raisoni EDUCATION

* Title:
Prediction of employee salary based on years of experience using linear regression.

* Aim:
A compan want to analyze the relationship b/w on employee's years of experience and their salary. develope a simple linear regression model that predicts its salary of an employee based on the no. of years they have worked. additionally evaluate the performance of the model using standard regression metrics.

* Objective:

1. To explore the relationship below an employee years of experience and their salary.
2. To develop a simple Linear regression model to predict salarly based on years of experience.
3. To evaluate the performance of regression model using standard regression metrics.

Outcomes-
1. A predictive model that can estimate an employee's salary based on their year of experience.
Understanding of how well the model performs through the use of regression models

3. Insights into the strength of relationship b/w years of experice of salary.

Tools Required:
- Python programming lang.
- Libraries, pandas, numpy.
- Jupyter net or any python IDE

Theory:-
Simple Linear regression is a statistical method that allows us to summarize and study relationships between two continuous (quantitative) variables:
One variable, denoted x, is regarded as the predictor, explanatory, or independent variable.
The other variable, denoted y, is regarded as the response, outcome, or dependent variable.

$Y = \beta_0 + \beta_1 x$
Y - dependent variable (salary)
x - independent variable (yrs of exp.)
β_0 - intercept.

Code:

```
import pandas as pd
import mamly as pp
import matplotlib.pylot as plt
From sklearn.model_selection import train_
From sklearn.Linear_model import Linear
From sklearn.metrics import mean_absolute
_error, mean_squared_errer,
  r2_score.
import seaborn as sns.

data =
```

"Years of experience": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
"Salary": [40000, 45000, 50000, 55000, 60000,
65000, 70000, 75000, 80000, 8500...
}

Visualize the data
sns.scatterplot(x='Years_of_experience', y='salary', data=df)
plt.title("Years of experience VS salary")
plt.xlabel('Years of experience')
plt.ylabel('salary')
plt.show()

x = df["Years of experience"].values.reshape(-1,1)
y = df['salary'] # dependent variable.

Splitting the dataset into training and testing sets
x_train, y_train, x_test, y_test = train_test_split(x,y,test_size=0.2, random_state=42)

# initialize and train the linear regression model
model = LinearRegression()
model.fit(x_train, y_train)

# Predict on test set
y_pred = model.predict(x_test)

# Evaluate the model performance.
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

raisoni
EDUCATION

```
m2 = m2.score(Y_test, y_pred)
print(f"Mean Absolute Error: (MSE)")
print(f"Mean square error: (MSE):")
print(f"Root mean squared error: olonr(
print(f"R squared : (r2)")
```

Output:
Mean Absolute Error = 355.0
Mean squared Error = 122500.0
Root mean squared error = 3500.0
R-squared = 0.97

conclusion:

Model performance: The model shows a good fit with an required value of 0.98, which means that 95% of the variable is salary can be explained by the yearsor experience.

Insights: There is a strong positive regression b/w years of experience and salary and the simple Linear regression model effectively predicts the salary based on the, relationship.

conclusion:
Model performance: Model shows a good fit with R-squared value 0.95 which means 95% of the variation in salary can be explained by years of experiment.

raisoni EDUCATION

Experiment No-01.

Title: Linear Regression Analysis: Curve Fitting, Generalization, and Model Evaluation.

Aim: Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set. i-) Perform linear regression analysis with Least Squares Method. ii-) Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error. iii-) Verify the Effect of Data set Size and Bias - Varience Tradeoff. iv-) Apply Cross Validation and plot the graphs for errors. v-) Apply Subset selection Method and plot the graphs for errors. vi-) Describe your feeling in each case.

Objectives:

1. To generate a 2D dataset and split it into training and test sets.
2. To perform linear regression using the least squares method.
3. To evaluate training and test errors and analyze curve fitting and generalization error.
4. To investigate the effect of dataset size on bias-variance tradeoff.
5. To apply cross validation and subset selection techniques and analyze errors.

Problem statement:
Given a dataset of N points in a 2D space, the objective is to file a linear regression model using the least squares method and evaluate its performance. The analysis includes training and test error comparisons, bias-variance tradeoff examination, cross-validation implemen-tation, and subset selection impact on generalization error.

Outcomes:
1. A well-fitted linear regression model like with evaluated MSE for training and test data.
2. Understanding of curve fitting and generalization error through error plots.
3. Insights into bias-variance tradeoff as dataset size varies.
4. Visualization of cross-validation error and their implications.
5. Effect of subset selection on model performance and generalization.

Tools Required : 4GB RAM, Anaconda Notebook.

Theory:
Linear regression is a fundamental statistical method used for modeling the relationship between a dependent variable (response) and one or more.

from the training data and fails to generalize well.
- Optimal model: Achieves a balance between bias and variance, leading to low test error.

Cross-Validation:
Cross-validation helps assess model performance by dividing the dataset into multiple sebsets. The model is trained on different training subsets and validation on different test subsets ensuring that performance is consistent across different data splits.

Subset Selection:
Subset selection involves choosing a subset OF the available features or training samples to improve model simplicity and interpretability. By selecting the most relevant data points or features the model can avoid overfitting and improve generalization

Algorithm:
Step1: Data Generation: Create a 2D dataset with N points and add noise.
Step2: Data Splitting: Divide the dataset into training (80%) and test (20%) sets.
Step3: Linear Regression Model: Train the model using the least squares method
Step4: Compute MSE: Calculate training and test mean squared errors.
Step5: Plot Errors: Visualize training vs

test MSE to analyze generalization error.
Step6: Bias-Variance Analysis: Evaluate the effect of training set size on model performance.
Step7: Cross-Validation: Perform k-Fold cross-validation and plot errors.
Step 8: Subset selection: Train models on different subset sizes and analyze errors.
Step 9: Findings: Summerize the results and their implications.

Source code:

```
import numpy as np.
import matplotlib.pyplot as plt
From sklearn.model_selection import
train_test_split, KFold
From.sklearn.metrics import mean_
squared_error
def generate_data (N=100, noise = 0.5)
 np.random.seed (42)
 x = np.linspace (0, 10, N).reshape(-1,1)
 y = 2.5 * x.squeeze() + np.random.normal
 (0, noise, N)
 return x, y
def split_data (x, y, test_size=0.2);
 return train_test_split (x, y, test_size=
test_size, random_state = 42)
def linear_regression (x_train, y_train,
x_test, y_test);
model = LinearRegression()
```

```
model.fit(x_train, y_train)
y_train_pred = model.predict(x_train)
y_test_pred = model.predict(x_test)
print(f"Training MSE: {train_mse:.4f}
Test MSE: {test_mse:.4f}")
plt.bar(['Training MSE', 'Test MSE'],
[train_mse, test_mse], color=['blue',
'orange'])
plt.title('Traing and test MSE')
plt.show()
def bias_varience_tradeoff(x, y):
train_sizes = [10, 20, 40, 60, 80, 100]
train_errors = []
test_errors = []
For size in train_sizes:
x_train, x_test, y_train, y_test =
split_data(x[:size], y[:size])
y_train_pred, y_test_pred, _ = linear
_regression(x_train, y_train, x_test,
y_test)
train_errors.append(mean_squared
_error(y_train, y_train_pred))
test_errors.append(mean_squared_
error(y_test, y_test_pred))
plt.plot(train_sizes, train_errors,
label='Training MSE', marker='o')
plt.plot(train_sizes, test_errors,
label='Test MSE' marker='o')
plt.xlabel('Dataset size')
plt.ylabel('MSE')
plt.title('Effect of Dataset size on
MSE')
```

```
plt.legend()
plt.show()
def cross_validation(x, y, folds=5):
kf = KFold(n_splits=folds, shuffle=True,
random_state=42)
train_errors = []
test_errors = []
For train_index, test_index in kf.split
(x):
x_train, x_test = x[train_index], x
[test_index]
y_train, y_test = y[train_index],
y[test_index]
plt.bar(['Train Error', Test Error']
[np.mean(train_errors), np.mean
(test_errors)].
color = ['blue', 'orange'])
plt.title(f'{folds}-Fold cross Valida-
-tion Errors')
plt.show()
def subset_selection(x, y):
subsets = [10, 20, 50, 100]
errors = []
For subset in subsets:
X_train, X_test, y_train, y_test = split
_data(x[:subset], y[:subset])
_, y_test_pred, _ = linear_regression
(x_train, y_train, x_test, y_test)
errors.append(mean_squared_error(y_
test, y_test_pred))
plt.plot(subsets, errors, marker='o',
color='purple').
```

```
plt.xlabel ('Subset size')
plt.ylabel ('Test MSE')
plt.title ('Effect of Subset size on
Test MSE')
plt.show()
def main():
  X, y = generate_data()
X_train, X_test, y_train, y_test = split
_data(X, y)
y_train_pred, y_test_pred, _ = linear
_regression(X_train, X_test, y_train)
plot_mse(X_train, y_train, X_test,
y_test, y_train_pred, y_test_pred)
print("Effect of Dataset Size on Bias
-variance Tradeoff:")
bias_variance_tradeoff(X, y)
print("Cross Validation:")
cross_validation(X, y)
print("Subset Selection Method:")
subset_selection(X, y)
if __name__ == "__main__":
  main()
```

Conclusion:

Linear Regression is a key tool for modeling relationship between variable. Through curve fitting train to find the best line that separates the data. To ensure good performance on new data. It's imp. to balance fitting and generalization.

raisoni EDUCATION

Experiment No. 2.

Title: Study and Implement Multiple Linear Regression.

Objectives:
1. Develop a machine learning model to predict calories burned on exercise-related metrics.
2. Perform data preprocessing (handling missing values, encoding categorical variables)
3. Train and evaluate a Linear Regression model for calorie prediction.
4. Measure model performance using Mean Squared Error (MSE) and R^2 score.
5. Visualize actual vs predicted values to access model accuracy.

Problem Statement:
This practical builds a data-driven approach using linear regression to predict calorie expenditure based on exercise parameters such as duration, pulserate, and max pulse.

Outcomes:
1. A trained Linear Regression model that predicts calorie expenditure.
2. Evaluation metrics (MSE, R^2) to assess model accuracy.
3. A scatter plot of actual vs predicted values to visualize prediction quality.

Theory:
Multiple Linear Regression (MLR) is an extension of simple linear regression, where a dependent

variable (Y) is predicted based on multiple independent (X1, X2, ..., Xn). The equation of MLR is

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots \beta_0 X_0 + \epsilon$$

Where,

- Y = Dependent variable.
- $X_1 X_2 \dots X_0$ = Independent variable.
- β_0 = Intercept.
- $\beta_1, \beta_2 \dots \beta_0$ = coefficients of independent variables.
- ϵ = Error term.

The model learns these coefficients by minimizing the Mean Squared Error (MSE):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y - \hat{Y})^2$$

where Y is the actual value of $\hat{Y}$ is the predicted value.
The goodness-of-fit is measured by R^2 score.

$$R^2 = 1 \frac{SS_{res}}{SS_{tot}}$$

where SSres is the residual sum of squares and SStot is the total sum of squares.

Algorithm:

Step 1: Load Dataset

- Read data.csv using pandas
- Display the first few rows to understand the structure.

Step2: Data preprocessing.
- Handle missing values (drop or impute).
- Select Independent (x) and depedent (y) variables.
- Convert categorical Features using pd.get_dummies().

Step3: Split Data
- Use train_test_split() to split into training (80%) and testing (20%) sets.

Step 4: Train Linear Regression Model
- Initialize LinearRegression() from sklearn.
- Fit the model using X_train and y_train

Step5: Make Predictions
- Use model.predict(X_test) to generate predictions.

Step6: Evaluate Model.
- Compute MSE and R^2 score to measure accuracy.

Source Code:

```
import numpy as np
import pandas as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv("data.csv") # Replace with actual dataset
```

raisoni EDUCATION

```
print (df.dropna())
X = df['Duration', 'Pulse', 'Maxpulse']] #
 Replace with actual Feature names
y = df['Calories']
X = pd.get_dummies (X, drop_first = True)
X_train, X_test, y_train, y_test = train
_test_split (X, y, test_size = 0.2, random_
state = 42)
model = LinearRegression ()
model.fit (X_train, y_train)
y_pred = model.predict (X_test)
mse = mean_squared_error (y_test, y_pred)
r2 = r2_score (y_test, y_pred)
print (f"Mean Squared Error : {mse}")
print (f" R-squared Value: {r2}")
plt.xlabel ("Actual values")
plt.ylabel ("Predicted Values")
plt.title ("Actual vs predicted values")
plt.show ()
```

Output:

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0

Mean Squared Error : 4354.413282161514
R-squared value : 0.8664689420685995.

Conclusion:

Multiple linear regression extends simple linear regression by modeling the relationship between one dependant variable and multiple independent variable. It allows more accurate & realistic predictions when multiple factors influence the outcome.

raisoni
EDUCATION

Title: Implement Logistic Regression using any dataset.

Objectives:
1. To generate a synthetic dataset with two numerical features and a binary taryet variable.
2. To implement and evaluate a logistic regression model for classification.
3. To analyze model performance using accuracy confusion matrix and ROC-AUC score.
4. To visualize classification performance using the ROC curve.

Problem statement:
The objective of this project is to build a logis-tic regression model that can predict a binary outcome based on two independent numerical variables. The model's effictiveness will be accessed using accuracy, confusion matrix, and ROC-AUC score, with additional visualization techniques to interpret results.

Outcomes:
1) A synthetic dataset with meaningful feature-target relationships.
2) A trained logistic regression model capable of predicting binary outcomes.
3) Performance evaluation through accuracy, confusion matrix, and ROC-AUC score.
4) Graphical representation of model perfor--mance using ROC curve.

Tool Required: 4GB RAM, Anaconda, Notebook

Theory:
Logistic regression is a statistical method used for binary classification. It models the probability that a given input belongs to a perticular class using the sigmoid function, defined as:

$$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2)}}$$

where:
- $P(Y=1|X)$ $P(Y=1|X)$ $P(Y=1|X)$ is the probability of the target varible being 1.
- $\beta 0$ \beta_0 $\beta 0$ is the intercept, and $\beta_1, \beta 2$ \ beta_1, \beta_2 $\beta 1, \beta 2$ are coefficients for features $X1, X2 X_1, X_2 x_1, , x2$.
- The model optimizes these parameters using maximum likelihood estimation (MLE).
- The performance of logistic regression is often assessed using:
  - Accuracy: Ratio of correctly predicted samples.
  - Confusion Matrix: Breakdown of TP, FP, TN and FN.
  - ROC curve & AUC score: Measures the model's ability to discriminate between classes.

raisoni EDUCATION

Algorithm:

Step1: Generate Synthetic Dataset

1. Generate two independent numerical features x1, and x2.
2. Define a linear relationship with added noise to determine binary target y.
3. Store data in a Pandas DataFrame.

Step 2: Data Preprocessing.

1. Split data into training and testing sets.

Step 3: Train Logestic Regression Model

1. Initialize the logistic regression model
2. Train the model using the training data

Step 4: Make Predictions:

1. Predict target values on training and testing sets.
2. Generate probability scores for ROC-AUC evaluation.

Step 5: Evaluate Model performance.

1. Compute accuracy, confusion matrix, and classification report.
2. Calculate the ROC-AUC score.
3. Plot the ROC curve.

Score code:

```
import numpy as np
import pandas as np
import matplotlib.pyplot as plt
From sklearn.model-selection import train_test.
From sklearn.linear_model import Logestic Regressia
From sklearn.metrics import accuracy_score,
confusion_matrix, classification_report,
roc_curve, roc_auc_score.
```

```
np.random.seed(42)
X1 = np.random.rand(100)*10
X2 = np.random.rand(100)*5
y = (2*X1 + 3*X2 + np.random.randn(100)
   *2 > 20).astype(int)
data = pd.DataFrame({
  'X1': X1
  'X2': X2,
  'Y': y
 })
X = data[['X1','X2']]
y = data['Y'] # Target.
X_train, X_test, y_train, y_test = train_
test_split(X, y, test_size=0.5, random_
state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
y_test_proba = model.predict_proba
(X_test)[:, 1].
train_accuracy = accuracy_score(y_train,
y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred
conf_matrix = confusion_matrix(y_test, y_test_
pred)
roc_auc = roc_auc_score(y_test, y_test_proba)
print(f"Train Accuracy : {train_accuracy:.2f}")
print(f" Test Accuracy : {test_accuracy:.2f}")
print(f" ROC-AUC score: {roc_auc:.2f}")
print("\nConfusion Matrix :")
print(conf_matrix).
```

```
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_
auc:.2f})', color='blue')
plt.plot([0,1], [0,1], 'k--', label='Random
Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating characteristic
(ROC) Curve')
plt.legend()
plt.grid()
plt.show()
```

Output:

```
Train Accuracy: 0.93
Test Accuracy : 0.90
ROC-AUC Score: 0.98
Confusion Matrix
[[15 0]
 [3 12]].
```

Conclusion:

Logistic regression is a powerful classification algorithm used to predict categorical outcomes. By implementing it and real world dataset. we learn how to model the probability of class membership. evaluate performance using accurate prevision, recall and confusion matrix & understand decision boundaries.

raisoni EDUCATION

Title: Study and Implement Decision Tree.

Objectives:
1. Implement a Decision Tree classifier to classify the Iris dataset into tree species.
2. Evaluate model performance using accuracy, classification report, and confusion matrix.
3. Visualize the decision tree structure and Feature importance.

Problem statement:
The problem requires:
- Developing a supervised machine learning model (Decision Tree).
- Evaluating the performance and interpret-ability of the model
- Visualization the decision-making process.

Outcomes:
1. A trained Decision Tree classifier with optimal parameters.
2. A performance evaluation of the model using accuracy, confusion matrix, and classification report.
3. A graphical representation of the decision tree to illustrate the decision-making process.

Tools Required: 4GB RAM, Anaconda, Notebook.

Theory:
A Decision Tree is a supervised learning algorithm used for classification and regression tasks. It works by recursively splitting data based on feature values to maximize information gain (entropy reduction or Gini impurity). The key concepts include:
- Gini impurity: Measures how often a randomly chosen element would be incorrectly classified.
- Entropy: Measures the disorder or impurity of the dataset.
- Information Gain: The reduction in entropy after splitting on a particular feature.
- Pruning: Reducing overfitting by limiting the depth or complexity of the tree.

Algorithm:
Step 1: Load the Dataset.
- Import the load_iris() dataset from sklearn.datasets.
- Extract the load_iris() d variables (X) & target labels (y).
Step 2: Data Preprocessing.
- Split the dataset into training and testing sets using train_test_split()
- Normalize or scale features if necessary (not required for Decision Trees).
Step 3: Train Decision Tree Model
- Initialize the decision Tree classifier with criterion='gini' and max_depth=3.
- Train the model using fit(X_train, y_train)

raisoni EDUCATION

Step 4: Make Predictions
· Use the trained model to predict labels for X_test.
Steps: Evaluate Model performance
· Compute accuracy using accuracy_score (y_test, y_pred()).
· Generate a classification report with precision, recall, and F1-score.
· Plot a confusion matrix to analyze misclassifications
Step 6: Visualize the Model
· Use tree.plot_tree() to graphically represent the decision tree.
· Plot feature importance to identify which features contribute most.

Source code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.datasets import load_iris
data = load_iris()
x = data.data
y = data.target
x_train, x_test, y_train = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
dt_model = DecisionTreeClassifier(criterion='gini',
 max_depth=3, random_state=42)
dt_model.fit(x_train, y_train)
y_pred = dt_model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classifica-
-tion_report(y_test, y_pred))
tree.plot_tree(dt_model, feature_names=
data.feature_names,
class_names = data.target_names, Filled=
True)
plt.show()
```

Output:

Accuracy: 1.0

Classification Report:

	precision	recall	F1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Conclusion:

Decision Trees are intuitive and powerful models used for both classification and regression tasks. By studying and implementing them your understand how data is split based on features to make decisions.

raisoni EDUCATION

Title : Implement Naive Bayes Classifier on data set of your choice . Test and compare for accuracy and Precision.

Objectives:
1. To implement a Naive Bayes classifier for classifying the Iris dataset.
2. To evaluate the model's performance using accuracy , precision , and classification reports.
3. To understand how probablistic classification works with Gaussian Naive Bayes.

Problem statement:
Classifying flowers into different species based on given attributes.

Outcomes:
1. A trained Gausian Naive Bayes model that can accurately classify iris species.
2. Performance evaluation metrices including accuracy, precision , and a classification report.
3. A deeper understanding of Bayesian probability and how it applies to classification problems.

Tools Required: 4GB RAM , Anaconda , Notebook

Theory:
The Naive Bayes classifier is based on Bayes' Theorem , which states:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

where:

- $P(A|B)$ is the probability of class A given feature B.
- $P(B|A)$ is the probability of Feature BBB given class A.
- $P(A)$ is the prior probability of class A.
- $P(B)$ is the probability of feature B.

In Gaussian Navie Bayes, features are assumed to follow a normal (Gaussian) distribution:

$$P(x|A) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where:

- μ is the mean
- σ^2 is the variance of the feature values for a given class.

Algorithm:

1. Load the dataset : Import the Iris dataset
2. Preprocess the data: Extract features (x) and labels (y).
3. Split the dataset: Use train_test_split() to divide data into training (80%) and testing (20%) sets.
4. Train the model:
   Use GaussianNB() from sklearn.naive_bayes to train a Naive Bayes model.
5. Make predictions:
   Use predict() on the test set.

Source Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train-
test-split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrices import accuracy_score,
precision_score, classification_report
from sklearn.dataset import load_iris
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train.
test_split(X, y, test_size=0.2, random_state
=42)
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred,
average='macro')
classification_rep = classification_report'
[y_test, y_pred, target_names = iris.
target_names)
printlF" Accuracy: {accuracy:.2F}")
print(F" Precision: {precision:.2F}")
print("Classification Report:\n", classification
_rep)
```

raisoni EDUCATION

Output:

Accuracy: 1.00
Precision: 1.00
Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	1.00	1.00	9
virginica	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Conclusion:-
Naive Bayes is fast and efficient classification algo. effective for high dimension data Testing it and dataset showed good accuracy and precision.

raisoni EDUCATION

Title: Implement K Nearest Neighbour (KNN) classification.

Objectives:
1. To understand and implement the KNN classification algorithm.
2. To classify data points based on similarity to their nearest neighbors.
3. To evaluate the model using accuracy & other performance metrics.

Problem Statement:
Given a dataset with labled instances, the objective is to classify new, unseen instances by identifying the KKK nearest neighbors and assigning the most common label among them The classification should be performed effectively, and the choice of KKK should be optimized to avoid underfitting or overfitting.

Outcomes:
1. A trained KNN model capable of classifying in new data points.
2. Performance evaluation using metrics like accuracy.
3. Visualization of decision boundaries.

Tools Required: 4GB RAM, Anaconda, Notebook.

raisoni EDUCATION

Theory:
KNN is a non-parametric, instance-based learning algorithm. Given a datapoint:
1. Compute the distance between the new point and all existing labeled points in the dataset.
2. Select KKK nearest point.
3. Assign the most common label among these neighbors to the new points.

Common distance metrics:
- Euclidean Distance:
$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$
- Manhattan Distance.

$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$
- Minkowski Distance:
$d(p,q) = (\sum_{i=1}^{n} |p_i - q_i|^p)^{1/p}$

The choice of KKK is crucial:
- Small KKK → sensitive to noise, overfitting risk.
- Large KKK → Smother decision boundary, potential underfitting.

Algorithm:
Steps:
1. Load and preprocess the dataset.
2. Choose the value of KKK
3. Compute the distance between the test instance all training instance.
4. Identify the KKK nearest neighbors.
5. Assign the most frequent class label among the neighbour.

Source code:

```
import numpy as np
import matplotlib.pyplot as plt.
from matplotlib.colors import ListedColormap)
from sklearn.neighbors import KNeighborsClassi
-fer
from sklearn.datasets import load_iris
from sklearn.model_selection import train_
test_split
data = load_iris ()
X, y = data.data.data.target
X_reduced = X[:, :2]
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_reduced, y)
def plot_decision_bounday (x, y, model)):
cmap_light = ListedColormap(['#FFAAAA', '
#AAAAFF', '# AAFFAA'])
cmap_bold = ListedColormap(['#FF0000',
'#0000FF', '#00FF00'])
h = .02
x_min, x_max = X[:,0].min() - 1, X[:,0].maxy
+1
y_min, y_max = X[:, 1], min() - 1, X[:, 1].
max ()+1
xx, yy = np.meshgrid (np.arrange(x_min,
x_max, h),
  np.arrang (y_min, y_max, h))
z = model.predict (np.c_[xx.ravel () .yy.
 ravel()])
 z = Z.reshape(xx.shape)
 plt.figure (figsize = (8, 6))
plt.contourf (xx, yy, z, cmap = cmap_light, alpha = 0.8)
```

```
plt.scatter(X[:,0], X[:, 1], c=y, cmap=
cmap-bold, edgecolor='K', s=20)
plt.title("KNNN Decision Boundary")
plt.show()
plot_decision_boundary(X_reduced, y,
knn).
```

Conclusion:

K nearest neighbour is a simple, instance based classification algorithm that makes prediction based on proximity to training data points. Implementing KNN helps understand the impact of distant metrics and the choice of K on accuracy.

Title: Implement support vector Machine algorithm

Objectives:
1. Implement the support vector Machine(SVM) algorithm for classification.
2. Understand hyperplane separation and kernel trick for non-linear classification.
3. Evaluate SVM using accuracy.

Problem statement:
Develop a support vector machine (SVM) model to classify a given dataset into different categories. The dataset may contain non-linearly separable data, requiring the use of kernel functions. The goal is to Find the best hyperplane that maximizes the margin between different classes while minimizing misclassification.

Outcomes:
1. Successfully implement SVM for classification tasks.
2. Identify optimal hyperplane for given data.
3. Compare different kernel functions
4. Evaluate model performance using confusion matrix and accuracy metrics.

Tools Required: 4GB RAM, Anaconda, Notebook

Theory:
Hyperplane: A decision boundary seperation different classes.
Support vectors: Data points closest to the hyperplane, influencing its position.
Margin: Distance between the hyperplane & support vectors (maximize it)
Kernel Trick: Transforming non-linearly seper-able data higher-dimensional space.
Types of SVM:
- Linear SVM
- Non Linear SVM.

Algorithm:
Steps:
1. Input: Dataset with Features and labels
2. Preprocessing: Normalize/scale the data.
3. Train-Test Split: Divide dataset into training and testing sets.
4. Choose Kernel Function: Linear, Polynom-ial or RBF.
5. Train SVM Model: Use Scikit-learn's SVM classifier.
6. Predict: Classify test data.
7. Evaluate: compute accuracy, precision, recall, F1-score.
8. Optimize (Optional): Tune hyperparameters using GridSearchCV.

raisoni
EDUCATION

Source Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train-
test_split
from sklearn.preprocessing import standard
scalar
from sklearn.svm import SVC.
from sklearn.metrices import classification_
report, confusion_matrix
from sklearn.datasets import make_classificatio
X,y = make_classification(n_samples = 1000,
        n_Features = 2,
        n_informative = 2,
        n_redundant = 0,
        n_repeated = 0,
        n_classes = 2,
        random_state = 42)
plt.scatter(X[:,0], X[:,1], c=y, cmap ='coolwarm)
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.title('dataset Visualization')
plt.show()
X_train, X_test, y_train, y_test = train_test
-split(X, y, test_size=0.2, random_state=42)
scalar = StandardScalar()
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)
svm_model = SVC(kernel ='rbf', C =1.0,
gamma = 'scale')
```

```
svm_model.fit(x_train, y_train)
y_pred = svm_model.predict(x_test)
conf_matrix = confusion_matrix(y_test,
y_pred)
sns.heatmap(conf_matrix, annot=True,
fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
print(classification_report(y_test, y_pred
))
```

Conclusion:

Support vector machine is a powerful supervised learning algorithm that works well for both linear and non-linear classification. By Implementing SVM we observed how it maximizes the margin between classes and handles complex boundaries using kernel functions.

raisoni EDUCATION

Title: K-Means clustering for classification Prediction

Objectives:
1. Implement K-Means clustering to classify new data points.
2. Understand clustering based classification techniques.
3. Assign majority class labels to clusters for classification.
4. Predict the class for a new data point based on its cluster m
5. Develop a Python-based solution for clustering and classification.

Problem statement:
Given the following data, which specify classification for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1 = 0.906 and VAR2 = 0.606. using the result of K-means clustering with 3 means (i.e. 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Outcomes:

1. A manual K-Means clustering model that groups the given dataset into three clusters.
2. Majority class labels assigned to each cluster.
3. A classification prediction for the new data point.

Tools Required: 4GB RAM, Anaconda, Notebook.

Theory:

K-Means clustering is an unsupervised machine learning algorithm that partitions data into K clusters based on similarity.

The algorithm works as follows:

1. Select K initial centroids randomly
2. Assign each data point to the nearest centroid.
3. Update the centroids by computing the mean of all points in each cluster.
4. Repeat steps 2 and 3 until centroids no longer change significantly.
5. Assign a majority class label to each cluster.
6. Classify a new data point based on the cluster it belongs to.

Algorithm:
Steps:
1. Input: Dataset with (VAR1, VAR2) values and class labels.
2. Apply k-Means clustering to partition data into 3 clusters.
3. Determine the majority class for each cluster.
4. Assign labels to the clusters based on majority class.
5. Predict classification for a new data point by:
   1- Finding its closest cluster centroid.
   2- Assigning the corresponding majority class.
6. Output: Predicted classification.

Source Code:

```
import numpy as np
from sklearn.cluster import KMeans
from scipy.stats import mode
data = np.array([
  [1.713, 1.586],
  [0.180, 1.786],
  [0.353, 1.240],
  [0.940, 1.566],
  [1.486, 0.759],
  [1.266, 1.106],
  [1.540, 0.419],
  [0.459, 1.799],
  [0.773, 0.186].
])
```

```
classes = np.array([0,1,1,0,1,0,1,1,1])
kmeans = KMeans(n_clusters=3, random_
state=42,n_init=10)
clusters = kmeans.fit_predict(data)
cluster_labels = np.zeros(8)
for i in range(8):
  cluster_labels[i] = mode(classes[clusters
  == i], keepdims=True).mode[0]
new_point = np.array([[0.908, 0.606]])
new_cluster = kmeans.predict(new_point)
predicted_class = cluster_labels[new_cluster
[0]]
print(f"Predicted classification: {int(
predicted_class)}")
```

Output:
Predicted classification: 1

conclusion:
K means is an unsupervised learning algo. that groups the data into clusters based on similarity. When used for classification prediction. It helps reveal hidden patterns and structure in the data. Through it doesn't use labels directly.

raisoni
EDUCATION

Title: Unsupervised Learning: Implement K-Means Clustering and Hierarchical clustering on proper data set of your choice. Compare their convergence.

Objectives:
1. Implemehd K-means clustering and Hierarchical on a chosen dataset.
2. Compare the convergence behaviour of both clustering algorithms.
3. Evaluate clustering performance using appropriate metrics
4. Visualize clusters and analyze computational efficiency.

Problem statement:
This project aims to implement, compare and analyze both algorithms to understand how they converge and under what conditions one outperforms the other.

Outcomes:
Implementations of K-Means and Hierarchical clustering on a real-world dataset
Visual comparisons of clusters formed by both methods.
Evaluate of convergence speed, SSE, Silhouette score, and other metrics.

Tools Required: 4GB RAM, Anaconda, Notebook.

raisoni
EDUCATION

Theory:

K-Means Clustering

- Algorithm
  1. Choose the number of clusters K
  2. Randomly initialize k centroids
  3. Assign each data point to the nearest centroid.
  4. Recalculate centroids by averaging the assigned points.
  5. Repeate until convergence
- convergence: converges when centroids stabilize. Affected by initial centroid selection.
- Computational complexity: O(nkt) (n= points, k= clusters, t= interations)

Hierarchical Clustering.

- Types:
  1. Agglomerative (Bottom-Up):
     - start with individual points, merge closest pairs iteratively
  2. Divisive (Top-Down):
     - start with all points in one cluster, split recursively.
     - convergence: Defined by a dendrogram; doesn't require pre-defined k.
- Computational complexity: $O(n^2 \log n)$ (Higher than K-Means).

raisoni EDUCATION

Algorithm:
Step 1: Load Dataset
· Choose a dataset and preprocess it.
Step 2: Implement K-Means Clustering
· Use K-Means from sklearn.cluster with different K values
· Track inertia (SSE) to check convergence.
Step 3: Implement Hierarchical Clustering
· Apply Agglomerative clustering using scipy cluster hierarchy.
· Visualize dendrogram to determine the best number of clusters.
Step 4: Compare Convergence.
· Compare SSE, silhouette scores and runtime.
· Plot elbow method, cluster assignment, and dendrograms.

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
```

raisoni
EDUCATION

```
From sklearn.preprocessing import standard
Scalar
iris = load_iris()
x = iris.data
scalar = standard scalar()
x_scaled = scaler.fit_transform(x)
inertia = []
silhouette_scores = []
k_range = range(2, 10)
for k in k-range:
    kmeans = kMeans(n_clusters=k, random_
state=42, n_init=10)
    kmeans.fit(x_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_score.append(silhouette_
score(x_scaled, kmeans.labels_))
plt.figure(figsize=(10,4))
plt.plot(k_range, inertia, market='o',
 linestyle='--', label='Inertia')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (SSE)')
plt.title('Elbow Method for k-Means')
plt.legend()
plt.show()
plt.figure(figsize=(10,4))
plt.plot(k_range, silhouette_scores,
marker='o', linestyle='--', color='red',
label='silhouette score')
plt.xlabel('Number of clusters (k)')
plt.ylabel('silhouette score')
plt.title('silhouette score for k-Means')
plt.legend()
```

```
plt.show()
plt.figure(figsize=(10,6))
linked = linkage(X_scaled, method='ward')
dendrogram(linked, truncate_mode='level'
, p=5)
plt.title('Dendrogram For Hierarchical
clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
hc = AgglomerativeClustering(n_clusters=
linkage 'ward')
hc_labels = hc.fit_predict(X_scaled)
plt.figure(figsize=(10,5))
sns.scatterplot(x=X_scaled[:,0], y=X
_scaled[:,1], hue=hc_labels, palette=
'viridis', s=60)
plt.title("Hierarchical clustering-Cluster
visualization")
plt.show()
```

Conclusion:

K means and Heirarchical clustering are key unsupervised learning techniques used to discover patterns in unlabeled data. K-means is faster and scales well, covering quickly through iterative optimization.

Title: Study and Implement Bagging/ Boosting using Random Forests.

Objectives:
1. Understand the concepts of Bagging and Boosting and their impact on model performance.
2. Implement Bagging and Boosting techniques using Random Forests.
3. Compare the performance of Bagging and Boosting with standard Random Forest classifers.
4. Evaluate accuracy.

Problem statement:
Develop and analyze ensemble learning techniques using Random Forests to improve classification accuracy and reduce overfitting in machine learning models. Compare the effectiveness of these techniques on real-world datasets

Outcomes:
1. Implementation of Bagging and Boosting using Random Forests.
2. Performance comparision between Bagging, Boosting and standard Random Forest classifiers.
3. Evaluation metrices demonstrating improvement in accuracy metrics.

Tools Required: 4GB RAM, Anaconda, Notebook.

1. Visualize.

Theory:

Bagging and Boosting are ensemble learning techniques that improve the stability and accuracy of machine learning algorithms by combining multiple base learners.

- Bagging:
  - Creates multiple subsets of training data via bootstrapping.
  - Trains separate models on each subset independently
  - Aggregates predictions through averaging or majority voting
  - Example: Random Forest.
- Boosting:
  - Sequentially trains weak models, each focusing on errors of the previous ones.
  - ~~Assig~~ Aggregates predictions through averaging or majority voting
  - Example: Random Forest.

Algorithm:

1. Load and preprocess data.
2. Split data into training and testing sets.
3. Implement Random Forest classifier as a baseline.
4. Implement Bagging with Random Forest.
5. Implement Boosting with Random
6. Compare performance metrics
7. Visualize.

raisoni EDUCATION

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
From sklearn.model-selection import train
-test-split
From sklearn.ensemble import Random
ForestClassifier, BaggingClassifier.
    AdaBoostClassifier.
From sklearn.metrices import accuracy-score,
classification_report
From sklearn.dataset import make_classifi-
-cation.
data, labels = make_classification(n_samples.
1000, n_Feature = 20, random_state=42)
X_train, X_test, y_train, y_test =
train_test_split(data, labels, test_size
=0.2, random_state=42)
rf = RandomForestClassifier(n_estimator.
100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rF = rF.predict(X_test)
print(" Random Forest Accuracy:",
accuracy_score(y_test, y_pred_rF))
bagging = BaggingClassifier(estimator-
RandomForestClassifier(), n_estimators=50
random_state = 42)
bagging.fit(x_train, y_train)
y_pred_bagging = bagging.pred(x_test)
print("Bagging Accuracy:", accuracy_
score(y_test, y_pred_boosting))
```

independent variables. The objective is to find the best fitting line that minimizes the error between predicted and actual values.

Least Squared Method.

The least squares method is used to minimize the sum of squared residuals. The linear regression model follows the equation

$y = wx + b$

where:

- y is the predicted output.
- x is the input feature.
- w is the weight and
- b is the bias

Curve Fitting and Generalization Error:

Curve Fitting determines how well a model captures patterns in the data. An underFitted model has high bias, while an overFitted model has high variance. The generalization error measures has high bias, while an overFitted model has high variance. The generalization error measures how well a model performs on unseen data. A model should balance bias and variance to minimize test error.

Bias - Variance TradeOff:

- High Bias (underFitting): The model is too simple and fails to capture patterns in the data.
- High variance: The model learns noise

Output:
Random Forest Accuracy : 0.9
Bagging Accuracy : 0.88
Boosting Accuray : 0.86

Conclusion:
Random Forests use bagging to build multiple decision trees and combine their outputs for more stable and accurate predictions. Studying and implementing this approach demonstrates how ensemble methods reduce variance and improve generalization.

Write a program to classify simple binary data using a quantum circuit as a Feature map is a hybrid quantum classical approach.

Theory:
Basic concepts of Quantum Machine learning

QML integrate quantum computing with classical machine LA. It aims to take adv. of quantum phenomena such as superposition entanglement and interference. In a typical hybrid quantum classical approach, the quantum computer is responsible for tasks like Feature extraction, while the classical computs. handles optimizaty training & classification

Feature mapping via Quantum circuit
Feature mapping refers to the process of transforming classical data into a higher dimensional space to capture intricate relationships that may not be easily discernible in the original space. In the hybrid quantum-classical, quantum circuits are used to map classical data into quantum states.

Why quantum circuits for feature mapping?

classical data is usually represented as vectors of numbers, and classical ML. However, quantum circuits can map this data to quantum states in a way that could

provide new insights on enhanced learning. Quantum circuits can encode info in a quantum superposition of states which allows for the explanation of multiple data simultaneously.

* Steps in Quantum Feature Mapping

1. Initial Data encoding : The classical binary data is mapped to quantum states
2. Quantum gate application : Quantum gates are applied to Qubits these gates alter the state of qubits and create superposition.
3. Measurements : The final quantum state is measured. These outcomes are used as feature for classical classifers.

General workflow :-

1. Data preparation :
   Class binary data is represented as a vector of 0s and 1s for example (1, 0, 1) represents one data sample.

2. Quantum Feature Map (Quantum circuit) :
The classical data is passed through quantum circuit, which encodes the binary data into quantum state. The circuit might involves gates like hadamard gates.

raisoni
EDUCATION

3. Measurement
After applying the quantum operations the quantum circuit is measured collapsing the quantum state to classical bit strings. The result (e.g. counts of measured outcomes like 000, 001)

4. classical classifier:
The classical classifier is trained using the quantum extracted features. This classifier learns just like a traditional machine learning classifer.

Advantages:

1. High dimensional Representations :-
Quantum circuits can make classical data to a high dimensional Hibert space, where more complex pattern might emerge.
2. Quantum superposition: Quantum circuit flow allows for superposition, meaning a qubit can exist in multiple states at once.
3. parallelism: Quantum computing can potentially speed up some operations by exploiting quantum parallelism.

```
import numpy as np.
From qiskit.import Quantum circuit, Aer,
transpile assemble
from qiskit.providers.aer import AerSimulator
from sklearn.svm. import SVC
From sklearn.model.selection input train_test_split
```

```
from sklearn.metrics import accuracy_score.

#function to create a quantum circuit for
binary data encoding def create_quantum_
circuit(x):-
 #Create a quantum circuit with 3 qubits
    qc = QuantumCircuit(3)
     if bit==1:
    qc.x(i) #Apply x gate if the ith is 1

 #Apply hadamard gates to all qubits to
 create superposition
  qc.h([0,1,2])
 #return the quantum circuit
    return qc.

#Run the quantum circuit on the simulator
 result = simulator.run(qobj).result()
# Get the measurement results (counts)
 (wnts = result.get_counts)

 X = np.array([[1,0,1],[0,1,0],[1,1,0],[0,0,1]])
#binary features y = np.array([0,1,0,1,1])
# labels (binary classification)
# Create a quantum simulator
   simulator = Aer.simulator()

#prepare data by encoding each binary
input into a quantum circuit and
extracting features
quantum_features = []
 for sample in X:
```

```
qc = create_quantum_circuit(sample)
quantum_feature = extract_quantum_feature(qu_simulator)
quantum_features = np.array(quantum_features)

x_train, x_test, y_train, y_test = train_test_split(quantum_features)
y_test_size = 0.25, random_state = 42)
#classifier SVM classifier
classifier = SVC(kernel = 'linear')
# train the classifier
classifier.fit(x_train, y_train)
# Make predictions on the test set
y_pred = classifier.predict(x_test)
#calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"classification accuracy: {accuracy}")
```

conclusion:

Classifying simple binary data using a quantum circuit as a feature map in a hybrid quantum classical approach leverage the unique capabilities of quantum computing to enhance classical ML. Although qc is still in its infancy, hybrid model like these represent a promising path forward for computing the strength of both quantum of classical computing.

raisoni
EDUCATION

YEAR - I

Logistic Regression

Logistic Regression is a transformation of the linear regression model that allows us to probability. Model binary variable. It is also known as a 'generalised linear model that uses a legit link classification algorithm. In a classificated problem the target variable y can take only discrete values for a given set of feature Logistic Regression becomes a classification technique only which a decision threshold is bought into the pleture. The setting of a threshold value vary important aspect of logistic regression and is dependent on the classification problem itself.

Logistic Regression Equation.

We know the equation to straight line

$$y = b_0 + b_1 x_1 + b_2 x_2 + \dots\dots b_n th$$

For this divide the above equation by $(1-y)$. $y/(1-y)$; 0 For $y=0$ and infinity $y=1$

This is the final equation of logistic regression

$$\frac{y}{(1-y)} = \frac{0}{1-0} = \frac{0}{1} = 0 \quad \dots [y=0]$$

$$\frac{y}{(1-y)} \quad \frac{1}{(1-1)} = \frac{1}{0} = \infty \quad \dots [y=1]$$

raisoni
EDUCATION

VLAB II

KNN Algorithm.

In pattern recognition the knearest neighbour algorithm (KNN) is a non-parametric method proposed by Thomas cover used for classification and regression. In both cases the input consists of k-closet training example in the feature space. The output depends on whether KNN ~~is~~ used. ~~The output dep~~ for fication or regression.