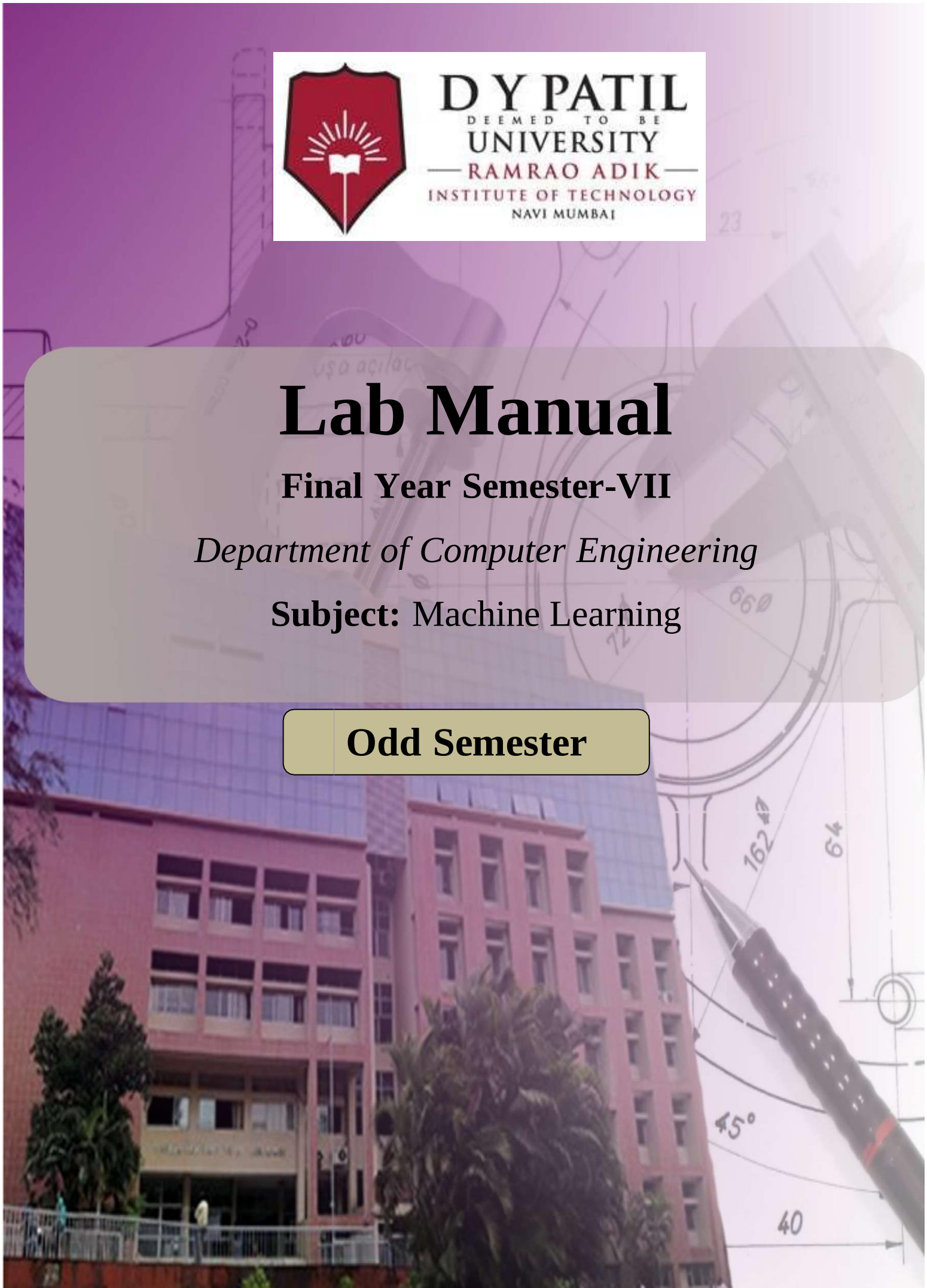


D Y PATIL
DEEMED TO BE
UNIVERSITY
RAMRAO ADIK
INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Lab Manual

Final Year Semester-VII

Department of Computer Engineering

Subject: Machine Learning

Odd Semester

Institute Vision, Mission & Quality Policy

Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems facedby our country and world at large.

Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

Quality Policy

ज्ञानधीनं जगत् सर्वम।

Knowledge is supreme.

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodologies.

Our Motto: If it is not of quality, it is NOT RAIT!

Dr. Vijay D. Patil
President, RAES

Department Vision & Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

- To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering.
- To provide the diverse platforms of sports, technical, co curricular and extracurricular activities for the overall development of student with ethical attitude.
- To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service.
- To collaborate with IITs, reputed universities and industries for the technical and overall uplifment of students for continuing learning and entrepreneurship.

Index

List of Experiments

Sr. No.	Experiments Name
1	To implement concept of bias over fitting and under fitting.
2	To implement Linear Regression.
3	To implement Logistic Regression.
4	To implement Ensemble learning (bagging/boosting).
5	To implement SVM.
6	To implement DB Scan.
7	To implement PCA/SVD.
8	**Mini project/Case study on any machine learning application.**

n

Course Outcome & Experiment Plan

Course Objectives:

1.	To introduce the basic concepts and techniques of Machine Learning.
2.	To acquire in depth understanding of various supervised and unsupervised algorithms
3.	To be able to apply various ensemble techniques for combining ML models.
4.	To demonstrate dimensionality reduction techniques.

Lab Outcomes:

CO1	To implement an appropriate machine learning model for the given application.
CO2	**Apply regression methods for learning relationships between features of the data.**
CO3	To implement ensemble techniques to combine predictions from different models.
CO4	**To implement learning with classification.**
CO5	**To implement different clustering methods in machine learning.**
CO6	To implement the dimensionality reduction techniques.

Experiment Plan:

Module No.	Week No.	Experiments Name	Course Outcome
1.	W1	To implement concept of bias over fitting and under fitting.	CO1
2.	W2	To implement Linear Regression.	CO2
3.	W3	To implement Logistic Regression.	CO2
4.	W4	To implement Ensemble learning (bagging/boosting)	CO3
5.	W5	To implement SVM	CO4
6.	W6	To implement DB Scan	CO5
7.	W7	To implement PCA/SVD	CO6
8.	**Mini project/Case study on any machine learning application**		

Mapping Course Outcomes (CO) - Program Outcomes (PO)

Subject Weight	Course Outcomes		Contribution to Program outcomes PO'S											
			1	2	3	4	5	6	7	8	9	10	11	12
THEORY 20%	CO1	To implement an appropriate machine learning model for the given application.	1	1	2	1	2	1		1				1
	CO2	**Apply regression methods for learning the relationships between features of the data.**		2	2	2	1	1	1					1
	CO3	To implement ensemble techniques to combine predictions from different models.		2	2	1	2	1			1			1
	CO4	**To implement learning with classification.**		2	2	1	3	1						1
	CO5	**To implement different clustering methods in machine learning.**		2	2	1	3	1						1
	CO6	To implement the dimensionality reduction techniques.		2	2	1	2		1				1	1
PRACTICAL 80%	CO1	To implement an appropriate machine learning model for the given application.	1	1	2	1	2	1		1				1
	CO2	**Apply regression methods for learning the relationships between features of the data.**		2	2	2	1	1	1					1
	CO3	To implement ensemble techniques to combine predictions from different models.		2	2	1	2	1			1			1
	CO4	**To implement learning with classification.**		2	2	1	3	1						1
	CO5	**To implement different clustering methods in machine learning.**		2	2	1	3	1						1
	CO6	To implement the dimensionality reduction techniques.		2	2	1	2		1				1	1

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
CSL7011	Machine Learning	03	02	--	03	01	--	04

Course Code	Course Name	Examination Sc eme		
		Term Work	Oral & Practical	Total
CSL7011	Machine Learning	25	25	50

Term Work:

1 Term work should consist of 6 experiments.
2 Journal must include one mini project/case study on any machine learning application.
3 The final certification and acceptance of term work ensures the satisfactory performance of laboratory work and minimum passing marks in term work.
4 Total 25 Marks (Experiments & Assignments: 15-marks, Attendance: 05-marks, mini project: 05-marks)

Oral & Practical exam.

Based on the entire syllabus **CSC7011** Machine Learning **and CSL7011**: Machine Learning Lab

Experiment No: 1

Over fitting and Under fitting

Experiment -1

Aim: To implement concept of bias over fitting and under fitting.

Software required: (Students should write **Software required** based on software used for implementing program e.g. Python(version) or Anaconda navigator etc)

Theory:

Over fitting, under fitting, and the bias-variance tradeoff are foundational concepts in machine learning. A model is **over fit** if performance on the training data, used to fit the model, is substantially better than performance on a test set, held out from the model training process. For example, the prediction error of the training data may be noticeably smaller than that of the testing data. Comparing model performance metrics between these two data sets is one of the main reasons that data are split for training and testing. This way, the model's capability for predictions with new, unseen data can be assessed.

When a model over fits the training data, it is said to have **high variance**. One way to think about this is that whatever variability exists in the training data, the model has "learned" this very well. A model with high variance is likely to have learned the noise in the training set. Noise consists of the random fluctuations, or offsets from true values, in the features (independent variables) and response (dependent variable) of the data. Noise can obscure the true relationship between features and the response variable. Virtually all real-world data are noisy.

If a model is not fitting the training data very well, this is known as **underfitting**, and the model is said to have **high bias**. In this case, the model may not be complex enough, in terms of the features or the type of model being used.

Implementation:

```
#!/usr/bin/env python
# coding: utf-8
# In[38]:
get_ipython().run_line_magic('matplotlib', 'inline')
#
# # Underfitting vs. Overfitting
#
# Model used: Linear regression with polynomial features to approximate nonlinear functions.
# In[39]:
import numpy as np  # For numerical calculation and matrix handling
import matplotlib.pyplot as plt  # For plotting
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures  # For pre-processing
from sklearn.linear_model import LinearRegression  # For Linear regression model
from sklearn.model_selection import cross_val_score  # For evaluation
np.random.seed(0)  # To control the random number generator
# ### Generate data for regression
# In[40]:
def gen_target(X):
 return np.cos(1.5 * np.pi * X)
# #### Define constants
# In[41]:
n_records = 30        # Total number of records
degrees = [1, 4, 30]   # Degree(s) of linear regression model
```

```
types = ['Underfitting', 'Perfect fitting', 'Overfitting']
# #### Generate features and targets
# In[42]:
X = np.sort(np.random.rand(n_records))    # Randomly generate data points (features)
y = gen_target(X) + np.random.randn(n_records) * 0.1  # Generate regression output with additive noise
# ### Build and Evaluate model
# #### Plotting function
#
# The plot shows the true function, the function approximated using linear regression model, and the records with additive noise used for building linear regression model. The model uses polynomial features of different degrees.
# In[43]:
def plot_test(X, y, deg, title=""):
    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="LR function (deg="+str(deg)+")")
    plt.plot(X_test, gen_target(X_test), '--r', label="True function")
    plt.scatter(X, y, facecolor="b", s=20, label="Training records")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title(title)
# - **Underfitting:** A linear function (polynomial with degree 1) is not sufficient to fit the training records.
# - **Overfiting:** A polynomial of degree > 1 that approximates the true function almost perfectly and for higher degrees the model learns the noise of the training data.
# **Underfitting**/**Overfitting** can be quantitatively evaluated by using cross-validation to calculate the mean squared error (MSE) on the validation set. The higher MSE, the less likely the model generalizes correctly from the training data.
# In[44]:
plt.figure(figsize=(14, 5))   # Generate figure window
for i, (deg, t) in enumerate(zip(degrees, types)):
    ax = plt.subplot(1, len(degrees), i + 1)  # Generate subplot for each degreee
    poly_feat = PolynomialFeatures(degree=degrees[i], include_bias=False)
    lr = LinearRegression()
    # Make regression pipeline
    pipeline = Pipeline(
        [
            ("poly_feat", poly_feat),
            ("lr", lr),
        ]
    )
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using 10-fold cross-validation and MSE
    scores = cross_val_score(pipeline, X[:, np.newaxis], y, scoring="neg_mean_squared_error", cv=10)
    # Plot results with original data
    plot_test(X, y, deg)
    print("Degree {}\nMSE = {:.3e}(+/- {:.3e}) \n".format(deg, -scores.mean(), scores.std()))
plt.show()
```

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Sample:

Reasons for over fitting are as follows-

- Data used for training is not cleaned and contains noise (garbage values) in it.
- The model has a high variance.
- The size of the training dataset used is not enough.
- The model is too complex.

Conclusion:

(Students should write conclusion on their own)

Sample:

From this experiment implementation, we conclude that under fitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the over fitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data.

Experiment -2

Aim: To implement Linear Regression.

Software required: (Students should write **Software required** based on software used for implementing program)

Theory: Linear regression is a basic predictive analytics technique that uses historical data to predict an output variable. It is popular for predictive modeling because it is easily understood. Linear regression models have many real-world applications in an array of industries such as economics (e.g. predicting growth), business (e.g. predicting product sales, employee performance), social science (e.g. predicting political leanings from gender or race), healthcare (e.g. predicting blood pressure levels from weight, disease onset from biological factors), and more.

The basic idea is that if we can fit a linear regression model to observed data, we can then use the model to predict any future values. There are two kinds of variables in a linear regression model:

- The input or predictor variable is the variable(s) that help predict the value of the output variable. It is commonly referred to as X.
- The output variable is the variable that we want to predict. It is commonly referred to as Y.

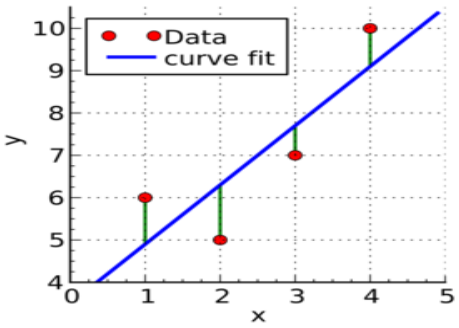
To estimate Y using linear regression, we assume the equation:

$Y_e = \alpha + \beta X$

where Y_e is the estimated or predicted value of Y based on our linear equation.

Our goal is to find statistically significant values of the parameters α and β that minimize the difference between Y and Y_e. If we are able to determine the optimum values of these two parameters, then we will have the line of best fit that we can use to predict the values of Y, given the value of X. How do we estimate α and β? We can use a method called ordinary least squares.

Ordinary Least Squares



Green lines show the difference between actual values Y and estimate values Y_e

The objective of the least squares method is to find values of α and β that minimize the sum of the squared difference between Y and Y_e. We will not go through the derivation here, but using calculus we can show that the values of the unknown parameters are as follows:

$$\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\alpha = \bar{Y} - \beta * \bar{X}$$

where $\bar{X}$ is the mean of X values and $\bar{Y}$ is the mean of Y values.

If you are familiar with statistics, you may recognize β as simply Cov(X, Y) / Var(X).

Implementation Steps:

1. Generate data with nonzero mean and standard deviation (X). Also, create random data by multiplying a constant and adding residual which is an actual data (Y).
2. Calculate the mean of X and Y

   Mean_X = X1+X2+…X_N/ N
   Mean_Y = Y1+Y2+…Y_N/ N
3. Calculate *α* and *β using*

$$\beta = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

$$\alpha = \bar{Y} - \beta * \bar{X}$$

4. Compute the predicted output y

$$Y_e = \alpha + \beta X$$

5. Plot regression against actual data

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Experiment -3

Aim: To implement Logistic Regression.

Software required: (Students should write **Software required** based on software used for implementing program)

Theory:

In statistics logistic regression is used to model the probability of a certain class or event. Logistic regression is similar to linear regression because both of these involve estimating the values of parameters used in the prediction equation based on the given training data. Linear regression predicts the value of some continuous, dependent variable. Whereas logistic regression predicts the probability of an event or class that is dependent on other factors. Thus, the output of logistic regression always lies between 0 and 1. Because of this property it is commonly used for classification purpose.

Logistic Model

Consider a model with features *x1, x2, x3 … xn*. Let the binary output be denoted by *Y*, that can take the values 0 or 1. Let *p* be the probability of *Y = 1*, we can denote it as *p = P(Y=1)*. The mathematical relationship between these variables can be denoted as:

$$ln(\frac{p}{1-p}) = b_0 + b_1x_1 + b_2x_2 + b_3x_3 \ldots b_nx_n$$

Here the term *p/(1−p)* is known as the *odds* and denotes the likelihood of the event taking place. Thus *ln(p/(1−p))* is known as the *log odds* and is simply used to map the probability that lies between 0 and 1 to a range between (−∞,+∞). The terms *b0, b1, b2…* are parameters (or weights) that we will estimate during training.

So we simplify the equation to obtain the value of p:

1. The log term ln on the LHS can be removed by raising the RHS as a power of e:

$$\frac{p}{1-p} = e^{b_0+b_1x_1+b_2x_2+b_3x_3\ldots b_nx_n}$$

2. Now we can easily simplify to obtain the value of p :

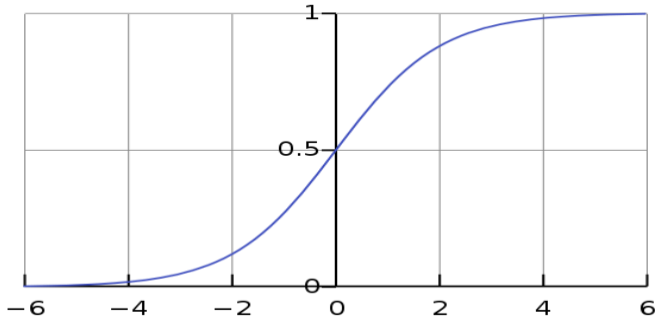
$$p = \frac{e^{b_0+b_1x_1+b_2x_2+b_3x_3\ldots b_nx_n}}{1+e^{b_0+b_1x_1+b_2x_2+b_3x_3\ldots b_nx_n}}$$

$$or$$

$$p = \frac{1}{1+e^{-(b_0+b_1x_1+b_2x_2+b_3x_3\ldots b_nx_n)}}$$

This actually turns out to be the equation of the Sigmoid Function which is widely used in other machine learning applications. The Sigmoid Function is given by:

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



Now we will be using the above derived equation to make our predictions. Before that we will train our model to obtain the values of our parameters b0, b1, b2… that result in least error. This is where the error or loss function comes in.

Loss Function

The loss is basically the error in our predicted value. In other words it is a difference between our predicted value and the actual value. We will be using the L2 Loss Function to calculate the error. Theoretically you can use any function to calculate the error. This function can be broken down as:

1. Let the actual value be y_i. Let the value predicted using our model be denoted as $\bar{y}_i$. Find the difference between the actual and predicted value.
2. Square this difference.
3. Find the sum across all the values in training data.

$$L = \sum_{i=1}^{n} (y_i - \bar{y_i})^2$$

Now that we have the error, we need to update the values of our parameters to minimize this error. This is where the "learning" actually happens, since our model is updating itself based on it's previous output to obtain a more accurate output in the next step. Hence with each iteration our model becomes more and more accurate. We will be using the ***Gradient Descent Algorithm*** to estimate our parameters. Another commonly used algorithm is the Maximum Likelihood Estimation.

The Gradient Descent Algorithm

You might know that the partial derivative of a function at it's minimum value is equal to 0. So gradient descent basically uses this concept to estimate the parameters or weights of our model by minimizing the loss function. For simplicity, assume that our output depends only on a single feature *x*. So we can rewrite our equation as:

$$\bar{y_i} = p = \frac{1}{1 + e^{-(b_0 + b_1 x_i)}} = \frac{1}{1 + e^{-b_0 - b_1 x_i}} \quad (1)$$

Thus we need to estimate the values of weights b0 and b1 using our given training data.

1. Initially let b0=0 and b1=0. Let L be the learning rate. The learning rate controls by how much the values of b0 and b1 are updated at each step in the learning process. Here let L=0.001.

2. Calculate the partial derivative with respect to b0 and b1. The value of the partial derivative will tell us how far the loss function is from it's minimum value. It is a measure of how much our weights need to be updated to attain minimum or ideally 0 error. In case you have more than one feature, you need to calculate the partial derivative for each weight b0, b1 … bn where n is the number of features.

$$D_{b_0} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times \bar{y_i} \times (1 - \bar{y_i})$$
$$D_{b_1} = -2\sum_{i=1}^{n}(y_i - \bar{y_i}) \times \bar{y_i} \times (1 - \bar{y_i}) \times x_i \quad (2)$$

3. Next we update the values of b0 and b1:

$$b_0 = b_0 - L \times D_{b_0}$$
$$b_1 = b_1 - L \times D_{b_1} \quad (3)$$

4. We repeat this process until our loss function is a very small value or ideally reaches 0 (meaning no errors and 100% accuracy). The number of times we repeat this learning process is known as iterations or epochs.

Steps:

1. Read and visualize the database
2. Divide the data to training set and test set (80:20)
3. Perform data normalization
4. Compute the method to make predictions using equation (1)
5. Evaluate the method to train the model using equations (2) and (3). Obtain b0 and b1
6. Train the model using training data
7. Make the predictions
8. Calculate accuracy

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Experiment -4

Aim: To implement Ensemble learning (bagging/boosting)

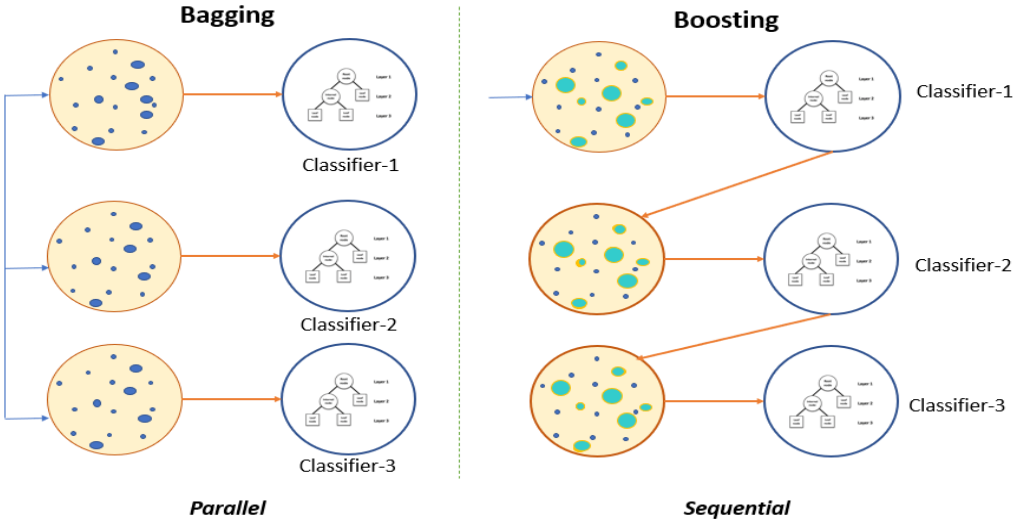
Software required: (Students should write **Software required** based on software used for implementing program)

Theory: Combine Model Predictions into Ensemble Predictions

The three most popular methods for combining the predictions from different models are:

- **Bagging.** Building multiple models (typically of the same type) from different subsamples of the training dataset.
- **Boosting.** Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.

In Bagging, multiple homogenous algorithms are trained independently and combined afterward to determine the model's average. Boosting is an ensemble technique, where we train multiple homogenous algorithms sequentially. These individual algorithms create a final model with the best results. The performance of one algorithm is influenced by the performance of the previously built algorithm.



Benefits of using Bagging algorithms

- Bagging algorithms improve the model's accuracy score.
- Bagging algorithms can handle overfitting.
- Bagging algorithms reduce bias and variance errors.
- Bagging can easily be implemented and produce more robust models.

The Bagging technique is also known as **Bootstrap Aggregation** and can be used to solve both classification and regression problems. In addition, Bagging algorithms improve a model's accuracy score. These algorithms prevent model overfitting and reduce variance.

Steps:

1. Read the database and perform pre-processing

2. Perform feature scaling (transforming a dataset to fit within a specific range)
3. Split the database into training and testing set (80:20)
4. Evaluate using single weak learner (Decision tree classifier) with 5-fold cross validation
5. Develop a boosting ensemble classifier using the weak learner (weak learner = Decision tree, number of weak learners used=100, training dataset for resampling =80, apply bootstrapping)
6. Train and evaluate this bagging model
7. Compare the performance of DT and bagging classifier.

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Experiment -5

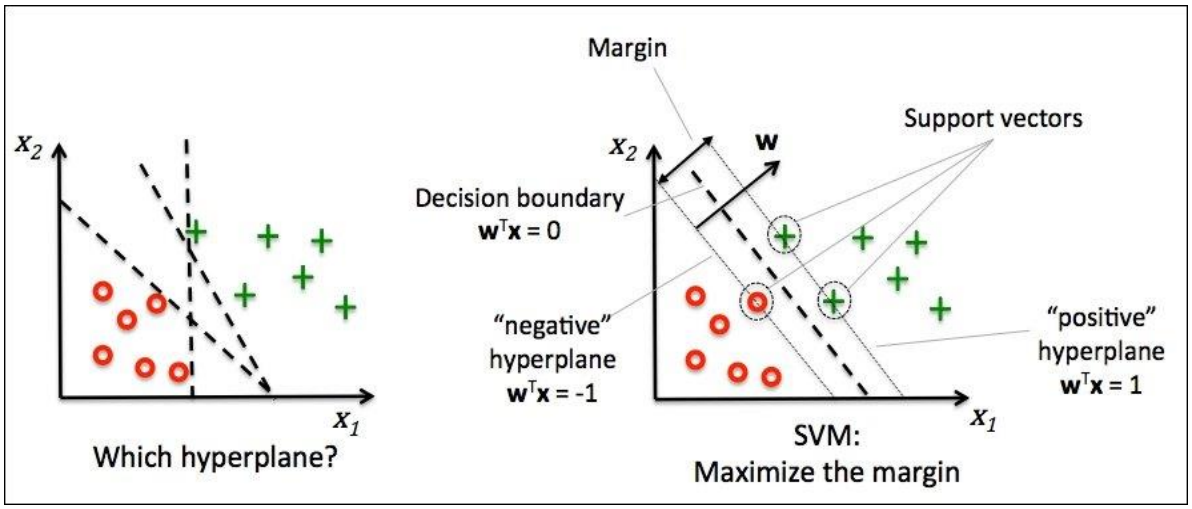
Aim: To implement SVM

Software required: (Students should write **Software required** based on software used for implementing program)

Theory:

Support Vector Machine aka **Support Vector Network** is a supervised machine learning algorithm used for classification and regression problems.
Hyperplane: A hyperplane or decision boundary/surface is an n-dimensional Euclidean space that distinctly separates the data points. The data points on either side of the hyperplane belong to different classes. If we have a p-dimensional feature set, the dimension of the hyperplane will be p-1.
Support Vectors: The individual data points that are close to the hyperplane are called the support vectors.
Margin: The width that the boundary could be increased before hitting a data point. In simple terms, the distance between the hyperplane and the support vectors is referred to as the Margin.
Linear separability: A dataset is linearly separable if there is at least one line that clearly distinguishes the classes.
Non-linear separability: A dataset is said to be non-linearly separable if there isn't a single line that clearly distinguishes the classes.

How does SVM work?



In SVM, the data of finite-dimensional space is mapped to much a higher dimension (p-dimension) and aims at finding the p-1 dimension hyperplane called a linear classifier. If the data is linearly separable, unlike logistic regression, in addition to finding the p-1 dimension hyperplane, SVM creates two parrel hyperplanes on either side that passes through the nearest data points(Support Vectors). The region bounded by these two hyperplanes is called the **margin.**
If the data is non-linearly separable, we need to apply transformations that map the original data to a much higher dimensional space. After transformation, the data would be linearly separable and we can easily find a hyperplane/decision boundary to classify.
Popular SVM Kernel functions:
1. Linear Kernel: It is just the dot product of all the features. It doesn't transform the data.
2. Polynomial Kernel: It is a simple non-linear transformation of data with a polynomial degree added.
3. Gaussian Kernel: It is the most used SVM Kernel for usually used for non-linear data.

4. Sigmoid Kernel: It is similar to the Neural Network with sigmoid activation function.

- Common kernel functions for SVM
  - linear $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$
  - polynomial $k(\mathbf{x}_1, \mathbf{x}_2) = (\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)^d$
  - Gaussian or radial basis $k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$
  - sigmoid $k(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\gamma\, \mathbf{x}_1 \cdot \mathbf{x}_2 + c)$

In simple terms, a kernel is nothing but a transformation that we apply to the existing features so that we can draw a classifier easily for non-linearly separable datapoints.

Steps:

8. Read the database and perform pre-processing
9. Perform feature scaling (transforming a dataset to fit within a specific range)
10. Split the database into training and testing set (80:20)
11. Instantiate Linear SVC object and train the linear SVC classifier using the training data (Choose C=1)
12. Test the model and compute test accuracy
13. Evaluate test accuracy using C = 1, 5, 10, 4, 70, 100
14. Perform similar experiment using RBF kernel (Choose different C and gamma) and compute accuracy
15. Also, compute confusion matrix and ROC curve

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Aim: To implement DB Scan

Software required: (Students should write **Software required** based on software used for implementing program)

Theory:

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

Clustering is an essential technique in machine learning and is used widely across domains and industries (think about Uber's route optimization, Amazon's recommendation system, Netflix's customer segmentation, and so on). Clustering is an unsupervised learning technique where we try to group the data points based on specific characteristics. There are various clustering algorithms with K-Means and Hierarchical being the most used ones. Some of the use cases of clustering algorithms include:

- Document Clustering
- Recommendation Engine
- Image Segmentation
- Market Segmentation
- Search Result Grouping
- and Anomaly Detection.

K-Means and Hierarchical Clustering both fail in creating clusters of arbitrary shapes. They are not able to form clusters based on varying densities. That's why we need DBSCAN clustering. It groups 'densely grouped' data points into a single cluster. It can identify clusters in large spatial datasets by looking at the local density of the data points. **The most exciting feature of DBSCAN clustering is that it is robust to outliers**. It also does not require the number of clusters to be told beforehand, unlike K-Means, where we have to specify the number of centroids.

DBSCAN requires only two parameters: *epsilon* and *minPoints*. *Epsilon* is the radius of the circle to be created around each data point to check the density and *minPoints* is the minimum number of data points required inside that circle for that data point to be classified as a **Core** point. In higher dimensions the circle becomes hypersphere, *epsilon* becomes the radius of that hypersphere, and *minPoints* is the minimum number of data points required inside that hypersphere. DBSCAN creates a circle of epsilon radius around every data point and classifies them into Core point, Border point, and Noise. A data point is a Core point if the circle around it contains at least 'minPoints' number of points. If the number of points is less than minPoints, then it is classified as Border Point, and if there are no other data points around any data point within epsilon radius, then it treated as Noise.

Reachability and Connectivity

These are the two concepts that you need to understand before moving further. Reachability states if a data point can be accessed from another data point directly or indirectly, whereas Connectivity states whether two data points belong to the same cluster or not. In terms of reachability and connectivity, two points in DBSCAN can be referred to as:

- Directly Density-Reachable

- Density-Reachable
- Density-Connected

Parameter Selection in DBSCAN Clustering

DBSCAN is very sensitive to the values of *epsilon* and *minPoints*. Therefore, it is very important to understand how to select the values of *epsilon* and *minPoints*. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm. The value of *minPoints* should be at least one greater than the number of dimensions of the dataset, i.e.,

minPoints>=Dimensions+1.

It does not make sense to take *minPoints* as 1 because it will result in each point being a separate cluster. Therefore, it must be at least 3. Generally, it is twice the dimensions. But domain knowledge also decides its value. The value of *epsilon* can be decided from the K-distance graph. The point of maximum curvature (elbow) in this graph tells us about the value of *epsilon*. If the value of *epsilon* chosen is too small then a higher number of clusters will be created, and more data points w ill be taken as noise. Whereas, if chosen too big then various small clusters will merge into a big cluster, and we will lose details.

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Aim: To implement PCA/SVD

Software required: (Students should write **Software required** based on software used for implementing program)

Theory:

Output Analysis:

(Students should write output analysis based on the output. Specify each output explicitly with output analysis)

Additional Learning:

(Students should write additional learning on their own based on what additionally they learnt after performing the experiment)

Conclusion:

(Students should write conclusion on their own)

Mini project/Case study on any machine learning application.

Experiment -8

Aim:

Software required:

Theory:

Conclusion:

(Students should write conclusion on their own)