are fundamental concepts in linear algebra that describe how a linear transformation affects certain vectors. An eigenvector is a non-zero vector that, when transformed by a matrix, only changes in scale, not direction. The corresponding eigenvalue is the factor by which the eigenvector is scaled. In essence, eigenvalues and eigenvectors provide a way to understand the fundamental behaviour of

linear transformations and systems, revealing their characteristic properties and behaviours.

Eigenvalues and eigenvectors have applications in various fields, including physics, engineering, computer graphics. The reason we focus on them is because they are also applied in data science, specifically in dimensionality reduction techniques such as Principal Component Analysis

Definition: For a square matrix *A*, if there exists a non-zero vector *u* and a scalar λ such that:

$$Au = \lambda u$$

The vector *u* is an eigenvector of the matrix *A* and the scalar λ is the corresponding eigenvalue.

The simplest way of calculating them is by solving the following equation, which is commonly referred to as the characteristic equation:

$$(A - \lambda I)u = 0$$

Which reduces to solving:

$$\det(A - \lambda I) = 0$$

After solving for λ, by finding the roots of the polynomial, λ is plugged into the equation to find the corresponding eigenvalue.

However, for large matrices or complex coefficients, this method is impractical. So, in practice, iterative algorithms are used. Common algorithms are the QR algorithm, Power iteration, Inverse iteration, and the Jacobi method.

4.3 Principal Component Analysis (PCA)

4.3.1 Introduction

[8], [9] Principal Component Analysis (PCA) is one of the earliest and most widely recognized techniques in multivariate data analysis. The concept was first introduced by Pearson in 1901 and later expanded by Hotelling in 1933. Although initially underutilized, PCA gained traction with the rise of electronic computing and is now a standard feature in nearly all statistical software.

PCA is a mathematical method that transforms a set of potentially correlated variables into a smaller set of uncorrelated variables known as principal components. While its roots are in multivariate statistics, PCA is regarded as a fundamental achievement in applied linear algebra and has become an essential tool for analyzing large, complex data sets. Beyond its role in exploratory data analysis, PCA is also used in tasks such as noise reduction, blind source separation, and data compression.

At its core, PCA applies a vector space transformation to reduce the dimensionality of data. Through projection, datasets with many variables can often be described effectively with only a few principal components. The main idea is to simplify data with numerous interrelated variables while preserving as much of the original variation as possible. This is done by constructing new variables, the principal components, which are uncorrelated and ordered so that the first few capture most of the variability in the dataset. Mathematically, computing principal components involves solving an eigenvalue-eigenvector problem for a symmetric, positive semi-definite matrix.

Today, PCA finds broad application across fields such as image and audio compression, medical research, and finance. Moreover, it has evolved into multiple variants, including Kernel PCA and Incremental PCA, which extend its utility to more complex data structures.

4.3.2 Algorithm explanation

Step 1: Standardization of the data.

The first step is to standardize the data, by subtracting the mean ($\bar{X}$) from the original data (X).

$$X_S = X - \bar{X}$$

Step 2: Find the covariance matrix (C) of the standardized data.

$$C = {X_S}^T X_S$$

Step 3: Compute the eigenvalues λ_i and eigenvectors v_i of the covariance matrix.

$$Cv_i = \lambda_i v_i \text{ (i = 1, 2, 3, …n n=number of features)}$$

Step 4: Sort the eigenvalues in descending order.

The eigenvalue of each eigenvector shows its ability to capture data variance. After sorting, we select a subset of the eigenvectors with the highest eigenvalues. This subset of eigenvectors is used to form the matrix V, commonly referred to as the loadings matrix. Usually, we select the subset of eigenvectors based on their explained variance ratio.

$$explained\ variance\ ratio = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j}$$

We then choose the number of eigenvectors for which the cumulative explained variance exceeds a threshold (e.g. 0.95).

Step 5: Transformation

With everything else out of the way, all that remains is to project the data into the principal component space.

$$P = XV$$

4.4 Singular Value Decomposition (SVD)

4.4.1 Introduction

[10] The Singular Value Decomposition (SVD) is a powerful matrix factorization method with roots dating back over a century. It was independently introduced by several mathematicians, who arrived at similar results using different approaches. Although the concept was known in the late 19th and early 20th centuries, it was not until the 1960s that efficient computational algorithms were developed, making SVD a practical and indispensable tool in numerical linear algebra.

Some of the classical mathematical applications of SVD include computing the Moore-Penrose pseudoinverse, performing low-rank matrix approximations, solving total least squares problems, and analysing the rank, range, and null space of a matrix. Beyond pure mathematics, SVD has found widespread use in science, engineering, and statistics. For example, it plays a central role in signal processing, process control, natural language processing (via latent semantic analysis), recommendation systems (such as those used by streaming services), and image compression. Its ability to extract dominant patterns in data while reducing noise makes it one of the most versatile decomposition techniques.

The enduring utility of SVD lies in its ability to balance mathematical rigor with broad practical applicability. Whether in classical numerical methods or modern machine learning applications, SVD remains a cornerstone of linear algebra and data-driven analysis.

4.4.2 Algorithm explanation

The SVD factorizes a m x n matrix X into an orthogonal matrix U (m x m), a diagonal matrix Σ (m x n) and an orthogonal matrix V (n x n).

$$X = U\Sigma V^T$$

Unlike the eigenvalue decomposition of a matrix, which is limited to square matrices, SVD can be used on any matrix.

Step 1: Find matrices XX^T and X^TX.

Step 2: Compute the eigenvalues and eigenvectors of both matrices.

Step 3: Find matrices U and V.

Matrix U contains the eigenvectors of XX^T (left singular vectors) and matrix V contains the eigenvectors of X^TX (right singular vectors). Because matrices U and V are supposed to be orthogonal, the eigenvectors are divided by their magnitude.

Step 4: Find matrix Σ.

Matrix Σ contains the root squares of the eigenvalues (singular values) along its diagonal and zeros in the remaining positions.

It is important to note that the standard SVD stops here. Although SVD has a lot of applications, in order to use it for dimensionality reduction an extra step, commonly referred to as “Truncation” needs to be taken. In Truncated SVD the eigenvalues and their corresponding eigenvectors are sorted in descending order. The new matrix $\tilde{\Sigma}$ contains the

largest singular values and the other singular values are replaced by zero. The new lower rank matrix $\widetilde{M}$ is the product of matrices U, $\tilde{\Sigma}$ and V^T.

$$\widetilde{M} = U\,\tilde{\Sigma}\,V^T$$

4.5 Linear Discriminant Analysis (LDA)

4.5.1 Introduction

[11] LDA is a statistical method rooted in Fisher's linear discriminant, first developed by Sir Ronald Fisher in the 1930s and later extended to multi-class problems by C. R. Rao. The purpose of LDA is to find a linear combination of input features that best distinguishes between two or more classes of labelled data. The fact that it uses labelled data means that it is supervised learning technique, unlike other dimensionality reduction techniques.

LDA operates as both a classification technique and a dimensionality reduction method. It projects data into a lower-dimensional space that enhances class separability. This is achieved by maximizing the distance between the mean values of different classes while minimizing the scatter of data points within each class. In effect, LDA identifies the directions (discriminant axes) in which the classes are most distinct.

The mathematical foundation of LDA involves constructing between-class and within-class scatter matrices and solving a generalized eigenvalue problem. The resulting discriminant vectors define the projection space, with the number of useful discriminants limited to one fewer than the number of classes.

Because of its balance of simplicity, interpretability, and effectiveness, LDA has been applied across diverse fields such as face and speech recognition, bioinformatics, and medical diagnostics. Its dual ability to classify and reduce dimensionality makes it particularly valuable for high-dimensional datasets requiring efficient class separation.

LDA is often compared to PCA, since they are both very potent dimensionality reduction methods. Their main difference is that PCA focuses on maximizing the variance in the data, while LDA focuses on maximizing the separability between different. Furthermore, as mentioned previously, LDA is a supervised learning method as opposed to PCA which is an unsupervised learning method.

4.5.2 Algorithm Explanation

Step 1: Separate samples based on class.

Step 2: Find the mean of the matrix for each class.

Step 3: Find the covariance matrix (S) for each class.

$$S_i = \frac{1}{N-1}\sum_{j=1}^{N}(x - \mu_i)(x - \mu_i)^T$$

Where μ_i the mean of the i-th class and N the number of features.

Step 4: Add the covariance matrices.

The sum of the covariance matrices S_w in called the within-class scatter matrix.

Step 5: Calculate the between-class scatter matrix (S_B).

- For 2 classes:

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$$

- For multiple classes:

$$S_B = \frac{1}{C}\sum_{i=1}^{C}(\mu_i - \mu)(\mu_i - \mu)^T$$

Where C is the number of classes and μ the mean of the means of each class.

Step 6: Solve the following eigenvalue-eigenvector problem.

$${S_W}^{-1}S_B w = \lambda w$$

Step 7: Transformation

Project the data into the optimal projection space w^T (matrix containing the eigenvectors with the highest corresponding eigenvalues).

$$X_{new} = w^T X$$

4.6 Independent Component Analysis (ICA)

4.6.1 Introduction

[12] ICA was first introduced in the 1980s by J. Hérault, C. Jutten, and B. Ans, who proposed an iterative real-time algorithm for signal separation. At the time, the method lacked a strong theoretical foundation, and the proposed algorithm proved inapplicable in many situations. As a result, ICA remained relatively obscure until 1994, when the term "Independent Component Analysis" was formally introduced and presented as a novel computational approach.

The primary objective of ICA is to recover hidden source signals from observed mixtures of data. These observed signals can take various forms, including images, financial time series, audio recordings, or biomedical measurements. By attempting to disentangle the underlying sources, ICA has found extensive applications in fields such as electroencephalogram (EEG) and functional MRI analysis, gene expression studies, audio signal separation, and other areas where mixed signals must be decomposed into their original components.

Conceptually, ICA can be viewed as an extension of Principal Component Analysis (PCA), though the two techniques differ fundamentally in their statistical criteria. PCA focuses on second-order statistics by maximizing variance and producing uncorrelated components. ICA, on the other hand, goes beyond this by exploiting higher-order statistics, such as kurtosis, to identify statistically independent components. This means that while PCA guarantees uncorrelated features, ICA seeks to separate signals into truly independent sources, offering a deeper level of data decomposition.

Through its ability to extract meaningful and independent features, ICA has become a valuable tool in data analysis, signal processing, and machine learning, complementing other dimensionality reduction and decomposition methods.

4.6.2 Algorithm explanation

The concept of ICA is, given a data matrix X, to find a matrix W such that

$$S = WX$$

where the components of S are as statistically independent as possible.

Step 1: Centering

Subtract the mean ($\bar{X}$) from matrix X.

Step 2: Whitening

After calculating the covariance matrix, find its eigenvalues and eigenvectors. The whitening is done as shown in the following equation:

$$X_W = \lambda^{-1/2} U^T X$$

Where λ is a diagonal matrix containing the eigenvalues of the covariance matrix and U the eigenvector matrix.

Step 3: Choose a non-linear function g(x).

Functions like tanh(x), $x * \exp(-\frac{u^2}{2})$, x^3 are usually chosen to introduce non-Gaussianity. To maximize non-Gaussianity metrics like kurtosis or negentropy are used.

Step 4: Find matrix W by using the following algorithm.

i) Initialize W randomly.

ii) Update W by using the following update rule:

$$w_{new} = E[x * g(w^T x)] - E[g'(w^T x] * w$$

iii) Orthogonalize W.

$$w = w - \sum_{i=1}^{k-1} (w^T w_i)\, w_i$$

iv) Normalize W.

$$w = \frac{w_{new}}{\|w_{new}\|}$$

Repeat those steps until convergence.

Step 5: Finally, compute the independent components matrix S.

$$S = WX$$

4.7 Non-Negative Matrix Factorization (NMF)

4.7.1 Introduction

[13] Positive Matrix Factorization (PMF) was introduced by Pentti Paatero and Unto Tapper in 1994, and later, Daniel D. Lee and H. Sebastian Seung extended this idea to develop Nonnegative Matrix Factorization (NMF). NMF is a powerful matrix decomposition method designed for multivariate data, operating under the constraint that all components remain nonnegative. Given a nonnegative matrix *X* of size n × m and a target rank *k*, NMF seeks an approximation X ≈ WH, where *W* (n × k) represents the basis or feature matrix, and *H* (k ×m) serves as the coefficient matrix. This ensures that the original matrix *X* is expressed as a product of two smaller nonnegative matrices.

NMF is widely used in clustering because its formulation naturally aligns with clustering principles. By applying NMF, the high-dimensional clustering problem on *X* s transformed into a reduced-dimensional clustering problem on *H.* Standard clustering techniques, such as k-means or spectral clustering, can then be applied. Over time, many improved variations of NMF have been proposed to achieve better results across diverse application areas.

What sets NMF apart from traditional decomposition methods are two main characteristics. First, the nonnegativity constraint makes the results easier to interpret, as the components can be viewed as genuine underlying parts of the original data. Second, NMF often produces sparse factorizations, meaning that many entries in the basis or coefficient matrices are zero, leading to compact, part-based representations. This property emphasizes local features and enhances interpretability. Moreover, NMF is particularly well-suited for large-scale datasets due to its straightforward computations, fast execution, efficient memory use, and clear interpretability compared to conventional approaches.

NMF has found applications across a wide spectrum of fields. Beyond modern disciplines like computer science, telecommunications, image compression, and remote

sensing, it is increasingly applied in traditional sciences such as biology, medicine, chemistry, physics, and psychology, where it provides valuable insights into complex data.

4.7.2 Mathematical Explanation

It is important to note that there are several different algorithms for NMF. The following algorithm is known as the Multiplicative Update Algorithm.

Step 1: Initialize matrices W and H

The elements of the matrices are usually initialized randomly. Their dimensions depend on the desired dimensions of the new data matrix.

Step 2: The NMF problem can be formulated as:

$$\min\|X - WH\|_F, \;\; subject\ to\ W_{ij} \geq 0, X_{ij} \geq 0 \quad \forall i, j.$$

To solve the NMF problem, the multiplicative update rule is applied, which is formulated as:

$$H_{next} = H * \frac{W^T X}{W^T W H}$$

$$W_{next} = W * \frac{X\, H^T}{W H H^T}$$

where H_{next} and W_{next} are the updated matrices H and W in each iteration.

The multiplicative update rule is applied until convergence or until the maximum number of iterations is reached.

Step 3: The new data matrix ($\tilde{X}$) is the product of matrices W and H.

$$\tilde{X} = WH$$

5 Neural Networks

5.1 Introduction

[14] An artificial neural network is a computational model inspired by how biological brains process information. The neural network of the human brain consists of many neurons which process and transmit information through connections called synapses. The brain strengthens or weakens those connections based on learning and experience, thus making humans able to perform complex tasks and to process information with very little effort. Similarly, artificial neural networks are built following a similar structure. Furthermore, they imitate how a biological neural network functions, by adjusting weights during training (analogous to a human's experience) to improve performance.

In 1943 Warren McCulloch and Walter Pitts built the first mathematical model of a neuron by introducing the idea that neuron can be modelled as simple binary threshold units. This model paved the way for further research in the field of artificial intelligence. Based on previous ideas, in 1958 psychologist Frank Rosenblatt built the Mark I Perceptron, the first implementation of a feed-forward artificial neural network, which was able to classify input patterns, mimicking basic brain function. In later years many breakthroughs were made in deep learning. Most notably, in 1967, Shun'ichi Amari published the first deep learning multilayer perceptron trained by stochastic gradient descent, which due to subsequent developments in hardware and hyperparameter tuning, was the currently dominant training technique. Nowadays, artificial neural networks are considered a core part of artificial intelligence and machine learning.

5.2 Architecture of a neural network

A neural network is composed of three types of layers, the input layer, hidden layers and the output layer.

Input layer: The input layer is the first layer of a neural network and it simply receives the data. Each neuron of the input layer corresponds to a feature of the data.

Hidden layers: These layers perform most of the computations. They are responsible for processing the data through weighted connections and applying nonlinear transformations. A neural network contains one or multiple hidden layers.

Output layer: The output layer is the final layer a neural network and it produces the final result. The format of the results varies depending on the specific task (classification or regression).

5.3 How neural networks work?

5.3.1 Forward Propagation

Firstly, the data is input into the network and passes through the network in the forward direction, from the input layer through the hidden layers and finally to the output layer. During this process, each neuron in a layer receives an input and multiplies it by a weight. These products are summed up and then a bias is added to the sum as shown below.

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_3 x_3 + b$$

Where w represents the weights, x the inputs and b the bias.

Afterwards, the result of the linear transformation (z) passes through an activation function. An activation function is a crucial function that introduces non-linearity into the system, enabling the network to identify more complex patterns. The most used activation functions are Rectified Linear Unit or “ReLU” (and its variations Leaky ReLU and PReLU), sigmoid and tanh. If the output of the activation function exceeds a given threshold it ‘’fires’’ (or activates) the neuron, passing the data to the next layer. This output becomes the input of the next neuron.

5.3.2 Loss calculation and backpropagation

After forward propagation, the network evaluates its performance. To do that, the network first calculates the difference between the actual output and the predicted output by using a loss function. Commonly used loss functions are mean squared error for regression tasks and cross-entropy loss for classification tasks. Then, the weights and

biases are updated using an optimization algorithm like gradient descent or Adam (adaptive moment estimation).

5.3.3 Iteration

This process is repeated many times over the dataset. In each iteration, the network adapts its parameters to better approximate the relationships within the dataset. This results in a reduction of the loss in each iteration, thereby improving the network's ability to make accurate predictions.

5.4 Hyperparameters of a neural network

Hyperparameters are the configurable settings of a neural network that are not learned from data but must be set before training begins. They greatly influence the performance, speed and accuracy of the model. The most important hyperparameters which are the activation function, the optimization algorithm and the loss function were mentioned before, so we will now focus on the other ones.

5.4.1 Learning rate

The learning rate controls how much the weights are updated during backpropagation. Although when using Adam optimizer the learning rate is constantly adjusted, gradient descent requires us to set the learning rate to a specific value. Choosing the learning rate can be crucial because a too small learning rate results in slow convergence while a too high learning can lead to overshooting. A typical range is between 0.001 to 0.1.

5.4.2 Weight initialization

Weight initialization determines how the starting weights of a neural network are set before training begins. It is important because poor weight initialization can lead to the weights shrinking or growing too much (vanishing or exploding gradients) when they are updated. The simplest way to initialize the weights is to follow a normal or a uniform distribution. Depending on the activation function, methods like Xavier and HE initialization can be applied.

Xavier (or Glorot) initialization aims to keep the variance stable across layer. This is achieved by drawing weights from a normal distribution with a specific variance, depending on the number of input and output neurons of each layer. Xavier initialization is better when the using sigmoid or tanh activation functions.

HE initialization, also known as Kaiming initialization, is well-suited for networks using ReLU (or its variations) activation functions. It helps address the "dying ReLU" problem, where the neurons become inactive due to zero or very small activations. It solves this problem by initializing the weights with a variance that is inversely proportional to the number of inputs/connections to a neuron.

5.4.3 Epochs

The number of epochs determines how many times the entire dataset will pass through the network. When the number is too low, we face the problem of underfitting, while a too high number results is overfitting.

5.4.4 Batch size

Batch size refers to the number of samples processed before updating the model. On the one hand, a larger batch size results in a faster training time. On the other hand, a smaller batch size results in better generalization. Common values are 32, 64, and 128.

5.4.5 Depth and width

The depth and the width define the model’s capacity. The depth of a neural network is the number of layers it contains and its width is the number of neurons in each layer. While more layers result in more abstraction, it also leads to a heavier computational load and the risk of overfitting.

5.4.6 Regularization

Regularization adds a penalty to the loss function to discourage overfitting by keeping weights small or sparse. L1 regularization, also known as LASSO, encourages sparsity by setting some weights to zero, effectively performing feature selection. L2

regularization, also known as Ridge, shrinks the weights smoothly towards zero, leading to smaller and more evenly distributed weights. There is also a technique called ElasticNet, a combination of L1 and L2 regularization, which balances sparsity and smoothness.

5.4.7 Dropout rate

Dropout is also considered a regularization technique, that prevents overfitting by randomly "dropping out" a subset of the neurons during each training step. Specifically, during training each neuron's output is kept with a probability *p* (commonly around 0.8). During testing, all neurons are used but their output is scaled down by *p*.

5.5 Types of neural networks

Neural networks come in various types. The most common ones are Feedforward Neural Networks (FNNs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs). Each architecture is designed for specific tasks and data structures. They vary in how data flows through the network, the types of layers they utilize, and their ability to handle sequential data or spatial information.

5.5.1 Convolutional neural networks (CNNs)

A CNN is a type of neural network designed to process grid-like data, such as images, where pixels are arranged in a 2D grid. Unlike traditional fully connected networks, CNNs take advantage of the spatial structure in data. Applications of CNNs include image classification, object detection, image segmentation, facial recognition and medical imaging, which is reason we focus on them. The following information is an overview of their unique architecture.

Input layer: The unique characteristic of a CNN's input layer is that it usually receives image data.

Convolutional layer: The convolutional layer is the core layer of a CNN's architecture and it is the key difference between them and other types of neural networks. It applies a set of filters that scan across the input image that detect patterns like edges, textures or shapes. Afterwards, each filter produces a feature map.

Activation function: An activation function is applied to the feature maps to introduce non-linearity. The most common activation function in CNNs is ReLU.

Pooling layer:

The pooling Layer reduces the spatial dimensions (height and width) of the feature maps while keeping the most important information. There are three types of pooling, max pooling, average pooling and global pooling.

Max pooling operates by taking the maximum value in each region (typically a 2x2 grid each time), effectively keeping the most prominent feature in each region. Similarly, average pooling takes the average of all values in each region, making it a smoother but generally less effective pooling technique than max pooling for image tasks. Finally, global pooling reduces the feature map into a single value by taking the average or the maximum value (global max pooling and global average pooling). It is often used before the output layer to make the output one-dimensional, without flattening.

The benefit of pooling is that it makes the model more computationally efficient and, most notably, translation invariant which is crucial when handling images.

Flattening: Flattening occurs after several convolutional and pooling layers and it reduces the feature maps into a 1D vector. Unlike global pooling, flattening is used when preserving all spatial information is necessary even though it makes the model less efficient. Another difference is that global pooling feeds its output directly into the output layer, whereas flattening requires a fully connected layer before the output layer.

Fully connected (dense) layer: Although it is a standard neural network layer, its purpose in CNNs is to make the final predictions, using the flattened data.

Output layer: The output layer uses a function to convert the output into a probability distribution over classes. For binary classification tasks the sigmoid function is used, while for multi-class classification tasks the sofmax function is used.

The Softmax function:

Given a vector $z = [z_1, z_2, \dots, z_n]$, which is called a vector of logits, the output for class *i* is:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

5.5.2 CNNs for medical imaging classification

5.5.2.1 Introduction

[15], [16] Medical imaging is a cornerstone of modern clinical diagnosis and an indispensable tool in life sciences research. Common modalities include X-rays, Magnetic Resonance Imaging (MRI), Computed Tomography (CT), and Positron Emission Tomography (PET), each providing critical insights into the structure and function of the human body. However, the vast size and complexity of these imaging datasets make manual analysis challenging, creating a pressing need for automated disease detection methods.

With the rapid progress of computer vision, deep learning has emerged as a transformative approach in this area. Researchers have increasingly focused on leveraging deep learning techniques to build computer-aided diagnostic (CAD) systems capable of analysing medical image data with high accuracy and efficiency.

Medical image classification is particularly vital in clinical practice, as it supports disease diagnosis, monitoring, and assessment of severity. Historically, classification relied on handcrafted features such as color, texture, and shape, which were extracted and selected before applying traditional machine learning algorithms, including Support Vector Machines (SVM) and Logistic Regression (LR). While these approaches provided useful results, their performance eventually plateaued due to limitations in feature representation and scalability.

The advent of deep learning, and more specifically Convolutional Neural Networks (CNNs), has dramatically advanced image-based classification tasks. CNNs, with their ability to automatically learn hierarchical feature representations from raw image data, have consistently demonstrated superior accuracy compared to traditional methods. Over recent years, the continuous evolution of CNN architectures has yielded impressive performance not only in natural image classification but also in medical imaging applications, where they significantly enhance the accuracy and reliability of diagnostic systems.

5.5.2.2 Medical image preprocessing

Medical images differ from natural images in structure, as they often contain irrelevant regions, background artifacts, and varying levels of noise. To address these issues, preprocessing is an essential step prior to analysis. Its purpose is to improve image quality, emphasize clinically important features, reduce unwanted noise, and ensure compatibility with the input requirements of computational models. By doing so,

preprocessing enhances the effectiveness of feature extraction, improves generalization across datasets, and increases the robustness of diagnostic models.

A variety of preprocessing techniques are commonly applied, including normalization, histogram equalization, denoising, resampling, local region highlighting, and segmentation. The choice of method depends on the imaging modality, since each type of medical image has unique structural and statistical characteristics.

In the case of X-rays, which are the input data later in the experiments, images are typically grayscale and exhibit a wide contrast range. Preprocessing X-rays aims to increase the visibility of the region of interest, suppress noise, and sharpen structural details to improve interpretability. Common approaches include contrast enhancement methods such as histogram equalization, as well as filtering and sharpening techniques, all of which contribute to clearer and more diagnostically useful images.

5.5.2.3 Contrast Limited Adaptive Histogram Equalization (CLAHE)

CLAHE is an image enhancement technique used to improve the contrast in images. It is a variation of the Adaptive Histogram Equalization (AHE) method. Unlike regular histogram equalization, which enhances contrast by redistributing the intensity of the entire image, AHE applies histogram equalization locally, on small regions called tiles. This adaptation makes it better at handling varying illumination across an image. Despite that, AHE tends to over-amplify noise in relatively homogeneous regions of an image. To tackle this problem, CLAHE expands further on regular histogram equalization by also introducing a contrast limiting step, in which it clips the histogram at a predefined threshold (called the clip limit) and redistributes the clipped pixels to enhance contrast more gently.

CLAHE is particularly useful for improving the visibility of features in low-light or unevenly illuminated photos. These properties make CLAHE a very potent image enhancement method, which is used not only in medical imaging but also remote sensing and night vision.

5.5.3 Recurrent Neural Networks (RNN)

A RNN is a type of neural network designed to handle sequential data, which are data where order matters, like text, audio or video frames. The thing that sets RNNs apart from other neural networks is that they feed information back into the network at each step. In other words, RNNs have a memory of past inputs that influences future outputs.

A RNNs architecture is similar to a typical feedforward network's architecture. The key difference is that the hidden layer consists of Recurrent Units, the core building blocks of an RNN. They are the components that process sequential data step by step, carrying information forward through a hidden state. Essentially, they have the ability to memorize information, allowing the network to detect dependencies from previous steps. The hidden state is represented mathematically as follows:

$$h_t = f(W_{xh} * x_t + W_{hh} * h_{t-1} + b_h)$$

Where:

- x_t: input at time t
- h_t: hidden state (memory) at time t
- W: weight matrices, shared across time steps
- b: bias
- f: activation function

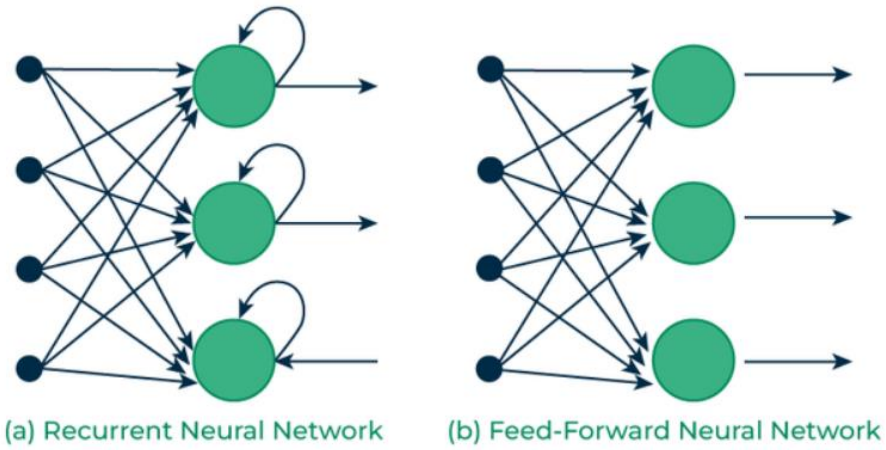


Figure 5.1: RNNs vs typical feed-forward NNs[4]

5.5.3.1 Types of RNNs

[17] RNNs come in several types depending on how the input, hidden states, and outputs are structured or modified. Thera are four main types: One-to-One, One-to-Many, Many-to-One and Many-to-Many

[4] https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/

One-to-One take a single input and produces a single output. It is pretty much a standard feedforward network and is often used in image classification. One-to-Many also takes a single input but its output is a sequence. Its most popular application is image captioning. Many-to-One is the complete opposite; it takes a sequential input and produces a single output. It is ideal for tasks like sentiment analysis. Finally, we have Many-to-Many. Both its input and output are sequential. This type of RNN is most commonly used in translation and speech recognition.

5.5.3.2 Variants of RNNs

Aside from the original form, the "vanilla" RNN, there are also several variations of RNNs. These are LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit) and bidirectional RNNs. LSTM introduces memory cells and each one has three gates; an input gate, a forget gate and an output gate. Collectively, they regulate how much information is added, discarded and finally output at each step. Their main benefit is that handle long-term dependencies well. GRU is a simplified version of LSTM that combines the input and forget gate into one. Compared to LSTM, GRU is more computationally efficient, while having a similar performance. Finally, bidirectional RNNs process the sequence in both directions, an attribute that is useful when the entire sequence is available.

5.5.4 RNNs in the medical field

[18], [19] Among the various deep learning architectures that the healthcare industry has adopted, RNNs have gained prominence due to their ability to process sequential data. In the medical field, sequential data come in the form of time-series data, images and patient health records.

Time series data is a sequence of data points collected and recorded over successive, equally spaced intervals in time, capturing how a phenomenon changes over time. Medical time series data include ECG and EEG signals and a patient's vital signs. In those cases, RNNs can help by detecting early signs of patient deterioration. When it comes to medical imaging, while CNNs dominate static image analysis (X-rays, MRIs, CT scans), RNNs are used when images form a sequence or temporal progression. Some examples are endoscopies and ultrasounds, where RNNs can help detect anomalies in live video frames. Finally, we have patient data. Clinical documents such as discharge summaries and progress notes are sources full of medically relevant information like medication prescriptions and diagnosis information.

Traditionally, all these data required manual analysis. This process can be not only time consuming, but also prone to human error. These challenges can be addressed with the use of deep learning methods and RNNs are uniquely suited to handle these kinds of data.

6 Experiments

6.1 Experiment 1: Combining dimensionality reduction techniques and classification algorithms.

[20] In the first experiment we used a dataset provided by CuMiDa. The dataset is about breast cancer and contains information about 35981 genes (our features) from 289 samples. This information was collected with the use of DNA microarrays, a modern technique used to collect information about multiple genes at once.

6.1.1 Machine learning for cancer diagnosis

[21], [22] One of the major research areas in the medical field is cancer research, due to its rapid growth in incidence and higher mortality rate compared to other diseases. Traditionally, cancer diagnosis involves various tests and procedures to determine not only if cancer is present but also its type and extend. Common diagnostic methods include physical exams, laboratory tests, imaging, endoscopy, and biopsy and a combination of those is often needed for a definitive diagnosis. Despite their extensive use, these conventional diagnostic methods have several limitations in their diagnostic ability.

To better understand and address the challenges of cancer diagnosis, researchers have increasingly turned to systematic approaches based on global gene expression analysis. Since gene expression levels reflect the activity of genes, they hold crucial information not only for disease prevention and treatment but also for understanding biological evolution and guiding drug discovery.

The introduction of microarray technology marked a turning point in this field, as it enabled the simultaneous measurement of thousands of genes in a single experiment. This technological advancement provided unprecedented opportunities for studying cancer at the molecular level and spurred significant progress in diagnostic research using gene expression data.

However, the vast amount of information contained in gene expression datasets makes them too complex to be interpreted and analysed manually. As a result, machine learning methods have become essential tools for extracting meaningful patterns from these high-dimensional data. Although the application of machine learning to gene expression-based cancer diagnosis is still a relatively young research area, early results have demonstrated strong potential, offering promising directions for improving accuracy and efficiency in clinical diagnostics.

6.1.2 Biological background information

To understand gene expression, it is first important to review some basics of molecular biology. Cells are the essential functional units of all living organisms, and their activities are governed by instructions stored in DNA. DNA is a double-stranded polymer built from four basic molecular components known as nucleotides. Each nucleotide is composed of a phosphate group, a deoxyribose sugar, and one of four nitrogen bases: adenine (A), guanine (G), cytosine (C), or thymine (T).

The specific order of these bases, referred to as the DNA sequence (for example, ATGAGTACTGA), encodes the instructions required to form an organism with its distinct traits. Because of this, DNA is often described as the blueprint of life, containing the complete set of information needed to build and sustain organisms ranging from bacteria to humans. The entire DNA content of an organism is known as its genome, which consists of numerous segments called genes. Each gene represents a defined portion of the genome with a particular function.

DNA not only serves as a template for its own replication but also provides the instructions for producing ribonucleic acid (RNA). Various forms of RNA are synthesized from the genome, the major types being messenger RNA (mRNA), transfer RNA (tRNA), and ribosomal RNA (rRNA). Gene expression refers to the process by which the information encoded in DNA is converted into functional products. This occurs in two main steps: (i) transcription, where a DNA sequence is copied into mRNA, and (ii) translation, where mRNA directs the assembly of amino acids into proteins that carry out cellular roles.

The level of gene expression reflects the number of RNA transcripts produced from a gene and is generally linked to the quantity of its corresponding protein. Distinct patterns of gene expression are observed under different biological conditions, such as embryonic development, cellular differentiation, and normal tissue responses. Importantly, abnormal gene expression is associated with disease. For example, cancers often arise from mutations in genes that regulate processes like the cell cycle, programmed cell death, and genome stability, leading to altered patterns of expression.

6.1.3 The DNA microarray technology

Research into the use of microarrays, a relatively recent technology that enables parallel measurement of thousands of genome-wide expression values, has demonstrated their effectiveness in identifying gene functions and improving cancer diagnosis. The central strength of this method lies in its ability to capture gene expression patterns on a large scale, providing insights that were previously unattainable with traditional molecular biology techniques.

Microarray technology builds upon classical probe hybridization methods but greatly expands their scope by allowing the simultaneous recording of expression levels from thousands of genes. The process involves immobilizing thousands of DNA or RNA sequences (probes) onto a solid surface, creating a microarray “chip.” When a biological sample (DNA or RNA extracted from tissues or cells) is applied to the chip, complementary sequences in the sample hybridize, or bind, to the probes. Typically, fluorescent labelling is used to visualize this binding, enabling researchers to determine which sequences are present and to what extent they are expressed.

Each data point produced from a microarray experiment corresponds to the ratio of expression levels for a single gene under two different conditions: one representing the experimental setting and the other serving as a reference. With *m* genes on a chip, this results in m expression level ratios, where the numerator reflects the gene’s expression under the condition of interest and the denominator represents its expression under the reference condition.

By providing such high-throughput measurements, microarrays have become a powerful tool for understanding gene activity, uncovering disease-related biomarkers, and advancing personalized approaches in cancer diagnosis and treatment.

6.1.4 Dimensionality reduction in cancer diagnosis

Despite the successful application of microarrays to efficiently extract gene expression data, using them for cancer classification can be challenging due to their unique nature.

The first challenge is that gene expression data contain information about thousands of genes. That combined with the fact that most available gene expression data have a small sample size, makes the task of classification algorithms very difficult. The high dimension of the data makes overfitting a major concern while also hurting the computational efficiency of the classification algorithms.

The second challenge comes from the inherent presence of noise in the dataset. Apart from any noise that is introduced during the various stages of collecting the data (technical noise), most genes aren’t related to cancer (biological noise).

The last challenge is related to the domain of their application. While the high accuracy of the classification is important, it isn’t the only goal we want to achieve. Biologists are interested in any additional information revealed in the process that can help them gain a better understanding of gene functions. This information is important because it can also be used in future biological studies.

All these challenges can be overcome with the use of dimensionality reduction methods. Projecting the data to a lower-dimensional space helps the classification algorithms by not only minimizing the risk of overfitting but by also reducing their computational speed. Furthermore, dimensionality reduction methods can reveal which features are most important for the classification task and specifically in our case which genes are relevant for the cancer diagnosis and which are not.

6.1.5 Initial testing and results

The workflow of the experiment is as follows:

i) Train/Test split (75%/25% split).

ii) Apply a dimensionality reduction method (PCA, LDA, ICA, Truncated SVD, NMF).

We tested keeping 5,10 and 20 components for all methods except LDA, since LDA keeps only one component when applied to a binary classification problem.

iii) Train a classifier (Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Random Forest).

iv) Evaluate the model's ability to make predictions on the test set.

The following tables show the accuracy of each combination of dimensionality reduction method and classifier.

	LR	KNN	SVM	RF
PCA	83.56%	84.93%	83.56%	86.30%
ICA	82.19%	86.30%	82.19%	86.30%
TRUNCATED SVD	83.56%	84.93%	73.97%	87.67%
NMF	82.19%	80.82%	80.82%	89.04%

Table 6.1: Accuracies for 5 components kept.

	LR	KNN	SVM	RF
PCA	90.41%	86.30%	87.67%	87.67%
ICA	87.67%	82.19%	80.82%	87.67%
TRUNCATED SVD	89.04%	84.93%	80.82%	86.30%
NMF	87.67%	84.93%	84.93%	89.04%

Table 6.2: Accuracies for 10 components kept.

	LR	KNN	SVM	RF
PCA	91.78%	86.30%	84.93%	87.67%
ICA	91.78%	79.45%	84.93%	89.04%
TRUNCATED SVD	89.04%	86.30%	89.04%	86.30%
NMF	86.30%	84.93%	86.30%	86.30%

Table 6.1: Accuracies for 20 components kept.

	LR	KNN	SVM	RF
LDA	90.41%	90.41%	87.67%	82.19%

Table 6.4: Accuracies for LDA (1 component).

The last table shows the best performing algorithms.

DR Method	Classifier	Components	Accuracy
PCA	LR	20	91.78%
ICA	LR	20	91.78%
PCA	LR	10	90.41%
LDA	LR	1	90.41%
LDA	KNN	1	90.41%

Table 6.2: Algorithms with the highest accuracy.

Although the difference in accuracy is relatively small (but significant nonetheless), we can conclude that PCA stands out as the overall best performing dimensionality reduction method and Logistic Regression stands out as the best classification algorithm.

6.1.6 Further analysis

Since PCA is the best performing dimensionality reduction method, we used it to see what additional information about the data it can provide.

6.1.6.1 Visualization

Although keeping only 2 components is not enough when the goal is to achieve the best classification accuracy, it makes it easier for us to visualize the data. The plot below shows our data when projected to a 2-dimensional space.

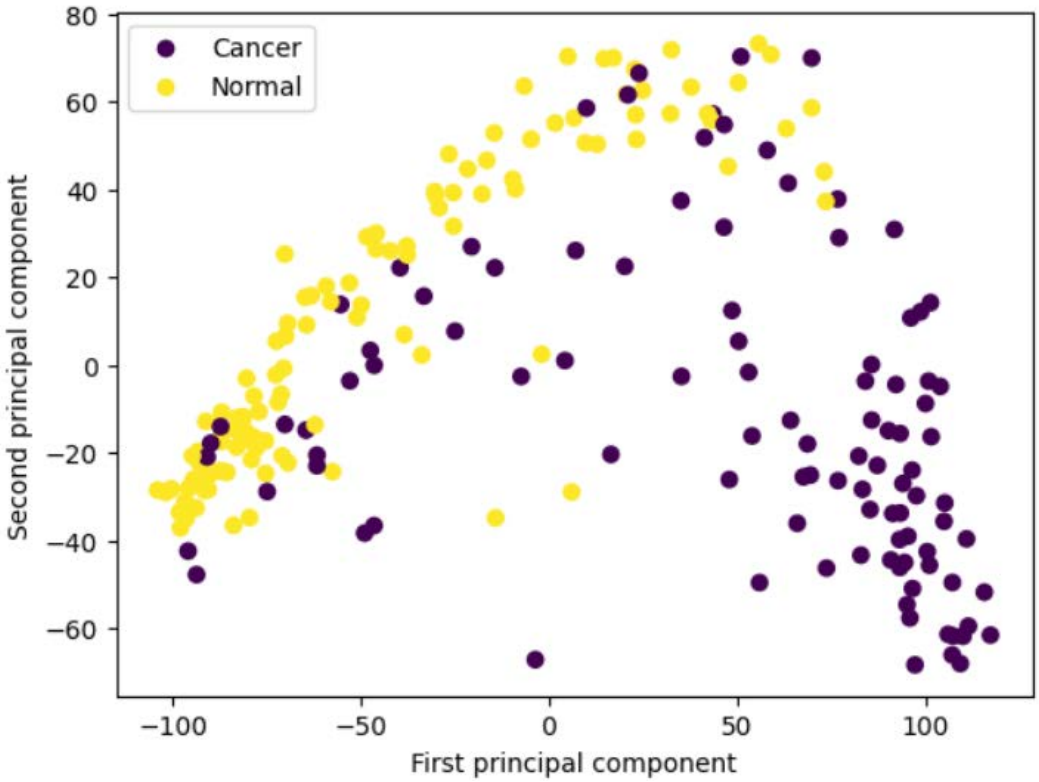


Figure 6.1: Plot of the data when 2 components are kept.

6.1.6.2 Explained Variance Ratio

Essentially, the Explained Variance Ratio tells us how much information (or variance) is retained when reducing the number of dimensions in our data. The plots below show the explained variance ratio and the cumulative explained variance ratio by the number of components kept.

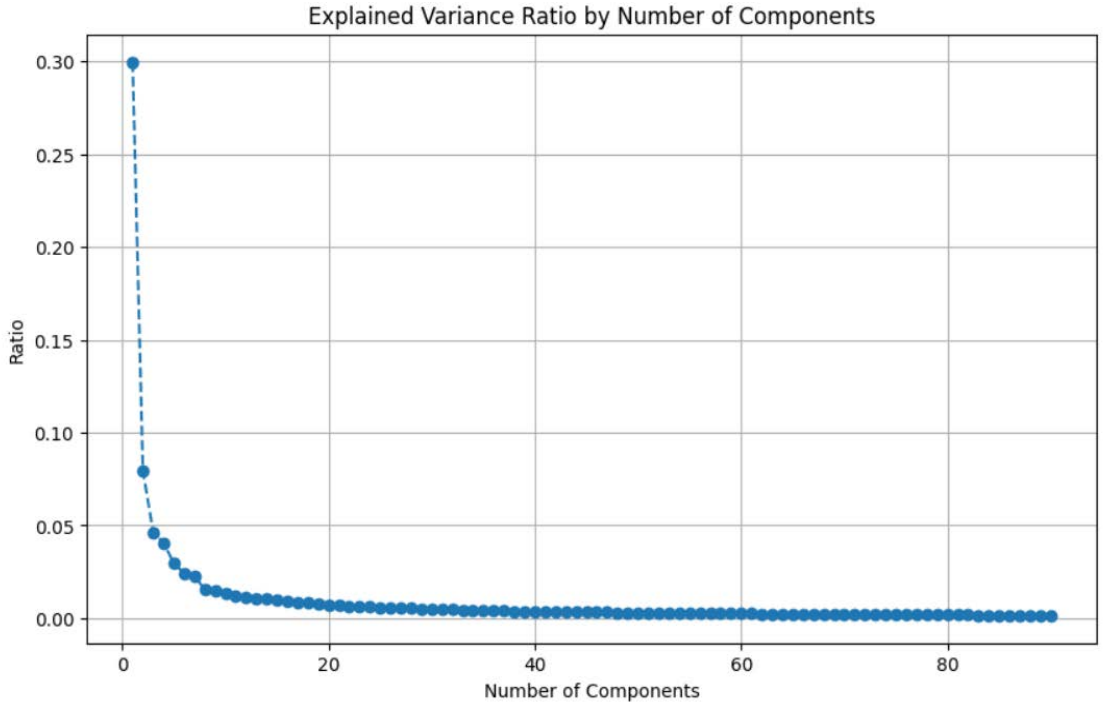


Figure 6.2: Plot of the explained variance ratio.

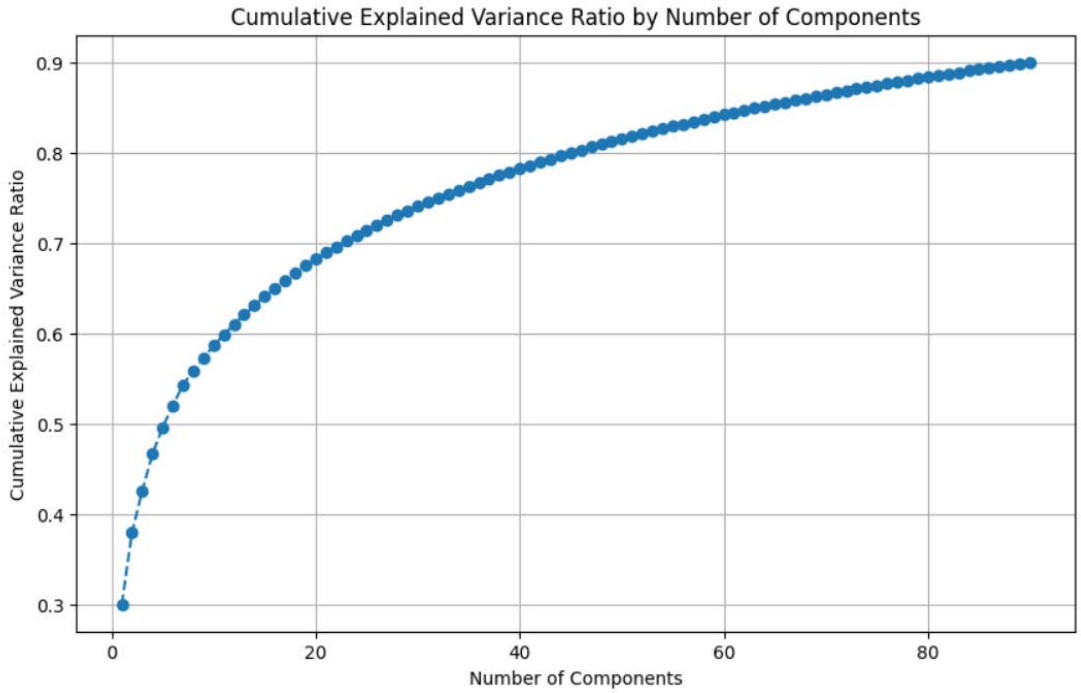


Figure 6.1: Plot of the cumulative explained variance ratio.

We can see the first few components (especially the first 2) preserve most of the variance of our data. In fact, the first 88 components capture 90% of the total variance of the original dataset. Considering that the original dataset contains information about 35981 genes, we can conclude that most of that information isn't relevant for the classification

task. That can be attributed to the fact that, as mentioned previously, most of the genes are probably not related to cancer.

6.1.6.3 Feature Selection

The principal components are a linear combination of all the features of our dataset. By looking at the absolute values of the loadings matrix, we can see the magnitude of influence that each feature has in the transformation of the original dataset into the principal components. In other words, we can get an idea about which genes are related to cancer the most.

Although this information will be different depending on the number of components kept, we will try keeping 5, 10 and 20 components like before. The following table shows the top 10 most important features (genes) for each case, to get a general idea.

	5 components	10 components	20 components
1	**NM_002652**	**NM_002652**	**NM_002652**
2	NM_152997	NM_002411	NM_058173
3	ENST00000390539	NM_058173	lincRNA:chr6:114722782-114743483_F
4	NM_000526	lincRNA:chr6:114722782-114743483_F	lincRNA:chrX:107264119-107280819_F
5	NM_002411	NM_006551	lincRNA:chr2:200984605-201020245_F
6	NM_006551	lincRNA:chr4:23771777-23778877_R	lincRNA:chr12:96953169-96992769_F
7	NM_006552	lincRNA:chrX:107264119-107280819_F	NM_002411
8	A_33_P3251412	lincRNA:chr12:96953169-96992769_F	lincRNA:chr4:23771777-23778877_R
9	ENST00000498435	NM_006552	NM_182908
10	ENST00000492167	lincRNA:chr2:200984605-201020245_F	lincRNA:chr9:2739400-2746875_F

Table 6.6: Most relevant features.

In general, this procedure can help us gain a better understanding of our dataset. In our case, this information is difficult to interpret because our features are reference sequences that correspond to a specific gene. Nevertheless, we dug deeper to understand what NM_002652 means, since it stands out as the most important gene according to our analysis.

NM_002652 corresponds to the PIP gene, which encodes the Prolactin-inducible protein. The protein is not normally expressed in breast tissue and its presence is a sign of apocrine metaplasia. Although it may be benign, it indicates an increased chance for the development of breast cancer in the future.

With that information in mind, we can be little more confident that with our analysis, we managed to correctly identify the genes that are related to breast cancer the most.

6.1.7 Conclusion

Summing up, the unique nature of gene expression data, and other similar medical data, makes handling them quite difficult. Fortunately, dimensionality reduction methods can alleviate this problem by not only helping classification algorithms fit to the dataset properly but also providing additional information about the data, a benefit that is really appreciated in the medical field. The variety in classification algorithms and dimensionality reduction methods can make choosing the right ones a difficult task. The correct choice depends heavily on the nature of data. Judging from this experiment, when handling gene expression data, it seems like Linear Regression and Principal Component Analysis have the edge over other methods.

6.2 Experiment 2: Classifying image data using CNNs.

6.2.1 Information about the dataset

[23] The dataset consists of chest X-rays, which are used to diagnose Pneumonia. It contains 5,863 X-ray images classified as “NORMAL” or “PNEUMONIA”. While it was already split into training, validation and test sets, the proportions were not ideal. So, we moved the images around to achieve an 80%/10%/10% split.

6.2.2 Preprocessing

Before being plugged into the CNN, the images underwent preprocessing using the CLAHE method. Below is an example of an image before and after CLAHE.

BEFORE AFTER

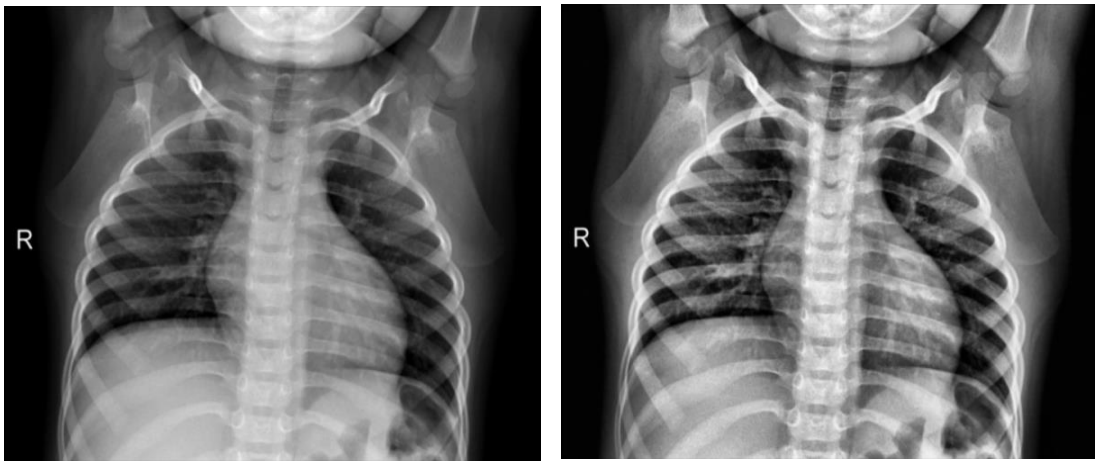


Figure 6.4: Images before and after CLAHE

6.2.3 The CNN's architecture

The CNN consists of:

- Input layer
- 3 Convolutional layers each followed by a Max Pooling layer
- 1 Dense layer
- Output layer

The CNN was built using Keras, an open-source library that provides a Python interface for artificial neural networks.

6.2.4 Hyperparameter choice

Choosing the hyperparameters of a neural network can be a difficult task. Fortunately, there is powerful tool in Keras' arsenal called KerasTuner. KerasTuner is hyperparameter optimization framework that solves the pain points of hyperparameter search. So, we used KerasTuner to find the most optimal:

- Number of units in the 3 convolutional layers and the dense layer (the number that defines the dimensionality of each layer)
- Activation function (ReLU, sigmoid, tanh)
- Learning rate
- Dropout rate

The optimizer and the loss function remain constant throughout the trials.

Optimizer: Adam

Loss function: Focal loss

Focal loss expands from binary cross-entropy loss by adding a modulation factor. This modulation reduces the influence of samples classified with high probability and increases the influence of samples classified with low probability. Essentially, focal loss is a modified version of cross-entropy designed to address problems with imbalance datasets (which is the reason it was preferred in our case).

$$FL(p, y) = -a\,(1 - p_t)^{\gamma}\log(p_t)$$

Where:

- y is the true label (0 or 1)
- p the predicted probability for class 1
- p_t = p if y = 1 and p_t = 1 – p if y = 0
- a a weight that balances the importance of positive vs negative samples
- γ>1 the focusing parameter (if γ=0 focal loss reduces to binary cross-entropy)

Activation function: Although the activation function for the convolutional layers and the dense layer changes, the activation function of the output layer remains the same. For binary classification problems like ours, the sigmoid function is preferred.

KerasTuner provides several algorithms for the search. The random search algorithm (20 trials, 5 epochs each) was chosen for efficiency.

6.2.5 Finding the best configuration

Below there is a summary of the trials:

Trial	conv1 units	conv2 units	conv3 units	dense units	dropout rate	learning rate	activation	accuracy	loss	precision	recall
0	128	128	448	224	0.2	0.0001	ReLU	0.7928	0.0275	0.817	0.9225
1	**32**	**224**	**288**	**160**	**0.4**	**0.0076**	**ReLU**	**0.815**	**0.0205**	**0.8258**	**0.946**
2	128	160	416	96	0	0.0001	tanh	0.7294	0.0296	0.7294	1
3	128	64	480	192	0.1	0.0001	ReLU	0.803	0.0219	0.8193	0.9366
4	128	64	416	224	0.2	0.0001	sigmoid	0.2705	0.7527	0	0
5	64	96	320	96	0	0.0017	sigmoid	0.7294	0.0295	0.7294	1
6	128	192	128	160	0	0.0073	ReLU	0.7294	0.0297	0.7294	1
7	96	192	320	160	0.4	0.0018	sigmoid	0.7294	0.0296	0.7294	1
8	32	224	288	192	0	0.0007	ReLU	0.791	0.0198	0.8318	0.8943
9	96	128	416	224	0	0.0037	tanh	0.7294	0.0311	0.7294	1
10	32	128	512	192	0.2	0.0044	ReLU	0.791	0.0247	0.7934	0.9647
11	96	224	448	160	0.4	0.0072	tanh	0.7294	0.0306	0.7294	1
12	32	96	192	64	0.1	0.009	ReLU	0.7294	0.0295	0.7294	1
13	32	96	352	192	0.1	0.005	tanh	0.7294	0.0337	0.7294	1
14	32	160	512	192	0	0.003	sigmoid	0.7294	0.0299	0.7294	1
15	32	192	224	256	0.4	0.0012	sigmoid	0.7294	0.0296	0.7294	1
16	32	96	512	128	0,2	0.0093	tanh	0.7294	0.0307	0.7294	1
17	96	224	192	256	0.4	0.0009	tanh	0.7294	0.03	0.7294	1
18	32	192	416	224	0.4	0.0007	sigmoid	0.7294	0.293	0.7294	1

Table 6.7: The configuration of each trial and their results on the validation set.

The best performing configuration is from trial 1 with a validation accuracy of 81.5%. One notable thing is that although most of the trials achieve 0.7294 validation accuracy, all trials that achieve more than that use ReLU as their activation function. In other words, all the top 5 performing models use ReLU.

Trial	activation	accuracy
1	ReLU	0.8150
3	ReLU	0.8030
0	ReLU	0.7928
10	ReLU	0.7910
8	ReLU	0.7910
2	tanh	0.7294
5	sigmoid	0.7294
7	sigmoid	0.7294
6	ReLU	0.7294
9	tanh	0.7294

Table 6.8: Top 10 trials.

6.2.6 Testing

So, because the model from trial 1 was the best performing one, it was the one kept and used on the test set. It achieved:

- Accuracy: 79%
- Precision: 79%
- Recall: 96%
- F1-score: 87%

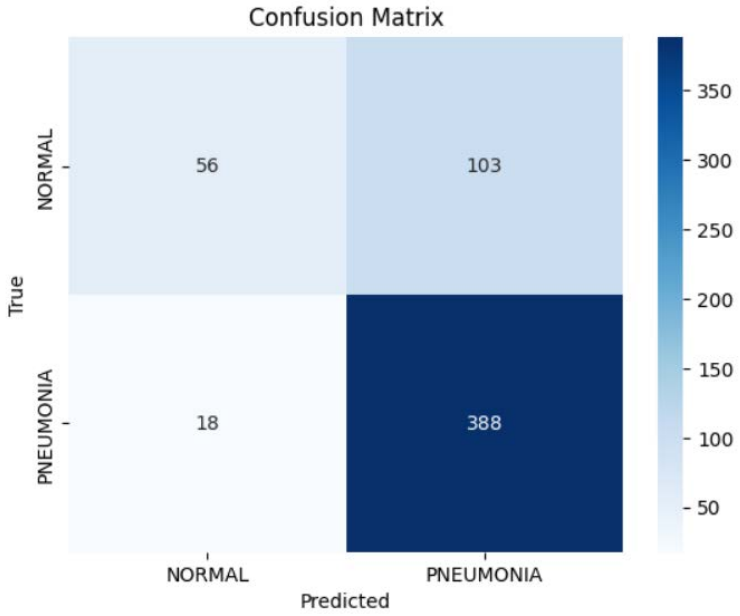


Figure 6.5: Confusion matrix.

To predict in which class every sample belongs to, we chose a decision threshold of 0.5. To evaluate the model at different decision thresholds we also plotted the ROC and precision-recall curves and calculated the AUC.

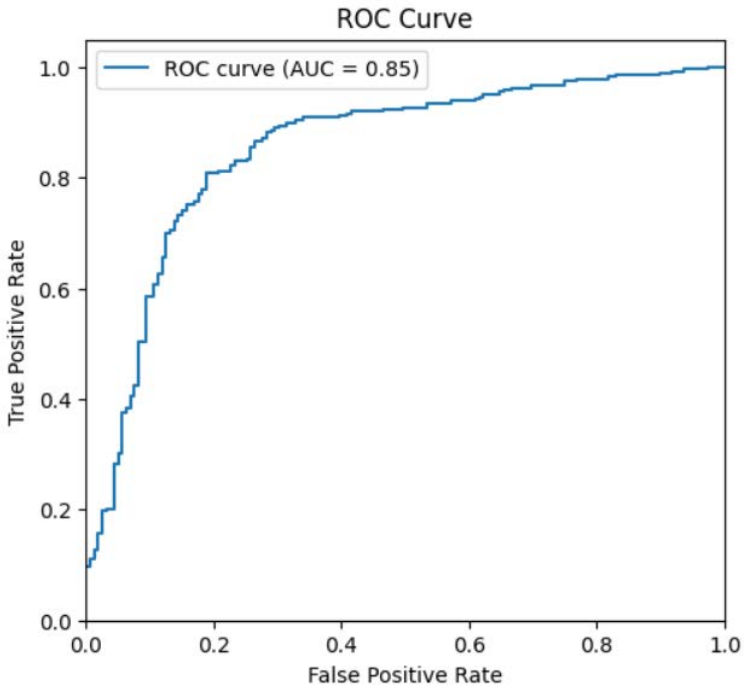


Figure 6.6: ROC Curve.

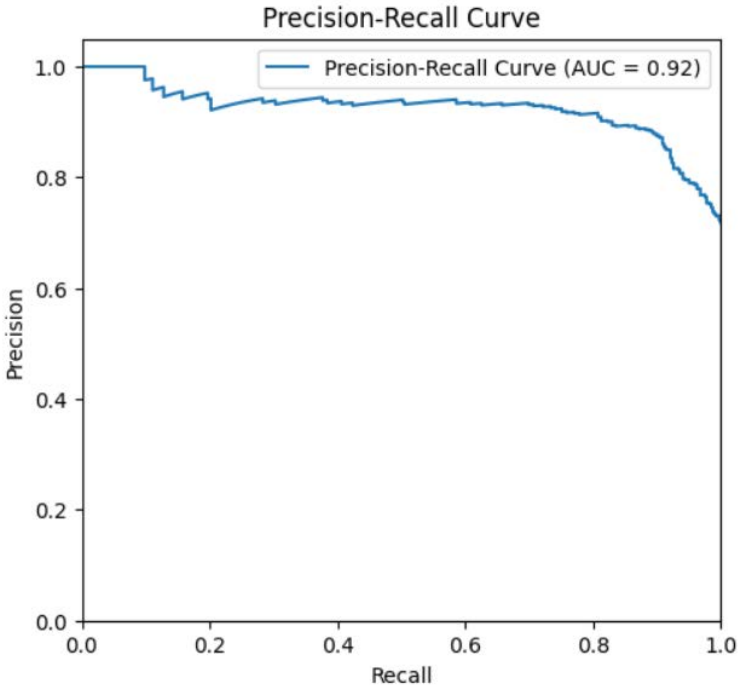


Figure 6.7: PR curve.

It is important to note that, because our dataset is imbalanced, a PR curve represents our model more accurately than the ROC curve.

6.2.7 Tuning

Afterwards, the model went through another set of training (5 more epochs). Ultimately, the tuned model was evaluated on the test set achieving:

- Accuracy: 81%
- Precision: 82%
- Recall: 94%
- F1-score: 88%
- ROC-AUC: 87%
- PR-AUC: 94%

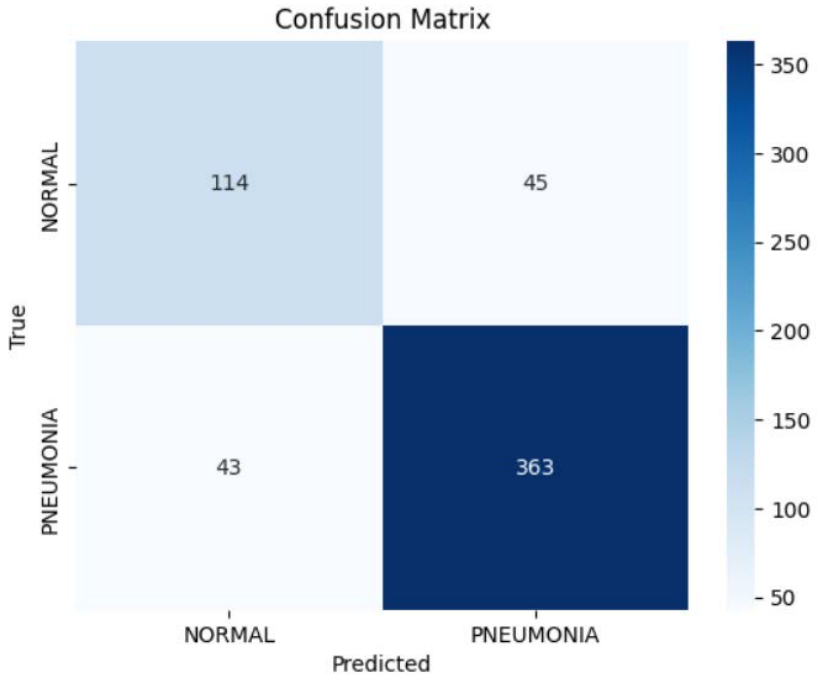


Figure 6.8: Confusion matrix of the tuned model.

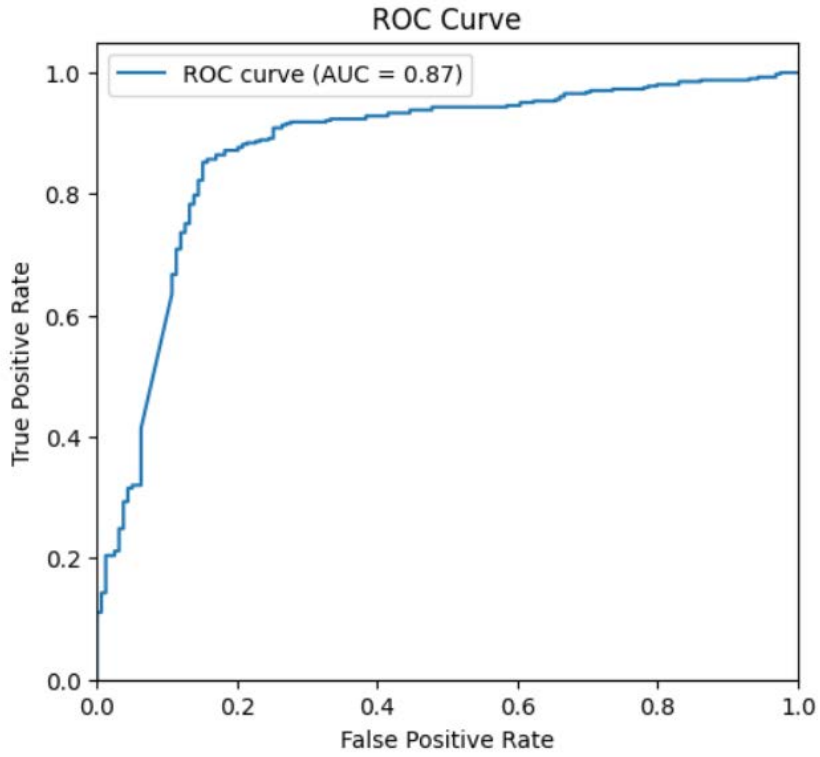


Figure 6.9: ROC curve of the tuned model.

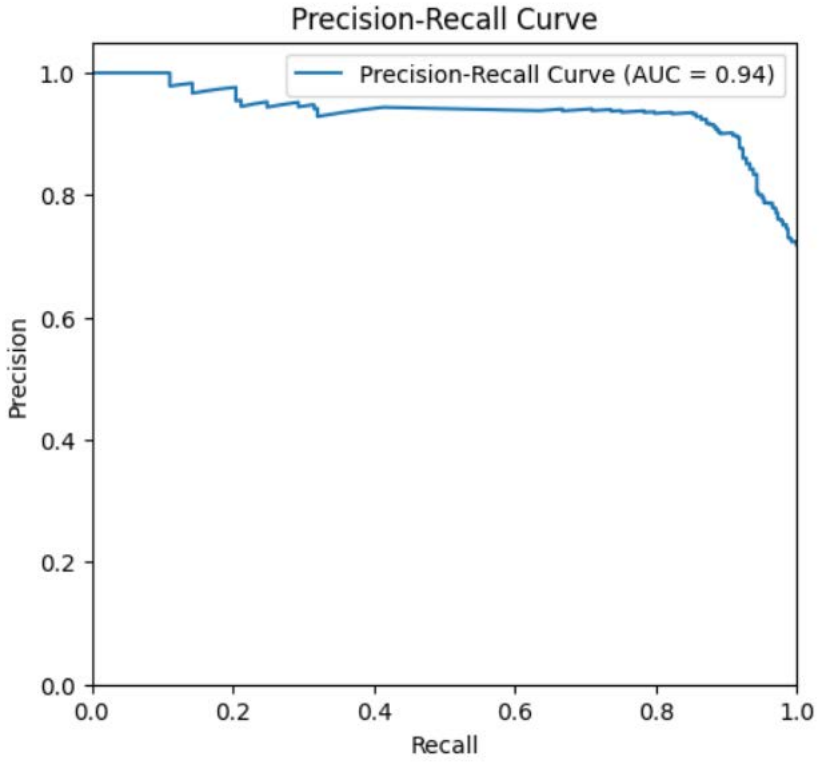


Figure 6.10: PR curve of the tuned model.

Overall, the tuned model performs better than the original model on almost every metric.

6.2.8 Choosing an optimal decision threshold and final testing

Although 0.5 is a common decision threshold it may not be the best in our case. So, finally, we used the ROC and PR curves to find the most optimal decision threshold.

1) Using the ROC curve and Youden's J statistic

By calculating J (TPR – FPR) for multiple decision thresholds, we conclude that the most optimal threshold, according to the ROC curve, is 0.7129. When making predictions using this decision threshold, instead of 0.5, the model achieves:

- Accuracy: 82%
- Precision: 91%
- Recall: 84%
- F1-score: 87%

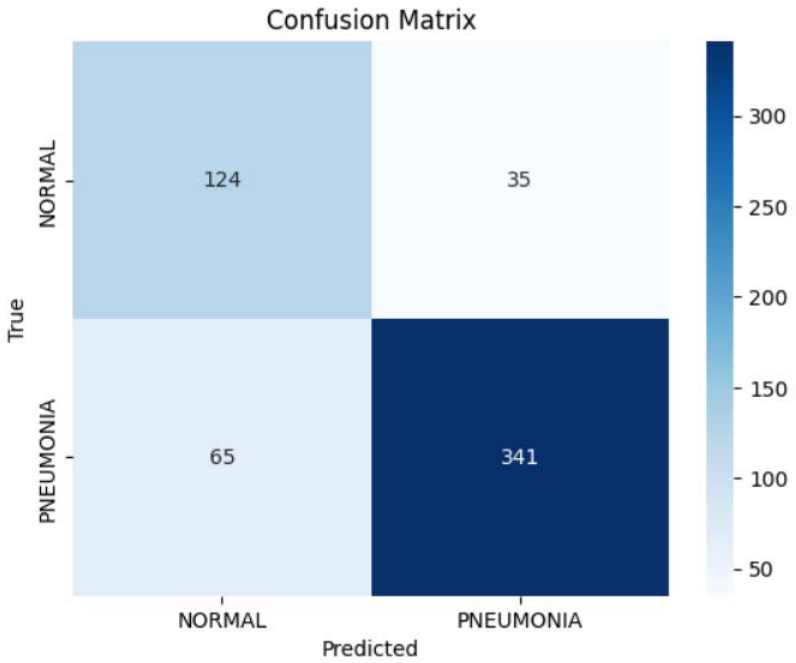


Figure 6.11: Confusion matrix (Decision Threshold: 0.7129)

2) Using the Precision-Recall curve

By calculating the F1-score for multiple decision thresholds, we conclude that the most optimal threshold, according to the PR curve now, is 0.5846. When making predictions using this decision threshold the model achieves:

- Accuracy: 83%
- Precision: 88%
- Recall: 88%
- F1-score: 88%

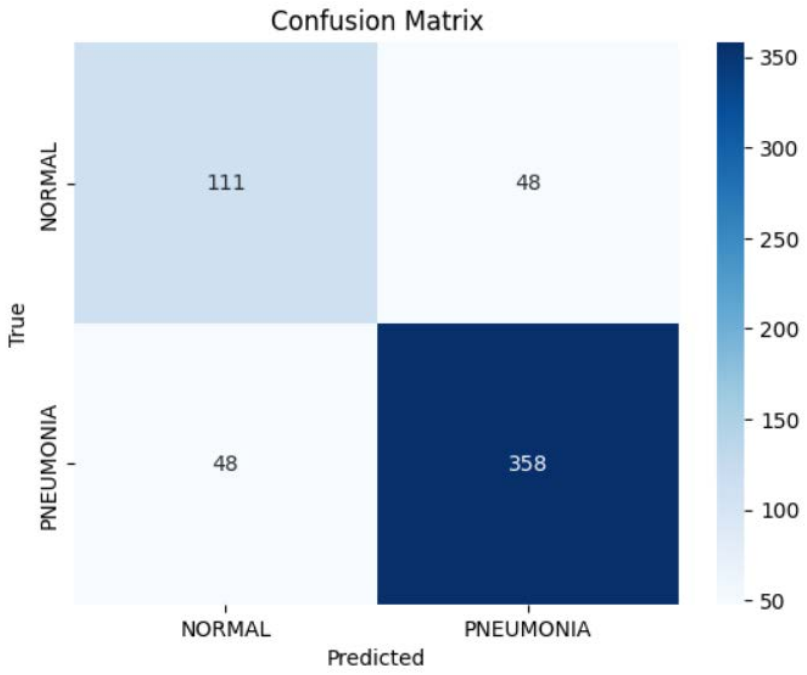


Figure 6.12: Confusion matrix (Decision Threshold: 0.5846)

6.2.9 Summary and conclusions

The table below shows the results of every model that was tested:

	BEST FROM TRIALS	TUNED (DT=0.5)	TUNED (DT=0.71)	TUNED (DT=0.58)
ACCURACY	79%	81%	82%	83%
PRECISION	79%	82%	91%	88%
RECALL	96%	94%	84%	88%
F1-SCORE	87%	88%	87%	88%

Table 6.9: Experiment 2 summary.

If we take into consideration every metric, the tuned model with a 0.58 decision threshold (chosen using the PR-curve) performs better overall. Ultimately, the decision should be made by taking into account the consequences of false positives and false negatives and considering if the trade-off is worth it.

6.3 Experiment 3: Sequential sentence classification using RNNs

6.3.1 Information about the dataset

[24] For this experiment the PubMed 200k RCT dataset was used. It contains about 2.3 million sentences, each labelled as one of the five following classes, representing the sentence's role in the medical abstract: background, objective, method, result and conclusion. We were only provided with a train and a test set so, we made changes to achieve an 80%/10%/10% train/test/validation split, like before.

6.3.2 The RNN's architecture

The RNN consists of:

- Embedding layer
- LSTM layer
- Dense layer
- Output layer

The RNN belongs to the Many-to-One category. It takes a sequence of words and outputs a class label.

6.3.3 Hyperparameter choice

Once again, KerasTuner was used to choose the right configuration. The parameters that were tuned are:

- Embedding layer's dimensions
- LSTM and dense layer units
- Dropout rate
- Learning rate
- Activation function

The optimizer and the loss function remain constant throughout the trials.

Optimizer: Adam

Loss function: Categorical cross-entropy, a variation of binary cross-entropy for multi-class problems.

Activation function: The activation function of the output layer has to remain the same. Instead of the sigmoid function, which was used on experiment 2, for multi-class problems the softmax function is preferred.

6.3.4 Finding the best configuration

Below there is a summary of the trials:

trial	embedding units	lstm units	dropout rate	dense units	activation	learning rate	accuracy	loss
0	192	64	0.5	64	tanh	0.0001	0.8555	0.3991
1	160	224	0.4	96	tanh	0.001	0.8612	0.3813
2	64	64	0.4	32	ReLU	0.0001	0.8471	0.4169
3	64	160	0.3	128	tanh	0.0001	0.8535	0.3989
4	128	224	0.3	112	ReLU	0.001	0.8623	0.3759
5	128	64	0.4	128	sigmoid	0.001	0.8583	0.3893
6	**64**	**192**	**0.4**	**128**	**ReLU**	**0.001**	**0.8636**	**0.3716**
7	160	256	0.4	80	tanh	0.0001	0.8585	0.3839
8	128	224	0.4	128	sigmoid	0.001	0.8615	0.3805
9	128	64	0.3	80	ReLU	0.01	0.765	0.632
10	224	192	0.3	112	sigmoid	0.0001	0.8593	0.3839
11	128	224	0.2	48	ReLU	0.0001	0.86	0.3832
12	64	64	0.5	80	sigmoid	0.001	0.8597	0.3896
13	192	64	0.1	64	sigmoid	0.0001	0.8565	0.3963
14	160	192	0.2	96	ReLU	0.01	0.7624	0.6499
15	192	224	0.3	64	sigmoid	0.0001	0.8605	0.3804
16	64	64	0.4	128	sigmoid	0.0001	0.8486	0.4128
17	64	192	0.2	80	tanh	0.001	0.8627	0.3787
18	96	160	0.3	64	ReLU	0.01	0.7538	0.6585
19	64	96	0.2	32	sigmoid	0.01	0.7772	0.6155

Table 6.10: The configuration of each trial and their results on the validation set (2).

The best performing configuration is from trial 6 with a validation accuracy of 86.36%.

6.3.5 Testing

Once again, the best model's performance was evaluated on the test set (note: The model went through another set of training like in experiment 2, but the model's performance was almost identical. 5 epochs were enough in this case). It achieved:

- Accuracy: 86%
- Precision (macro average): 82%
- Precision (weighted average): 86%
- Recall (macro average): 80%
- Recall (weighted average): 86%
- F1-score (macro average): 80%
- F1-score (weighted average): 86%

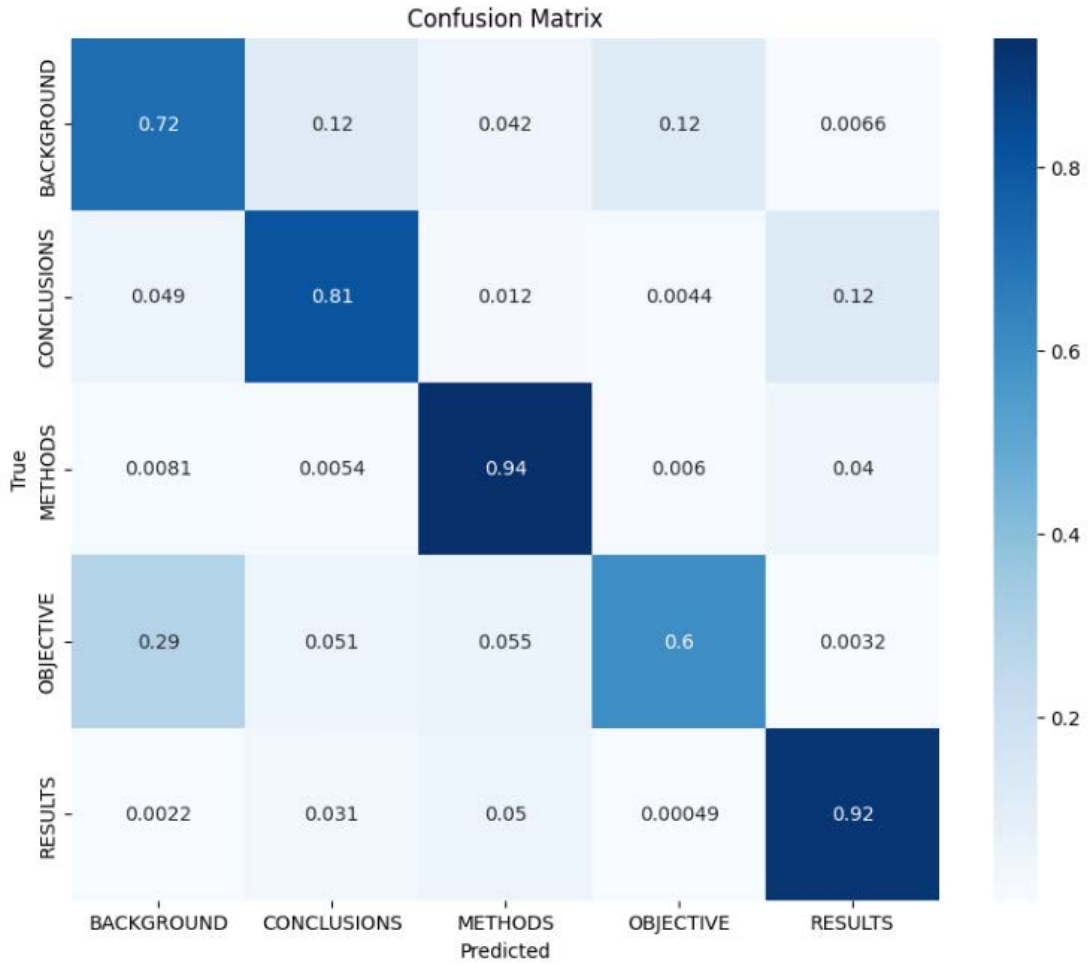


Figure 6.13: Confusion matrix.

We also plotted the ROC and PR curves, which were calculated using the One-vs-Rest approach.

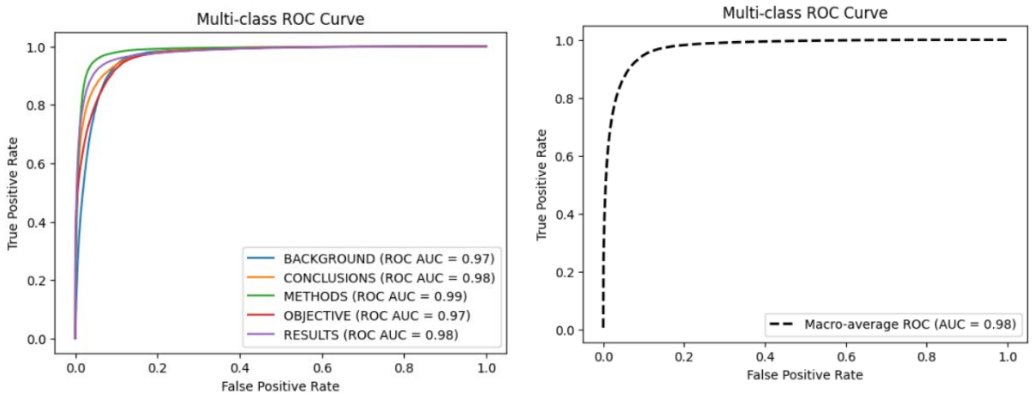


Figure 6.14: ROC curve for every class and the macro average.

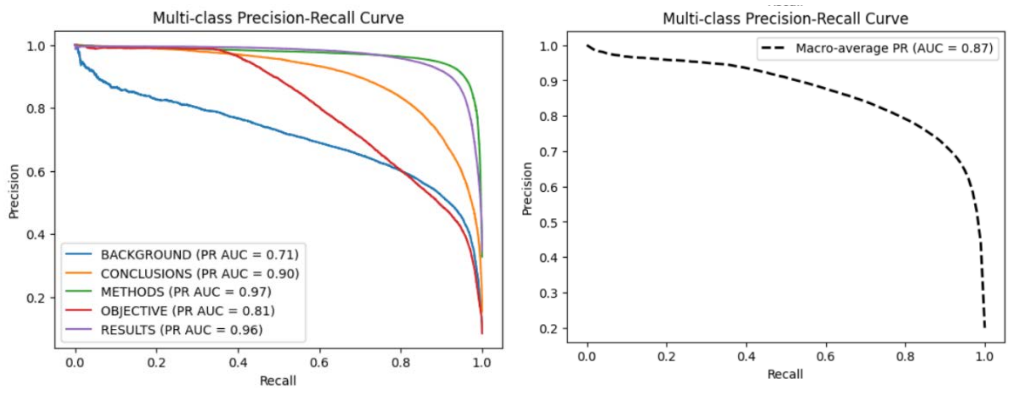


Figure 6.15: Precision-Recall curve for every class and the macro average.

The model achieved:

- Macro average ROC-AUC: 98%
- Macro average PR-AUC: 87%

6.3.6 Transfer Learning

[25] Transfer learning is a machine learning strategy in which a model trained on one task is reused as the starting point for a related task. Instead of training a model entirely from scratch, the knowledge and features learned in the initial task are carried over and adapted to the new one. This approach allows the model to learn more efficiently, achieve higher performance, and converge faster than if it had to extract all features independently.

One of the major benefits of transfer learning is its ability to reduce overfitting. Since the model already incorporates generalizable features from the source task, it does not rely as heavily on the limited data available for the target task. This makes transfer learning particularly valuable in scenarios where collecting and labelling large datasets is difficult, costly, or impractical. By leveraging pretrained models, the dependency on massive training sets is minimized, allowing complex models to be applied effectively even in data-constrained environments.

A key application area of transfer learning is in deep learning, where state-of-the-art models trained on large benchmark datasets can be fine-tuned for more specialized problems. This approach has been especially impactful in the medical domain. One of the central challenges in medical analysis is the shortage of annotated training data, as labelling often requires time-consuming input from highly skilled experts. Transfer learning addresses this limitation by reusing pretrained convolutional neural networks (CNNs) or other architectures and fine-tuning them on specific tasks.

Beyond medicine, transfer learning has been successfully applied in natural language processing, speech recognition, and many other domains, often serving as a bridge between general-purpose models and domain-specific applications. Its continued growth reflects a shift in machine learning toward reusability and efficiency, where knowledge gained in one domain fuels progress in another.

In this experiment, we tried transfer learning by deploying 2 publicly accessible models and fine-tuning them.

6.3.7 Fine-tuning the PubMedBERT model

First, we tried the PubMedBERT model, which is a model trained on data very similar to our dataset. Since domain mismatch is not a problem, fine-tuning the PubMedBERT model should yield great results.

So, we loaded the PubMedBERT model and fine-tuned it to fit our dataset. The now fine-tuned model achieved:

- Accuracy: 89%
- Precision (macro average): 84%
- Precision (weighted average): 89%
- Recall (macro average): 83%
- Recall (weighted average): 89%
- F1-score (macro average): 83%
- F1-score (weighted average): 89%
- Macro average ROC-AUC: 98%
- Macro average PR-AUC: 90%

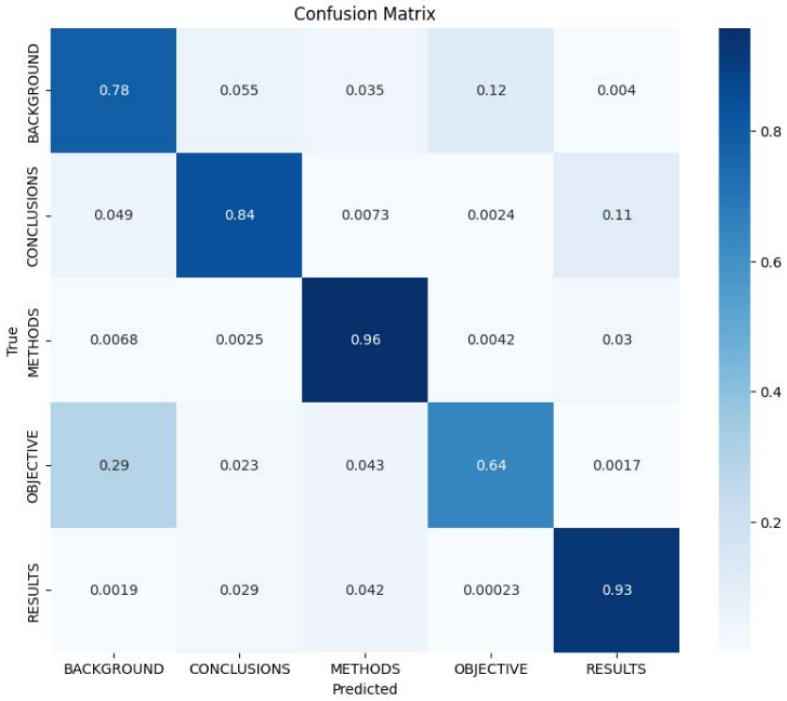


Figure 6.16: Confusion matrix of the fine-tuned PubMedBERT model.

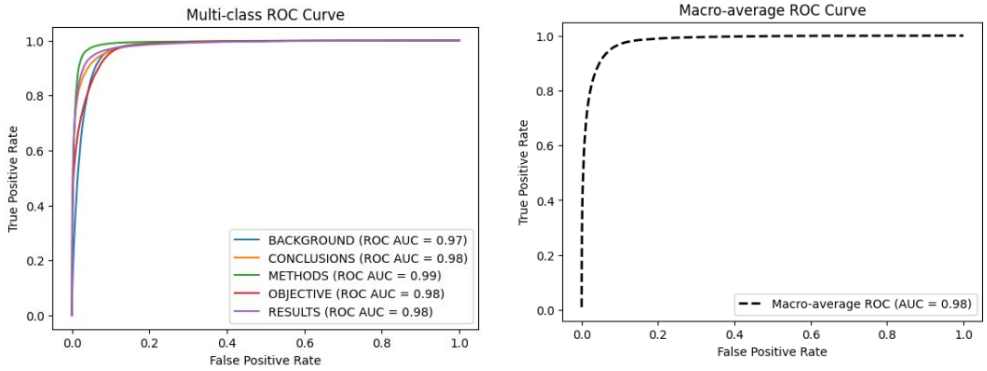


Figure 6.17: ROC curve for every class and the macro average (fine-tuned PubMedBERT model).

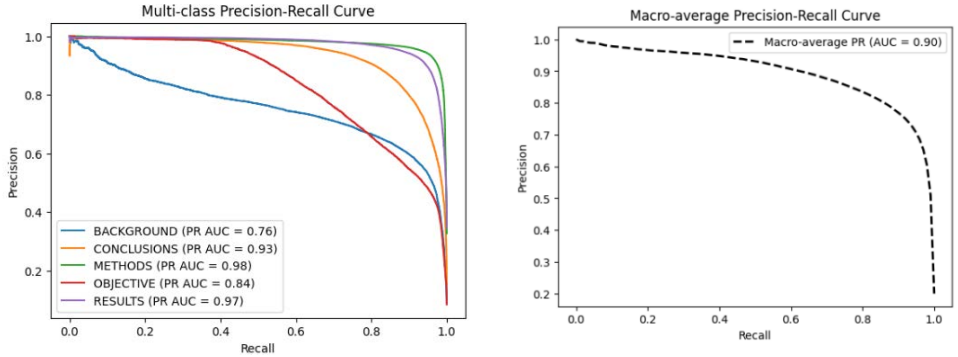


Figure 6.18: Precision-Recall curve for every class and the macro average (fine-tuned PubMedBERT model).

The fine-tuned PubMedBERT model performed better on every metric than our model made from scratch.

6.3.8 Fine-tuning the BioBERT-v1.1 model

The second model we tried was the BioBERT-v1.1 model. It was trained on large bodies of biomedical texts and it was designed for tasks such as biomedical named entity recognition, relation extraction, and question answering. The data it was trained are not as similar to ours, like in the case of PudMedBERT, but they are similar enough to give the model a shot.

So, we loaded the BioBERT-v1.1 model and, like before, fine-tuned it to fit our dataset. The now fine-tuned model achieved:

- Accuracy: 89%
- Precision (macro average): 84%
- Precision (weighted average): 89%
- Recall (macro average): 83%
- Recall (weighted average): 89%
- F1-score (macro average): 83%
- F1-score (weighted average): 89%
- Macro average ROC-AUC: 98%
- Macro average PR-AUC: 90%

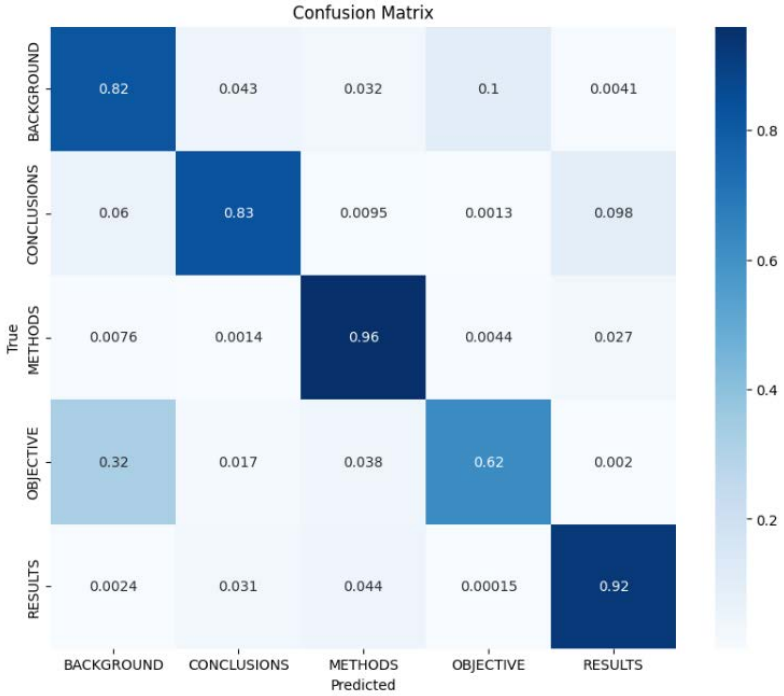


Figure 6.19: Confusion matrix of the fine-tuned BioBERT-v1.1 model.

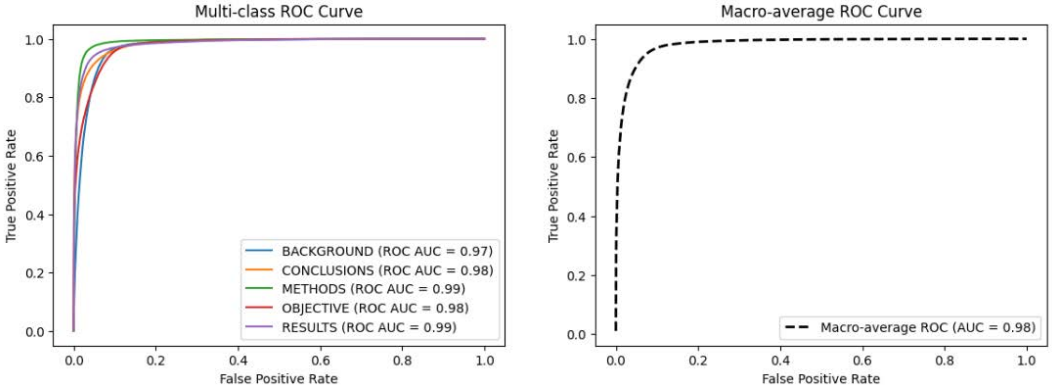


Figure 6.20: ROC curve for every class and the macro average (fine-tuned BioBERT-v1.1 model).

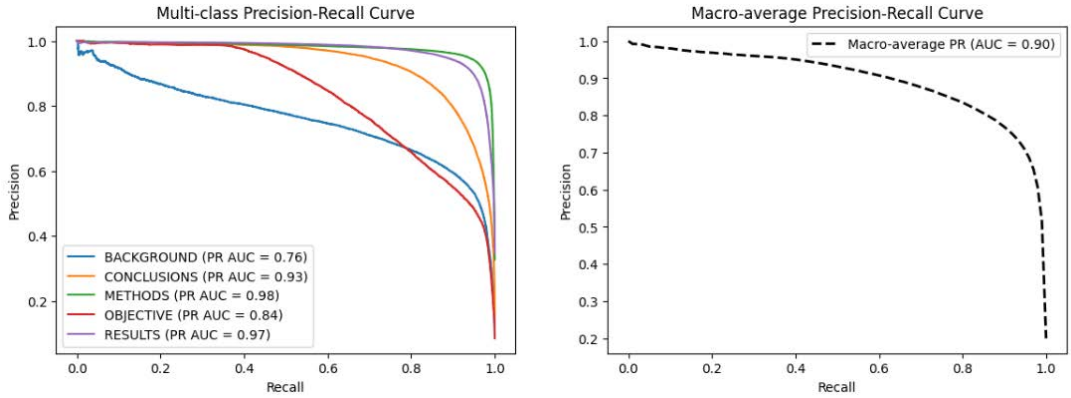


Figure 6.21: Precision-Recall curve for every class and the macro average (fine-tuned BioBERT-v1.1 model).

The fine-tuned BioBERT-v1.1 model also performed better than our model on every metric, with results very similar to the fine-tuned PubMedBERT model's. Their only differences are when it comes to per-class results, which can be seen in the confusion matrices.

6.3.9 Summary and conclusions

Even though our dataset was pretty large, hence data availability was not an issue, the use of publicly available, state-of-the-art models still proved beneficial. Finding the best architecture and the most optimal hyperparameter configuration can be tricky. For that reason, working with neural networks can be complicated. That makes transfer learning an appealing approach, which can yield great results if used properly.

7 Conclusion

Summing up, we covered several machine learning methods and discussed how they can be used on medical data. As we experimented with them, we realized that even the relatively simple algorithms and models we used can yield promising results. It shows that machine learning is a powerful tool, a tool we should become accustomed to. Even though we focused on its applications in the medical field, machine learning has a wide range of applications and it has already been adopted by many fields. As it continues to develop, it is likely that machine learning will find its way in every field, not only the ones that are scientific in nature, thus becoming eventually an integral part of our lives.

Bibliography

[1] "Precision and Recall in Machine Learning - GeeksforGeeks." Accessed: Sep. 03, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/precision-and-recall-in-machine-learning/

[2] "Precision-Recall Curve - ML - GeeksforGeeks." Accessed: Sep. 03, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/precision-recall-curve-ml/

[3] "Calculating Precision and Recall for Multiclass Classification Using Confusion Matrix - GeeksforGeeks." Accessed: Aug. 26, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/calculating-precision-and-recall-for-multiclass-classification-using-confusion-matrix/

[4] "What is the k-nearest neighbors algorithm? | IBM." Accessed: Jun. 15, 2025. [Online]. Available: https://www.ibm.com/think/topics/knn

[5] "What Is Random Forest? | IBM." Accessed: Jun. 15, 2025. [Online]. Available: https://www.ibm.com/think/topics/random-forest

[6] "What Is Logistic Regression? | IBM." Accessed: Jun. 15, 2025. [Online]. Available: https://www.ibm.com/think/topics/logistic-regression

[7] V. Kecman, "Support Vector Machines-An Introduction." [Online]. Available: www.springerlink.com

[8] S. Mishra *et al.*, "Principal Component Analysis," *International Journal of Livestock Research*, p. 1, 2017, doi: 10.5455/ijlr.20170415115235.

[9] "What Is Principal Component Analysis (PCA)? | IBM." Accessed: May 06, 2025. [Online]. Available: https://www.ibm.com/think/topics/principal-component-analysis

[10] "Singular Value Decomposition (SVD) - GeeksforGeeks." Accessed: Sep. 03, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/singular-value-decomposition-svd/

[11] "What Is Linear Discriminant Analysis? | IBM." Accessed: May 12, 2025. [Online]. Available: https://www.ibm.com/think/topics/linear-discriminant-analysis

[12] A. Tharwat, "Independent component analysis: An introduction," 2018, *Emerald Group Holdings Ltd.* doi: 10.1016/j.aci.2018.08.006.

[13] Y. T. Guo, Q. Q. Li, and C. S. Liang, "The rise of nonnegative matrix factorization: Algorithms and applications," *Inf Syst*, vol. 123, p. 102379, Jul. 2024, doi: 10.1016/J.IS.2024.102379.

[14] "What is a Neural Network? | IBM." Accessed: Jul. 10, 2025. [Online]. Available: https://www.ibm.com/think/topics/neural-networks

[15] L. Cai, J. Gao, and D. Zhao, "A review of the application of deep learning in medical image classification and segmentation," *Ann Transl Med*, vol. 8, no. 11, p. 713, Jun. 2020, doi: 10.21037/ATM.2020.02.44.

[16] C. Chen, N. A. Mat Isa, and X. Liu, “A review of convolutional neural network based methods for medical image classification,” *Comput Biol Med*, vol. 185, p. 109507, Feb. 2025, doi: 10.1016/J.COMPBIOMED.2024.109507.

[17] “Introduction to Recurrent Neural Networks - GeeksforGeeks.” Accessed: Sep. 06, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/

[18] Y. Luo, “Recurrent neural networks for classifying relations in clinical notes,” *J Biomed Inform*, vol. 72, pp. 85–95, Aug. 2017, doi: 10.1016/J.JBI.2017.07.006.

[19] D. Ribeiro dos Santos, “Application of Recurrent Neural Networks (RNNs) in Medical Diagnostics,” p. 255, Aug. 2024, doi: 10.34894/VQ1DJA.

[20] B. C. Feltes, E. B. Chandelier, B. I. Grisci, and M. Dorn, “CuMiDa: An Extensively Curated Microarray Database for Benchmarking and Testing of Machine Learning Approaches in Cancer Research,” *Journal of Computational Biology*, vol. 26, no. 4, pp. 376–386, Apr. 2019, doi: 10.1089/CMB.2018.0238.

[21] Y. Lu and J. Han, “Cancer classification using gene expression data,” 2003.

[22] A. Islam, M. M. Rahman, E. Ahmed, F. Arafat, and M. Fazle Rabby, “Adaptive feature selection and classification of colon cancer from gene expression data: An ensemble learning approach,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Jan. 2020. doi: 10.1145/3377049.3377070.

[23] D. Kermany, K. Zhang, and M. Goldbaum, “Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification,” vol. 2, 2018, doi: 10.17632/RSCBJBR9SJ.2.

[24] F. Dernoncourt and J. Y. Lee, “PubMed 200k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts,” Oct. 2017, Accessed: Aug. 27, 2025. [Online]. Available: https://arxiv.org/pdf/1710.06071

[25] “What is Transfer Learning? - GeeksforGeeks.” Accessed: Aug. 27, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/ml-introduction-to-transfer-learning/