**People's Democratic Republic Of Algeria**
**Ministry of Higher Education and Scientific Research**

**University ECHAHID HAMMA LAKHDAR - EL OUED**
**Fculty OF Exact Sciences**
**Computer Science department**

**End of study memory Presented for the Diploma of**

ACADEMIC MASTER

Domain : **Mathematics and Computer Science**
Industry : **Computer Science**
Specialty : **Distributed Systems and Artificial Intelligence**

***Theme***

# Utilizing Deep Learning Models for Gene Selection from Microarray Expression Data

**Presented by :**
Chennouf Souhail
Messaoudi Abdrrazzak

| | | |
|---|---|---|
| DR, Ben Ali Abdelkamel | Supervisor | Univ. El Oued |
| DR. Bouhamed Mohammed Mounir | President | Univ. El Oued |
| DR. Gharbi Kadour | Examiner | Univ. El Oued |

Academic year
2023/2024

# شكر و عرفان

بسم الله و الصلاة و السلام على رسول الله

اولا نحمد الله و نشكره على توفيقه و منحي القوة والصبر لإتمام هذا المشروع بنجاح.

شكرًا لأستاذ المشرف *بن علي عبدالكامل* على توجيهنا و تقديم النصح لنا ،

و شكر خاص لوالديّ الكريمين و لأصدقائي الأعزاء الذين كانوا دائمًا بجانبي، يشجعوني ويساندوني في كل الظروف ،

كما لا أنسى شكر أساتذتي الأفاضل على توجيهاتهم ونصائحهم القيمة

كما أعبر عن شكري الجزيل

وامتناني العميق لكل من ساهم في إنجاح هذه المذكرة

# Abstract

Gene selection from microarray gene expression data is an essential task for accurate cancer classification. This study explores the application of deep learning models, including AutoEncoder, and other deep learning-based feature selection methods, to identify the smallest possible set of genes that maintain high predictive performance. Selection of relevant genes is essential in most gene expression studies, with the aim of identifying the most important genes for specific classification tasks. To test the classification accuracy, we implement several machine and deep learning models such as Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest, and Multi-Layer Perceptrons (MLP). These models are trained on selected genes to predict classification accuracy.
This study highlights the potential of combining deep learning-based feature selection with advanced classification models to improve cancer diagnosis and treatment strategies. These findings contribute to the field of precision medicine and computational biology by demonstrating the benefits of leveraging cutting-edge machine learning techniques to analyze genomic data.

**Key words**: Gene selection, Microarray expression data , Deep learning models, Cancer classification, Precision medicine .

# Résumé

La sélection de gènes à partir des données d'expression génique des micropuces est une tâche essentielle pour une classification précise du cancer. Cette étude explore l'application de modèles d'apprentissage profond, notamment AutoEncoder, et d'autres méthodes de sélection de fonctionnalités basées sur l'apprentissage profond, pour identifier le plus petit ensemble possible de gènes qui maintiennent des performances prédictives élevées. La sélection de gènes pertinents est essentielle dans la plupart des études d'expression génique, dans le but d'identifier les gènes les plus importants pour des tâches de classification spécifiques.
Pour tester l'exactitude de la classification, nous implémentons plusieurs modèles d'apprentissage automatique et profond tels que le plus proche voisin (KNN), la machine à vecteurs de support (SVM), la forêt aléatoire et les perceptrons multicouches (MLP). Ces modèles sont formés sur des gènes sélectionnés pour prédire l'exactitude de la classification.
Cette travail de mémoire met en évidence le potentiel de combiner la sélection de fonctionnalités basée sur l'apprentissage profond avec des modèles de classification avancés pour améliorer les stratégies de diagnostic et de traitement du cancer. Ces découvertes contribuent au domaine de la médecine de précision et de la biologie computationnelle en démontrant les avantages de tirer parti des techniques d'apprentissage automatique récentes pour analyser les données génomiques.

**Key words** :sélection de gènes, données d'expression de micropuces, modèles d'apprentissage profond, classification du cancer, médecine de précision.

# ملخص

يعد اختيار الجينات من بيانات تعبير الجيني لمصفوفة الدقيقة مهمة حاسمة لتصنيف السرطان بدقة. تستكشف هذه الدراسة تطبيق نماذج التعلم العميق، بما في ذلك التشفير التلقائي و طرق أخرى لاختيار الميزات تعتمد على التعلم العميق، لتحديد أصغر مجموعة ممكنة من الجينات الذي يحافظ على الأداء التنبؤي العالي. يعد اختيار الجينات ذات الصلة أمرًا ضروريًا في معظم دراسات التعبير الجيني، بهدف تحديد الجينات الأكثر أهمية لمهام التصنيف المحددة.

لاختبار دقة التصنيف ، قمنا بتنفيذ العديد من نماذج التعلم الآلي والعميق مثل ( Random Forest, MLP, KNN , SVM ) يتم تدريب هذه النماذج على جينات مختارة للتنبؤ بدقة التصنيف.

يسلط هذا العمل الضوء على إمكانية الجمع بين اختيار الميزات القائمة على التعلم العميق ونماذج التصنيف المتقدمة لتحسين تشخيص السرطان واستراتيجيات العلاج. تسهم هذه النتائج في مجال الطب الدقيق والبيولوجيا الحسابية من خلال إظهار فوائد الاستفادة من تقنيات التعلم الآلي الحديثة لتحليل البيانات الجينومية.

**الكلمات المفتاحية:** اختيار الجينات، بيانات تعبير مصفوفة الدقيقة، نماذج التعلم العميق، تصنيف السرطان، الطب الدقيق.

# Contents

# List of Figures

# List of Tables

# General Introduction

This study focuses on using deep learning models for gene selection from microarray gene expression data with the aim of improving cancer classification. Gene selection from microarray gene expression data is a critical aspect of cancer classification and other biological studies. Microarray technology allows researchers to measure the expression levels of thousands of genes simultaneously, providing a wealth of data for analysis. However, not all genes are equally informative for a particular classification task or biological question. Therefore, the goal of gene selection is to identify a subset of genes that are most relevant and informative for accurately classifying samples (e.g., cancer vs. normal).

Various methods have been developed for gene selection, ranging from traditional statistical techniques to more advanced machine learning approaches, including deep learning models. These methods aim to identify the smallest set of genes that can achieve high predictive performance, thus reducing the dimensionality of the data and improving computational efficiency while maintaining or even enhancing classification accuracy. Deep gene selection methods leverage deep learning architectures, such as AutoEncoders, to extract meaningful features from gene expression data. AutoEncoders are neural networks designed to learn compact representations of input data, which can be used to identify important features (genes) for classification tasks.

The study's structure consists of three main chapters. The first chapter introduces the concept of microarrays and their importance in gene expression analysis, in addition to explaining the classification process and the significance of feature selection in improving classification accuracy. The second chapter discusses machine learning and deep learning, focusing on models such as SVM, RF, KNN, MLP, AutoEncoders and their roles in analyzing data. The third chapter aims to use deep learning models for gene selection from microarray data, where the original data is transformed into a compressed representation using the ReLU activation function, allowing for an effective representation of key aspects and primary information without significant loss of information.

Different machine learning models (SVM, KNN, RF, MLP) were applied to the data extracted (selected genes) from the AutoEncoder and the original data, followed by a comparison of results to evaluate the performance of the AutoEncoder. Through this methodology, the study seeks to improve the accuracy of cancer classification , contributing to the development of more effective data analysis techniques in molecular biology. Overall, gene selection from microarray gene expression data is a crucial, especially in cancer classification, where identifying the most relevant genes can lead to a better understanding of disease mechanisms, improved diagnostic tools, and targeted therapies.

Chapter 1

# Classifying with Microarray Data : Features and Selection

## Introduction

Feature selection and classification are essential steps in the analysis of microarray data. Various methods exist to perform these tasks, aiming to identify the most relevant features and improve the accuracy of classification models. Some of the widely used techniques for feature selection and classification in microarray data analysis include evolutionary methods, machine learning algorithms, and population-based stochastic search approaches. These methods are employed to identify predictive genes and improve the accuracy of classification models, particularly in the multi-class classification of microarray data. The primary objective of feature selection is to identify the most appropriate number of features that are relevant for the classification task. The enormous dimensionality of microarray data, which can range from 6000 to 60,000 characteristics, makes feature selection a crucial step in the analysis process. Various feature selection and feature extraction methods are being widely used to remove redundant and irrelevant features, thereby facilitating the classification of new data. The goal of these methods is to address the challenges posed by the size of the data and the complex relations among different genes in microarray analysis.

## 1.1 What is a Microarray ?

Scientists know that a mutation or alteration in a particular gene's DNA may contribute to a certain disease. However, it can be very difficult to develop a test to detect these mutations, because most large genes have many regions where mutations can occur. For example, researchers believe that mutations in the genes BRCA1 and BRCA2 cause as many as 60 percent of all cases of hereditary breast and ovarian cancers. But there is not one specific mutation responsible for all of these cases. Researchers have already discovered over 800 different mutations in BRCA1 alone.The DNA microarray is a tool used to determine whether the DNA from a particular individual contains a mutation in genes like BRCA1 and BRCA2. The chip consists of a small glass plate encased in plastic. Some companies manufacture microarrays using methods similar to those used to make computer

microchips. On the surface, each chip contains thousands of short, synthetic, single-stranded DNA sequences, which together add up to the normal gene in question, and to variants (mutations) of that gene that have been found in the human population [4].

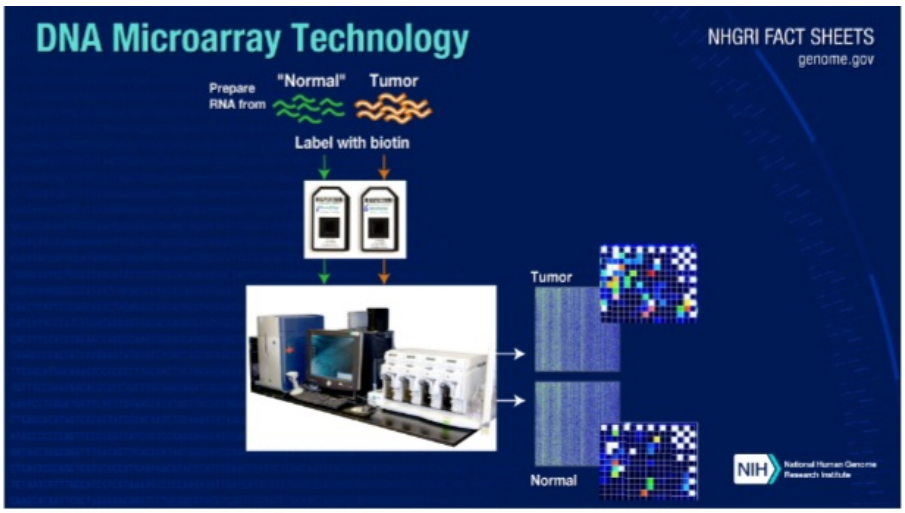


Figure 1.1: DNA Microarray Technology.

### 1.1.1 What is a microarray used for ?

When they were first introduced, DNA Microarrays were used only as a research tool. Scientists continue today to conduct large-scale population studies - for example, to determine how often individuals with a particular mutation actually develop breast cancer, or to identify the changes in gene sequences that are most often associated with particular diseases. This has become possible because, just as is the case for computer hips, very large numbers of 'features' can be put on Microarray chips, representing a very large portion of the human genome. Microarrays can also be used to study the extent to which certain genes are turned on or off in cells and tissues. In this case, instead of isolating DNA from the samples, RNA (which is a transcript of the DNA) is isolated and measured. Today, DNA microarrays are used in clinical diagnostic tests for some diseases. Sometimes they are also used to determine which drugs might be best prescribed for particular individuals, because genes determine how our bodies handle the chemistry related to those drugs. With the advent of new DNA sequencing technologies, some of the tests for which microarrays were used in the past now use DNA sequencing instead. But microarray tests still tend to be less expensive than sequencing, so they may be used for very large studies, as well as for some clinical test [4].

### 1.1.2 How does Microarray work ?

To determine whether an individual possesses a mutation for a particular disease, a scientist first obtains a sample of DNA from the patient's blood as well as a control sample - one that does not contain a mutation in the gene of interest.The researcher then denatures the DNA in the samples - a process that separates the two complementary strands of DNA into single-stranded molecules. The next step is to cut the long strands of DNA into smaller, more manageable fragments and then to label each fragment by attaching a fluorescent dye (there are other ways to do this, but this is one common method). The individual's DNA is labeled with green dye and the control - or normal - DNA is labeled with red dye. Both sets of labeled DNA are then inserted into the chip and allowed

to hybridize - or bind - to the synthetic DNA on the chip.If the individual does not have a mutation for the gene, both the red and green samples will bind to the sequences on the chip that represent the sequence without the mutation (the "normal" sequence).If the individual does possess a mutation, the individual's DNA will not bind properly to the DNA sequences on the chip that represent the "normal" sequence but instead will bind to the sequence on the chip that represents the mutated DNA [4].

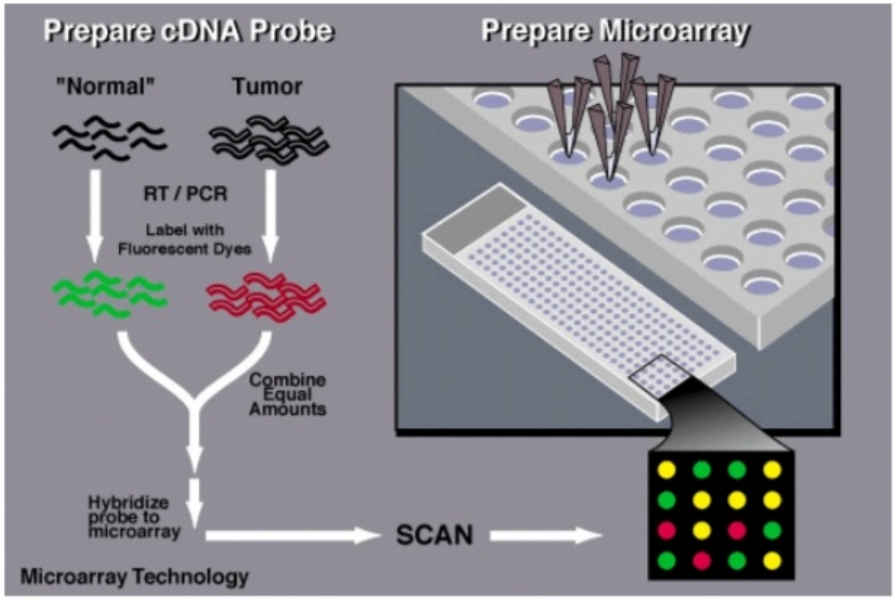


Figure 1.2: Microarray Technology.

## 1.2 What is Classification ?

Microarray classification is a technique used in bioinformatics to classify biological samples based on their gene expression levels. It's a type of supervised learning task, where a classification algorithm is trained on a set of samples with known classifications (e.g., cancerous or healthy tissue) and then used to predict the classification of new samples [24].
Supervised machine learning techniques are extensively used in the field of cancer classification using microarray gene expression data. These methods involve developing algorithms that are trained on labeled datasets to predict outcomes. Various studies have explored the efficiency of different classification methods, such as Support Vector Machines (SVM), RBF Neural Nets, MLP Neural Nets, Bayesian, Decision Tree, and Random Forest methods [24].
Analysis of DNA microarray data is done through the following steps. Firstly, data are preprocessed using feature selection techniques to remove noisy, redundant features and get only informative ones. Then the resultant subset is used to train the learning model to diagnose cancer subtypes as illustrated in Figure 1.3 [21].

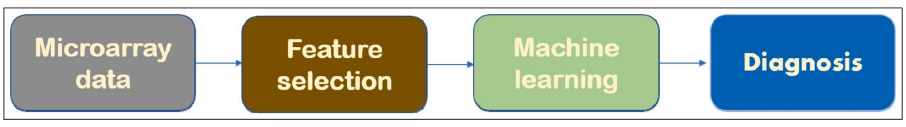


Figure 1.3: The pipeline of microarray analysis.

### 1.2.1 What is Gene Classification ?

Gene classification refers to the organization of genes according to their functions, structures, or other attributes. This process involves identifying patterns and relationships between genes to facilitate understanding their roles in biological processes and to aid in the interpretation of complex datasets, particularly in the context of microarrays. Microarray technology allows researchers to measure the expression levels of thousands of genes simultaneously, generating vast amounts of data that necessitate sophisticated analytical tools for gene classification [23][27].
Various statistical and machine learning methods are employed for gene selection and classification. Some examples include [23][27]:

- Random Forest (RF): RF is a powerful machine learning technique that builds multiple decision trees and combines their predictions to improve accuracy.
- Support Vector Machines (SVM): SVM construct hyperplanes to separate classes in multi-dimensional spaces.
- K-Nearest Neighbors (KNN): KNN assigns a test instance to the class of its k-closest neighbors.
- Ensemble Learning: Combining multiple classifiers can lead to improved accuracy and stability.
- Filtered, Wrapped, and Embedded Modeling Strategies: These methods aim to find the most informative genes for classification tasks.

## 1.3 What is feature selection ?

Feature selection is the process of automatically or manually select the features that have an impact on the prediction to [21]:

- Reduce overfitting: overfitting means the model doesn't generalize well from our training data to unseen data due to noise and redundancy in the data. The model will be well generalized when removing such data.
- Improves accuracy: train the model with less misleading data will improve the accuracy.
- Reduce training time : The smaller the number of features, the less computation time required for training.
- Offer biologists with insights about the mechanism between gene signature and diseases.

Feature selection can be classified based on the integration between the selection algorithm and the implemented model for classification into four main categories, as shown in Figure 1.4 [21].

### 1.3.1 Filter approaches

Filter approaches are divided into univariate and multivariate. Univariate feature selection examines each feature individually to measure the strength of an association between the features and the outcome variable. Common types are mutual information (MI) and information gain (IG) [21].

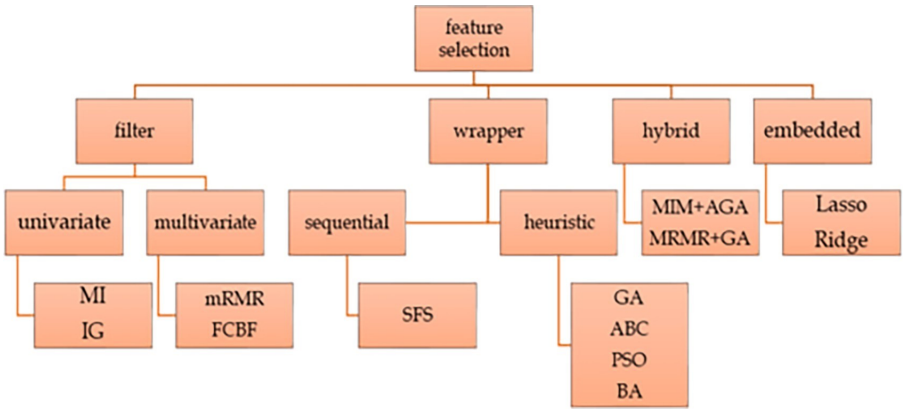


Figure 1.4: The taxonomy of the feature selection techniques.

- MI measures the correlation between the two variables. In other words, measures how much information one variable (X) knows about another one (Y). In gene selection, it measures the correlation between gene and classification category. The larger the value of MI, the more informative the genes are .
- IG is a statistical property that measures how infor mative a feature is. Highly related features to class are those with the information, while unrelated features give no information. To determine the value of IG, entropy value which is the impurity of the given samples is used. Then a threshold is set and features which value higher than the threshold are selected .

Multivariate evaluates features in the context of others, the most typically used techniques are:

- Minimum Redundancy and Maximum Relevance (mRMR) tends to select highly correlated features with the class and lowly between themselves. It may not proper to select both features that are highly relevant and highly correlated, as they wouldn't add more information due to high correlation, but they would increase the model complexity and make it susceptible to overfitting .
- Correlation-based Feature Selection (CFS) ranks features based on the correlation due to the heuristic evaluation functions. Features are evaluated according to the hypothesis "Good feature subset contains features that are highly correlated with the classification and yet uncorrelated to each other", which means a low correlation with the class refers to irrelevant features while informative features are strongly correlated .
- Fast Correlation Based Filter (FCBF) is a multivariate algorithm that bases on symmetrical uncertainty (SU) to select highly correlated features with the class. Then it applies heuristics to remove the redundant features and maintain relevant ones to the class .

### 1.3.2 Wrapper methods

Wrapper methods are using learning algorithms to select the optimal subset of features. They have better accuracy than filter methods, but they are intended for a particular learning algorithm, tend to overfit and they are very computationally intensive, as the model has to train each subset. This approach can be categorized into sequential selection algorithms and heuristic search algorithms. Sequential selection algorithms remove or add one feature at a time based on the classifier performance until a subset of the desired k features is reached that gives maximum accuracy.

Common techniques are sequential forward selection and sequential backward selection. The most utilized heuristic search algorithms are [21]:

- Genetic Algorithm (GA)
- Artificial Bee Colony (ABC)
- Particle Swarm Optimization (PSO)
- Bat Algorithm (BA)

### 1.3.3 Embedded techniques

Embedded methods learn which features best contribute to the accuracy of the model while the model is created. The most common types of embedded feature selection methods are regularization methods [21].

### 1.3.4 Hybrid approach

The hybrid approach can be any combination of any number of same or different methods of feature selection to combine the advantages of both approaches and overcome or handle the drawback of each approach individually. A combination is usually a filter-wrapper approach that gets the benefit of fast computational of filter approach to remove redundant features and high performance of wrapper approach. It also less prone to overfitting than wrapper but it is classifier specific [21].

## Conclusion

Gene selection from microarray expression data is pivotal for enhancing the accuracy and interpretability of classification models. Given the vast dimensionality of microarray data, ranging from thousands to tens of thousands of genes, efficient feature selection methods are essential to identify the most relevant genes that contribute to the classification task. Techniques such as evolutionary methods, machine learning algorithms, and population-based stochastic search approaches have shown significant promise in addressing these challenges. By effectively removing redundant and irrelevant features, these methods not only improve the performance of classification models but also facilitate a deeper understanding of the underlying biological processes.
Ultimately, the successful application of these feature selection techniques can lead to more accurate predictions and insights, advancing the field of genomics and personalized medicine.

Chapter 2

# Machine and Deep learning

## Introduction

In recent years, artificial intelligence technology, particularly in the domain of machine learning, has witnessed remarkable and widespread advancements. The goal of machine learning is to develop models capable of automatically extracting patterns and knowledge from data without direct human intervention. This field encompasses a wide range of techniques and models, including logistic regression, decision trees, and artificial neural networks.
Among the branches of machine learning, "deep learning" stands out as one of the most advanced and effective methods. Deep learning relies on multi-layered artificial neural networks, which excel in extracting and analyzing information from data in a detailed and profound manner. Deep learning is considered the cornerstone in applications such as speech and image recognition, natural language processing, and medical diagnosis.
This remarkable advancement in machine learning and deep learning has opened up vast horizons for scientific research across various domains. With the increasing importance of these technologies, researchers are increasingly looking to utilize them in their scientific studies, aiming to solve complex problems and gain a deeper understanding of intricate phenomena in numerous scientific and applied fields.

## 2.1 Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression analysis, developed in the 1990s by Vladimir N. Vapnik and his colleagues. SVMs classify data by finding an optimal line or hyperplane that separates data points into different categories with the largest margin possible, making them robust in handling both linear and nonlinear classification tasks. They are widely used in various fields, including natural language processing, image classification, security domains, and cancer research, due to their excellent generalization performance and ability to detect subtle trends in complex data. SVMs can also analyze layered geophysical structures underground and predict the seismic liquefaction potential of soil, making them valuable in civil engineering applications [19].

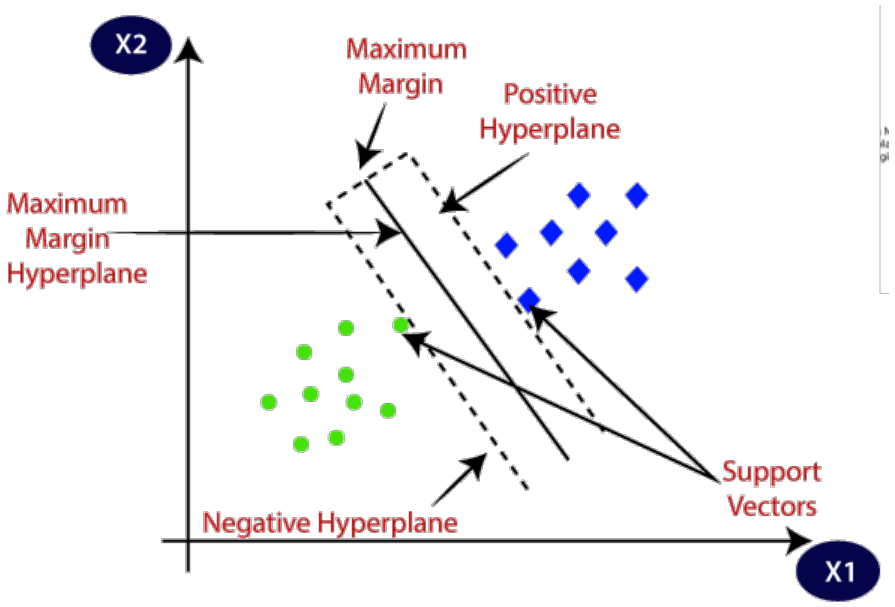


Figure 2.1: Classification of data by support vector machine (SVM).

### 2.1.1 How does SVM work ?

SVMs work by finding an optimal hyperplane that separates data points into different categories with the largest possible margin as shown in Figure 2.1 [19].
The key steps are:

- Mapping data to high-dimensional space: For non-linear data that is not easily separable, SVMs use kernel functions to map the data into a higher-dimensional feature space where the data becomes linearly separable.
- Finding the optimal hyperplane: SVMs determine the optimal hyperplane (decision boundary) that maximizes the margin between the closest data points of different classes. This hyperplane is defined by the weight vector w and bias term b, such that $wx + b = 0$.
- Identifying support vectors: The data points closest to the hyperplane are called support vectors. These support vectors define and determine the optimal hyperplane.
- Maximizing the Margin: SVMs aim to find the hyperplane that has the maximum margin, or distance, between the support vectors of different classes. This allows the SVM to generalize well to new, unseen data.
- Handling non-linear data: For non-linear data, SVMs use the "kernel trick" to efficiently map the data into higher dimensions without the computational expense of explicitly transforming the data.

### 2.1.2 Advantages and disadvantages

Here are the key advantages and disadvantages of support vector machines (SVMs):
Advantages [13][12]:

- Effective in high-dimensional spaces and can handle many features.

- Robustness to noise: SVMs are robust to noise in the data, as the decision boundary is determined by the support vectors, which are the closest data points to the boundary.
- Generalization: SVMs have good generalization performance, which means that they are able to classify new, unseen data well.
- Versatility: SVMs can be used for both classification and regression tasks, and it can be applied to a wide range of applications such as natural language processing, computer vision, and bioinformatics.
- Sparse solution: SVMs have sparse solutions, which means that they only use a subset of the training data to make predictions. This makes the algorithm more efficient and less prone to overfitting.

Disadvantages [13]:

- Lack of transparency in the results, as the model is not easily interpretable.
- Long training times, especially for large datasets.
- Difficulty in handling overlapping or noisy classes.
- Challenging to choose the appropriate kernel function.
- Hyperparameter tuning (e.g., cost, gamma) can be difficult.
- Cannot directly provide probability estimates, only class predictions.

## 2.2 What is Random Forest ?

A Random Forest is a supervised learning algorithm that creates an ensemble of multiple decision trees, combining them to make more accurate predictions. The algorithm operates by constructing a multitude of decision trees during training, where for classification tasks, the output is the class selected by most trees, and for regression tasks, the mean or average prediction of the individual trees is returned. Random Forests correct for decision trees' tendency to overfit and provide a robust method for prediction across various applications [17].

### 2.2.1 How does Random Forest work ?

Here is an explanation of how the Random Forest algorithm works (see Figure 2.2):

1. Bagging: Random Forest uses the bagging (bootstrap aggregating) technique, where it creates multiple samples of the training data with replacement. This introduces randomness and diversity into the model.
2. Decision Tree Building: For each sample, a decision tree is built. When splitting a node, Random Forest considers a random subset of features instead of all features, further adding to the diversity of the trees.

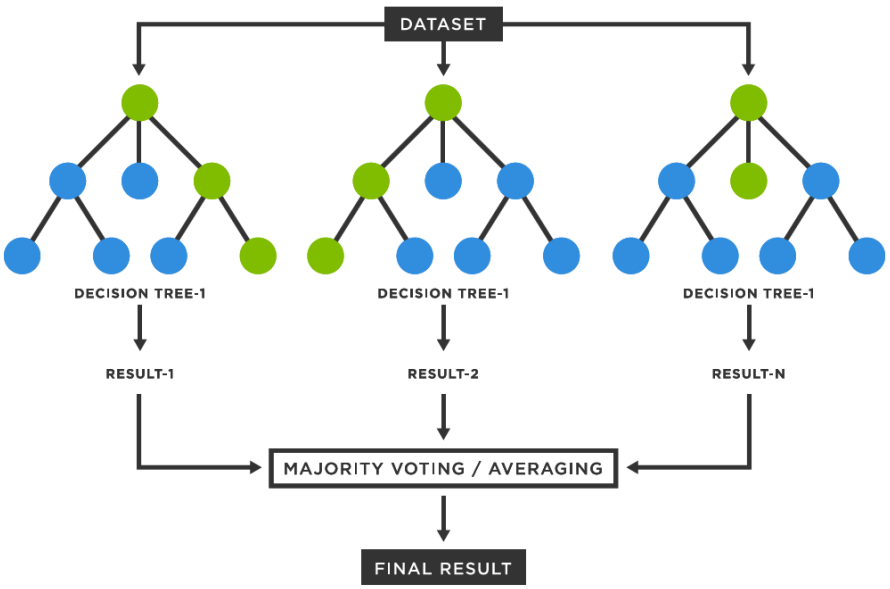


Figure 2.2: Random Forest Simplified.

3. Prediction Aggregation: Once the forest of decision trees is built, new data is passed through all the trees. For classification, the final prediction is the majority vote across all the trees. For regression, the final prediction is the average of all the tree predictions.

The randomness introduced at the feature selection stage, combined with the ensemble of multiple decision trees, makes Random Forest more robust and less prone to overfitting compared to a single decision tree.
Additionally, Random Forest provides a way to measure the importance of each feature in the prediction, which can be useful for feature selection [17].

### 2.2.2 Random Forest advantages and disadvantages

The key advantages and disadvantages of the Random Forest algorithm are:
Advantages [15]:

- Robust and Accurate: Random Forest is a robust algorithm that can handle noisy data and outliers. It is one of the most accurate machine learning algorithms, performing well on both classification and regression tasks.
- Handles Diverse Data: Random Forest can work with a variety of data types, including numeric and categorical variables. It does not require feature scaling and can handle missing values.
- Feature Importance: Random Forest provides a measure of feature importance, which can be useful for feature selection and understanding the underlying patterns in the data.
- Scalable and Parallelizable: Random Forest can handle large, high-dimensional datasets. Its training process can be parallelized, making it efficient for large-scale applications.
- Cannot directly provide probability estimates, only class predictions.

Disadvantages [15]:

- Interpretability: Random Forest models can be less interpretable than a single decision tree, as the prediction involves the combined output of multiple trees.
- Computational Complexity: Training a Random Forest model, especially with a large number of trees or on a large dataset, can be computationally expensive and require significant memory resources.
- Prediction Time: Making predictions with a Random Forest model can be slower compared to some other algorithms, particularly for real-time or latency-sensitive applications.
- Overfitting Potential: While Random Forest is generally less prone to overfitting, it can still overfit the data if the number of trees or the depth of the trees is too high.

## 2.3 What is Multi-Layer Perceptrons (MLPs) ?

A Multi-Layer Perceptron (MLP) is a type of artificial neural network with multiple layers, including input, output, and hidden layers. It uses backpropagation to adjust weights, enabling it to learn and minimize errors. MLPs are known for their ability to handle non-linear data and are essential in deep learning. They consist of interconnected neurons with various activation functions, allowing them to process complex information. MLPs excel at distinguishing non-linearly separable data, unlike the original perceptron. MLPs are crucial in modern machine learning for tasks like image processing and sentiment analysis [9].

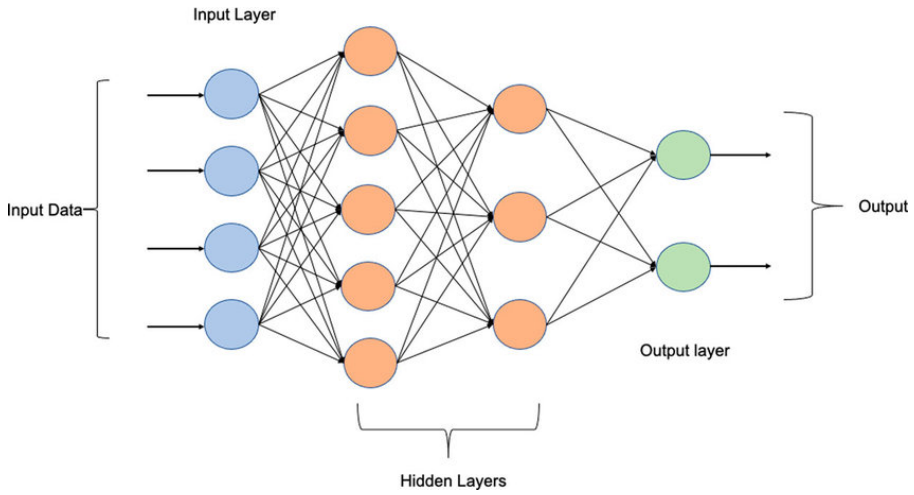


Figure 2.3: Multi-layer perceptron basic Architecture.

### 2.3.1 How does MLPs works ?

MLPs are a type of artificial neural network consisting of multiple layers of interconnected nodes called neurons. They have an input layer, one or more hidden layers, and an output layer.
The key aspects of how MLPs work are:

1. Information Flow: Data flows through the network in a feedforward manner, from the input layer, through the hidden layers, to the output layer.

2. Weighted Connections: Neurons in adjacent layers are connected by weighted connections, which determine how much influence one neuron's output has on another neuron's input.

3. Activation Functions: Each neuron applies a non-linear activation function to the weighted sum of its inputs before passing the result to the next layer. Common activation functions include sigmoid, tanh, and ReLU.

4. Learning via Backpropagation: MLPs are trained using the backpropagation algorithm, which adjusts the weights between neurons to minimize the error between the network's outputs and the desired outputs.

5. Ability to Learn Complex Patterns: The multiple hidden layers allow MLPs to learn complex, non-linear relationships in data, making them powerful models for tasks like classification, regression, and pattern recognition.

In summary, MLPs use a feedforward architecture with weighted connections and non-linear activation functions to learn complex patterns in data through backpropagation, a supervised learning technique [9].

### 2.3.2 Advantages and disadvantages of MLPs

Advantages [10]:

1. Versatility: MLPs are flexible and can handle various types of input data, including continuous or categorical variables, making them adaptable to different scenarios.

2. Generalization: MLPs can effectively generalize to new, previously unseen data when properly trained, making them suitable for real-world applications.

3. Scalability: MLPs can be scaled up by adding additional hidden layers or nodes to enhance model performance on complex tasks.

4. Nonlinear Modeling: MLPs can model complex nonlinear interactions between inputs and outputs, making them valuable for a wide range of applications.

5. Black Box Model: MLPs are often referred to as "black boxes" due to their ability to make predictions without revealing the underlying decision-making process, which can be advantageous in certain contexts.

Disadvantages [10]:

1. Overfitting: MLPs can easily overfit the training data if the model is too complex or if the training data is limited, leading to reduced generalization performance.

2. Slow Training: Training an MLP can be computationally time-consuming, especially for large datasets or deep networks, impacting the efficiency of the learning process.

3. Hyperparameter Optimization: Optimizing hyperparameters such as the number of nodes, layers, activation functions, and learning rate is crucial for achieving optimal performance with MLPs.

4. Limited Interpretability: MLPs' complex architecture can make it challenging to interpret how the network makes predictions, limiting the transparency of the model.

5. Dependency on Training Data: MLPs heavily rely on the availability of representative training data, which can be a limitation in scenarios with limited or biased data.

## 2.4 What is K-nearest neighbor (KNN) ?

The K-nearest neighbor (KNN) algorithm is a supervised machine learning method used for classification and regression tasks. It operates by assigning new data points to the majority set within its neighbors for classification or predicting based on the average of values closest to the query point for regression. KNN is versatile, non-parametric, and adaptable across various fields like finance, healthcare, and recommendation systems due to its simplicity and accuracy [7].

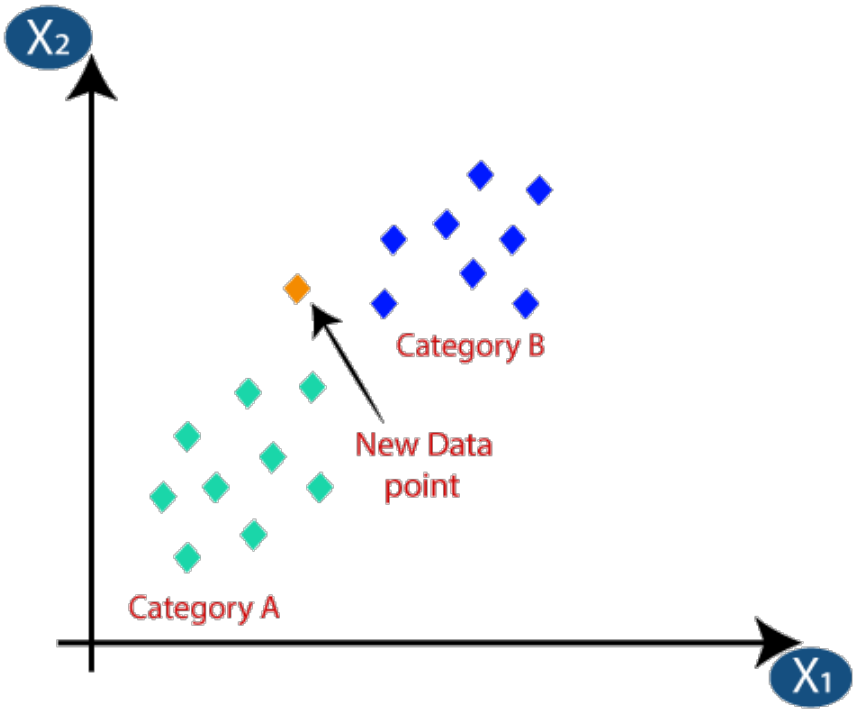


Figure 2.4: Classification of data by k-nearest neighbor (KNN).

### 2.4.1 How does KNN works ?

Here is an explanation of how the K-nearest neighbor (KNN) algorithm works [7][8]:

1. KNN is a supervised machine learning algorithm that can be used for both classification and regression tasks.
2. For classification, KNN assigns a new data point to the majority class among its K nearest neighbors.
3. For regression, KNN predicts the value of a new data point based on the average of the values of its K nearest neighbors.
4. The algorithm works by:
    - Storing the entire training dataset.

- Calculating the distance between the new data point and all the training examples using a distance metric like Euclidean distance.
- Identifying the K nearest neighbors to the new data point based on the calculated distances.
- For classification, assigning the most common class label among the K neighbors.
- For regression, predicting the value as the average of the values of the K neighbors.

5. KNN is a non-parametric algorithm, meaning it does not make any assumptions about the underlying data distribution. This makes it versatile and adaptable across various domains.
6. The choice of the K value, the number of nearest neighbors considered, is an important hyperparameter that needs to be tuned for optimal performance.
7. KNN is considered a "lazy" algorithm as it simply stores the training data and performs the necessary calculations when a new data point needs to be classified or predicted, without building any generalized model.

### 2.4.2 Advantages and disadvantages of KNN

Advantages [18]:

- Simplicity and Effectiveness: KNN is a simple and powerful algorithm, making it a popular choice in machine learning tasks.
- Non-Parametric Approach: KNN is a non-parametric algorithm that does not make assumptions about the underlying data distribution.
- Versatility: KNN is widely used in various fields like healthcare, scientific studies, finance, and recommendation systems.
- Interpretable Results: KNN provides interpretable results by looking at the nearest neighbors.
- Adaptability: KNN adjusts easily to new training data and does not require building an explicit model.

Disadvantages [18]:

- High Computational Cost: KNN can be computationally expensive, especially for large datasets, leading to time complexity issues.
- Sensitivity to Hyperparameters: The choice of K value and distance metric can significantly impact KNN's performance.
- Curse of Dimensionality: KNN's performance degrades as the number of features increases.
- Imbalanced Datasets: KNN can struggle with imbalanced data and missing values, affecting its accuracy.
- Lack of Explanation: KNN does not provide explanations or confidence for its predictions, limiting interpretability in some applications.
- Slower Performance: KNN can be slower compared to other algorithms as the dataset grows.

## 2.5 What is Convolutional Neural Networks (CNNs) ?

Convolutional Neural Networks (CNNs) are specialized deep learning algorithms used for tasks like image classification, detection, and segmentation. They work by applying convolution and pooling layers to extract features from input data, enabling them to identify patterns regardless of position or scale. CNNs, inspired by the human visual cortex, consist of layers like input, convolutional, pooling, and fully connected layers. They excel in computer vision tasks and can be applied to various domains beyond image analysis [6].

### 2.5.1 How does CNNs work ?

Convolutional neural networks (CNNs) are a type of deep learning algorithm that are particularly well-suited for analyzing visual data, such as images and videos. The key distinguishing features of CNNs are:

- Convolutional layers: The core building block of a CNN is the convolutional layer, where the majority of the computation occurs. In this layer, the input image is convolved with a set of learnable filters (also called kernels or feature detectors) that extract various features from the image, such as edges, shapes, and textures. These filters are applied across the entire image, creating feature maps that highlight the presence of the detected features.
- Parameter sharing: The weights of the filters in the convolutional layers are shared across the entire image, rather than having a unique set of weights for each pixel. This reduces the number of parameters in the network and allows the CNN to efficiently process high-resolution images.
- Pooling layers: After the convolutional layers, pooling layers are often used to reduce the spatial dimensions of the feature maps, while preserving the most important features. This helps to make the network more robust to small translations and distortions in the input.
- Fully connected layers: The final layers of a CNN are typically fully connected layers, which take the high-level features extracted by the convolutional and pooling layers and use them for classification or regression tasks.
- Backpropagation and parameter optimization: During training, the weights of the filters in the convolutional layers are optimized using backpropagation and gradient descent, allowing the CNN to automatically learn the most effective features for the given task.

By leveraging the spatial structure of the input data and the ability to learn effective features, CNNs have demonstrated state-of-the-art performance in a wide range of visual recognition tasks, such as image classification, object detection, and semantic segmentation [14].

### 2.5.2 Advantages and disadvantages of CNNs

Advantages [2]:

- Good at detecting patterns and features in images, videos, and audio signals.
- Robust to translation, rotation, and scaling invariance.

- Automatic feature extraction, no need for manual feature engineering.
- Can handle large amounts of data and achieve high accuracy.

Disadvantages [2]:

- Computationally expensive to train and require a lot of memory.
- Can be prone to overfitting if not enough data or proper regularization is used.
- Requires large amounts of labeled data.
- Interpretability is limited, it's hard to understand what the network has learned.
- Limited effectiveness for sequential data like text or speech.
- Training takes a long time.

## 2.6 What is AutoEncoder ?

An Autoencoder is a type of artificial neural network used for supervised learning to efficiently compress input data into essential features and reconstruct the original input from this compressed representation. It consists of an encoder that compresses the data into a latent space and a decoder that reconstructs the input. Autoencoders aim to discover latent variables in data and are used for tasks like data compression, image denoising, anomaly detection, and feature extraction. Variants like Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs) adapt the architecture for generative tasks like image generation [16].

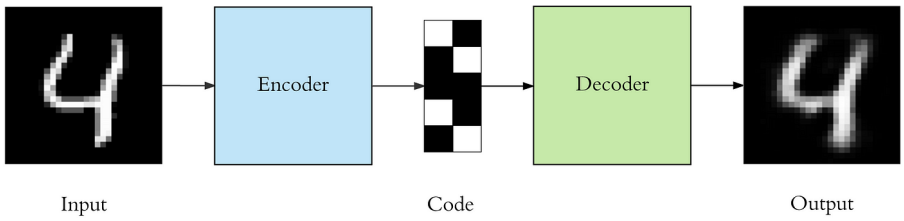


Figure 2.5: Schema of Autoencoder architecture.

### 2.6.1 Types of Autoencoders

1. Standard Autoencoder : This is the basic form of an autoencoder, consisting of an encoder and a decoder. It aims to reconstruct the input data while learning a compressed representation in the latent space. It's often used for dimensionality reduction and feature learning.
2. Denoising Autoencoder: Denoising autoencoders are trained to reconstruct clean data from noisy input. By adding noise to the input data and training the autoencoder to remove it, they learn robust features and are useful for data denoising and feature extraction.
3. Sparse Autoencoder: Sparse autoencoders introduce sparsity constraints on the activations of neurons in the hidden layers. This encourages the network to use only a subset of neurons, resulting in a more compact representation. They are useful for feature selection and creating sparse representations.

4. Variational Autoencoder (VAE): VAEs are probabilistic autoencoders that model the data distribution in the latent space. They aim to generate new data samples by sampling from the learned distribution. VAEs are used for generative modeling and have applications in generating new images, text, and more.

5. Contractive Autoencoder: Contractive autoencoders impose a penalty on the model's weights to ensure that the learned representations are robust to small variations in the input data. They are helpful in tasks where stability and robustness are critical, such as anomaly detection.

6. Adversarial Autoencoder (AAE): AAEs combine autoencoders with generative adversarial networks (GANs). They use a GAN-like architecture to learn the distribution of the latent space, making them suitable for generative tasks while still performing encoding and decoding.

7. Sequence-to-Sequence Autoencoder: These autoencoders are designed for sequential data, such as time series or text. They encode and decode sequences, making them suitable for tasks like sequence-to-sequence prediction, machine translation, and text summarization.

8. Variational Recurrent Autoencoder (VRAE): VRAEs combine VAEs with recurrent neural networks (RNNs). They can generate sequences while capturing the uncertainty in the data, making them useful for tasks like generating diverse sequences and handling missing data.

9. Stacked Autoencoders: Stacked autoencoders consist of multiple autoencoders stacked on top of each other. Each layer learns higher-level features. They are used for deep feature learning and have applications in image recognition and classification.

10. Convolutional Autoencoder: Convolutional autoencoders use CNN layers in both the encoder and decoder. They are tailored for image data and are effective in tasks like image denoising, super-resolution, and image generation [5].

### 2.6.2 How does AutoEncoder works ?

Autoencoders are a type of neural network architecture designed to efficiently compress (encode) input data into a low-dimensional latent representation and then reconstruct (decode) the original input from this compressed code.The key components of an Autoencoder are:

1. Encoder: This part of the network compresses the high-dimensional input data into a lower-dimensional latent space representation.

2. Bottleneck: The compressed latent space representation, which is the smallest part of the network and contains the most essential features of the input.*

3. Decoder: This part of the network takes the latent space representation and reconstructs the original input, essentially decompressing the data.

The Autoencoder is trained in an unsupervised manner to minimize the reconstruction error between the input and the output. This forces the network to learn an efficient encoding of the data in the bottleneck layer.By constraining the size of the bottleneck to be smaller than the input, the Autoencoder is encouraged to learn a compressed, low-dimensional representation that captures the most salient features of the data.Variants like Variational Autoencoders (VAEs) and Adversarial Autoencoders (AAEs) adapt this basic architecture for generative tasks like image synthesis.The key goal of an

Autoencoder is to discover the minimum number of important features (latent variables) that can accurately reconstruct the original input data [3].

### 2.6.3 Advantages and disadvantages of AutoEncoder

The key advantages and disadvantages of Autoencoders are:
Advantages [1]

- Automatic feature learning - can automatically learn features from data, eliminating the need for manual feature engineering.
- Effective at processing large and complex datasets, making them powerful for big data analytics.
- Can handle both structured and unstructured data types.
- Can provide meaningful predictions even with incomplete data.
- Effective at processing sequential data like time series, speech, and text.

Disadvantages [1]:

- Computationally expensive to train and require substantial computational resources like powerful GPUs.
- Can be prone to overfitting if not enough data or proper regularization is used.
- Require large amounts of labeled data.
- Interpretability is limited, it's hard to understand what the network has learned.*
- Training takes a long time.

## Conclusion

The use of neural networks (Autoencoders) to select genes from microarray expression data is considered an important advance in improving the accuracy and efficiency of classification models. These networks rely on deep learning techniques to reduce dimensionality and extract important features from complex data, leading to better performance of predictive models and a deeper understanding of biological processes.
Thanks to the evolution of deep learning, significant progress can now be made in understanding gene functions and diagnosing diseases with greater accuracy and efficiency. This advancement is not merely an improvement in technology but signifies a qualitative leap in scientific research and our progress towards a deeper understanding of genes and their impact on human health.

Chapter 3

# Utilizing Deep Learning Models for Gene Selection from Microarray Expression Data

## Introduction

This chapter focuses on implementing the proposed approach for gene selection from microarray gene expression data using deep learning models. It overviews the resources, programming language, and development environment used. The chapter outlines the step-by-step implementation process, including dataset preprocessing, machine and deep learning techniques selection. The second part presents the experimental results, evaluating the performance of the model and comparing different techniques. Overall, this chapter serves as a guide for implementing the proposed approach, providing insights into the methodology and enabling further research in gene selection from microarray gene expression data using deep learning.

## 3.1 Related Work

DNA Microarray technology (also known as DNA chips or gene chips) is a powerful tool that helps researchers monitor the gene expression level in an organism. Microarray data analysis provides valuable results which contribute towards solving gene expression profile problems. One the most important applications of Microarray data analysis is cancer classification. Cancer may be a genetic disease; the analysis of cancer pathobiology is the analysis of genes that cause cancer, i.e. the gene whose mutation is responsible for cancer. This reflects the changes in the expression level of various genes. However, classifying the gene expression profile is a challenging task and considered as (NP)-Hard problem . Hence, not all genes contribute to the presence of cancer. A vast number of genes are irrelevant or insignificant to clinical diagnosis. Therefore, incorrect diagnoses can be reached when all the genes are used in Microarray gene expression classification. There are two main issues related to the analysis of Microarray data; first, the dataset in Microarray is high-dimensional which means it contains several thousand genes (features) and it has low data sparsity, meaning it has a low number of samples, usually tens of samples. Second, gene expression data has a high complexity; genes are directly or indirectly correlated to each other. Standard machine learning

methods did not perform well because these methods are best suited when there are more samples than features.
In an attempt to overcome these issues, dimension reduction or feature (gene) selection algorithms have been applied. Generally, the gene selection methods are categorized into three categories: filter, wrapper, and embedded methods. The filter method involves each feature being evaluated individually by using its general statistical properties. The wrapper method uses learning techniques to select the optimal feature subset. The quality of the wrapper technique is estimated by the accuracy of the specific classifier. The wrapper approach usually employs evolutionary or bio-inspired algorithms to guide the search process. The embedded method searches for the optimal feature subset and is built in the classifier; the search space is combined in the hypothesis space. Recently, hybrid and ensemble methods were added to the general framework of feature selection. A hybrid approach is built to take advantage of both filter and wrapper approaches. Thus, it combines the computational efficiency of the filter approach with the high performance of the wrapper approach.
Bio-inspired evolutionary methods have widely applied the wrapper approach in feature selection for Microarray data analysis and demonstrate a superior performance [22].

## 3.2 Our Method

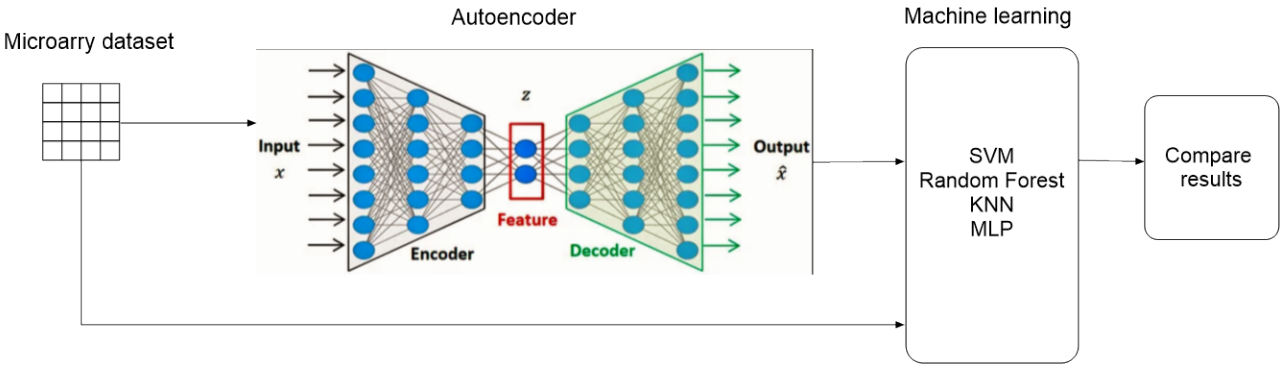


Figure 3.1: The global architecture.

### 3.2.1 Our methodology

Gene selection from microarray gene expression data is particularly crucial in cancer classification, as it helps identify the most important set of genes that can lead to accurate classification of cancer. The aim of this study is to utilize AutoEncoder models for this gene selection task.
In the encoding stage, the original data is transformed into a compressed representation using the ReLU activation function. This is achieved by converting negative values in the data to zero, while retaining positive values unchanged. This compressed representation contains important and useful features extracted from the original data, allowing for the effective representation of key aspects and primary information without significant loss of information.
We implement different machine learning models (SVM ,KNN ,RF ,MLP) on data extracted (selected genes) from the Autoencoder and the original data, followed by comparison of results to evaluate the Autoencoder performance .

### 3.2.2 Microarray data

The data generated by Microarray experiments are arranged and kept as large matrices (N × M). Each Microarray data matrix comprises of the samples illustrated in rows and the genes (features) in the columns as shown in Table 1.

| | G1 | G2 | ... | GM |
|---|---|---|---|---|
| S1 | x11 | x12 | ... | x1M |
| S2 | x21 | x22 | ... | x2M |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| SN | xN1 | xN2 | ... | xNM |

Table 3.1: Microarray data matrix format.

Microarray data is of the form of N by M matrix and is very huge. Where N is number of rows and indicates the samples, M is number of columns and indicates the genes. Every single cell has a certain value of gene expression in a sample xij denotes the expression level of the gene j and the condition or sample i. Where j ranges between 1 and M, and i from 1 to N [24].
Microarray data use in our study were downloaded from a previous study https://csse.szu.edu.cn/staff/zhuxx/Datasets.html

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 2 | 3.839861 | 0 | 24.04692 | 7.905669 | 17.90997 | 13.45714 | 0 | 4.975826 | 2.473785 | 5.979985 | 0 | 0 | 0 | 4.580473 | 0 | 23.80817 |
| 3 | 4.517811 | 0 | 21.33401 | 6.516234 | 20.50557 | 13.58315 | 0 | 4.610992 | 4.430159 | 7.818872 | 0 | 0 | 0 | 6.160771 | 0 | 28.11932 |
| 4 | 6.85046 | 0 | 34.50131 | 10.03505 | 28.29859 | 20.41666 | 0 | 7.219848 | 5.175944 | 9.850866 | 0 | 0 | 0 | 8.099148 | 0 | 38.38424 |
| 5 | 4.280307 | 0 | 20.33156 | 5.906265 | 17.88835 | 12.31069 | 0 | 4.422999 | 3.662994 | 6.694126 | 0 | 0 | 0 | 5.271389 | 0 | 24.61346 |
| 6 | 4.249156 | 0 | 21.19797 | 6.405954 | 17.76131 | 12.5848 | 0 | 4.800563 | 3.402012 | 6.452298 | 0 | 0 | 0 | 5.065952 | 0 | 24.67905 |
| 7 | 4.678671 | 0 | 28.14524 | 8.944498 | 21.75882 | 16.22989 | 0 | 6.029787 | 3.679595 | 7.326771 | 0 | 0 | 0 | 6.001855 | 0 | 29.61986 |
| 8 | 5.174715 | 0 | 27.75602 | 8.636828 | 22.34323 | 16.19557 | 0 | 5.987959 | 3.797747 | 7.657112 | 0 | 0 | 0 | 6.269964 | 0 | 30.22207 |
| 9 | 3.556911 | 0 | 21.29498 | 6.862957 | 16.43435 | 12.30536 | 0 | 4.546751 | 2.588097 | 5.519258 | 0 | 0 | 0 | 4.465137 | 0 | 22.1781 |
| 10 | 6.236949 | 0 | 31.62724 | 9.553729 | 26.86421 | 18.90197 | 0 | 6.865654 | 5.019405 | 9.675694 | 0 | 0 | 0 | 7.744433 | 0 | 36.58131 |
| 11 | 7.001209 | 0 | 36.20128 | 10.89308 | 29.95868 | 21.4805 | 0 | 7.810993 | 5.377606 | 10.4991 | 0 | 0 | 0 | 8.579531 | 0 | 40.71703 |
| 12 | 6.056415 | 0 | 31.62221 | 9.631957 | 26.02851 | 18.85093 | 0 | 6.701936 | 4.513146 | 8.9102 | 0 | 0 | 0 | 7.517235 | 0 | 35.01035 |
| 13 | 7.082032 | 0 | 39.71295 | 12.32506 | 30.95247 | 22.8194 | 0 | 8.244793 | 5.135604 | 10.38193 | 0 | 0 | 0 | 8.665105 | 0 | 41.63302 |
| 14 | 6.559806 | 0 | 31.86619 | 9.306338 | 26.01661 | 18.58242 | 0 | 6.688837 | 4.728529 | 9.194542 | 0 | 0 | 0 | 7.349856 | 0 | 35.33345 |
| 15 | 5.383187 | 0 | 26.0923 | 7.796774 | 24.08013 | 16.2816 | 0 | 5.571684 | 5.091091 | 8.979658 | 0 | 0 | 0 | 7.322834 | 0 | 32.92473 |
| 16 | 6.812194 | 0 | 36.73535 | 11.19801 | 29.18675 | 21.22386 | 0 | 7.80093 | 5.223092 | 9.998606 | 0 | 0 | 0 | 8.296007 | 0 | 39.70109 |
| 17 | 4.792292 | 0 | 24.06771 | 7.047134 | 21.1081 | 14.76686 | 0 | 5.208789 | 4.44903 | 7.718379 | 0 | 0 | 0 | 6.460526 | 0 | 29.12152 |
| 18 | 4.040301 | 0 | 18.94194 | 5.8331 | 17.72685 | 11.82333 | 0 | 4.332439 | 4.048644 | 6.752339 | 0 | 0 | 0 | 5.464778 | 0 | 24.95665 |
| 19 | 5.73058 | 0 | 29.07862 | 8.696566 | 24.05729 | 17.20905 | 0 | 6.395794 | 4.68486 | 8.58423 | 0 | 0 | 0 | 7.027691 | 0 | 33.23388 |
| 20 | 7.383297 | 0 | 35.46505 | 10.47851 | 29.78563 | 20.82301 | 0 | 7.410206 | 5.042089 | 10.50195 | 0 | 0 | 0 | 8.494238 | 0 | 39.67849 |
| 21 | 6.304728 | 0 | 31.3029 | 9.266648 | 28.23298 | 19.16698 | 0 | 6.598119 | 6.049369 | 10.47106 | 0 | 0 | 0 | 8.59469 | 0 | 38.73715 |
| 22 | 6.065281 | 0 | 28.95197 | 8.736653 | 26.5914 | 17.99443 | 0 | 6.334775 | 5.750902 | 10.04588 | 0 | 0 | 0 | 8.047892 | 0 | 36.71815 |
| 23 | 7.201243 | 0 | 33.65773 | 9.800052 | 29.35989 | 20.12905 | 0 | 7.166995 | 5.708666 | 10.83535 | 0 | 0 | 0 | 8.649639 | 0 | 39.80015 |
| 24 | 5.442058 | 0 | 26.31504 | 7.88134 | 23.90943 | 16.02095 | 0 | 5.504909 | 5.235831 | 9.027652 | 0 | 0 | 0 | 7.085712 | 0 | 32.95963 |
| 25 | 6.703013 | 0 | 31.85604 | 9.271043 | 27.68939 | 19.2472 | 0 | 6.849667 | 5.223404 | 10.05759 | 0 | 0 | 0 | 8.13173 | 0 | 37.54189 |

Figure 3.2: data after reduction.

### 3.2.3 step 2: Autoencoder

This code builds a coding model using a neural network with the TensorFlow/Keras library. The steps are as follows:

#### 3.2.3.1 Reading the Data:

Pandas library is used to read a CSV file containing data. Data is divided into features and labels.

#### 3.2.3.2 Building the model:

- The model is built using Dense layers in the Keras library.

- The number of layers, the size of each layer, and the activation function (ReLU activation function is used here) are specified.

#### 3.2.3.3 Compiling the model:

The model is compiled using the 'Adam' optimizer and `'binary_crossentropy'` loss function.

#### 3.2.3.4 Training the Model:

- The model is trained on the processed data.
- The number of epochs, batch size, and shuffle option are specified.

#### 3.2.3.5 Using the Model:

- The model is used to encode the available data.
- The encoded data is converted to a DataFrame using Pandas.

#### 3.2.3.6 Saving the data:

The encoded data is saved in an Excel file.
The code : Annex 1

### 3.2.4 Performance evaluation

#### 3.2.4.1 SVM Model

An SVM model on encrypted data and then evaluates the model's performance using several metrics and plots a confusion matrix.
Reading the Data:

- The encrypted data is read from an Excel file using the Pandas library.
- Features and labels are extracted from the data.

Data Splitting:

- The data is split into training and testing sets using the `train_test_split` function from the scikit-learn library.

Model Initialization:

- An SVM model is created using the scikit-learn library.

Model Training:

- The model is trained on the training data using the fit function.

Model Evaluation:

- The trained model is used to predict labels for the test data.
- A set of metrics is calculated to evaluate the model's performance, such as accuracy.

The code : Annex 2

#### 3.2.4.2 Random Forest Model

Data Reading :

- Reads data from a CSV file named 'Ovarian.csv' using the Pandas library.

Feature and Label Extraction:

- Extracts features and labels from the dataset.

Data Splitting:

- Splits the data into training and testing sets using the `train_test_split` function from scikit-learn.

Random Forest Model Building:

- Builds a Random Forest model using the RandomForestClassifier class from scikit-learn.
- Sets the number of estimators to 100 and the random state to 42.

Model Training:

- Trains the Random Forest model on the training data using the fit method.

Model Prediction:

- Uses the trained Random Forest model to make predictions on the test data using the predict method.

Model Evaluation:

- Calculates the accuracy of the Random Forest model using the `accuracy_score` function from scikit-learn.

The code : Annex 3

#### 3.2.4.3 KNN Model

Data Loading :

- Reads data from a CSV file named 'Ovarian.csv' using the Pandas library.

Feature and Label Extraction:

- Extracts features and labels from the dataset.

Data Splitting :

- Splits the data into training and testing sets using the `train_test_split` function from scikit-learn.

Feature Scaling:

- Standardizes the features by removing the mean and scaling to unit variance using StandardScaler from scikit-learn.

- Standardization ensures that features are on the same scale, which is important for the KNN algorithm.

KNN Classifier Initialization:

- Initializes the KNN classifier using the KNeighborsClassifier class from scikit-learn.
- The number of neighbors (K) is set to 5, but you can adjust this value as needed.

Model Training:

- Trains the KNN classifier on the scaled training data using the fit method.

Model Prediction:

- Uses the trained KNN classifier to make predictions on the scaled test data using the predict method.

Model Evaluation:

- Calculates the accuracy of the KNN classifier using the `accuracy_score` function from scikit-learn.

The code : Annex 4

#### 3.2.4.4 MLP Model

Data Reading :

- Reads data from a CSV file named 'Ovarian.csv' using the Pandas library.

Feature and Label Selection:

- Selects features and labels from the dataset.

Data Splitting:

- Splits the data into training and testing sets using the `train_test_split` function from scikit-learn.

MLP Model Building:

- Builds a Multi layer Perceptron (MLP) model using the Sequential API from TensorFlow/Keras.
- The model consists of an input layer with 128 units and ReLU activation function, a hidden layer with 64 units and ReLU activation function, and an output layer with 1 unit and sigmoid activation function for binary classification.

Model Compilation:

- Compiles the model using the Adam optimizer and binary cross-entropy loss function.
- The accuracy metric is used to monitor the model's performance during training.

Model Training:

- Trains the model on the training data for 10 epochs with a batch size of 32.

- Utilizes 20% of the training data for validation.

Model Evaluation:

- Predicts the labels for the test data.
- Converts the predicted probabilities into binary predictions using a threshold of 0.5.
- Calculates the accuracy of the model using the `accuracy_score` function from scikit-learn.

The code : Annex 5

### 3.2.5 Experimental setup

This experiment aims to study the effect of Autoencoder on the performance of four classification algorithms on two types of data: colon data and ovarian data. The data was reduced to 50%, 25%, and 10% using Autoencoder , and then the performance of the algorithms was evaluated on each dataset.
Methodology:

1. Data Collection: Two datasets were collected: colon data and ovarian data.
2. Autoencoder: Autoencoder was used to reduce the data size to 50%, 25%, and 10%.
3. Performance Evaluation: The performance of four classification algorithms was evaluated on each dataset (SVM, KNN, MLP, RF).
4. Results Analysis: The evaluation results were analyzed to identify the best classification algorithm for each dataset.

## 3.3 Results and comparison

### 3.3.1 charts

#### 3.3.1.1 SVM Model

**Ovarian**
The ROC curve is a graphical plot that shows the performance of a binary classifier model at different threshold values. The ROC curve plots the true positive rate (TPR) on the y-axis against the false positive rate (FPR) on the x-axis FIGURE 3.3 .
Curve analysis:
In the image provided, we can see that there are four ROC curves representing four different models (all, 50%, 25%, 10%). Each curve is characterized by an area under the curve (AUC) that indicates the overall model performance. The larger the AUC, the better the model performance.
Key findings:
All models obtained a perfect AUC (1): This indicates that all curves reach the upper left corner of the graph, indicating that 100% true positive rate and 0% false positive rate are achieved at a certain threshold. This scenario raises some doubts, as it is unrealistic in most machine learning applications. It is likely that the models may have been overfitting with the data on which they were trained.

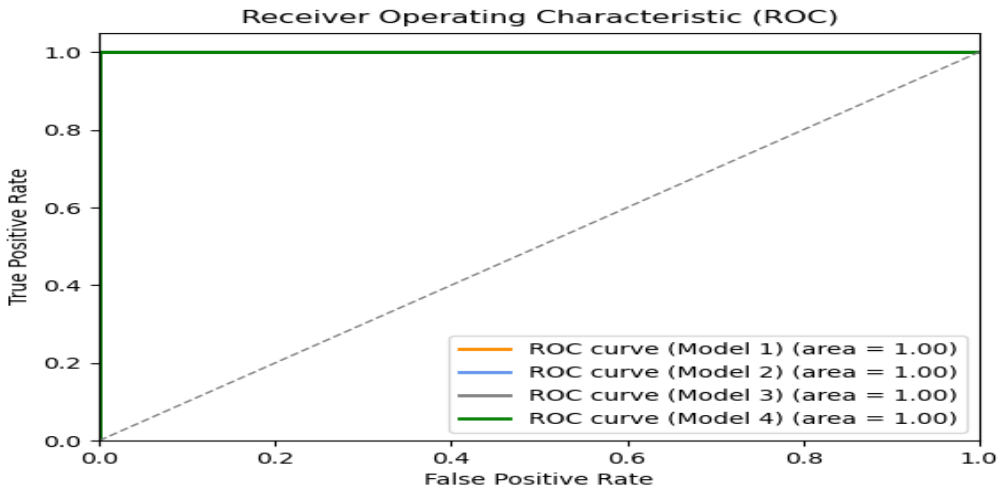


Figure 3.3: SVM chart of Ovarian.

There are no fundamental differences between the models: The curves do not show any noticeable differences between the four models, as they all overlap completely.
While the ROC curves themselves do not indicate any statistically significant trends or patterns between models, the ideal AUC scores for all models raise questions about their ability to generalize to new data.

**Colon**

The ROC curve has four curves that correspond to four different models. The area under the curve

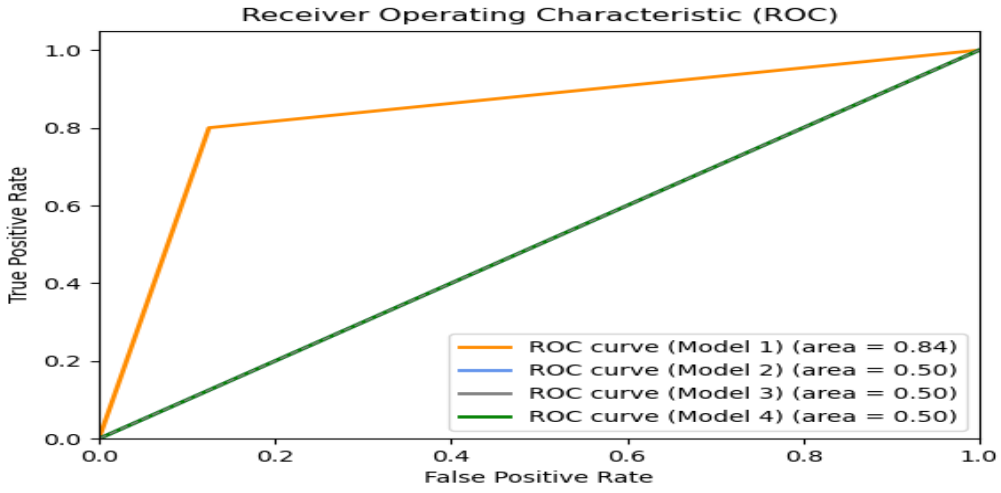


Figure 3.4: SVM chart of Colon.

(AUC) is also determined for each model. A larger AUC indicates better model performance.
The ROC curve for Model 1 has the largest area under the curve (0.84) among the four models. This means that Model 1 has the best overall performance in terms of balancing the true positive rate and false positive rate.
Model 2, Model 3, and Model 4 all have the same area under the curve (0.50), which is equal to the diagonal line. This is the line where TPR equals FPR and corresponds to a random classifier. In other words, the performance of these three models is no better than random guessing.
Overall, the ROC curves show that Model 1 significantly outperforms the other three models.

#### 3.3.1.2 Random Forest Model

**Ovarian**

All four models achieve a high level of performance.

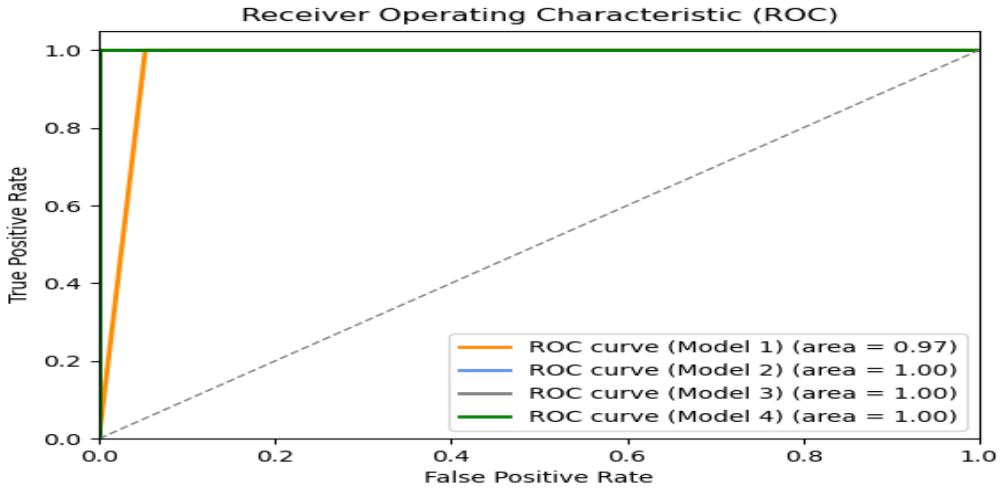


Figure 3.5: Random Forest chart of Ovarian.

This is because all the curves reach close to the top-left corner of the graph. This corner represents a perfect classification result, where the model can correctly identify all positive and negative cases (TPR = 1 and FPR = 0).
Models 2, 3, and 4 appear to have a perfect classification performance (area = 1.00).
Their ROC curves completely overlap the top-left corner of the graph. This suggests that these models might be indistinguishable in their performance on this specific dataset.
Model 1 (area = 0.97) performs well but not perfectly. While its curve reaches towards the top-left corner, it doesn't quite reach it. This indicates that Model 1 might make a few mistakes in classification compared to the other models.

**Colon**

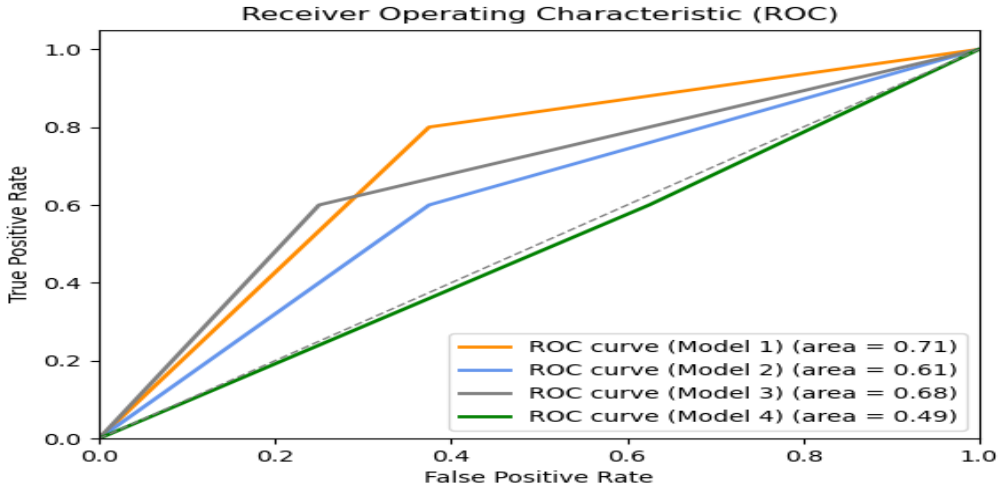


Figure 3.6: Random Forest chart of Colon.

Model 1 (area = 0.71):

This curve demonstrates an acceptable performance. It starts off by increasing the TPR rapidly at a low FPR, which is a desirable characteristic. However, as the curve progresses towards the top-right corner of the graph, the TPR increases at a slower rate and doesn't reach a perfect score (TPR = 1). This indicates that Model 1 might struggle to correctly classify all positive instances as the FPR increases.
Model 2 (area = 0.61) and Model 3 (area = 0.68):
The performance of these models is lower than Model 1. Their curves show a similar trend, where the TPR increases slowly throughout the range of the FPR. This suggests that both models might frequently misclassify positive or negative instances.
Model 4 (area = 0.49):
This model performs the worst among the four. Its curve approaches a diagonal line rising from the bottom left to the top right corner. An ROC curve that follows a diagonal line represents a random classifier, meaning it performs no better than randomly guessing the classifications.
Overall, the ROC curves in the image suggest that Model 1 is the most reliable model for classifying the data. However, its performance is not ideal, and all the models could potentially benefit from further improvement.

#### 3.3.1.3 KNN Model

**Ovarian**

By analyzing the area under the curve (AUC) and the overall shape of each curve, we can observe

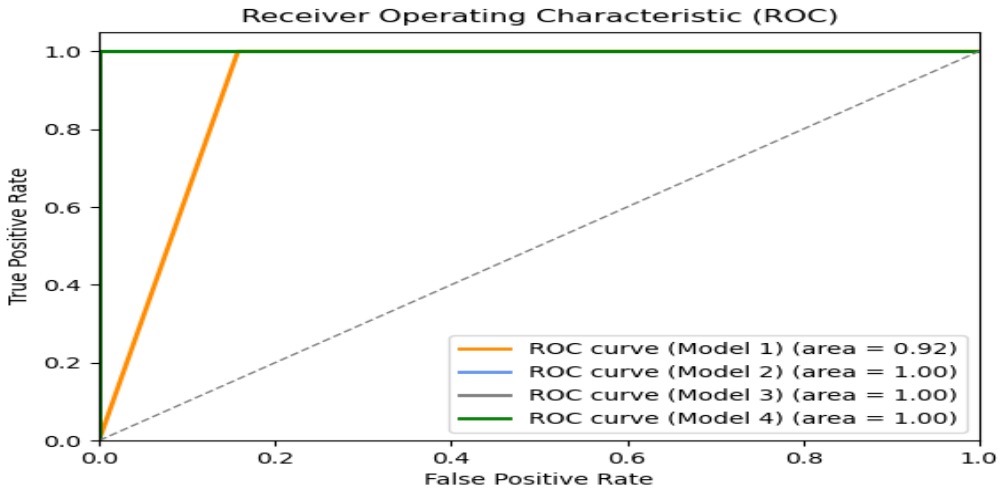


Figure 3.7: KNN chart of Ovarian.

the following trends:
Models 2, 3, and 4 achieve perfect classification (area = 1.00).
Their ROC curves completely overlap the top-left corner of the graph. This indicates that these models can ideally distinguish between positive and negative cases in this specific dataset, with a TPR of 1 (classifying all positive cases correctly) and an FPR of 0 (no false positive classifications).
Model 1 performs well but not perfectly (area = 0.92). While its curve reaches towards the top-left corner, it falls slightly below the other curves. This suggests that Model 1 might make a few mistakes in classification compared to the other models, resulting in a TPR slightly lower than 1 and potentially a small number of false positive classifications.

**Colon**

By analyzing the area under the curve (AUC) and the overall shape of each curve, we can observe

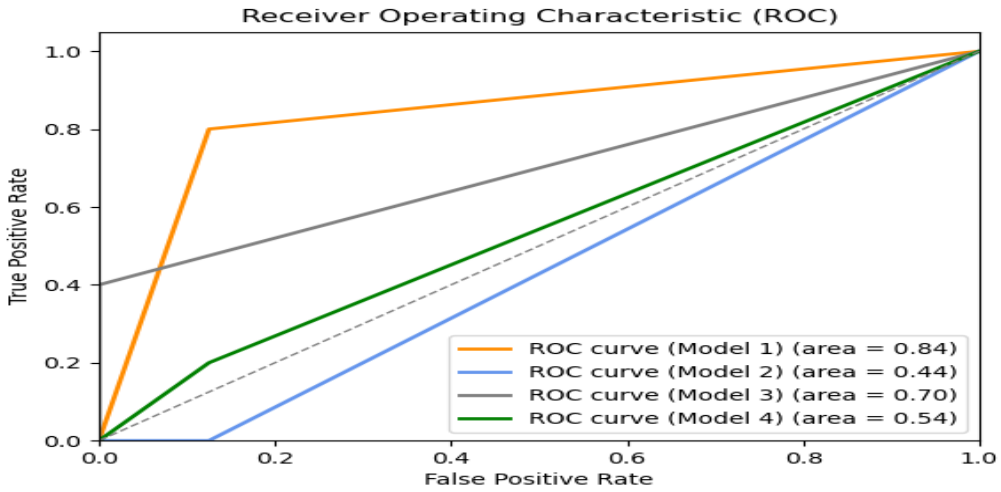


Figure 3.8: KNN chart of Colon.

the following trends:

Model 1 (area = 0.84): demonstrates acceptable performance. Its curve starts off well, with a rapid increase in TPR at a low FPR. This is a desirable characteristic, indicating the model can correctly identify many positive cases early on without mistakenly classifying negative cases. However, as the curve progresses towards the top-left corner, the TPR increases at a slower rate and doesn't reach a perfect score (TPR = 1). This suggests that Model 1 might struggle to correctly classify all positive instances as the FPR increases.

Model 2 (area = 0.44) : has significantly lower performance compared to Model 1. Its curve approaches a diagonal line rising from the bottom left corner to the top right corner. An ROC curve that follows a diagonal line represents a random classifier, meaning its performance is no better than randomly guessing the classifications. In this case, Model 2 performs poorly at distinguishing between positive and negative cases.

Model 3 (area = 0.70): shows intermediate performance between Model 1 and Model 2. Its curve follows a similar trend to Model 1, but with a less pronounced initial increase in TPR and a steeper rise in FPR at later stages. This suggests Model 3 might make more mistakes than Model 1, both in terms of missing positive cases and incorrectly classifying negative cases.

Model 4 (area = 0.54): also exhibits poor performance, similar to Model 2. Its curve is closer to the diagonal line than Model 3, indicating a weak ability to differentiate between positive and negative cases. Overall, the ROC curves in the image suggest a clear hierarchy in performance: Model 1 > Model 3 > Model 2 and Model 4. While Model 1 performs well, there's still room for improvement in correctly classifying all positive cases. Models 2 and 4 show a significant weakness in distinguishing between positive and negative instances.

#### 3.3.1.4 MLP Model

**Ovarian**

Key findings:

All models obtained a perfect AUC (1): This indicates that all curves reach the upper left corner of

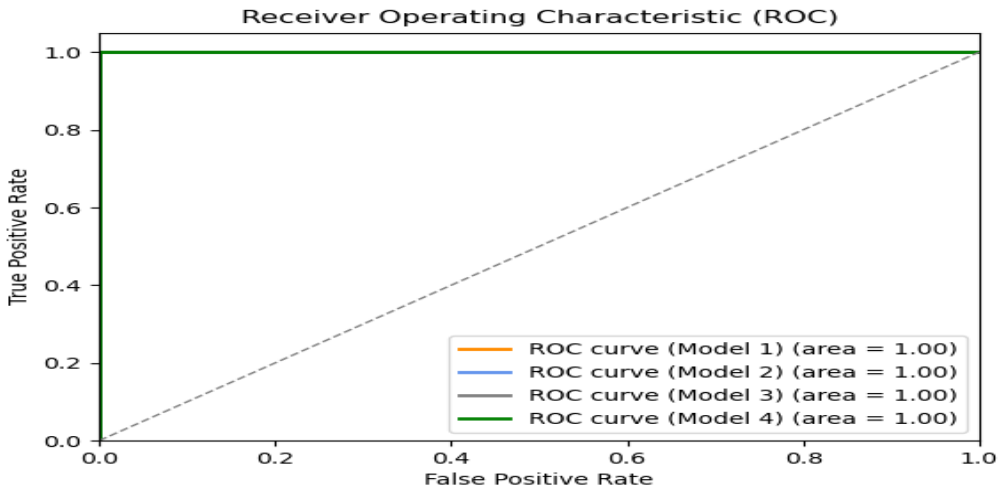


Figure 3.9: MLP chart of Ovarian.

the graph, indicating that 100% true positive rate and 10% false positive rate are achieved at a certain threshold. This scenario raises some doubts, as it is unrealistic in most machine learning applications. It is likely that the models may have been overfitting with the data on which they were trained.
There are no fundamental differences between the models: The curves do not show any noticeable differences between the four models, as they all overlap completely.
While the ROC curves themselves do not indicate any statistically significant trends or patterns between models, the ideal AUC scores for all models raise questions about their ability to generalize to new data.

**Colon**

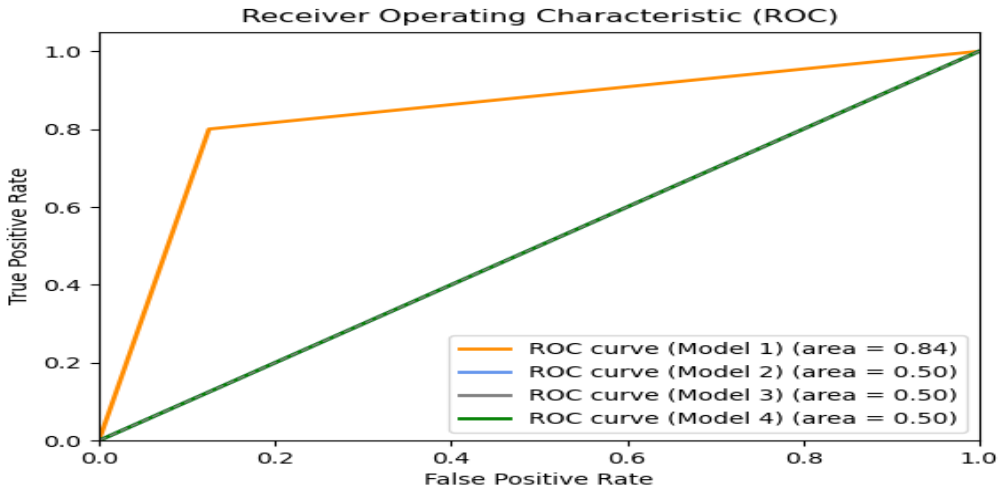


Figure 3.10: MLP chart of Colon.

In the ROC curves three of the models (Model 1, Model 3 and Model 4) show a clear upward trend, starting from the lower left corner (FPR = 0, TPR = 0) and reaching a point close to the upper left corner (FPR = 0, TPR = 1). . This indicates that the true positive rate increases as the false positive rate increases. These curves also have a relatively smooth arc shape.
Model 2 deviates from this pattern. It starts from the lower left corner and reaches the ideal result (TPR = 1) very quickly, with a steeper curve than other models. This suggests that Model 2 may be a

poor classifier overall. All cases may be classified as positive, regardless of whether they are actually positive or negative. This would result in a high true positive rate, but would also result in a very high false positive rate.
The area under the ROC curve (AUC) is a measure that summarizes the performance of a classifier. An ideal classifier will have an AUC of 1.0, while a random classifier will have an AUC of 0.5. The AUC for each model is provided in the graph legend. Here, Model 1 has the highest AUC (0.84), followed by Models 3 and 4 (both at 0.50), and finally Model 2 (0.50). This confirms that Model 1 achieves the best performance among the four models presented.
In conclusion, the ROC curves in the image show that Model 1 is the best performing classifier, followed by Models 3 and 4 (which appear to perform similarly), and finally Model 2.

| Algo | Data set | All | 50% | 25% | 10% |
|---|---|---|---|---|---|
| SVM | Ovarian | 1 | 1 | 1 | 1 |
| | Colon | 0.92 | 0.64 | 0.69 | 0.61 |
| RF | Colon | 0.71 | 0.46 | 0.69 | 0.61 |
| | Ovarian | 0.98 | 1 | 1 | 1 |
| KNN | Colon | 0.92 | 0.64 | 0.69 | 0.61 |
| | Ovarian | 0.91 | 1 | 1 | 1 |
| MLP | Colon | 0.84 | 0.61 | 0.61 | 0.38 |
| | Ovarian | 1 | 1 | 1 | 1 |

Table 3.2: comparison of accurcy.

## 3.4 Summary:

This analysis provides a detailed breakdown of the performance of four classification algorithms (SVM, RF, KNN, MLP) on two different datasets ("Ovary" and "Colon") after reducing the data size using the autoencoder technique at different percentages (50%, 25%, and 10%).

### 3.4.1 Key Findings:

**SVM:** Achieved the best overall performance on both datasets, with perfect accuracy scores (1) on the Ovary dataset across all data subsets (all, 50%, 25%, 10%). On the Colon dataset, SVM performed well on the all and 25% data subsets (1, 0.69) but not as well on the 50% and 10% subsets (0.64, 0.61).
**RF:** Also performed well, with perfect accuracy scores (1) on the Ovary dataset across all data subsets. On the Colon dataset, Ran For did not perform as well as SVM, but it still achieved a respectable accuracy of 0.71 on the all data subset.
**KNN:** Had similar performance to SVM on the Colon dataset (0.92, 0.64, 0.69, 0.61) but slightly lower on the Ovary dataset (0.91) though still achieving perfect accuracy scores on the remaining subsets (1, 1, 1).
**MLP:** Performed the worst overall. While it achieved perfect accuracy scores on the Ovary dataset (1, 1, 1, 1), its performance on the Colon dataset was consistently lower across all data subsets (0.84, 0.61, 0.61, 0.38).

#### 3.4.1.1 Impact on Model Selection

The choice of the optimal model depends on the specific dataset and the available data percentage.
**If data is available in full (100%):** SVM is the best choice for both datasets.
**If data is available at 50%:** SVM is still a good choice for both datasets, but Ran For can also be considered.
**If data is available at 25%:** SVM is still a good choice for the Ovary dataset, but KNN may be a better choice for the Colon dataset.
**If data is available at 10%:** SVM is still a good choice for the Ovary dataset, but KNN or Ran For may be better choices for the Colon dataset.

#### 3.4.1.2 Conclusion

These analyses have shown that the SVM algorithm provides the best overall performance on both datasets ("Ovary" and "Colon") across all data reduction percentages (50%, 25%, and 10%).

## 3.5 Hardware and Software

### 3.5.1 Hardware environment (Hardware)

We present the detailed experiments and evaluation steps to test the effectiveness of our model. All experiments were performed on an Acer PC equipped an Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz 2.40 GHz processor and 4 GB RAM and an Intel(R)HD Graphics GPU.

### 3.5.2 Software environment (Software)

#### 3.5.2.1 Python

Is an interpreted high-level programming language for programming for general use. Created by Guido van Rossum, and first published in 1991. It is based on a design philosophy that emphasizes the readability of the code, in particular by using meaningful spaces. It provides constructs for programming clear at small and large scale. Python offers a dynamic type system and management automatic memory. It supports multiple programming paradigms,including object-oriented, imperative, functional and procedural, and has a library extended and complete standard. Python is an open-source, high-level programming language, developed for use with a wide range of operating systems It is called as the most powerful programming language due to its dynamic nature and diversified. Python is easy to use with a super simple syntax that is very encouraging to beginner learners, and very motivating for seasoned users. contains several libraries popular artificial intelligence: (Keras, Tensorflow, Numpy, Pandas, OPENcv, Pytorch, etc.. . .) [11].

#### 3.5.2.2 Pandas

Is a robust and well-liked open-source Python data manipulation and analysis toolkit. It offers adaptable data structures that let users manage and analyze structured data effectively, like

DataFrame and Series. For activities including data cleansing, filtering, grouping, merging, reshaping, and visualization, Pandas provides a large number of methods and functions. The list of authoritative sources in the bibliography for Pandas in Python includes academic papers, books, government documents, online tutorials, and articles that provide in-depth explanations of the features, capabilities, and best practices of Pandas. These sources are helpful tools for Python data workers who want to learn more about Pandas and how to use its robust capabilities to glean insightful information from datasets [26].

#### 3.5.2.3 Sklearn

A well-known machine learning library for Python is called scikit-learn, also known as sklearn. It offers a wide variety of tools and methods for jobs like model selection, regression, clustering, and dimensionality reduction. A user-friendly interface for developing and testing machine learning models is provided by scikit-learn, which is built on top of existing scientific Python tools like NumPy and SciPy. It refers to a collection of reliable sources that explore the ideas, algorithms, and usage patterns related to scikit-learn, including academic papers, textbooks, government documents, tutorials, and articles. These publications offer useful insights into the theory of machine learning, real-world implementation strategies, and best practices for using scikit-learn successfully. For developers, academics, and practitioners wishing to use scikit-learn to create accurate and reliable machine learning models in Python, the bibliography is an invaluable resource [25].

#### 3.5.2.4 Tensorflow

Is an end-to-end open source platform for building applications machine learning. It is a symbolic mathematical library that usesa flow of data and differentiable programming to perform various tasks focusedon the training and inference of deep neural networks. It allows developers to create machine learning applications using various tools, libraries andcommunity resources. Currently, the most famous deep learning library in the world is TensorFlow from Google. Google uses machine learning in all of its products tosearch engine optimization, translation, image captions or recommendations. TensorFlow offers multiple levels of abstraction so you can choose the right one best for your needs. Build and train models using the high-level Keras API, making it easy to get started with TensorFlow and machine learning [20].

## Conclusion

This chapter delved into the implementation of a deep learning-based approach for gene selection from microarray gene expression data, aiming to enhance cancer classification accuracy. The chapter commenced with an overview of the resources, programming language, and development environment employed. Subsequently, it meticulously outlined the step-by-step implementation process, encompassing data preprocessing and the selection of suitable machine and deep learning techniques. The latter part of the chapter presented the experimental results, evaluating the performance of the proposed model and comparing different techniques. Overall, this chapter serves as a comprehensive guide for implementing the proposed approach, offering valuable insights into the methodology and paving the way for further research in gene selection from microarray gene expression data using deep learning.

# General Conclusion

This study demonstrated the effectiveness of deep learning, especially autoencoders, in selecting useful genes from microarray chip data for cancer classification. By reducing data dimensionality and extracting relevant features, the proposed approach improved classification accuracy and deeper understanding of basic biological processes compared to traditional methods.
The study provided a step-by-step guide to implementing this deep learning-based gene selection approach, explaining the necessary resources, programming environment, and step-by-step procedures. In addition, analyzes presented with SVM classification demonstrated the superiority of this method across different levels of data reduction.

- The autoencoder has been shown to be effective in selecting features from microarray slide data, significantly improving cancer classification accuracy.
- Reducing the dimensionality of data using autoencoders has improved the efficiency of classification models.
- Autoencoder provided more discriminating features, leading to a better understanding of the biological processes underlying cancer.

This work makes a valuable contribution to the field of cancer staging by demonstrating the capabilities of deep learning, especially autoencoder, in selecting useful genes from microarray chip data. These findings could contribute to the development of more accurate and efficient classification models, which may lead to improved cancer diagnosis and treatment.
In conclusion, these results suggest that the deep learning-based gene selection approach can be applied to other diseases, opening the way for new research areas in the field of precision medicine.

# Bibliography


[1] Autoencoders -machine learning. https://www.geeksforgeeks.org/auto-encoders/.

[2] Convolutional neural network. https://www.engati.com/glossary/convolutional-neural-network.

[3] Deep inside: Autoencoders. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f.

[4] Dna microarray technology fact sheet. https://www.genome.gov/about-genomics/fact-sheets/DNA-Microarray-Technology.

[5] An introduction to autoencoders: Types, applications and working. https://blog.gopenai.com/an-introduction-to-autoencoders-types-applications-and-working-a66d9599e172.

[6] Introduction to convolution neural network. https://www.geeksforgeeks.org/introduction-convolution-neural-network/.

[7] k-nearest-neighbours. https://www.geeksforgeeks.org/k-nearest-neighbours/.

[8] k-nearest-neighbours (knn). https://www.pinecone.io/learn/k-nearest-neighbor/.

[9] Multilayer perceptron. https://en.wikipedia.org/wiki/Multilayer$_perceptron$.

[10] Multilayer perceptron. https://deepchecks.com/glossary/multilayer-perceptron/.

[11] Python. https://www.python.org/doc/essays/blurb/.

[12] Support vector machine in machine learning. https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/.

[13] svm-advantages-disadvantages-applications. https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications/.

[14] What are convolutional neural networks? https://www.ibm.com/topics/convolutional-neural-networks.

[15] What are the advantages and disadvantages of random forest? https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-random-forest/.

[16] What is an autoencoder? https://www.ibm.com/topics/autoencoder.

[17] What is random forest? https://www.ibm.com/topics/random-forest.

[18] What is the k-nearest neighbors (knn) algorithm? https://www.ibm.com/topics/knn.

[19] What are support vector machines (svms)? https://www.ibm.com/topics/support-vector-machine., 27 December 2023.

[20] M. Abadi. Tensorflow: learning functions at scale. *SIGPLAN Not.*, 51(9):1, sep 2016.

[21] M. Abd-Elnaby, M. Alfonse, and M. Roushdy. Classification of breast cancer using microarray gene expression data: A survey. *Journal of Biomedical Informatics*, 117:103764, 2021.

[22] N. Almugren and H. Alshamlan. A survey on hybrid feature selection methods in microarray gene expression data for cancer classification. *IEEE Access*, 7:78533–78548, 2019.

[23] R. Díaz-Uriarte. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 2006/01/06.

[24] M. A. Hambali, T. O. Oladele, and K. S. Adewole. Microarray cancer feature selection: Review, challenges and research directions. *International Journal of Cognitive Computing in Engineering*, 1:78–97, 2020.

[25] B. Komer, J. Bergstra, and C. Eliasmith. *Hyperopt-Sklearn*, pages 97–111. 05 2019.

[26] W. Mckinney. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*, 01 2011.

[27] K. Rezaee, G. Jeon, M. R. Khosravi, H. H. Attar, and A. Sabzevari. Deep learning-based microarray cancer classification and ensemble gene selection approach. *IET Systems Biology*, 16(3-4):120–131, 2022.

listings xcolor

Annex 1

```
import pandas as pd
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
data = pd.read_csv('CoLon.csv')
features = data.iloc[:, :-1]
labels = data.iloc[:, -1]
# Building the model
input_data = Input(shape=(features.shape[1],))
encoded = Dense(128, activation='relu')(input_data)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(200, activation='relu')(encoded) # Specify encoder values 50%, 25%, 10%
encoder = Model(input_data, encoded)
# Assemble the model
encoder.compile(optimizer='adam', loss='binary_crossentropy')
# training model
encoder.fit(features, labels, epochs=50, batch_size=32, shuffle=True)
# Encode the data using the trained model
encoded_data = encoder.predict(features)
# Convert the encoded data to a DataFrame
encoded_df = pd.DataFrame(encoded_data)
encoded_df.to_excel('enco_Colon10%.xlsx', index=False)
```

Listing 3.1: Autoencoder

Annex 2

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn import metrics
# Load the dataset
data = pd.read_csv('Ovarian .csv')
# Separate the labels and the features
labels = data.iloc[:, -1]
features = data.iloc[:, :-1]
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test =train_test_split(features,labels, test_size=0.2,
    random_state=42)
# Create an SVM model
svm_model = SVC()
# Train the SVM model using the training data
svm_model.fit(X_train, y_train)
# Predict the labels for the test data
y_pred1 = svm_model.predict(X_test)
# Calculate the accuracy of the model
Accuracy = metrics.accuracy_score(y_test, y_pred1)
print("Accuracy:", Accuracy)
```

Listing 3.2: SVM Model

Annex 3

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
# Load the dataset
data = pd.read_csv('Ovarian .csv')
# Separate the labels and the features (using the first 2000 features)
labels = data.iloc[:, -1]
features = data.iloc[:, :2000]
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
    random_state=42)
# Create a Random Forest Classifier model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the Random Forest model using the training data
rf_model.fit(X_train, y_train)
# Predict the labels for the test data
rf_predictions = rf_model.predict(X_test)
# Calculate the accuracy of the model
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
```

Listing 3.3: Random Forest Model

Annex 4

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score ,confusion_matrix
from sklearn import metrics
# Load the dataset
data = pd.read_csv('Ovarian .csv')
labels = data.iloc[:, -1]
features = data.iloc[:, :-1]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
    random_state=42)
# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Initialize the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors (K)
    as needed
# Train the classifier
knn.fit(X_train_scaled, y_train)
# Make predictions
y_pred = knn.predict(X_test_scaled)
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Listing 3.4: KNN Model

Annex 5

```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn import metrics
# Load the dataset
data = pd.read_csv('CoLon.csv')
# Separate the labels and the features
labels = data.iloc[:, -1]
features = data.iloc[:, :-1]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
    random_state=42)
# Define the neural network model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),  # First hidden layer
     with 128 neurons
    Dense(64, activation='relu'),  # Second hidden layer with 64 neurons
    Dense(1, activation='sigmoid')  # Output layer with 1 neuron for binary
    classification
])
# Compile the model with Adam optimizer and binary crossentropy loss function
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Make predictions on the test data
y_pred = model.predict(X_test)
y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred]  # Convert probabilities to
    binary predictions
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred_binary)
print("Accuracy:", accuracy)
```

Listing 3.5: MLP Model

Annex 6

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
# Calculate the false positive rate (FPR), true positive rate (TPR), and thresholds for
    each model's ROC curve
fpr1, tpr1, thresholds1 = roc_curve(y_test, y_pred1)
roc_auc1 = auc(fpr1, tpr1)
fpr2, tpr2, thresholds2 = roc_curve(y_test, y_pred2)
roc_auc2 = auc(fpr2, tpr2)
fpr3, tpr3, thresholds3 = roc_curve(y_test, y_pred3)
roc_auc3 = auc(fpr3, tpr3)
fpr4, tpr4, thresholds4 = roc_curve(y_test, y_pred4)
roc_auc4 = auc(fpr4, tpr4)
# Plotting the ROC curves for each model
plt.figure()
```

```
plt.plot(fpr1, tpr1, color='darkorange', lw=2, label='ROC curve (Model 1) (area = %0.2f)' % roc_auc1)
plt.plot(fpr2, tpr2, color='cornflowerblue', lw=2, label='ROC curve (Model 2) (area = %0.2f)' % roc_auc2)
plt.plot(fpr3, tpr3, color='gray', lw=2, label='ROC curve (Model 3) (area = %0.2f)' % roc_auc3)
plt.plot(fpr4, tpr4, color='green', lw=2, label='ROC curve (Model 4) (area = %0.2f)' % roc_auc4)
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--') # Diagonal line representing random chance
# Setting the limits and labels for the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
# Save the plot as a PNG file
plt.savefig('ROC.png')
# Display the plot
plt.show()
```

Listing 3.6: Graph ROC