CZECH TECHNICAL

UNIVERSITY

IN PRAGUE

FACULTY OF

MECHANICAL ENGINEERING

BACHELOR

THESIS

COMPARISON OF THE SVM AND THE MLP NN METHODS FOR A DATA CLASSIFICATION

BURAK

UNSAL

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ünsal Burak** Personal ID number: **461712**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Instrumentation and Control Engineering**

Study program: **Theoretical Fundamentals of Mechanical Engineering**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Comparison of the SVM and the MLP NN methods for a data classification

Bachelor's thesis title in Czech:

Srovnání metody podpůrných vektorů s použitím neuronových sítí pro klasifikaci dat

Guidelines:

1. Explain the theory of SVM and the theory of MLP network.
2. Develop the algorithms and programs for both the methods.
3. Describe the selected sample of data.
4. Apply the described methods for the selected sample of data.
5. Evaluate the precision and velocity of the classification process for both the methods.

Bibliography / sources:

1. Mariette Awad, Rahul Khanna: Support Vector Machines for Classification. in: Efficient Learning Machines (pp.39-66), 2015, DOI:10.1007/978-1-4302-5990-9_3
2. Raj Bridgelall: Introduction to Support Vector Machines. Lecture Notes, 2017. Available online: https://www.ugpti.org/smartse/resources/downloads/support-vector-machines.pdf
3. Vladimír Hlaváč: Ukázka rozpoznávání řeči s využitím klasifikace neuronovou sítí. Nové metody a postupy v oblasti přístrojové techniky, automatického řízení a informatiky 2020
4. Neural Net Pattern Recognition. Matlab documentation online: www.mathworks.com/help/deeplearning/ref/neuralnetpatternrecognition-app.html

Name and workplace of bachelor's thesis supervisor:

Ing. Vladimír Hlaváč, Ph.D., U12110.3

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **29.10.2021** Deadline for bachelor thesis submission: **14.01.2022**

Assignment valid until: _____

Ing. Vladimír Hlaváč, Ph.D.	Head of department's signature	prof. Ing. Michael Valášek, DrSc.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt	Student's signature

I confirm that the diploma (bachelor's) work was disposed by myself and independently, under leading of my thesis supervisor. I stated all sources of the documents and literature.

In Prague ……

……

Burak Unsal

Acknowledgment

I acknowledge the assistance of my thesis supervisor Vladimir Hlavac, in various aspect of the research work.

Abbreviations

SVM	Support vector machine
ANN	Artificial Neural Networks
RBF	Radial Basis Function
SUM	Summation
RELU	Rectified Linear Activation Unit
CNN	Convolution Neural Networks
FFT	Fast Fourier Transform
SKLEARN	Scikit-Learn
CSV	Coma-Separated Variables

Table of Contents

1 Introduction

Vowel classification is a important task in speech processing with various applications such as speech recognition, speech synthesis, and speech analysis. Accurate classification of vowels is fundamental for the development of effective speech systems that can interpret and respond to human speech. In recent years, machine learning algorithms such as Support Vector Machines (SVMs) and Multi-layer perceptrons (MLPs) have been widely utilized for vowel classification.

This thesis compares the performance of SVMs and MLPs in vowel classification. The dataset used in the study includes sounds of different vowels and consonants, which were preprocessed and divided into training and testing sets. The SVM model was trained with a linear kernel, while the MLP model was trained using the default configuration. Both models' performance was evaluated using the f1 score and compared to determine the most appropriate algorithm for vowel classification.

The thesis also covers the fundamental concepts of machine learning and artificial intelligence, including support vector machine theory, kernel methods, and activation functions used in neural networks. Popular Python libraries such as sci-kit-learn, numpy, matplotlib, and pandas were utilized to evaluate the performance of various algorithms and parameters. The objective of this thesis is to provide a comprehensive comparison of SVMs and MLPs for vowel classification and to showcase the strengths and weaknesses of each method.

2 Support Vector Machine

The foundations of support vector machines were laid in 1963 by Vladimir Vapnik and Alexey Chernozemic. Although, it was published by Vladimir Vapnik, Bernhard Baser and Isabelle Guyan for the first time in 1992. The advantages include high accuracy, complex decision modelling, working with multiple independent variables, both being able to apply to non-linearly separable and non-separable data, and less overfitting compared to other methods. The disadvantages can be counted as failure to produce probability estimates. It is used in various fields such as object and handwriting recognition, time series prediction tests, and bioinformatics.

Support vector machines are used in many different areas today. For example, weather forecasts of meteorology, patients can be diagnosed according to similar patients, in face recognition systems, pattern recognition systems, security cameras, and in many different industries. Also, features such as working with multi and single data and transforming the 2D space into higher-dimensional problems are the most important features that distinguish the support vector machine from other classification methods.

In our study, I will explain the support vector machines in detail and examine how we can easily classify the given complex datasets using the kernel method. Next, we will examine the methods of implementing the information given in practice with the Python programming language. Finally, I will present two practical examples covering the topics that were examined.

2.1 Machine learning

To understand machine learning, firstly we need to understand learning. Although Learning is a very large concept, in its most general meaning we can define it as a result of education and acquired knowledge of cognitive behaviour forms seen in our behaviour. Similarly, machine learning is the realization of this learning process by computers. The computer can learn about a situation and then make decisions about future events and find solutions to problems by learning.

Statistical learning is the basis of many machine learning algorithms. Although it is the basis of learning algorithms, it is used to obtain the necessary information from experimental data and obtain future problems. Machine learning algorithms are often categorized as supervised or unsupervised. The biggest difference between them is that if the information is given as labelled, it is called a supervise algorithm. Supervised datasets consist of one or more attribute groups. There are two types of supervised technique those are classification and regression. Classification is the process of predicting a class label from predefined attribute groups. Regression is the predicting continuous value from dependent on single or multi-labelled values. Unsupervised is the classification of unlabelled information according to various similarities and various analyses. The most important difference between them is that unsupervised algorithms are using unlabelled information. In unsupervised algorithms, clustering methods are generally used.

Support vector machines are one of the most popular machine learning algorithms. Support vector machines are used frequently in the industry, because it can solve linear and nonlinear complicated problems with high accuracy. Support vector machines are a supervised classification method based on mathematical statistical methods. It uses mapping to transform

the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane.

2.2 Classification

As we mentioned in the previous section, classification is a tool for machine learning, and its basis is mathematical statistics. Along with the support vector machines, there are different methods such as artificial neural networks, k-neighbourhood points, logistic regression etc. This is the process that assignment of objects of similar characteristics to designated subgroups. It is distributing various classes that defined on a data set. The classification algorithms learn the distribution pattern from the given training data and then try to classify it correctly.

2.3 Hyperplane

For the support vector machine classification, we need to draw a line between two attribute tuples and it's named a hyperplane. The location of line should be located at the farthest point from the two attributes tuple.

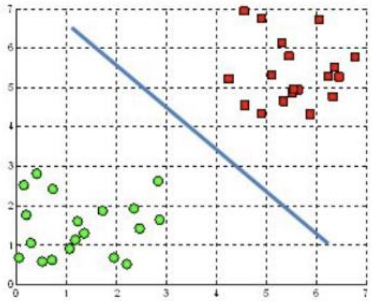

Fig. 1 Separator dividing two attributes group [1]

2.4 The Maximal Margin Classifier

A natural choice is the maximal margin hyperplane which is the separating hyperplane that is farthest from the training observations. That is, we can compute the distance from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the margin. The maximal margin hyperplane is the separating hyperplane for which the margin is largest that is, it is the hyperplane that has the farthest minimum distance to the training observations. We can then classify a test observation based on which side of the maximal margin hyperplane it lays. This is known as the maximal margin classifier

If $\beta_0, \beta_1, \ldots, \beta_p$ are the coefficients of the maximal margin hyperplane, then the maximal margin classifier classifies the test observation x^* based on the sign of

$$f(x^*) = \beta_0 + \beta_1 x^*_1 + \beta_2 x^*_2 + \cdots + \beta_p x^*_p \quad (1)$$

.

We now consider the task of constructing the maximal margin hyperplane based on a set of n training observations $x_1, \ldots, x_n \in R^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$. Briefly, the maximal margin hyperplane is the solution to the optimization problem.

Maximize M

$$\beta_0 + \beta_1, \dots \dots, \beta_p$$

Subject to ; $\sum_{J=1}^{P} \beta_j^2 = 1$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \forall i = 1, \dots, n. \quad (2)$$

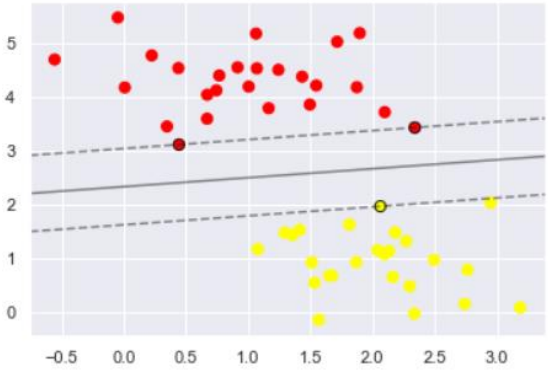

Fig. 2 Hyperplane maximizing two attributes group [2]

2.5 Soft margin SVM

Soft-margin SVM is an improvement over hard-margin SVM, this improvement make the classifier able to classify data accurately even if there was a noisy data. In other words, there is an issue with the hard margin SVM because not every dataset can be separated linearly, due to the outlier points. The outlier points have two cases; one of them is the case that a data point is closer to the other class points than its class and this causes reducing in the margin, another case happens when a point is among the other class points and this case breaks the linear separability.

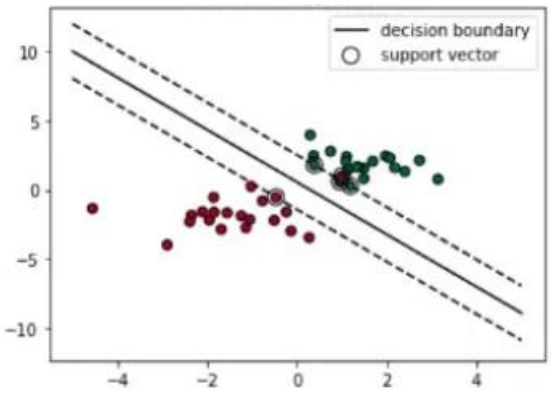

Fig. 3 Showing outline point in the graph [3]

2.6 Kernel Methods

2.6.1 Linear Kernel Method

The linear kernel is a simple and efficient kernel function that can be used with SVMs. It is often used when the data is linearly separable, which means that it can be separated into different classes by a linear decision boundary. The linear kernel is also relatively easy to compute, as it does not require any additional operations beyond the inner product. The disadvantage of the linear kernel is that it may not be able to separate more complex relationships in the data, such as non-linear patterns. In these cases, it may be necessary to use a more sophisticated kernel function, such as a polynomial kernel or an RBF kernel, to get better performance.

The linear kernel is a type of kernel function that is defined as the inner product of the input vectors. Mathematically, it can be written as:

$$k(x, y) = < x, y > \tag{3}$$

where x and y are the input vectors and <,> represents the inner product. The inner product of two vectors is a scalar value that represents the dot product of the vectors, which is defined as the sum of the products of the corresponding elements. For example, if x and y are vectors of length n, the inner product can be calculated as:

$$< x, y > \equiv \quad x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n \tag{4}$$

Let's give an easy example to using linear kernel method;

```
def linear_kernel(x, y):
    return sum(x[i] * y[i] for i in range(len(x)))
```

This function takes two input vectors x and y as arguments and returns their inner product, which is calculated as the sum of the products of the corresponding elements.

```
X = [[1, 2], [3, 4], [5, 6]]
y = [1, -1, 1]
```

This code will generate a synthetic dataset with 3 samples and 2 features and use the linear kernel function to compute the kernel values for all pairs of samples in the dataset. The kernel values are stored in a 3x3 matrix.

```
kernel_values = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

This variable means that each element of the matrix is initially set to 0. The matrix has 3 rows and 3 columns, corresponding to the 3 samples in the synthetic dataset. The purpose of this initialization is to create a matrix that can store the kernel values for all pairs of samples in the dataset.

```
for i in range(3):
    for j in range(3):
        kernel_values[i][j] = linear_kernel(X[i], X[j])
```

The kernel values are computed using the linear_kernel function, which takes two input vectors x and y, and returns their inner product.

```
print(kernel_values)
[[5, 11, 17], [11, 25, 39], [17, 39, 61]]
```

Obtained the result from the given dataset.

```
def linear_kernel(x, y):
    return sum(x[i] * y[i] for i in range(len(x)))
X = [[1, 2], [3, 4], [5, 6]]
y = [1, -1, 1]

# Compute the kernel values
kernel_values = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
for i in range(3):
    for j in range(3):
        kernel_values[i][j] = linear_kernel(X[i], X[j])

print(kernel_values)
 [[5, 11, 17], [11, 25, 39], [17, 39, 61]]
```

2.6.2 Polynomial Kernel Method

The polynomial kernel is a more flexible kernel function than the linear kernel, as it can capture more complex relationships in the data. It is often used when the data is not linearly separable and can be particularly effective when the data has a non-linear structure. The disadvantage of the polynomial kernel is that it can be computationally expensive, as it requires raising the dot product to power. In addition, the choice of the degree d can have a significant impact on the performance of the model, and finding the optimal value for d can be challenging.

The polynomial kernel is a type of kernel function that is defined as the dot product of the input vectors raised to a specific power. Mathematically, it can be written as:

$$(x, y) = (< x, y > + c)^d \tag{5}$$

where x and y are the input vectors, c is a constant parameter, and d is the degree of the polynomial. The inner product <x, y> is a scalar value that represents the dot product of the vectors, which is defined as the sum of the products of the corresponding elements. For example, if x and y are vectors of length n, the inner product can be calculated as:

$$< x, y > \equiv \quad x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n \tag{6}$$

Let's give an easy example to using polynomial kernel method;

```
def polynomial_kernel(x, y, c, d):
    return (sum(x[i] * y[i] for i in range(len(x))) + c) ** d
```

This function takes two input vectors x and y as arguments, as well as the constant parameter c and the degree d of the polynomial. It returns the dot product of the vectors raised to the power d, which is calculated as the sum of the products of the corresponding elements plus the constant c.

```
X = [[1, 2], [3, 4], [5, 6]]
y = [1, -1, 1]
```

This code will generate a synthetic dataset with 3 samples and 2 features and use the linear kernel function to compute the kernel values for all pairs of samples in the dataset. The kernel values are stored in a 3x3 matrix.

```
c = 1
d = 3
kernel_values = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

This variable means that each element of the matrix is initially set to 0. The matrix has 3 rows and 3 columns, corresponding to the 3 samples in the synthetic dataset. The purpose of this initialization is to create a matrix that can store the kernel values for all pairs of samples in the dataset. c variable is representing the constant value and the d value is representing the number of degrees of the polynomial.

```
for i in range(3):
    for j in range(3):
        kernel_values[i][j] = polynomial_kernel(X[i], X[j], c, d)



print(kernel_values)
```

```
[[1331, 103889, 5764801], [103889, 823543, 27462529], [5764801, 27462529, 1073741825]]
```

The kernel values were computed using the polynomial kernel function

2.6.3 RBF Kernel Method

RBF kernel is a widely used kernel function that can capture complex relationships in the data. It is particularly effective when the data is not linearly separable and has been used successfully in a variety of applications, including image classification, speech recognition, and natural language processing. A disadvantage of the RBF kernel is that it can handle high-dimensional data efficiently, as it only depends on the distance between the vectors. However, the choice of the hyperparameter sigma (σ) can have a significant impact on the performance of the model, and finding the optimal value for sigma can be challenging. The radial basis function (RBF) kernel, also known as the Gaussian kernel, is a type of kernel function that is defined as the exponential of the negative Euclidean distance between the input vectors. Mathematically, it can be written as:

$$k(x, y) = e^{\frac{-\|x-y\|^2}{2\sigma^2}} \tag{7}$$

where x and y are the input vectors, ||x - y|| is the Euclidean distance between the vectors, and sigma is a hyperparameter that controls the width of the kernel. The Euclidean distance between the vectors is defined as the square root of the sum of the squared differences between the corresponding elements.

Let’s give an easy example of using the RBF kernel method;

```
def rbf_kernel(x, y, sigma):
    distance = sum((x[i] - y[i]) ** 2 for i in range(len(x)))
    return math.exp(-distance / (2 * sigma ** 2))
```

This function takes two input vectors x and y as arguments, as well as the hyperparameter sigma that controls the width of the kernel. It returns the RBF kernel value for the vectors, which is calculated as the exponential of the negative Euclidean distance between the vectors divided by $2\ \sigma^2$.

```
X = [[1, 2], [3, 4], [5, 6]]
y = [1, -1, 1]

# Compute the kernel values
sigma = 1
kernel_values = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
for i in range(3):
    for j in range(3):
        kernel_values[i][j] = rbf_kernel(X[i], X[j], sigma)

print(kernel_values)
```

This is the result of the Kernel RBF:

```
[[1.0,   0.6065306597126334,   0.1353352832366127],   [0.6065306597126334,   1.0,
0.6065306597126334], [0.1353352832366127, 0.6065306597126334, 1.0]]
```

2.7 Comparison of kernel methods

Kernel Type	Advantages	Disadvantages
Polynomial	Can model non-linear relationships between input vectors	Can be computationally expensive to evaluate and may produce unstable results when the degree is set too high
Linear	Fast to compute and scales well with large datasets	Only able to model linear relationships between input vectors
RBF	Can model non-linear relationships between input vectors and can handle high-dimensional data well	Sensitive to the choice of the kernel width parameter, which can be difficult to fit

2.7.1 Example 1)

The dataset consists of 150 samples of three different species of iris flowers: Iris setosa, Iris versicolor, and Iris virginica. Each sample consists of four features: sepal length, sepal width, petal length, and petal width, all measured in centimeters. The goal is to be able to predict the species of an iris flower based on these four features. [4]

```
# Load the iris dataset as an example
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

# Train an SVM model with a linear kernel
linear_svm = svm.SVC(kernel='linear')
linear_svm.fit(X_train, y_train)

# Train an SVM model with a polynomial kernel
poly_svm = svm.SVC(kernel='poly', degree=3)
poly_svm.fit(X_train, y_train)

# Train an SVM model with an RBF kernel
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)

# Evaluate the accuracy of each model
linear_accuracy = linear_svm.score(X_test, y_test)
poly_accuracy = poly_svm.score(X_test, y_test)
rbf_accuracy = rbf_svm.score(X_test, y_test)

# Compare the accuracy results
print("Linear kernel accuracy: ", linear_accuracy)
print("Polynomial kernel accuracy: ", poly_accuracy)
print("RBF kernel accuracy: ", rbf_accuracy)
```

2.7.2 Example 2)

The Digits dataset consists of 1797 8x8 grayscale images of handwritten digits (0-9). Each image is represented as a 64-dimensional vector, with each dimension corresponding to the pixel intensity at a particular location in the image. The goal is to be able to classify each image as one of the 10 digits based on the pixel intensities. [4]

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

# Load the dataset
X, y = datasets.load_digits(return_X_y=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

# Train an SVM model with a linear kernel
linear_svm = svm.SVC(kernel='linear')
linear_svm.fit(X_train, y_train)

# Train an SVM model with a polynomial kernel
poly_svm = svm.SVC(kernel='poly', degree=3)
poly_svm.fit(X_train, y_train)

# Train an SVM model with an RBF kernel
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)

# Evaluate the accuracy of each model
linear_accuracy = linear_svm.score(X_test, y_test)
poly_accuracy = poly_svm.score(X_test, y_test)
rbf_accuracy = rbf_svm.score(X_test, y_test)

# Compare the accuracy results
print("Linear kernel accuracy: ", linear_accuracy)
print("Polynomial kernel accuracy: ", poly_accuracy)
print("RBF kernel accuracy: ", rbf_accuracy)
```

2.7.3 Example 3)

The breast cancer dataset consists of 569 samples of breast cancer tumors, each with 30 features representing characteristics of the tumor. The goal is to classify each tumor as either benign (not cancerous) or malignant (cancerous). [5]

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

# Load the dataset
X, y = datasets.load_breast_cancer(return_X_y=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train an SVM model with a linear kernel
linear_svm = svm.SVC(kernel='linear')
linear_svm.fit(X_train, y_train)

# Train an SVM model with a polynomial kernel
poly_svm = svm.SVC(kernel='poly', degree=3)
poly_svm.fit(X_train, y_train)

# Train an SVM model with an RBF kernel
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)

# Evaluate the accuracy of each model
linear_accuracy = linear_svm.score(X_test, y_test)
poly_accuracy = poly_svm.score(X_test, y_test)
rbf_accuracy = rbf_svm.score(X_test, y_test)

# Compare the accuracy results
print("Linear kernel accuracy: ", linear_accuracy)
print("Polynomial kernel accuracy: ", poly_accuracy)
print("RBF kernel accuracy: ", rbf_accuracy)
```

2.7.4 Example 4)

The wine dataset consists of 178 samples of wine, each with 13 features representing the chemical characteristics of the wine. The goal is to classify each wine as one of three types of wine based on these chemical characteristics. [4]

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics

# Load the dataset
X, y = datasets.load_wine(return_X_y=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Train an SVM model with a linear kernel
linear_svm = svm.SVC(kernel='linear')
linear_svm.fit(X_train, y_train)

# Train an SVM model with a polynomial kernel
poly_svm = svm.SVC(kernel='poly', degree=3)
poly_svm.fit(X_train, y_train)

# Train an SVM model with an RBF kernel
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)

# Evaluate the accuracy of each model
linear_accuracy = linear_svm.score(X_test, y_test)
poly_accuracy = poly_svm.score(X_test, y_test)
rbf_accuracy = rbf_svm.score(X_test, y_test)

# Compare the accuracy results
print("Linear kernel accuracy: ", linear_accuracy)
print("Polynomial kernel accuracy: ", poly_accuracy)
print("RBF kernel accuracy: ", rbf_accuracy)
```

2.8 Linear Kernel

Name of Dataset	Number of Iterations	Max Accuracy	Min Accuracy	Average Accuracy
Breast Cancer Dataset	10	0.98	0.96	0.97
Digits Dataset	10	0.97	0.97	0.97
Iris Dataset	10	0.96	0.92	0.94
Wine Dataset	10	0.97	0.91	0.93

2.9 Polynomial Kernel

Name of Dataset	Number of Iterations	Max Accuracy	Min Accuracy	Average Accuracy
Breast Cancer Dataset	10	0.95	0.88	0.92
Digits Dataset	10	0.99	0.98	0.99
Iris Dataset	10	0.98	0.97	0.97
Wine Dataset	10	0.72	0.63	0.68

2.10 RBF Kernel

Name of Dataset	Number of Iterations	Max Accuracy	Min Accuracy	Average Accuracy
Breast Cancer Dataset	10	0.92	0.88	0.90
Digits Dataset	10	0.93	0.88	0.90
Iris Dataset	10	0.96	0.92	0.94
Wine Dataset	10	0.96	0.92	0.94

3 Artificial Neural Networks

Artificial neural networks originated with the idea of creating a mathematical artificial intelligence inspired by the human brain. Neurons in the human brain consist of axons, dendrites, synapses, and nuclei. The axon is the electrically active body where the output pulses are generated and the conduction on the body is unidirectional. It is system output. Dendrites are electrically passive arms that pick up signals from other cells. It is system input. Synapse provides the connection of axons of the cell with other dendrites. The Nucleus provides periodic reproduction of signals along the Axon. Artificial neural networks are created by imitating biological neurons in the human brain using mathematical formulas. Neurons connect and form neural networks.

Artificial neural networks were first created by Warren McCulloch and Walter Pitts using mathematical logic-created threshold logic units. The concept of perceptron originated with F. Rosenblatt in 1950. The first calculators were used by Farley and Wesley A Clark to simulate Hebbian networks. By 1970, multi-layered neural networks emerged, inspired by the xor problems. Artificial neural networks are widely used today in areas such as face recognition, speech recognition, self-driving cars, the security industry and the health industry. Face recognition technology is used in smartphones, cameras, security cameras, etc. Voice recognition allows us to save time in phone banking. The Deep Learning method offers great comfort in autopilot applications in cars that park themselves.

If we summarize the working principle in general, each input has a weight value. Each input value entered by the system is multiplied by its weight and collected in the input function, and if it reaches a certain threshold value, the signal is transmitted to the activation function. Then the output function is obtained. This algorithm is called the perceptron and is used in machine learning for supervised learning. There are two types of perceptron networks, single-layer, and multi-layer. Multilayer perceptrons are used for problems that cannot be separated linearly.

3.1 Perceptron

Perceptron is a supervised algorithm based on binary. It is used to classify datasets that can be linearly separable. Perceptron algorithm also has one or several inputs. Each input is aggregated by multiplying with the weight functions, and the output data is obtained by using the activation function when reaching a certain threshold value. Then the group of attributes to which it belongs can be predicted. Then the data can be classified into attribute groups. There are two types of perceptron, divided into single-layer and multi-layer.

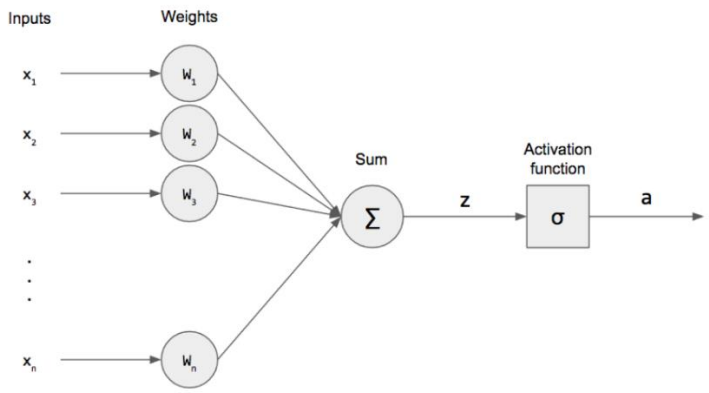

Fig. 4 Perceptron [6]

3.2 Single Layer perceptron

Single-layer perceptron is used for solving linearly separable problems. It is consisting of inputs and output. Layers can have one or more neurons. The computation of a single layer perceptron is performed over the calculation of the sum of the input vector each value multiplied by the corresponding element of a vector of the weights.

$$y(x) = f(\sum_{i=1}^{n} w_i x_i) \tag{8}$$

All weight values (w) are multiplied by the input values(x) which are connected by themselves then all the values summed each other f(x). Then the total input value f(x) is used with the activation function and the output value y(x) is obtained.

3.3 Multi-layer perceptron

It is developed due to single-layer perceptron failure to solve nonlinear problems. Multi-layer perceptron consists of input layers where information is entered, an output layer, and one or more hidden layers. There are several layers which are located between the input and output layers which are named hidden layers. Having more hidden layers in a neural network provides increased hierarchical feature learning, enhanced feature extraction, better modeling of complex decision boundaries, and improved generalization abilities. These advantages allow the network to handle more complex tasks and achieve higher performance on a wide range of problems. The disadvantage of adding more hidden layers is that it may increase the complexity of the problem and increase the risk of overfitting.

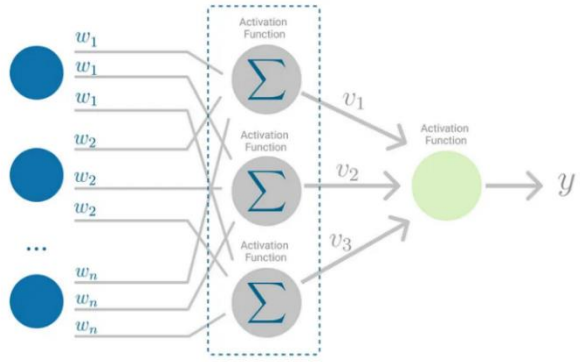

Fig. 5 Multi-layer perceptron [7]

3.4 Activation functions

An activation function in a neural network is a non-linear function that is applied to the output of each neuron before passing the result to the next layer. The purpose of an activation function is to introduce non-linearity into the output of each neuron, which allows neural networks to model complex relationships and non-linear data distributions.

Activation functions play a crucial role in determining the output of a neural network, as well as its ability to learn and generalize to unseen data. Some of the most commonly used activation functions in neural networks are:

Sigmoid (logistic function)**:** maps any input value to a value between 0 and 1, which can be interpreted as a probability. This activation function is often used for binary classification problems. [8]

Tanh (Hyperbolic tangent): maps input values to the range of -1 and 1. This activation function is similar to the sigmoid function but outputs values that are centred around zero, which can be useful in certain architectures. [9]

Leaky ReLU: similar to ReLU but with a small slope for negative inputs. This activation function addresses the problem of "dying ReLU," where neurons in the network never activate if the inputs are negative. [10]

Softmax: used for multiclass classification problems, maps a vector of real numbers to a probability distribution over classes. The softmax function is often used in the final layer of a neural network to make predictions. [11]

Choosing the right activation function depends on the specific problem being solved and the requirements of the neural network architecture. For example, the sigmoid function may work well for binary classification problems.

3.4.1 Sigmoid Function

The sigmoid function is a mathematical function that maps any input value to the range of 0 and 1. It has the characteristic "S" shaped curve and is defined as:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1} \tag{9}$$

where e is the base of the natural logarithm.

In artificial neural networks, the sigmoid activation function is often used for binary classification problems, where the goal is to predict one of two classes. The output of the sigmoid function can be interpreted as a probability and a threshold can be applied to determine the final binary prediction.

It's important to note that the sigmoid function can suffer from saturation when the input is large, leading to slow convergence during training.

Example)

Let's say have an input value x = 2. The sigmoid function can be calculated as follows:

$$S(x) = \frac{1}{1+e^{-x}} = S(x) = \frac{1}{1+0.135} = S(x) = \frac{1}{1.135} \approx 0.87$$

Here is the graph for the sigmoid function.

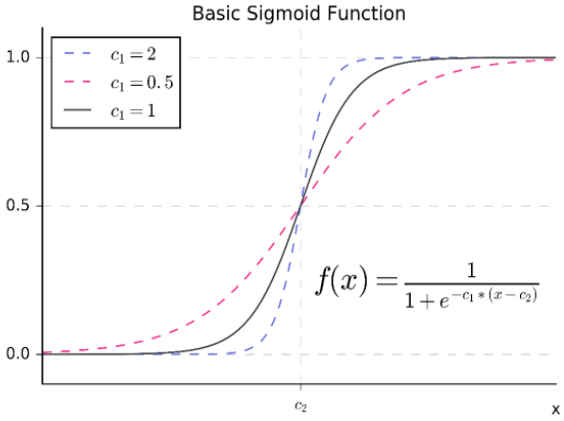

Fig. 6 Basic Sigmoid Function graph [12]

3.4.2 Hyperbolic Tangent Function

The hyperbolic tangent (tanh) is a commonly used activation function in artificial neural networks. It is a non-linear function that maps any real-valued number to the range [-1, 1]. The hyperbolic tangent function has a similar shape to the sigmoid function, with the main difference being that the tanh outputs values in the range [-1, 1], while the sigmoid outputs values in the range [0, 1].

The formula for the hyperbolic tangent function is:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1} \tag{10}$$

Let's say have an input value x = 3. The hyperbolic tangent function can be calculated as follows:

$$\frac{e^{2x}-1}{e^{2x}+1} = \frac{e^{6}-1}{e^{6}+1} = \frac{403.42879-1}{403.42879+1} = 0.99505$$

The graph for the hyperbolic tangent function is on Fig. 7.

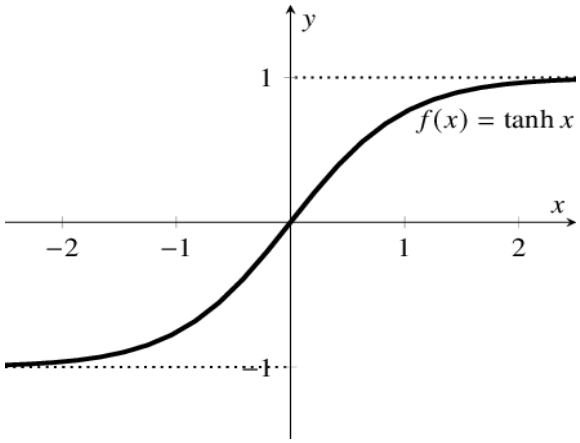

Fig. 7 Hyperbolic tangent graph [13]

3.4.3 Leaky ReLU Function

Leaky ReLU is an activation function used in artificial neural networks. It is a modification of the standard ReLU function, which returns 0 for all negative input values and returns the input value for all positive input values. Leaky ReLU solves the issue of the standard ReLU "dying" (i.e., always outputting 0) for negative input values by adding a small positive slope for negative inputs, instead of returning 0.

A leaky ReLU activation function can be expressed mathematically as follows:

$$f(x) = max(\alpha x, x) \tag{11}$$

Here is a simple mathematical example of the leaky ReLU activation function:

Let's assume α is set to 0.01.

If the input x is 5 functions will return 5

$f(x) = \max(\alpha * 5, 5)$

If the input x is -5, the leaky ReLU function will return -0.05

The max function returns the maximum value between the two inputs, so in the first case, the function returns 5 because it's the larger value, and in the second case, the function returns -0.05 because it's the larger value between 0.01 * -5 and -5. [14]

The graph for the leaky ReLU function is on Fig. 8.

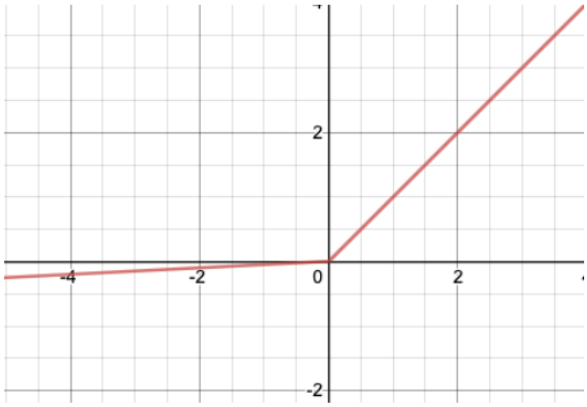

Fig. 8 Leakly ReLU function [15]

3.4.4 SoftMax Function

The softmax function is a commonly used activation function in the output layer of neural networks when the task involves multi-class classification. Given a vector of k real numbers, the softmax function computes the probability distribution over k classes, where the elements of the output vector sum to 1.

The mathematical formula for the softmax function is as follows:

$$S(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \; for \; i = 1 \dots k \tag{12}$$

where j is the i-th element of the input vector z, e is the exponential function, and k is the number of classes. The softmax function transforms each input into a non-negative number that can be interpreted as a probability, and the sum of all the elements in the output vector is equal to 1.

Here's an example of the softmax function;

Consider an input vector z = [2, 1, 0]. The softmax function will calculate the probability distribution over 3 classes as follows:

$$\frac{e^2}{e^2 + e + e\text{^}0} = \frac{7.389}{11099} = 0.7109$$

$$\frac{e^1}{e^2 + e + e\text{^}0} = \frac{2.718}{11099} = 0.2447$$

$$\frac{e^0}{e^2 + e + e\text{^}0} = \frac{1}{11099} = 0.0444$$

So, the output of the softmax function applied to z is [0.7109, 0.2447, 0.0444], which represents a probability distribution over 3 classes where class 1 has the highest probability (0.7109), class 2 has the second highest probability (0.2447), and class 3 has the lowest probability (0.0444). [16]

Here is the graph of the probability for the softmax function for a given example using python;

```
import numpy as np
import matplotlib.pyplot as plt

z = [2, 1, 0]
softmax = np.exp(z) / np.sum(np.exp(z))

plt.bar([1, 2, 3], softmax)
plt.xlabel('Class')
plt.ylabel('Probability')
plt.title('Softmax Function Output')
plt.show()
```

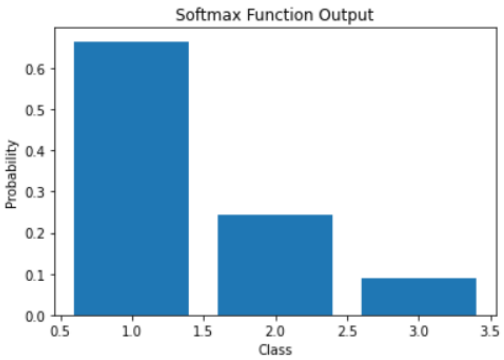

Fig. 9 Softmax function [17]

3.5 Backpropagation

If we examine the training of multi-layered artificial neural networks in three simple stages, the first step is feed-forward evaluation, the second step is the activation function as explained in the previous section, and the last step is the back-propagation phase. It is not possible to use a back-propagation algorithm for classifications that can be separated linearly. There are connections between layers in a neural network that connect the neuron in each layer to the neurons in the next layer, and each connection has a numerical value. These values are called weights. The backpropagation algorithm helps to find the best weight values that fits our neural networks using the backpropagation method to obtain the optimum cost function (loss function).

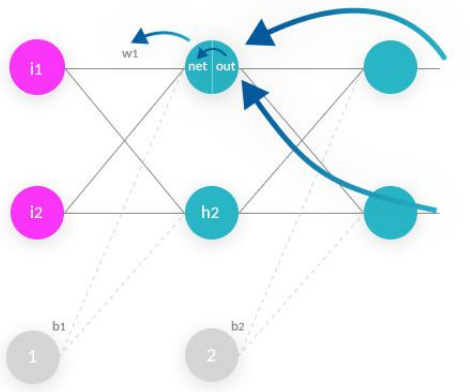

Fig. 10 Backpropagation [18]

3.6 Cost Function

A cost function is a mathematical model that helps us measure the performance of an artificial intelligence algorithm. A cost function is a value calculated by taking the differences between expected and calculated values. There are many different types of activation. The choice should be made according to the type of data given in artificial neural networks and the function of the activation function used. Some of the most commonly used types of activation functions are mean squared error and cross-entropy.

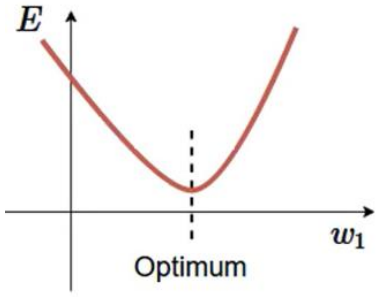

Fig. 11 Cost function graph [*]

3.6.1 Mean squared error loss

Mean square error is the average error between predicted values and actual values. Mean square error is used as a cost function for linearly separable and ReLU activation functions.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \acute{Y}_i)^2. \quad (13)$$

Mean square error loss is the difference between the actual value (Y) and the estimated value (Ý) in a measurement

3.6.2 Cross entropy error loss.

Cross entropy information technology originated, and it is used to solve the probability difference between 2 different groups of attributes. Cross entropy is used in machine learning methods such as logistic regression and artificial neural networks. It is used in sigmoid and softmax activation functions in an artificial neural network. Low-probability events contain large amounts of information while High probability events contain little information. To find the amount of information an event contains, we need to find its probability, and we can find the probability by the given formula.

$$h(x) = -\log\left(P(x)\right) \tag{14}$$

In the given formula h (x) represents the information as the event (x) and P (x) the probability. The cross-entropy between two probability distributions p and q is defined as

$$H(p,q) = -\sum p(x) log q(x) \tag{15}$$

3.7 Gradient Descent

A gradient is a mathematical expression that helps to find the slope on the line by calculating the rate of change of a function. It is used to find the optimum cost function in artificial neural network applications. As will be seen in the graph, we need to calculate the slope of specific points. If the value we obtained is positive it moves in an upward direction if it is a negative value it moves in a downward direction.

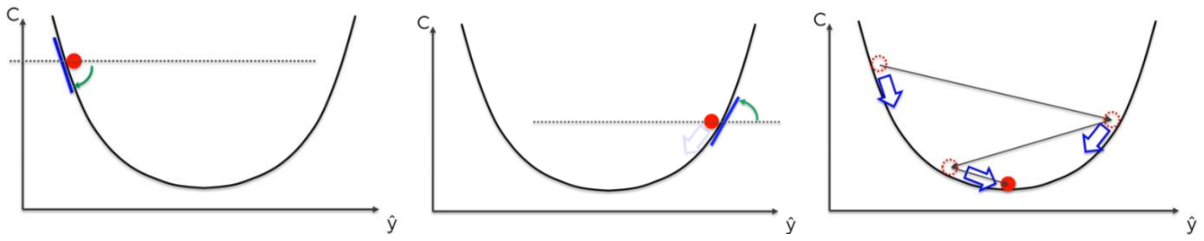

Fig. 12 Gradient descent [19]

We can obtain the slope by calculating the partial derivative of each point. We can calculate those points with the help of the chain rule.

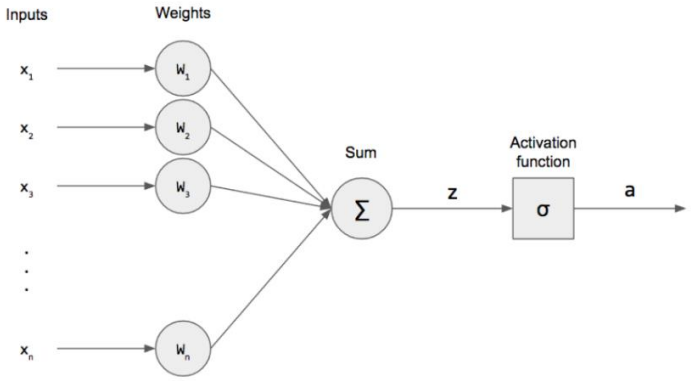

Fig. 13 Perceptron [6]

In the given figure x values represent input values,w weights, z is the net input function, σ represents the activation function, p weighted input and a value represents output. The partial derivative of the cost function calculates concerning first weight with the given formula.

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w1}} = \frac{\partial \mathrm{E}}{\partial \mathrm{a}} * \frac{\partial \mathrm{a}}{\partial \mathrm{z}} * \frac{\partial \mathrm{z}}{\partial \mathrm{p1}} * \frac{\partial \mathrm{p1}}{\partial \mathrm{w1}} \tag{16}$$

After calculating the partial derivative of the cost function concerning the net input function following result is obtained.

$$\frac{\partial \mathrm{E}}{\partial \mathrm{a}} = \frac{\partial}{\partial a}(y-a)^2 = -(y-a) = a-y \tag{17}$$

The following result is obtained after the partial derivative of the net input function concerning the sigmoid activation function with the given formula.

$$\frac{\partial a}{\partial z} = \frac{\partial a}{\partial z} * \frac{1}{1+e^{-z}} = (1-a) * a \tag{18}$$

The following result is obtained after calculating the partial derivative of the total input function multiplied by the weight values concerning weighted input with the given formula.

$$\frac{\partial \mathrm{z}}{\partial \mathrm{p1}} = \frac{\partial}{\partial \mathrm{p1}} \sum_{\mathrm{i}=0}^{\mathrm{n}} \ \mathrm{pi} = 1 \tag{19}$$

The following result is obtained after calculating the partial derivative of weighted input concerning weight with the given formula.

$$\frac{\partial \mathrm{p1}}{\partial \mathrm{w1}} = \frac{\partial}{\partial \mathrm{w1}} x_1 w_1 = x_1 \tag{20}$$

Finally, the following result is obtained by applying the chain rule.

$$\frac{\partial \mathrm{E}}{\partial \mathrm{w1}} = \frac{\partial \mathrm{E}}{\partial \mathrm{a}} * \frac{\partial \mathrm{a}}{\partial \mathrm{z}} * \frac{\partial \mathrm{z}}{\partial \mathrm{p1}} * \frac{\partial \mathrm{p1}}{\partial \mathrm{w1}} = (a-y) * (1-a) * a * x_1 \tag{21}$$

The formula will allow us to obtain the most suitable weight values for the neurons.

3.8 Convolution neural networks

The main purpose of the Convolutional neural network is to preserve the main feature of the given data and to make highly accurate and computational speed predictions and classifications. Image recognition is used in many areas. Some of these areas such as self-driving cars, object and person identification in social media platforms such as Facebook and Instagram, face recognition in security cameras, and person identification in mobile phone cameras, and is becoming more and more important every day. There are few useful software to perform convolution neural networks from a computer. Some of this software is the matvconvnet tool for MATLAB, caffe for c ++, PyTorch, TensorFlow, and theano libraries for python. Also, there are few architectures commonly used in the field of Convolutional neural networks. Some of those popular architectures are lenet, Alexnet, zfnet, vggnet, and resnet.

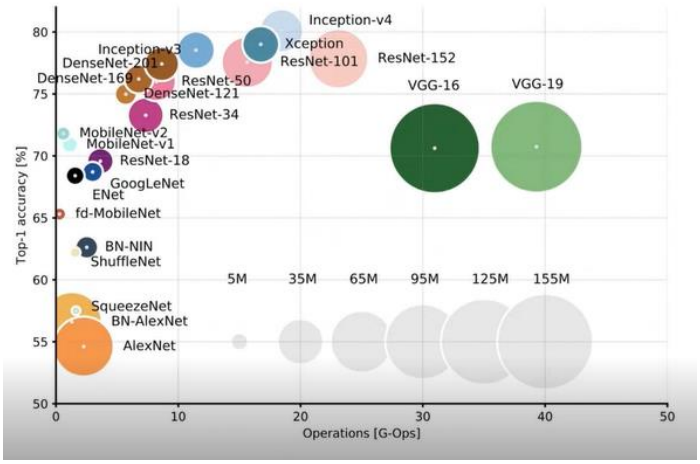

Fig. 14 Architecture of neural networks [20]

Convolution neural networks consist of 4 stages. The first step is the convolution operation and the relu layer. This step explains what a convolution network is and how it works. The second step is pooling. In this step, it will be explained how the algorithm is made more efficient and faster with the concepts of max and average pooling. The third step is flattening. In this step, it is explained how it converts the matrix operations and simplified with the average pooling step to vector operations. Finally, our last step full connection is the last stage before making predictions and classifications with artificial neural networks.

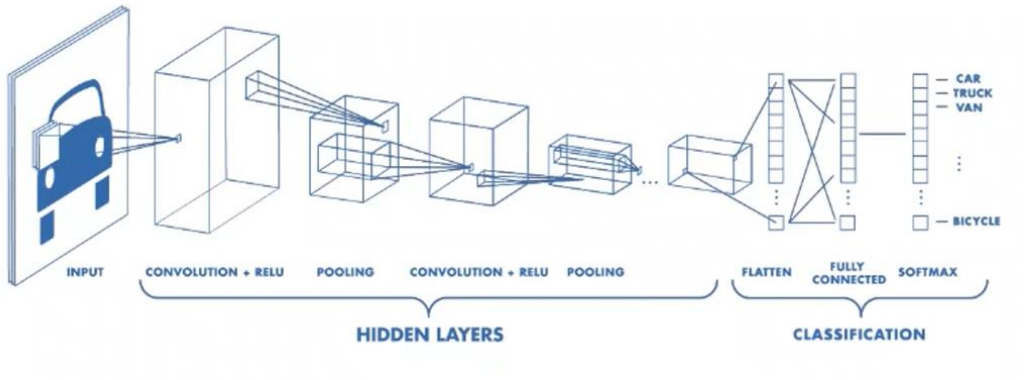

Fig. 15 Convolution structure [21]

3.9 Convolution Operation

An image consists of a large number of pixels. It is possible to classify or make predictions by connecting each pixel that is numbered by the computer to a neuron. If one image consists of too many same pixels algorithm says that it is the same image. Instead of connecting every input pixel to one neuron, it can connect single patches to neurons and it will determine the same feature this will be more efficient than connecting every pixel to one neuron. These patches are called a filter. Also, useful features may be found in more than one place in an image So, so it makes sense to slide a filter all over the image in the hope of extracting that feature in different parts of the image using the same filter. For example, an image 5 * 5 matrices sized with 25 pixels overlap a 2 * 2 filter and the filter with the pixel is sliding in the x and y directions, and the feature map is created by adding the values which are multiplied by each pixel. Feature maps consist of 16 elements and it is a 4x4 matrix.

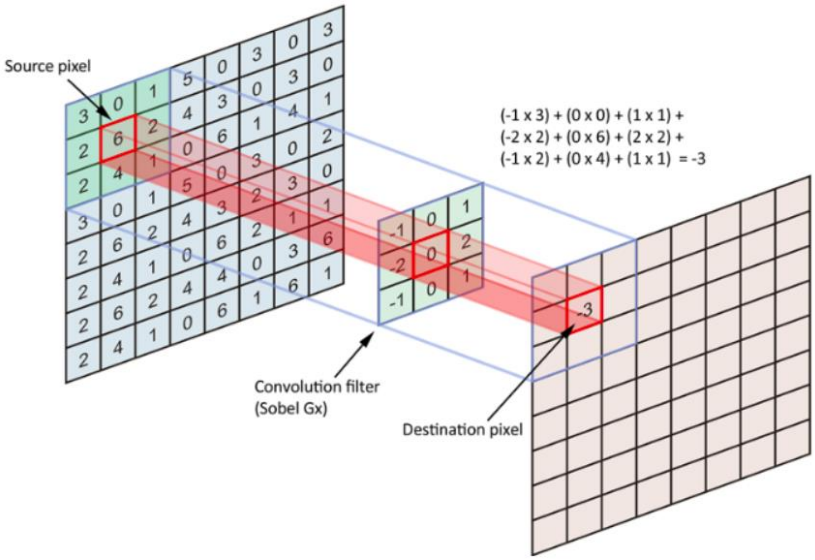

Fig. 16 Filter [22]

After the convolution pooling flatten and full connection processes are performed, the process continues as in artificial neural networks and the filter values are updated with the backpropagation algorithm for the best result. Also, special filters are used to detect lines and edges in the images provided.

3.9.1 Pooling

It is a method used to predict and classify a given input value with maximum computational speed without losing main features. The most commonly used method is max pooling. Max pooling is the method to make smaller matrix operations by taking the maximum value after the convolution process. Another pooling method is average pooling. In the average pooling method, it is performed by taking a mean value instead of the maximum value in the given matrix.

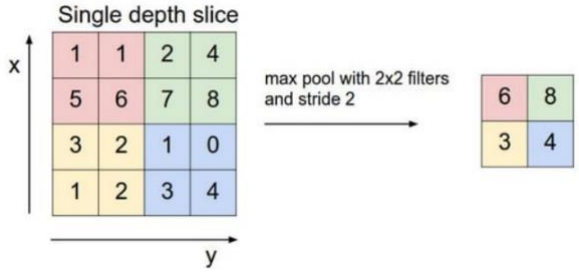

Fig. 17 Max pooling with 2x2 filters and stride 2 [23]

3.9.2 Flattening

Flattening is the easiest step for convolutional neural networks. Basically, the pooled matrix is transformed into only one row as in the figure below.

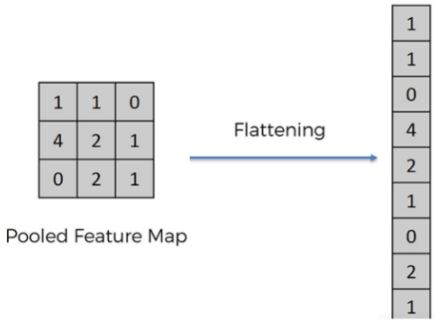

Fig. 18 Flattening of a 3x3 image matrix into a 9x1 vector [24]

3.9.3 Full connection

The full connection step is the phase of connecting flattened pixels to neurons for classification and prediction. Connecting flattened pixels to neurons using the artificial neural network and performing classification and prediction as explained in the previous sections of this project. The activation function and the cost function should be selected according to the type of images we have and the desired output values. For example, if the output value expected from the images imported to the environment is based on two different predictions value then a binary-based activation function can be selected. The selected cost function plays a critical role in making high-precision predictions due to it allows the selection of appropriate filters with a backpropagation algorithm.

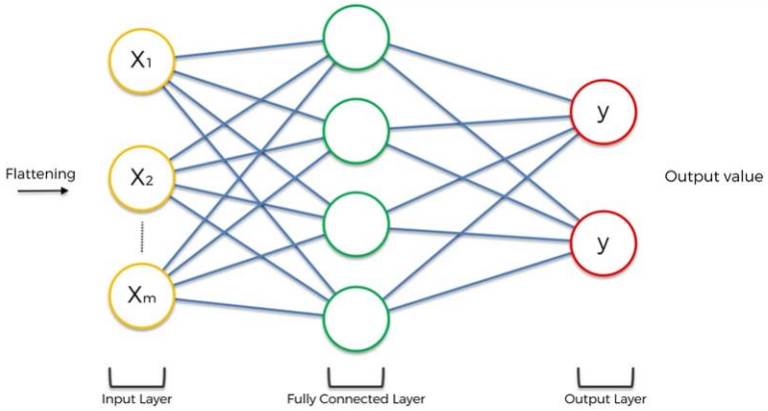

Fig. 19 Full connection [25]

4 Sample Data Description

Speech recognition using spectrum analysis has been studied since before the spread of computers in the 1970s. The use of computational methods is already described by Professor Josef Psutka in 1995. One of the basics is the use of a fast Fourier transform and a non-uniform (logarithmic) distribution band when the sum of the amplitudes in these bands creates a vector that is characteristic of the voice. In this way, at least spoken vowels can be distinguished with minimal error under the conditions of a speaker. This sample data will use a comparison of both methods support vector machine and artificial neural networks. This section will describe how to obtain these vectors for the sample data to use in both methods.

4.1 Acquisition of data for recognition

The data comes from [26], where it has been obtained using the sound card and the sound recorder application that is included in the Windows operation system. The data was recorded in a wav format that is not packaged. Voices that can be pronounced in length have been selected for the recording seconds after certain subjective consideration of the vowels a, e, i, o, u, and the consonant s.

4.2 Frequency conversion and division into bands

The application for data preparation [26] was written in Pascal, where the powerful RAD environment Lazarus is available. A sample FFT solution for Delphi is possibly found on the Internet but was used from the author's archives.

The program allows you to load directly wav files in mono or stereo version for 16-bit samples. After loading it you need to set whether it is a mono or stereo recording and in the latter case which channel to read. All analyses were performed for stereo recording and left channel. Afterward, 4096 vowels were taken for each letter. This process was repeated for each letter vowel. The sample obtained was output as an Excel file.

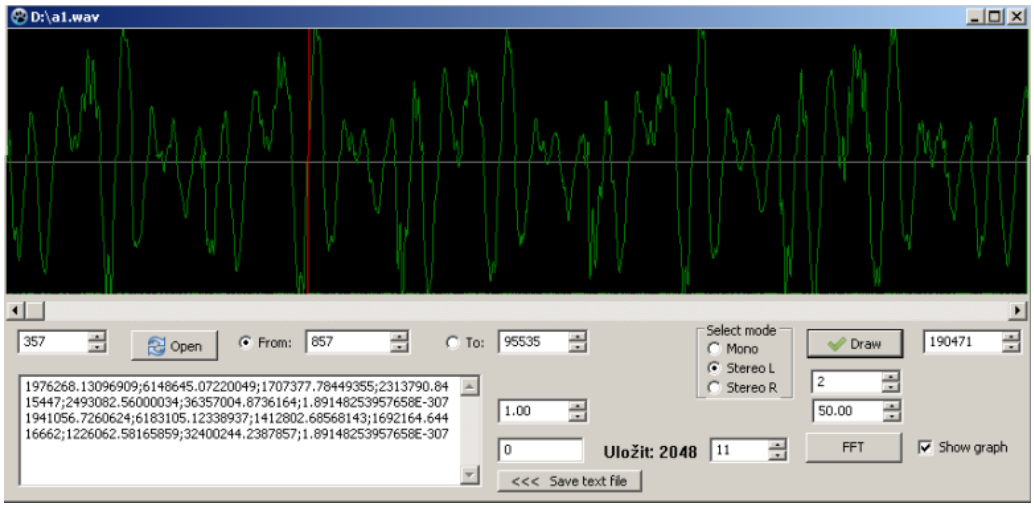

Fig. 20 Voice signal in the time domain [26]

5 Appliance of the Methods in Python

5.1 Machine learning applications using Python

We do not need to implement complex mathematical formulas to use machine learning applications in practice. We can apply quickly mathematical statistics models to the data given by using such libraries; sci-kit learn, pandas, numpy, matplot, and scipy and we can also make data visualization for simple machine learning applications.

Pandas: This library is one of the most preferred library and it allows us to import the data set in CSV format for machine learning applications. Also, it converts to dictionary, list, and arrays to a data frame in Python. [27]

NumPy: NumPy name is combined from the first three letters of NumPy and the first two letters of python. NumPy library is one of the useful libraries that are frequently used in machine learning applications. The main purpose of the Numpy library is to work with multidimensional arrays. A one-dimensional array is called a vector and multidimensional arrays occur in matrices. [17]

Matplot: Matplot library is one of the most known libraries of python, it allows us to easily create complex graphs with only a few lines of code and it is frequently used in machine learning applications. [28]

Scikit learns: sci-kit learn is necessarily using the machine learning library including supervised and unsupervised learning methods. The scikit learn library works compatible with numpy, pandas, matplot, and it is easily implemented with pre-defined phyton codes using such as classification, regression, clustering, etc. scikit learn library plays an important role in the svm classifier, which is the field of our article. The data given with the codes pre-defined from the library can be easily analyzed, classified and also data visualities can be done with the help of the mat plot library. [29]

We extracted the Fourier transform values given with the excel file to the phyton environment and defined each sound to the environment then we normalized the given values. We chose the linear core method because of its high accuracy rate, and then we completed the classification process with the help of support vector machines. Finally, we realize the f1 score test to measure our accuracy rate and printed the result.

5.2 Support Vector Classifier by Using Python

Python has a very important role in supervised support vector classifications. Support vector classifications are frequently used libraries; numpy, matplot, and sci-kit learn. We will talk about how to implement given data using support vector machines via python.

Firstly, it starts the process by implanting the given libraries into our Python environment.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

We import the sample data sets that we will use in the classification process in the environment of Python using the Scikit-learn library. We can also extract the CSV files and edit the data using panda's libraries for classification applications. After that the using Python code below we are setting the number of samples, the density of samples, the color of the samples, and the position of the sample on the graph

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```

we import the support vector classifier class from the sklearn library in the following section. Then, we specify the type of the kernel function and C parameter to our environment. Since we do a linear separable classification, we choose a linear kernel and determine our parameter. Selecting the parameter selection in high numbers will help us avoid misclassification. Finally, we fit the clf value we set to X, y.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.svm import SVC
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
clf = SVC(kernel='linear', C=1000)
clf.fit(X, y)
```

To maximize hyperplane in linear classification, we need to use two decision boundaries and the code we wrote below will help us determine decision boundaries for the x and y axis.

```
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
```

The code found in Syntax allows us to define a linear space between the x and y axis and create square meshes and define a decision function with our clf value, which we have previously defined using the svc linear kernel function.

```
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
```

```
Z = clf.decision_function(xy).reshape(X.shape)
```

We draw the XX, YY, and Z decision boundaries that we have already determined, and the code above allows us to adjust the color type and thickness of our decision line. Finally, we draw our hyperplane with the boundaries we have determined. This is the dividing line that maximizes the margin between the two sets of points.

```
ax.contour(XX, YY, Z, colors='k',
levels=[-1, 0, 1], alpha=0.5,
linestyles=['--', '-', '--'])
ax.scatter(clf.support_vectors_[:, 0],
clf.support_vectors_[:, 1],
s=100, linewidth=1, facecolors='none');
```

As a result, we created a linearly separable classification group consisting of two different classification groups and fifty samples in each of the two classification groups. Using the sci-kit learn library, we separated the two groups and maximized the margin separating the hyperplane, and obtained the output as shown in the picture below.

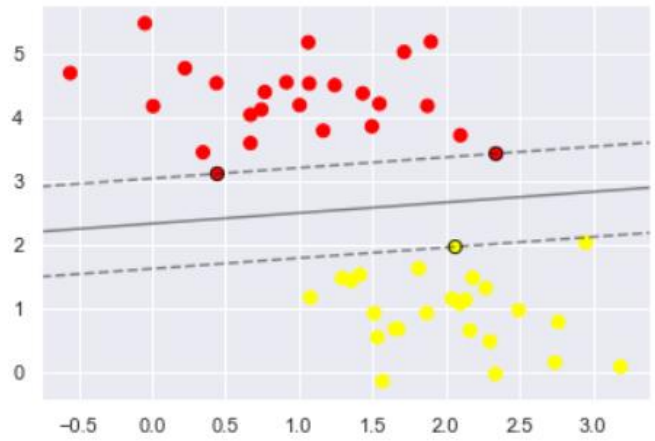

Fig. 21 Hyperplane maximizing two attributes group

5.3 Kernel SVM

I work on the case that is linearly separable in the previous section. In non-linear classifications, similar to linear separable situations, the kernel method selection which is included in the sci-kit learns library will be applied. Depending on the given data, the sigmoid, the polynomial or gaussian kernel can be used. First, we start by importing the libraries we will use.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.svm import SVC
import pandas as pd
```

We can import the data we want to classify using our pandas library into our environment. We can display the first 5 lines in our data with the head command.

```
df = pd.read_csv("samples.csv")
df.head()
```

After that data preprocessing will be done depending on our data. We will import our test and separation class from scit learn model selection library and we will test and split the given data.

```
X = samplesdata.drop('Class', axis=1)
 y = samplesdata['Class']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

For choosing appropriate kernel model depending on the given data we use svc class from our scikit learn library then we specify one of the rbf , Gaussian or polynomial kernel methods.

```
clf = SVC(kernel='sigmoid')
clf.fit(X, y)
```

Finally, we perform the classification using the kernel method with the command given below.

```
y_pred = clc.predict(X_test)
```

5.3.1 Example 1)

Obs.	X_1	X_2	Y
1	3	4	Red
2	2	2	Red
3	4	4	Red
4	1	4	Red
5	2	1	Blue
6	4	3	Blue
7	4	1	Blue

Fig. 22 Example Table

(a) given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label.

(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane

(c) On your sketch, indicate the margin for the maximal margin hyperplane.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = [1,1,1,1,0,0,0]
clf = svm.SVC(kernel='linear', C = 2)
clf.fit(X,y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
```

```
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```

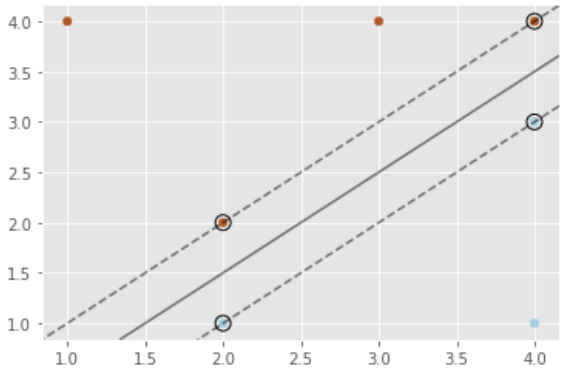

Fig. 23 Graph shows the exercise output value (own work).

5.3.2 Example 2)

Cancer types obtained from patients in the given dataset are given in 10 different parameters. By examining these data, we will create a new class and classify using the kernel method.

```
import pandas as pd
import pylab as pl
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import svm
cell_df = pd.read(r"C:\Users\burak\Downloads\cell_sample.csv')
cell_df.head(10) # printing the first 10 row
cell_df.head(10)
cell_df.dtypes # we are checking if the all values are numerical
cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnul
l()]  #we are converting the Barenuc row all values numerical
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes
feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpi
Size', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
X = np.asarray(feature_df) #  defining the X values for 9 cancer cell types
 in the numpy array
cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class']) #  defining the Class values for y variabl
e which includes 2(benign cells ),4(malignant cells)
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, r
andom_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)  # creating the test and t
raning set for predefined X and y and printing
clf = svm.SVC(kernel='rbf') #choosing the our kernel method
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
predicted [0:20]  # printing the first 20 predicted value
```

This is the our printed first 10 column for the imported training data set ;

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

Fig. 24 Output value from the 9th column from the syntax

This is the predicted first 20 values ;

Classification Result : Array([2, 4, 2, 4, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4, 4, 2, 2, 2, 2, 2])

5.4 Multi-layer perceptron using Python

5.4.1 Example 1)

Cancer types obtained from patients in the given dataset are given in 10 different parameters. By examining these data, we will create a new class and classify using artificial neural networks.

```
1. cell_df  = pd.read_csv('cell_samples.csv')
2. cell_df.dtypes # we are checking if the all values are numerical
3. cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnul
   l()]  #we are converting the Barenuc row all values numerical
4. cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
5. feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpi
   Size', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
6. X = np.asarray(feature_df) #  defining the X values for 9 cancer cell types
    in the numpy array
7. cell_df['Class'] = cell_df['Class'].astype('int')
8. y = np.asarray(cell_df['Class']) #  defining the Class values for y variabl
   e which includes 2(benign cells ),4(malignant cells)
9. X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, r
   andom_state=4)  #  test size 20 percent and the train size 80 percent choos
   en
10. model=MLPClassifier( hidden_layer_sizes=(150,100,50), random_state=1,) # 3
    hidden layer choosen
11. model.fit(X_train,y_train)
12. model.predict(X_test)
13. model.predict(X_train)
14. #using the f1 score test to know our accuracy of classifier and printing th
    e report for each attribute group
15. report = classification_report(model.predict(X_test),y_test)
16. print(report)
```

```
              precision    recall  f1-score   support

           2       0.94      1.00      0.97        85
           4       1.00      0.90      0.95        52

    accuracy                           0.96       137
   macro avg       0.97      0.95      0.96       137
weighted avg       0.97      0.96      0.96       137
```

Fig. 25 The f1 score

6 Evaluation of the Sample Data in Python

The file has been processed by a fast Fourier transform (FFT), with 4096 points (to 4096 frequencies). Then the frequency domain has been aggregated, not evenly, ranges of frequencies are in the heads of the columns (as 34 - 99). Each row represents one sample. The last frequency range is nonsense, because the FFT of real number series is symmetrical. So the first six numbers in each of the rows can be used for a supervised learning. Each of the vowels is on a separate sheet. On the first sheet, some normalization is done, with a demand of the overall sum being 10000. This normalized value would be better, because the sound intensity should be ignored while using the FFT.

```
1    import pandas as pd
2    import numpy as np
3    from sklearn.model_selection import train_test_split
4    from sklearn.metrics import classification_report, accuracy_score
5    from sklearn.svm import SVC
6    from sklearn.neural_network import MLPClassifier
7    from sklearn.metrics import f1_score
8    df = pd.read_excel("fft.xlsx",sheet_name=None)

9    # defining the each sound from importing file
10   a=df["a"]
11   e = df["e"]
12   i =df["i"]
13   o=df["o"]
14   u=df["u"]
15   s=df["s"]

16   # we normalized data by dividing sum of all columns and multiply by 10000
     each value
17   ss= (s.apply(lambda x: x/x.sum(axis=0), axis=1))* 10000
18   us = (u.apply(lambda x: x/x.sum(axis=0), axis=1))* 10000
19   os = (o.apply(lambda x: x/x.sum(axis=0), axis=1))* 10000
20   iss = (i.apply(lambda x: x/x.sum(axis=0), axis=1))* 10000
21   es = (e.apply(lambda x: x/x.sum(axis=0), axis=1))* 10000

22   # we take first 50 columns of normalized data for each sound
23   ssound = ss.iloc[:50,:]
24   usound = us.iloc[:50,:]
25   osound = os.iloc[:50,:]
26   isound = iss.iloc[:50,:]
27   asound = a.iloc[:56,10:17]
28   esound = es.iloc[:50,:]

29   # specifying class number for each attribute groups
30   ssound["class"]=5
31   usound["class"]=4
32   osound["class"]=3
33   isound["class"]=2
34   esound["class"]=1
35   asound["class"]=0

36   # equalizing the all sounds columns and then merging the whole sounds using
     concat function
37   asound.columns= esound.columns
38   isound.columns=asound.columns
39   osound.columns=esound.columns
40   usound.columns=asound.columns
```

```
ssound.columns=asound.columns
combine = pd.concat([asound,esound,isound,osound,usound,ssound])
# SVM Classifier
svm_accuracies = []
for i in range(10):
X_train,X_test,y_train,y_test = train_test_split(combine.iloc[:,:-
1],combine.iloc[:,-1],test_size=0.8)
model = SVC(kernel = 'poly',C = 10000, gamma = 'auto')
model.fit(X_train,y_train)
svm_predictions = model.predict(X_test)
svm_report = classification_report(svm_predictions, y_test,
output_dict=True)
svm_accuracies.append(svm_report['weighted avg']['f1-score'])

svm_avg_accuracy = np.mean(svm_accuracies)
print("SVM Classifier Average Accuracy: ", svm_avg_accuracy)


# MLP Classifier
mlp_accuracies = []
for i in range(10):
X_train,X_test,y_train,y_test = train_test_split(combine.iloc[:,:-
1],combine.iloc[:,-1],test_size=0.8)
mlp =
MLPClassifier(hidden_layer_sizes=(150,100,50),activation="tanh",random_state
=1)
mlp.fit(X_train, y_train)
mlp_predictions = mlp.predict(X_test)
mlp_report = classification_report(mlp_predictions, y_test,
output_dict=True)
mlp_accuracies.append(mlp_report['weighted avg']['f1-score'])

mlp_avg_accuracy = np.mean(mlp_accuracies)
print("MLP Classifier Average Accuracy: ", mlp_avg_accuracy)
```

The results of the SVM and MLP classifiers show that the SVM classifier performs better in terms of accuracy with an average accuracy of approximately 0.93 compared to the MLP classifier which has an average accuracy of approximately 0.86. This indicates that the SVM classifier is better suited for this problem as it can correctly classify the data with a higher degree of accuracy. The SVM classifier uses a polynomial kernel and has a high value of regularization parameter "C" which helps in avoiding overfitting, and a value of 'auto' for gamma. This helps in finding a balance between underfitting and overfitting and results in high accuracy. On the other hand, the MLP classifier uses multiple hidden layers of 150,100,50 nodes each. Increasing the number of hidden layers and increasing the number of iterations may result in higher accuracy for the MLP classifier. However, this may also increase the computational cost and result in overfitting.

7 Conclusion

The concepts of machine learning and artificial intelligence were introduced and their application in vowel classification was discussed in this thesis. The theory of support vector machines (SVM), including kernel methods, and the different activation functions used in neural networks were covered. The SVM model was trained using three different kernel methods, while the MLP model was trained using different activation functions. Python libraries such as sci-kit-learn, NumPy, matplotlib, and pandas were utilized with toy datasets to compare the performance of support vector machines and neural networks. Different datasets were used to evaluate the accuracy of different kernel methods for SVM and different activation functions for neural networks.

This thesis highlights the importance of selecting the appropriate machine-learning algorithm for vowel classification tasks. Both SVM and MLP have their strengths and weaknesses, and the choice between the two depends on the specific requirements and limitations of the project. SVM is computationally efficient and can handle large datasets, making it suitable for classification systems. Neural networks, on the other hand, can learn complex non-linear relationships in data and are well-suited for classification tasks. Two different machine learning models, Support Vector Machine (SVM) and Multi-layer Perceptron (MLP) were trained on a dataset consisting of sounds of different vowels. Data has been processed by a fast Fourier transform (FFT), with 4096 points (to 4096 frequencies). Then the frequency domain has been aggregated, not evenly, ranges of frequencies are in the heads of the columns (as 34 - 99). Each row represents one sample. The data was normalized and split into training and testing sets, with 80% used for testing and 20% for training. Different Kernel methods and activation functions are trialed for better accuracy for both methods.

The results of this experiment suggest that SVM can be more effective than neural networks in vowel classification tasks. However, it's important to note that the difference in performance could vary based on the specific dataset and parameters used. Best accuracy results were obtained using polynomial kernel for SVM and tanh activation function for the artificial neural networks. The results showed that the SVM model had a higher average accuracy compared to the MLP model, with an average accuracy of 93.01% compared to 86.15%.

8 References

[1] R. Raj, “https://www.enjoyalgorithms.com,” [Online]. Available: https://www.enjoyalgorithms.com/blog/support-vector-machine-in-ml.

[2] P. Bhapkar, "medium," 5 3 2018. [Online]. Available: https://medium.com/@prashantbhpkr/demystifying-support-vector-machines-35fca3ecac17.

[3] L. Chen, "Medium.com," 17 11 2018. [Online]. Available: https://medium.com/bite-sized-machine-learning/support-vector-machine-explained-soft-margin-kernel-tricks-3728dfb92cee.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel and E. Duchesnay, "scikit-learn," 10 12 2011. [Online]. Available: https://scikit-learn.org/stable/datasets/toy_dataset.html.

[5] W. Wolberg, W. Street and O. Mangasarian, "Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates.," *Cancer Letters,* pp. 77(2-3):163-71, 15 3 1994.

[6] A. Dalal, "medium.com," 15 8 2020. [Online]. Available: https://medium.com/analytics-vidhya/basic-of-neural-network-956b8f190f3a.

[7] C. Bento, "Towards Data Science," 21 9 2021. [Online]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141.

[8] T. LEIBOVICH-RAVEH, D. LEWIS, S. AL-RUBAIEY, K. AL-RUBAIEY and D. ANSARI, "A new method for calculating individual subitizing ranges," 2018. [Online]. Available: https://www.researchgate.net/publication/325868989.

[9] A. Jacobs, R. Pfitscher, R. Santos, M. Franco, E. John Scheid and L. Granville, "Artificial neural network model to predict affinity for virtual network functions.," 2018.

[10] A. L. Maas, A. Y. Hannun and A. Y. N. (2014), 2014. [Online]. Available: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[11] Y. Sako, 2018. [Online]. Available: https://medium.com/@u39kun/is-the-term-softmax-driving-you-nuts-ee232ab4f6bd.

[12] T. Leibovich-Raveh, D. Lewis, S. Al-Rubaiey, Kadhim and D. Ansari, "A new method for calculating individual subitizing ranges.," 2018.

[13] A. Jacobs, R. Pfitscher, R. Santos, M. Franco, E. Scheid and L. Granville, "Researchgate," 2018. [Online]. Available: https://www.researchgate.net/publication/326279910_Artificial_neural_network_model_to_predict_affinity_for_virtual_network_functions.

[14] Greatlearningteam, "mygreatlearning," 22 3 2022. [Online]. Available: https://www.mygreatlearning.com/blog/relu-activation-function/#:~:text=ReLU%20activation%20function%20formula&text=ReLU%20function%20is%20its%20derivative,range%20from%200%20to%20infinity..

[15] S. Singh, "deeplearninguniversity," [Online]. Available: https://deeplearninguniversity.com/leaky-relu-as-an-activation-function-in-neural-networks/.

[16] V. Zhou, "victorzhou," 22 7 2019. [Online]. Available: https://victorzhou.com/blog/softmax/.

[17] S. V. D. Walt, S. C. Colbert and G. Varoquaux, "numpy," 2011. [Online]. Available: https://numpy.org/.

[18] "Indiantechwarriors," [Online]. Available: https://indiantechwarrior.com/backpropagation-in-neural-networks/.

[19] D. Das, "http://dhrubajitdas44.blogspot.com/," [Online]. Available: http://dhrubajitdas44.blogspot.com/2018/01/how-to-minimize-cost-function-in-neural.html.

[20] E. Culurciello, "https://towardsdatascience.com/," [Online]. Available: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba.

[21] "bs.org.tr," [Online]. Available: https://www.bs.org.tr/blog/yapay-zeka-kapsul-aglari/96.

[22] R. Hachilif, "researchgate," [Online]. Available: https://www.researchgate.net/figure/2D-convolution-with-filter-size-3x3-49_fig4_334974839.

[23] Ş. İ. Serengil, "https://bilisim.io/," [Online]. Available: https://bilisim.io/2018/01/07/konvolusyonel-noral-aglara-kisa-bir-giris/.

[24] S. Team, "https://www.superdatascience.com/," [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

[25] S. Team, "https://www.superdatascience.com/," [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection.

[26] V. Hlaváč, "Ukázka rozpoznávání řeči s využitím klasifikace neuronovou sítí," in *Nové metody a postupy v oblasti přístrojové techniky, automatického řízení a informatiky 2020*, Zámek Lobeč, 2020.

[27] W. McKinney, "pandas," 2010. [Online]. Available: https://pandas.pydata.org/.

[28] J. D. Hunter, "matplotlib," 2007. [Online]. Available: https://matplotlib.org/.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel and E. Duchesnay, "Scikit-learn," 12 10 2011. [Online]. Available: https://scikit-learn.org/stable/.

[30] A. Jacobs, R. Pfitscher, R. Santos, M. J. Franco, E. Scheid and L. Granville, "Artificial neural network model to predict affinity for virtual network functions," 2018/04/01.

[31] T. Leibovich-Raveh, D. Lewis, S. K. Al-Rubaiey and D. Ansari, "A new method for calculating individual subitizing ranges," 2018/06/20.

[32] T. Hastie, R. Tishirani and J. Friedman, The element of Statistical Learning, 2009.

[33] M. Stewart, "“Simple Introduction to Convolutional Neural Networks”, Towards Data Science, February 2019.," [Online]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.

[34] M. Sardogan, A. Tuncer, Y. Ozen and P. Leaf, "Disease Detection and Classification used on CNN with LVQ Algorithm," in *Institute of Science and Engineering, Department of Computer Engineering, 3rd international conference on computer science and engineering*, 2018.

[35] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.