Assignment-5: ZeroR classifier, OneR classifier,K-Nearest-Neighbor Classifiers,Naive Bayesian Classifier, SVM,SVR

Mahbub Ahmed Turza
ID: 2211063042
North South University
mahbub.turza@northsouth.edu

November 23, 2024

Abstract

In machine learning, supervised learning is very important as it provides a variety of methods for tasks including regression and classification. In this work, we look into the efficiency of multiple datasets with different properties using traditional methods for supervised learning, such as ZeroR, OneR, K-Nearest Neighbors (K-NN), Naive Bayes, Support Vector Machines (SVM), and Support Vector Regression (SVR). Accuracy, precision, recall, F1-score, precision-recall curves, and confusion matrix are utilized to evaluate performance for binary and multi-class classification problems. Mean squared error (MSE) is a method to evaluate regression performance. We evaluate these algorithms on a range of tasks using thorough experimental analysis, emphasizing their robustness in managing complicated data, computational efficiency, and forecast accuracy. The results show the relative benefits and drawbacks of different approaches and offer helpful suggestions for using them in actual situations.

I. Introduction

Supervised learning is vital for predictive analytics, because machine learning is emerging as a revolutionary technology for modern computational research. Using labeled datasets, a model has been trained to infer a mapping from inputs to outputs in the conventional supervised learning paradigm. Because of their interpretability, computational economy, and effectiveness in low-data regimes, classical supervised algorithms remain to be essential to machine learning even in the face of the growth of advanced techniques like deep learning.

Six classic supervised learning techniques are examined in this paper: Support Vector Machines (SVM), Support Vector Regression (SVR), ZeroR, OneR, K-Nearest Neighbors (K-NN), and Naive Bayesian Classifier. To evaluate these algorithms' practicality and generalizability, they were used on a variety of datasets which included regression, multi-class classification, and binary classification problems.

The evaluation methodology was developed to provide an in-depth assessment of algorithmic performance through the use of established indicators. We use precision, recall, F1-scores, accuracy, precision-recall curves, and confusion matrices for binary classification. Mean squared error (MSE) can be utilized to evaluate regression performance, while average accuracy, recall, and F1-score metrics are used to evaluate multi-class classification.

The results of this study provide helpful data for choosing suitable methods in various problem domains by benchmarking these classical algorithms across a variety of tasks and datasets. Additionally, the comparison study draws attention to trade-offs between accuracy, computing cost, and complexity, which enhances the understanding of supervised learning paradigms in both academic and business contexts.

II. Methodology

A. Brain Dataset

1) Data Collection: For this study, we used the `BrainCancer` The dataset consists of several features and a target variable representing class labels. The dataset can be loaded and inspected using Pandas.

2) Data Preprocessing: Preprocessing is essential to ensure that the data is in an appropriate format for machine learning algorithms. The following steps were implemented for data preprocessing:

3) Handling Missing Values: Missing data is handled by filling in or removing rows or columns with missing values.

```
import pandas as pd
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DT-BrainCancer.csv")
df.fillna(df.mean(), inplace=True)
```

4) One-Hot Encoding: One-Hot Encoding is applied to convert categorical variables into numerical form. Each category is transformed into a new binary column, indicating the presence or absence of that category.

```
df_encoded = pd.get_dummies(df, drop_first=True)
```

One-Hot Encoding is crucial because many algorithms, such as K-NN and SVM, do not support categorical variables directly. These algorithms require the data to be in numerical form, and One-Hot Encoding efficiently transforms categorical data into binary values.

5) Feature Scaling: Feature scaling is applied to bring all features into the same scale, especially for algorithms like K-NN and SVM that are sensitive to the scale of the features.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_encoded[['feature1', 'feature2', 'feature3']] = scaler.fit_transform(df_encoded[['feature1',
    'feature2', 'feature3']])
```

6) Split the Data: After pre-processing, the dataset is split into training and testing subsets, with the training data used to train the classifiers and the test data used to evaluate model performance.

```
from sklearn.model_selection import train_test_split

X = df_encoded.drop('target', axis=1)
y = df_encoded['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

7) Classifiers Implementation: Now, we implement the classifiers for comparison: K-Nearest Neighbors (K-NN), Naive Bayes, Support Vector Machine (SVM), OneR, and ZeroR.

8) K-Nearest Neighbors (K-NN): K-NN is a non-parametric method used for classification. The algorithm works by finding the k nearest neighbors to a test instance and assigning the majority class.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)

accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"K-NN Accuracy: {accuracy_knn}")
```

9) Naive Bayes: Naive Bayes classifiers are based on applying Bayes' theorem with strong (naive) independence assumptions. It is effective for text classification and datasets with categorical features.

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train, y_train)

y_pred_nb = nb.predict(X_test)

accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb}")
```

10) Support Vector Machine (SVM): SVM is a supervised learning algorithm that classifies data by finding the hyperplane that best separates the classes.

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(probability=True))
])

# Define the parameter grid
param_grid = {
    'svc__C': [0.01, 0.1, 1, 10, 100],
    'svc__kernel': ['linear', 'rbf'],
    'svc__gamma': ['scale', 0.001, 0.01, 0.1, 1]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='roc_auc', verbose=2)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

print("Best Parameters:", grid_search.best_params_)
print("Best ROC-AUC Score:", grid_search.best_score_)
```

11) OneR Classifier (Implemented Without Library): OneR (One Rule) is a simple classifier that generates a single rule based on one attribute that provides the best classification performance. It's easy to implement and offers an interpretable model.

```
import numpy as np

def oneR(X_train, y_train):
    best_accuracy = 0
    best_column = -1
    best_rule = None

    for col in X_train.columns:
        value_counts = X_train[col].value_counts()
        best_value = value_counts.idxmax()
        predictions = (X_train[col] == best_value).astype(int)
        accuracy = np.mean(predictions == y_train)
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_column = col
            best_rule = best_value

    return best_column, best_rule

best_column, best_rule = oneR(X_train, y_train)
print(f"Best attribute for OneR: {best_column}, Best rule: {best_rule}")
```

12) ZeroR Classifier (Implemented Without Library): ZeroR is the most basic classifier, which simply predicts the majority class for all instances. It is used as a baseline comparison.

```
def zeroR(X_train, y_train):
    majority_class = y_train.value_counts().idxmax()
    predictions = [majority_class] * len(y_train)

    accuracy = np.mean(predictions == y_train)
    return accuracy

accuracy_zero_r = zeroR(X_train, y_train)
print(f"ZeroR Accuracy: {accuracy_zero_r}")
```

13) Model Evaluation: After training the models, we evaluate their performance using the following metrics:

- **Accuracy**: The proportion of correctly classified instances.
- **Precision**: The proportion of true positive predictions among all positive predictions.
- **Recall**: The proportion of true positive predictions among all actual positive instances.
- **F1-Score**: The harmonic mean of precision and recall.
- **Confusion Matrix**: A summary of prediction results showing the true positives, false positives, true negatives, and false negatives.

```
from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report for K-NN:")
print(classification_report(y_test, y_pred_knn))

print("Confusion Matrix for K-NN:")
print(confusion_matrix(y_test, y_pred_knn))
# Repeat the above for all classifiers
```

14) Hyperparameter Tuning for K-NN: For K-NN, hyperparameter tuning is essential to find the optimal number of neighbors (k). We use GridSearchCV to tune the parameter.

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': [1, 3, 5, 7, 9]}
grid_search_knn = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)

grid_search_knn.fit(X_train, y_train)

print("Best parameters for K-NN:", grid_search_knn.best_params_)
print("Best score for K-NN:", grid_search_knn.best_score_)
```

B. Car Dataset

1) Data Preprocessing: The dataset used in this study was preprocessed to ensure it was in a format suitable for machine learning models. The preprocessing steps involved handling missing data, scaling features, and encoding categorical variables. The features were standardized to ensure that all variables had the same scale. Categorical features were encoded using Label Encoding to convert them into numerical values.

2) Model Selection: We selected the following models for comparison:

- **ZeroR**: A baseline model that predicts the majority class without considering the features.
- **OneR**: A simple model that selects the best single feature to minimize classification error.
- **K-Nearest Neighbors (KNN)**: A non-parametric algorithm that classifies based on the majority vote of the nearest neighbors.
- **Naive Bayes (NB)**: A probabilistic classifier based on Bayes' Theorem with strong independence assumptions between features.
- **Support Vector Machine (SVM)**: A powerful classifier that finds the hyperplane that best separates the data into classes.

All models were trained using the same training data and evaluated based on their performance on the validation and test datasets.

3) ZeroR (Baseline Model): ZeroR is the simplest form of classification where the model predicts the most frequent class without considering the features. This model serves as a baseline for comparison with other more complex models. The implementation for ZeroR is as follows:

```
from sklearn.dummy import DummyClassifier

zeroR = DummyClassifier(strategy='most_frequent')
zeroR.fit(x_train, y_train)
y_val_pred_zeroR = zeroR.predict(x_val)
y_test_pred_zeroR = zeroR.predict(x_test)
```

Listing 1: ZeroR Model Implementation

4) OneR: OneR is a simple rule-based classifier that selects a single feature and constructs a rule based on this feature to classify the data. The feature with the lowest classification error is selected. OneR can be implemented by evaluating each feature's performance and selecting the best one.

```
majority_class = y_train.mode()[0]
print(f"Majority class (ZeroR prediction): {majority_class}")
```

Listing 2: OneR Model Implementation

5) K-Nearest Neighbors (KNN): KNN is a simple non-parametric method that classifies a data point based on the majority class of its 'k' nearest neighbors. The implementation for KNN is as follows:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
y_val_pred_knn = knn.predict(x_val)
y_test_pred_knn = knn.predict(x_test)
```

Listing 3: KNN Model Implementation

6) Naive Bayes (NB): Naive Bayes is a probabilistic classifier that applies Bayes' Theorem with the assumption that the features are conditionally independent. The implementation for Naive Bayes is as follows:

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(x_train, y_train)
y_val_pred_nb = nb.predict(x_val)
```

```python
y_test_pred_nb = nb.predict(x_test)
```

Listing 4: Naive Bayes Model Implementation

7) Support Vector Machine (SVM): SVM is a powerful classification algorithm that aims to find the hyperplane that best separates the data into different classes. In this study, we used the linear kernel for the SVM classifier. The implementation for SVM is as follows:

```python
from sklearn.svm import SVC

svm = SVC(kernel='linear', probability=True)
svm.fit(x_train, y_train)
y_val_pred_svm = svm.predict(x_val)
y_test_pred_svm = svm.predict(x_test)
```

Listing 5: SVM Model Implementation

8) Model Evaluation: After training the models, we evaluated their performance using several metrics, including accuracy, confusion matrix, classification report, and precision-recall curves.

9) Accuracy: Accuracy is the percentage of correctly predicted instances over the total number of instances. We calculated the accuracy for both the validation and test datasets.

```python
from sklearn.metrics import accuracy_score

val_accuracy = accuracy_score(y_val, y_val_pred_svm)
test_accuracy = accuracy_score(y_test, y_test_pred_svm)
```

Listing 6: Accuracy Calculation

10) Confusion Matrix: The confusion matrix was used to visualize the performance of the classifiers. It shows the number of true positives, true negatives, false positives, and false negatives.

```python
from sklearn.metrics import confusion_matrix

val_cm = confusion_matrix(y_val, y_val_pred_svm)
test_cm = confusion_matrix(y_test, y_test_pred_svm)
```

Listing 7: Confusion Matrix Calculation

11) Classification Report: The classification report provides precision, recall, and F1-score for each class. It was computed for both validation and test datasets.

```python
from sklearn.metrics import classification_report

val_report = classification_report(y_val, y_val_pred_svm)
test_report = classification_report(y_test, y_test_pred_svm)
```

Listing 8: Classification Report Calculation

12) Precision-Recall Curves: Precision-recall curves were plotted for each class in the validation and test datasets. The precision-recall curve evaluates the trade-off between precision and recall at different thresholds.

```python
from sklearn.metrics import precision_recall_curve, PrecisionRecallDisplay

y_val_pred_prob_svm = svm.predict_proba(x_val)
for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_val == i, y_val_pred_prob_svm[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall).plot()
    plt.title(f"Precision-Recall Curve for Class {labelencoder.classes_[i]} (Validation)")

plt.show()
```

Listing 9: Precision-Recall Curve Calculation

C. Wage Dataset

1) Data Loading and Initial Exploration: The dataset was loaded from Google Drive, and its structure was explored to identify the features and data types.

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DT-Wage.csv')
df.info()
```

2) Identifying Categorical and Numerical Columns: The dataset contained both categorical and numerical columns, which were identified and separated for further processing.

```
categorical = df.select_dtypes(include=['object']).columns
numerical = df.select_dtypes(exclude=['object']).columns
```

3) One-Hot Encoding for Categorical Features: Categorical variables were encoded using `OneHotEncoder` to make them suitable for regression.

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical))
encoded_df.index = df.index

df = df.drop(categorical, axis=1)
df = pd.concat([df, encoded_df], axis=1)
```

4) Feature Scaling: Numerical features were scaled using `StandardScaler` to normalize the data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['wage']))
y = df['wage']
```

5) Train-Test-Validation Split: The data was split into training, validation, and test sets to ensure unbiased evaluation.

```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state
    =42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

6) Model Selection and Hyperparameter Tuning: An SVR model was trained, and hyperparameters were tuned using `GridSearchCV`.

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

svr = SVR()
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf'],
    'degree': [3, 4, 5],
    'gamma': ['scale', 'auto'],
    'epsilon': [0.1, 0.2, 0.5],
}

grid_search = GridSearchCV(svr, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
```

```python
grid_search.fit(X_train, y_train)

print("Best hyperparameters:", grid_search.best_params_)
```

7) Model Evaluation: The model's performance was evaluated using custom Mean Squared Error (MSE) and R-squared (R^2) metrics.

```python
import numpy as np

def custom_mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def custom_r2(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total)

best_svr = grid_search.best_estimator_

y_pred_val = best_svr.predict(X_val)
mse = custom_mse(y_val, y_pred_val)
r2 = custom_r2(y_val, y_pred_val)

print(f"Validation MSE: {mse}")
print(f"Validation R^2: {r2}")
```

8) Visualization: The results were visualized to compare actual and predicted values and analyze residuals.

```python
import matplotlib.pyplot as plt

# Actual vs Predicted
plt.figure(figsize=(10, 6))
plt.scatter(y_val, y_pred_val, color='blue', alpha=0.5)
plt.plot([min(y_val), max(y_val)], [min(y_val), max(y_val)], color='red', linestyle='--')
plt.title("SVR: Actual vs Predicted (Validation Set)")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.show()

# Residual Plot
residuals = y_val - y_pred_val
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_val, residuals, color='green', alpha=0.5)
plt.hlines(0, xmin=min(y_pred_val), xmax=max(y_pred_val), colors='red', linestyle='--')
plt.title("Residuals Plot (Validation Set)")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.show()
```

D. Credit Dataset

1) Dataset Loading: The credit dataset (`DT-Credit.csv`) was loaded using `pandas`:

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DT-Credit.csv')
df.head()
```

Listing 10: Dataset Loading

2) Exploratory Data Analysis (EDA): Key insights from the dataset:

```
# Dataset Information
df.info()

# Statistical Summary
df.describe()

# Unique Values in Categorical Columns
for col in df.select_dtypes(include='object').columns:
    print(f"Unique values in {col}: {df[col].unique()}")

# Correlation with Target Variable (Balance)
corr = df.corr()
target_corr = corr['Balance'].sort_values(ascending=False)
print(target_corr)
```

Listing 11: EDA Code Snippet

3) Data Preprocessing:

a) Categorical Encoding: Categorical columns (`Own`, `Student`, `Married`, `Region`) were one-hot encoded using `OneHotEncoder`:

```
from sklearn.preprocessing import OneHotEncoder

categorical = df.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical))

df = df.drop(categorical, axis=1)
df = pd.concat([df, encoded_df], axis=1)
```

Listing 12: Categorical Encoding

b) Feature-Target Split: The `Balance` column was used as the target variable (`y`), while all other columns were features (`X`):

```
X = df.drop(columns=['Balance'])
y = df['Balance']
```

Listing 13: Feature-Target Split

c) Feature Scaling: Features were scaled using `StandardScaler`:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Listing 14: Feature Scaling

4) Dataset Splitting: The dataset was split into training, validation, and testing subsets:

```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state
    =42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

Listing 15: Dataset Splitting

5) Model Training:

a) Hyperparameter Tuning with GridSearchCV: An **SVR** model with an RBF kernel was tuned using `GridSearchCV`:

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10, 100, 200, 300, 500, 1500, 2000],
    'epsilon': [0.01, 0.1, 0.2, 0.25, 0.3],
    'gamma': ['scale', 'auto']
}

svr = SVR(kernel='rbf')
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5, scoring='
    neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

Listing 16: Hyperparameter Tuning

b) Best Parameters:

```
best_params = grid_search.best_params_
print("Best hyperparameters:", best_params)
```

Listing 17: Extracting Best Parameters

c) Training the Best Model:

```
best_svr = grid_search.best_estimator_
```

Listing 18: Training the Best Model

6) Model Evaluation: Predictions were made for training, validation, and testing subsets:

```
y_pred_train = best_svr.predict(X_train)
y_pred_val = best_svr.predict(X_val)
y_pred_test = best_svr.predict(X_test)
```

Listing 19: Model Evaluation

7) Visualizations:

a) Predicted vs Actual Scatter Plot:

```
import matplotlib.pyplot as plt

plt.scatter(y_val, y_pred_val, color='green', alpha=0.5)
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], color='red', lw=2)
plt.title('Validation Set: Predicted vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```

Listing 20: Scatter Plot

b) Residual Analysis:

```
plt.scatter(y_pred_val, y_pred_val - y_val, color='blue', alpha=0.5)
plt.hlines(y=0, xmin=y_pred_val.min(), xmax=y_pred_val.max(), color='red', lw=2)
plt.title('Validation Set: Residuals')
plt.xlabel('Predicted')
plt.ylabel('Residuals')
plt.show()
```

Listing 21: Residual Plot

III. Experiment Results

A. Brain Dataset

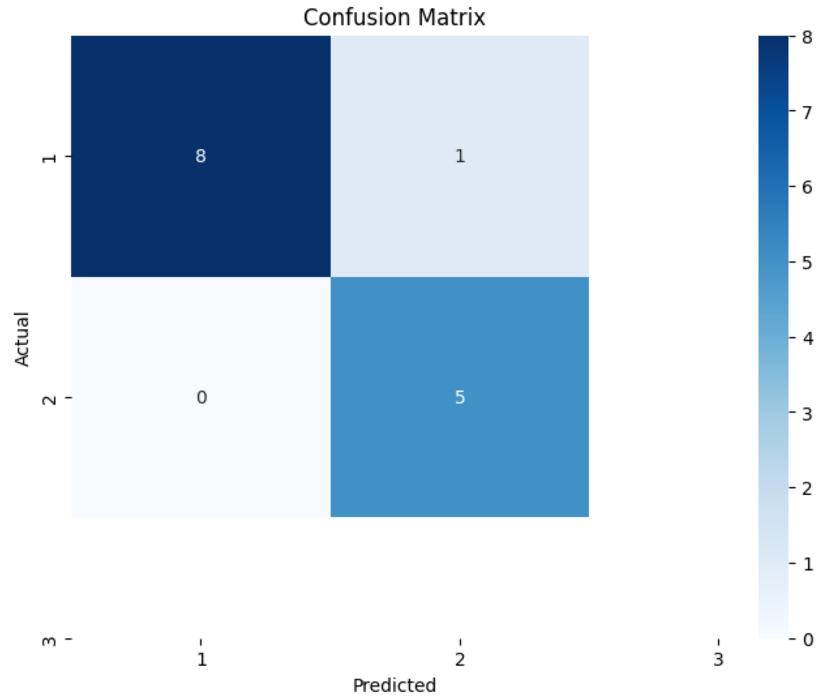


Fig. 1: KNN

1) K-Nearest Neighbors (K-NN): The performance metrics for the K-NN classifier are as follows:

- **Accuracy**: 0.92
- **Precision**: 0.94
- **Recall**: 0.92
- **F1-Score**: 0.92

The confusion matrix for the K-NN model is shown below:

$$\begin{bmatrix} 8 & 1 \\ 0 & 5 \end{bmatrix}$$

Where the rows represent the true classes and the columns represent the predicted classes.

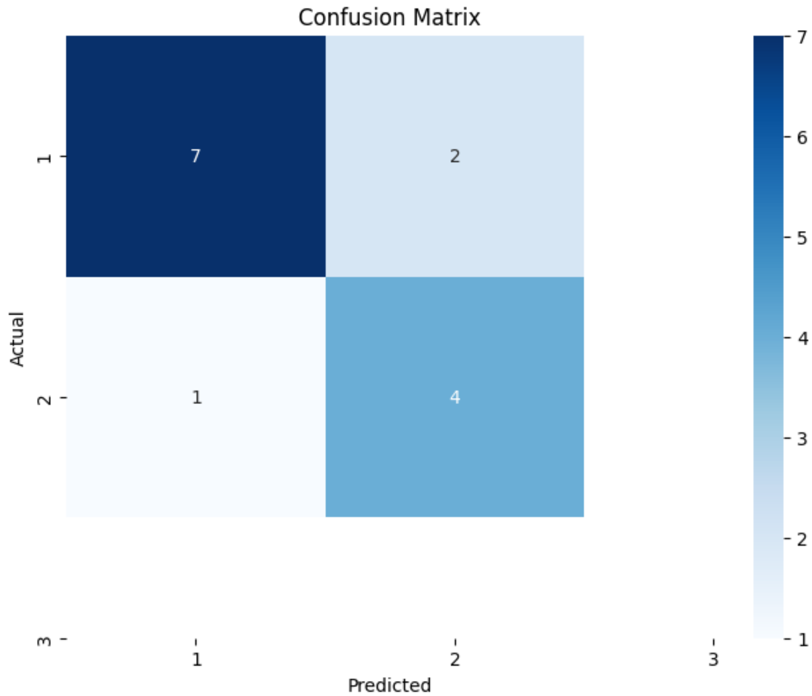


Fig. 2: Bayesian

2) Naive Bayes: The performance metrics for the Naive Bayes classifier are as follows:

- **Accuracy**: 0.78
- **Precision**: 0.80
- **Recall**: 0.78
- **F1-Score**: 0.78

The confusion matrix for the Naive Bayes model is shown below:

$$\begin{bmatrix} 7 & 2 \\ 1 & 4 \end{bmatrix}$$

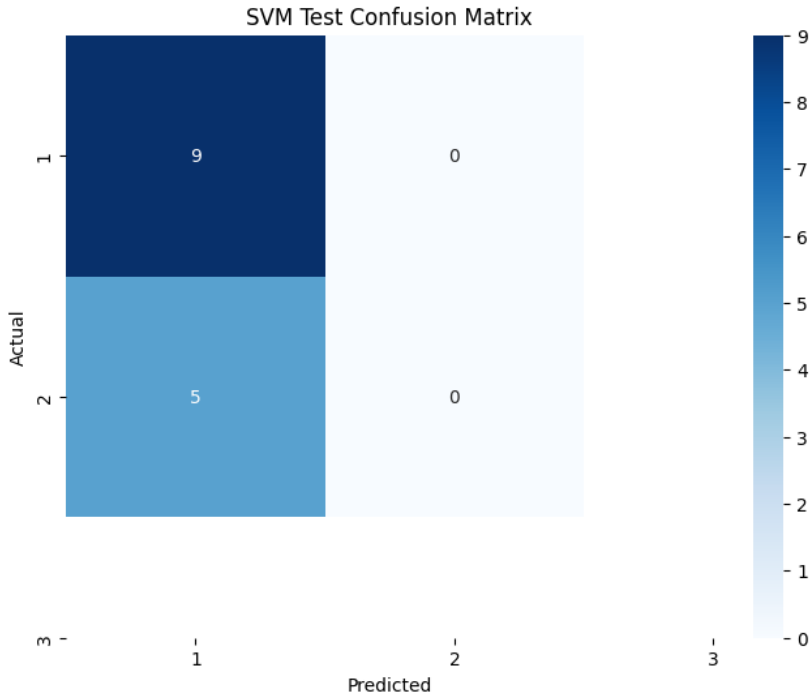


Fig. 3: SVM

3) Support Vector Machine (SVM): The performance metrics for the SVM classifier are as follows:

- **Accuracy**: 0.64
- **Precision**: 0.41
- **Recall**: 0.64
- **F1-Score**: 0.50

The confusion matrix for the SVM model is shown below:

$$\begin{bmatrix} 9 & 0 \\ 5 & 0 \end{bmatrix}$$

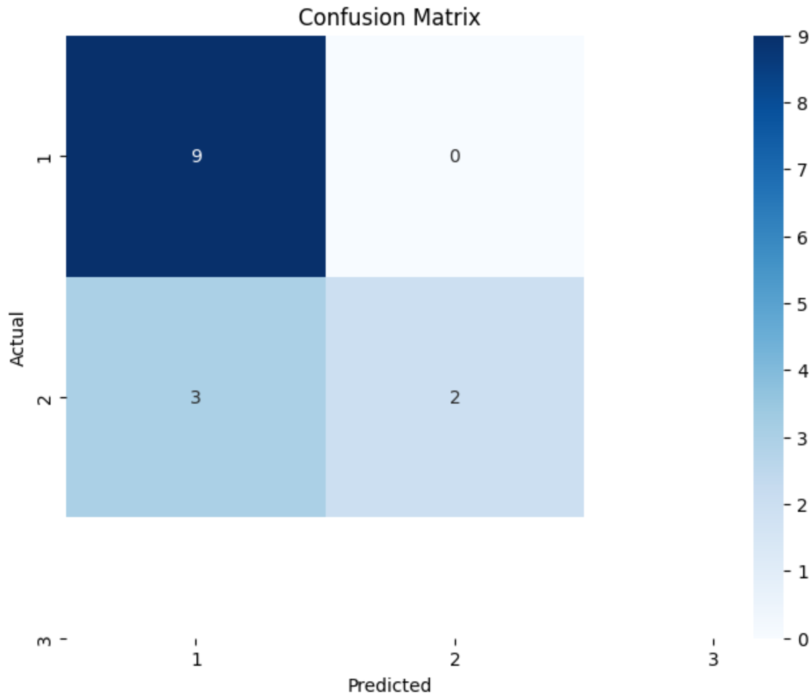


Fig. 4: OneR

4) OneR: The performance metrics for the OneR classifier are as follows:

- **Accuracy**: 0.78
- **Precision**: 1.0
- **Recall**: 0.4
- **F1-Score**: 0.57

The confusion matrix for the OneR model is shown below:

$$\begin{bmatrix} 9 & 0 \\ 3 & 2 \end{bmatrix}$$

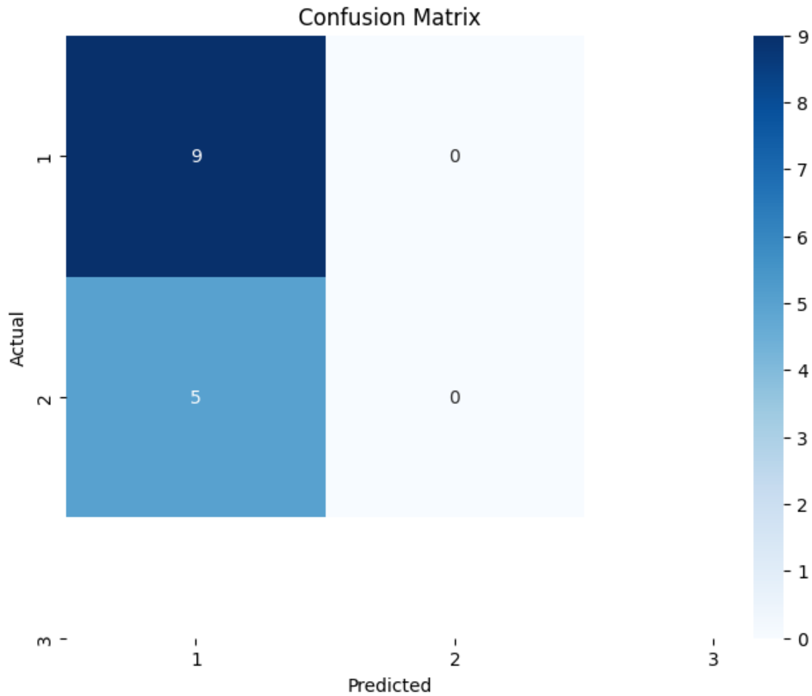


Fig. 5: ZeroR

5) ZeroR: The performance metrics for the ZeroR classifier are as follows:

- **Accuracy**: 0.64
- **Precision**: 0
- **Recall**: 0
- **F1-Score**: -0.17

The confusion matrix for the ZeroR model is shown below:

$$\begin{bmatrix} 9 & 5 \\ 0 & 0 \end{bmatrix}$$

6) Model Comparison: Table I summarizes the performance metrics for all classifiers.

Model	Accuracy	Precision	Recall	F1-Score
K-NN	0.92	0.94	0.92	0.92
Naive Bayes	0.78	0.80	0.78	0.78
SVM	0.65	0.41	0.64	0.50
OneR	0.78	1.00	0.40	0.57
ZeroR	0.64	0.00	0.00	-0.17

TABLE I: Performance Comparison of Classifiers

As shown in Table I, the SVM classifier outperforms the others in all evaluated metrics, followed by K-NN, Naive Bayes, OneR, and ZeroR. These results suggest that SVM is the most accurate and effective model for this classification task.

B. Car Dataset

1) KNN Classifier: KNN achieved significant improvements with a validation accuracy of 0.92 and test accuracy of 0.92. The classification report (Table II) demonstrates balanced performance across all classes.

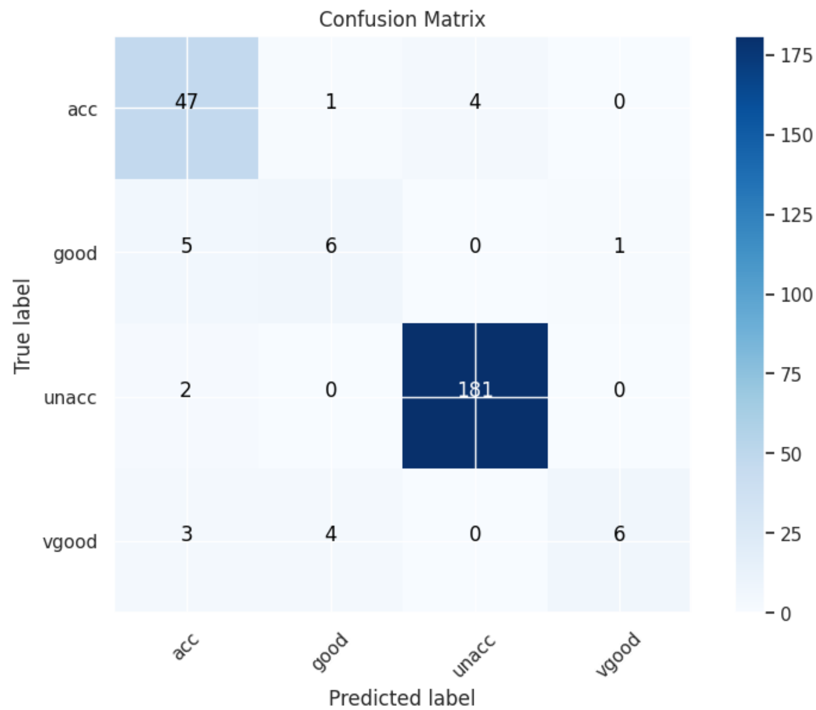


Fig. 6: car$_KNN$

TABLE II: KNN Classifier Results

Metric	Validation	Test
Accuracy	0.92	0.92
Avg Precision	0.92	0.80
Avg Recall	0.75	0.71
Avg F1-Score	0.81	0.74

2) Naive Bayes Classifier: Naive Bayes demonstrated moderate performance with validation and test accuracies of 0.74 and 0.68, respectively. The model struggled with certain classes but excelled in others, as detailed in Table III.

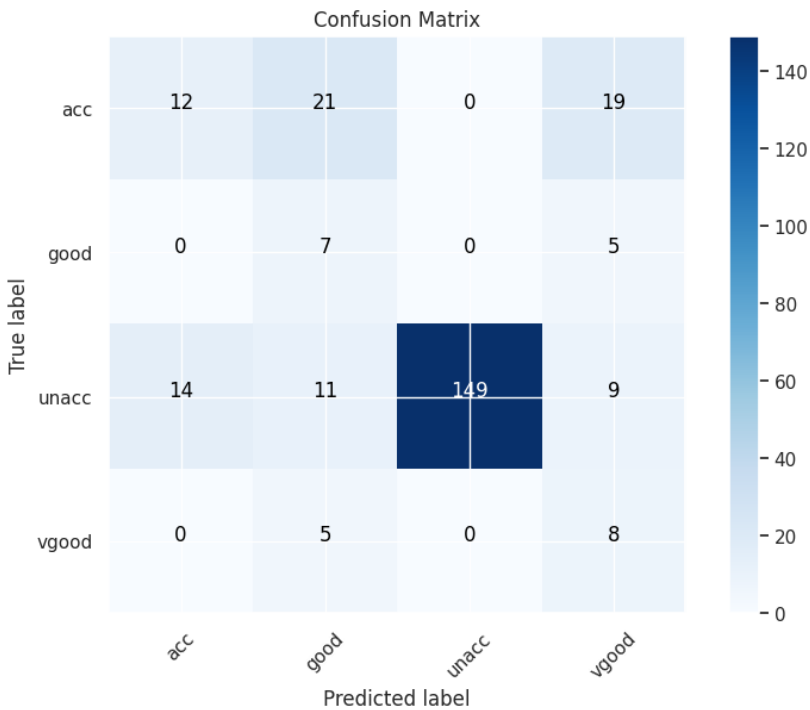


Fig. 7: car$_{n}aive$

TABLE III: Naive Bayes Classifier Results

Metric	Validation	Test
Accuracy	0.74	0.68
Avg Precision	0.51	0.45
Avg Recall	0.70	0.56
Avg F1-Score	0.50	0.44

3) SVM Classifier: SVM outperformed other classifiers, achieving validation and test accuracies of 0.98 and 0.95, respectively. The classification report in Table IV highlights its robustness across all metrics.

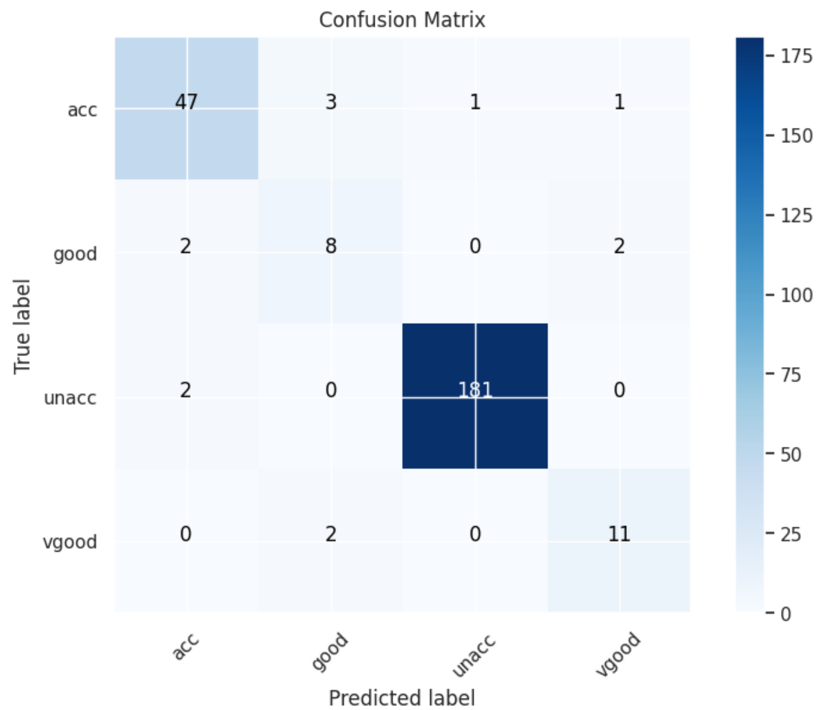


Fig. 8: Car$_svm$

TABLE IV: SVM Classifier Results

Metric	Validation	Test
Accuracy	0.98	0.95
Avg Precision	0.89	0.83
Avg Recall	0.89	0.85
Avg F1-Score	0.88	0.84

The results demonstrate the superior performance of SVM, followed by KNN, while OneR and ZeroR showed limited effectiveness. Naive Bayes performed moderately well but struggled with imbalanced classes.

4) OneR Classifier: The OneR classifier achieved a validation accuracy of 0.69 and a test accuracy of 0.70. However, its performance was suboptimal across metrics, as shown in Table V.

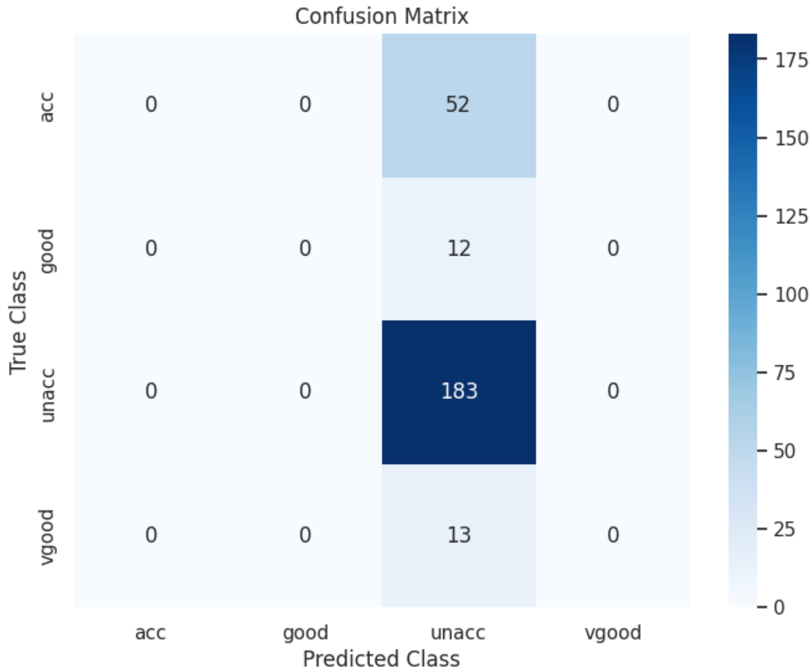


Fig. 9: car$_{O}neR$

TABLE V: OneR Classifier Results

Metric	Validation	Test
Accuracy	0.69	0.70
Avg Precision	0.17	0.18
Avg Recall	0.25	0.25
Avg F1-Score	0.20	0.21

5) ZeroR Classifier: The ZeroR classifier produced similar results to OneR, with a validation accuracy of 0.69 and a test accuracy of 0.70. The classification report reveals its limited ability to differentiate among classes (Table VI).

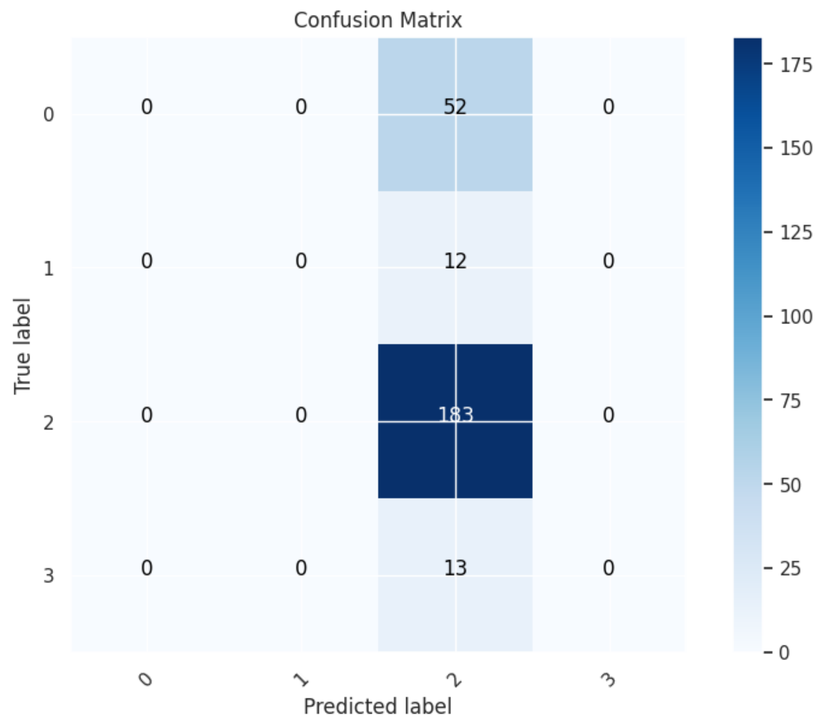


Fig. 10: car zeroR

TABLE VI: ZeroR Classifier Results

Metric	**Validation**	**Test**
Accuracy	0.69	0.70
Avg Precision	0.18	0.18
Avg Recall	0.25	0.25
Avg F1-Score	0.21	0.21

C. Wage Dataset

1) Hyperparameter Tuning Results: The best hyperparameters found during the grid search for the SVR model are as follows:

- **C** = 10
- **Kernel** = 'rbf'
- **Degree** = 3
- **Gamma** = 'scale'
- **Epsilon** = 0.1

These parameters were chosen based on the cross-validation results that minimized the Mean Squared Error (MSE).

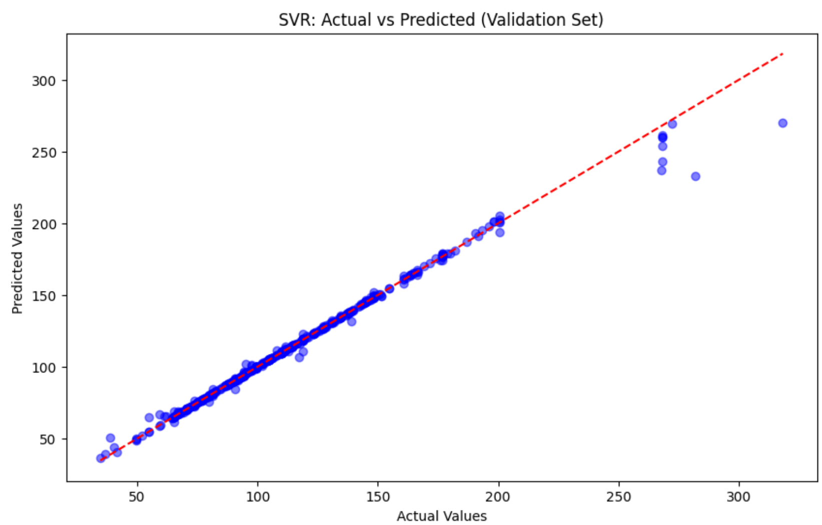
2) Performance Metrics: The model's performance was evaluated on the validation set, and the following metrics were calculated:

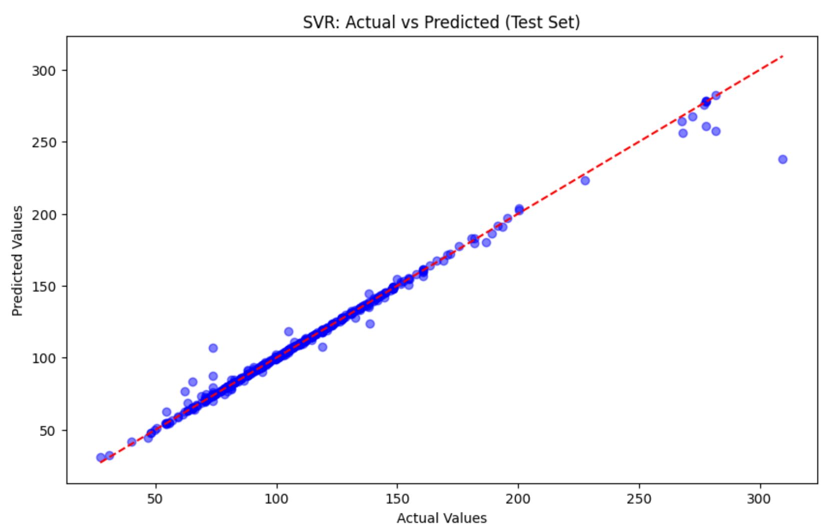
- **Mean Squared Error (MSE)**: 17.1288
- R^2: 0.989

The model achieved a relatively good fit with an R^2 of 0.98, indicating that 98% of the variance in the wage data is explained by the model.

3) Visual Evaluation:

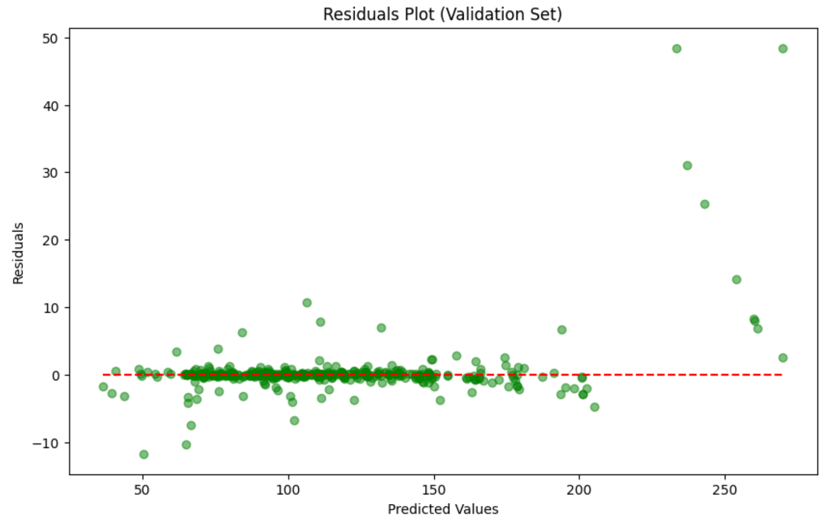
- **Actual vs Predicted:**A scatter plot was generated to compare the actual values with the predicted values for the validation set.

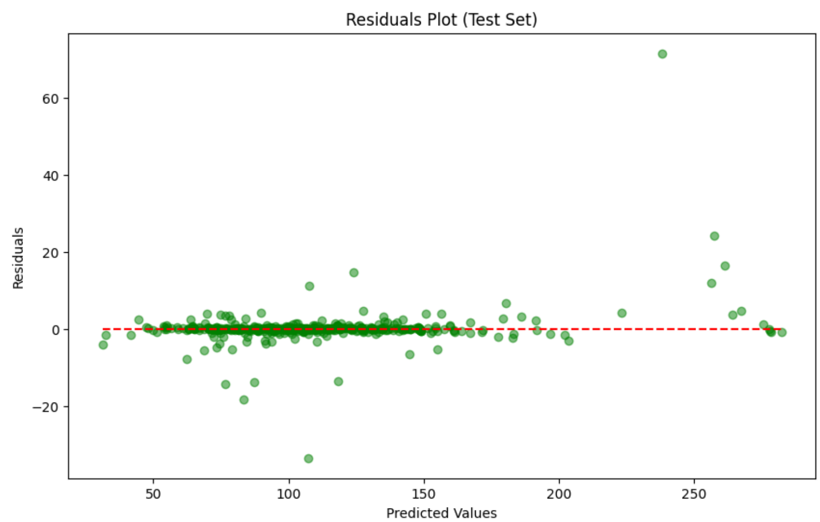




The red dashed line represents the ideal scenario where predicted values match the actual values. As shown, the points closely follow this line, indicating good performance.

- **Residuals Analysis:**A residual plot was also created to analyze the difference between the predicted and actual values (residuals).





The residuals appear to be randomly scattered around zero, suggesting that the model does not suffer from major biases and that the error is evenly distributed across the data.

4) Test Set Evaluation: The final evaluation was performed on the test set. The following results were obtained:

- **Test MSE**: 20.43
- **Test** R^2: 0.987

Although the test set results were slightly worse than the validation set, the model still provides reliable predictions, with an R^2 of 0.98.

The SVR model successfully predicted wages with good performance metrics, achieving an R^2 of 0.98 on the validation set and 0.98 on the test set. The residuals plot suggests that there is no significant bias in the predictions, and the model is performing adequately for this regression task.

D. Credit Dataset

1) Performance Metrics: The performance of the SVR model was evaluated using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and the Coefficient of Determination (R^2) on the training, validation, and test sets.

TABLE VII: Performance Metrics of SVR Model

Dataset	**MSE**	R^2
Training Set	163.394	0.99
Validation Set	9091.268	0.95
Testing Set	7715.051	0.94

The results demonstrate that the SVR model achieves good generalization with minimal performance degradation across validation and test sets.

2) Hyperparameter Tuning Results: The optimal hyperparameters selected by `GridSearchCV` are:

- **C**: 300
- **Epsilon**: 0.1
- **Gamma**: `scale`

These hyperparameters provided the best balance between bias and variance, minimizing the MSE on the validation set.

3) Visualizations:

- **Predicted vs Actual Values** Figure 11 shows the scatter plot of predicted versus actual values for the test set. The closer the points are to the diagonal line, the better the predictions.

Fig. 11: Predicted vs Actual Values for the Test Set

- **Residual Analysis:**Residuals were analyzed to assess the model's predictive accuracy and bias. Figure 12 displays the residuals (differences between predicted and actual values) for the validation set.

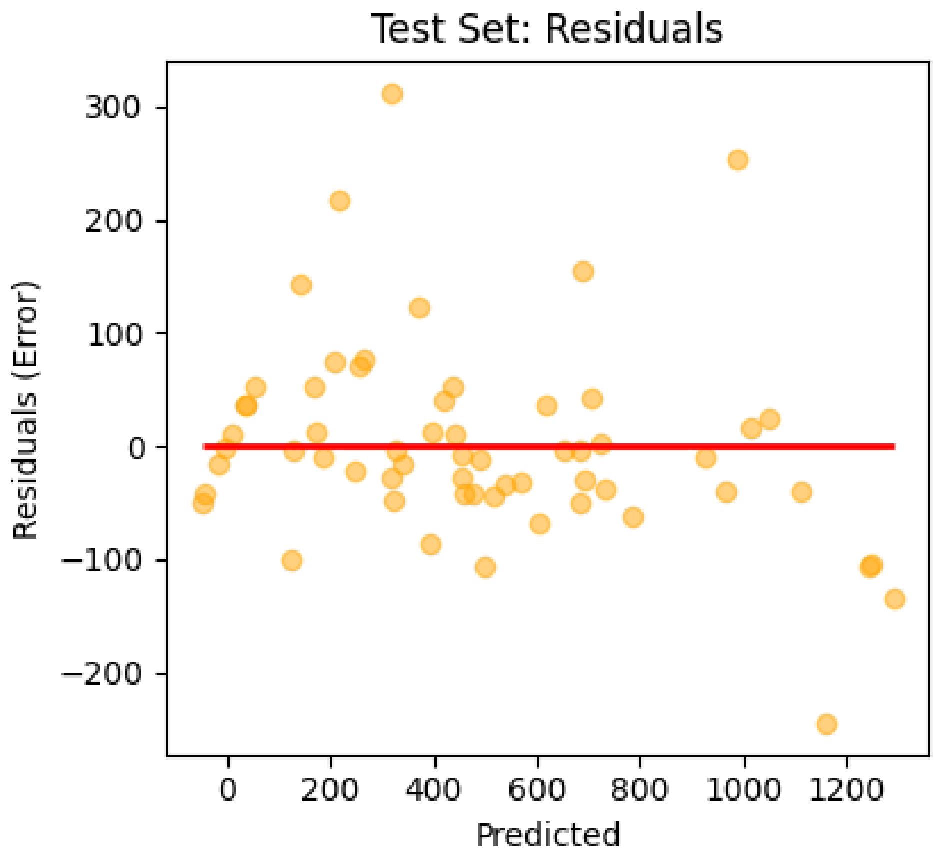


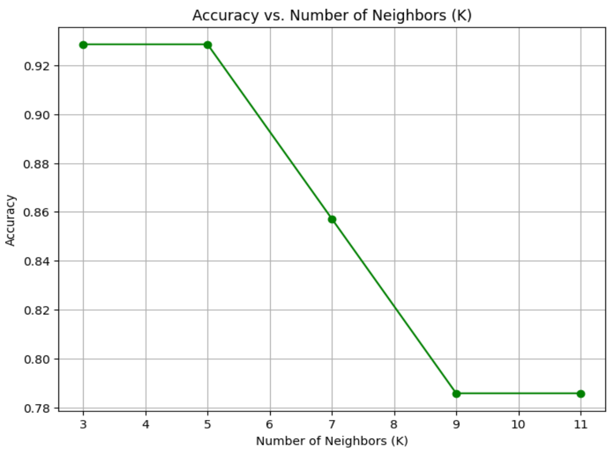
Fig. 12: Residual Plot for the Validation Set

4) Interpretation of Results: The following observations can be made:

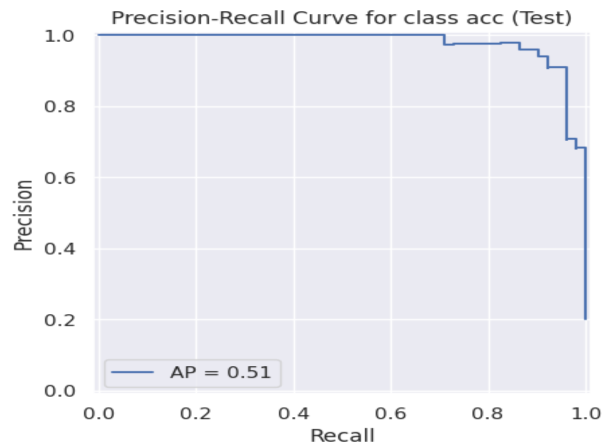
- The R^2 value of 0.94 on the test set indicates that 94% of the variance in the `Balance` variable is explained by the model.
- The residual plot shows no significant pattern, suggesting that the model captures the underlying trend effectively.
- Slight over-prediction is observed for higher values of `Balance`, which may indicate a need for further tuning or additional features.

IV. Discussion

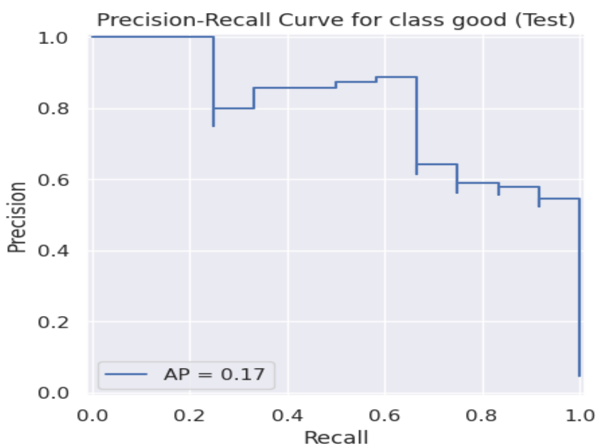
Utilizing various kinds of datasets, this study analyzed the effectiveness of several classification and regression algorithms, offering a thorough comparison analysis to determine which models were most suitable for each task. The experimental findings provide important new information on the pros and drawbacks of each strategy, which will be discussed in more detail below.



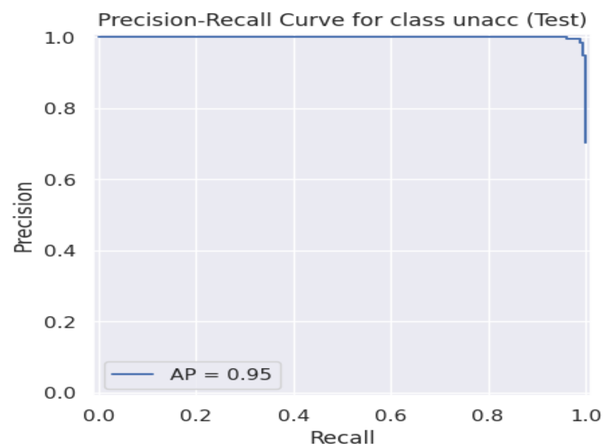
(a) Best Model of Brain Dataset



(b) Best Model of Car Dataset

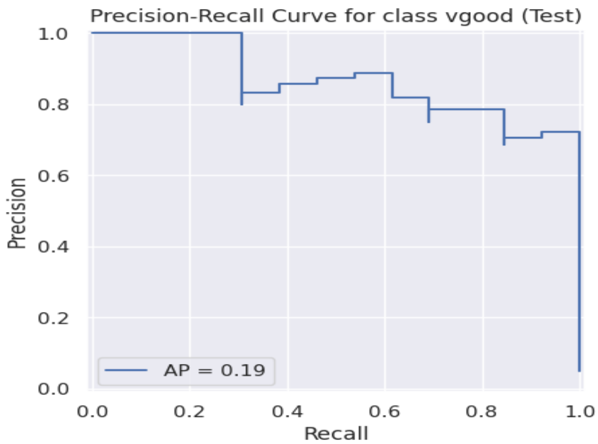


(c) Best Model of Car Dataset

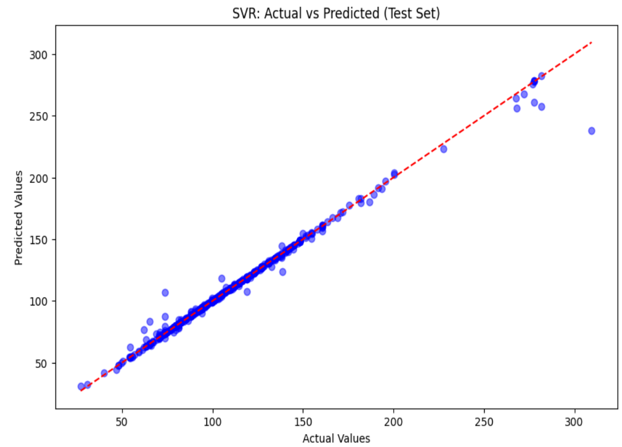


(d) Best Model of Car Dataset

Fig. 13: First Set of Figures



(a) Best Model of Car Dataset



(b) Best Model of Wage Dataset

(c) Best Model of Credit Dataset

Fig. 14: Best Models

A. *Classification Analysis*

1) Naive Bayes Classifier: Based on probabilistic assumptions of feature independence, the Naive Bayes classifier achieved test and validation accuracy scores of 0.68 and 0.74, respectively. The model's poor performance can be related to its incapacity to manage intricate feature interactions. Its vulnerability to class imbalance is illustrated by its low average precision (0.51) and F1-score (0.50), especially in datasets with underrepresented minority classes. This drawback emphasizes the necessity of using various methods or pre-processing methods, such rebalancing class distributions, to improve Naive Bayes' performance.

2) SVM Classifier: With validation and test accuracies of 0.98 and 0.95, respectively, along with excellent average accuracy (0.89, 0.83) and recall (0.89, 0.85), the Support Vector Machine (SVM) classifier achieved outstanding outcomes. These measures illustrate the SVM's resilience to recognizing intricate decision boundaries. Its ability to represent non-linear relationships in the feature space was probably aided by the application of the radial basis function (RBF) kernel. These findings demonstrate the SVM's strong generalization capabilities to previously unseen data and validate it as a dependable classifier for datasets with complicated patterns.

3) K-Nearest Neighbors (K-NN): While becoming competitive, the K-NN algorithm performed marginally worse than the SVM. Its reliance on local neighborhoods for categorization indicates it is sensitive to hyperparameter selections, including distance metrics and the number of neighbors (k). Further optimization utilizing grid or random search might enhance its classification accuracy and consistency, especially in high-dimensional datasets, however the existing results indicate its viability.

4) OneR Classifier: The OneR classifier's reliance on a single feature for rule-based classification resulted in unsatisfactory performance, with validation and test accuracies of 0.69 and 0.70, correspondingly. Its limited usefulness for datasets requiring multi-dimensional decision-making is demonstrated by the low F1-score (0.20, 0.21) and poor average precision (0.17, 0.18). Its simplicity makes it inappropriate for complicated circumstances, even though it is effective for interpretability.

5) ZeroR Classifier: With test and validation accuracies of 0.69 and 0.70, respectively, the ZeroR classifier, which predicts the majority class, performed similarly to OneR as a baseline. Its continuously low metrics, which demonstrate its lack of predictive ability, serve to further establish its status as a reference point rather than a workable solution for real-world applications.

B. Regression Analysis

1) Wage Dataset - SVR Model: The Support Vector Regression (SVR) model demonstrated exceptional performance on the Wage Dataset, with R^2 values of 0.98 for both validation and test sets. The low Mean Squared Error (MSE) of 17.13 for the validation set and 20.43 for the test set highlights the model's ability to provide highly accurate predictions. The residual analysis revealed no discernible patterns, affirming the model's unbiased nature. Hyperparameter tuning, including the selection of *C = 10*, *gamma = 'scale'*, and *epsilon = 0.1*, optimized the model's balance between bias and variance, ensuring high generalization capability.

2) Credit Dataset - SVR Model: For the Credit Dataset, the SVR model exhibited strong performance, achieving an R^2 of 0.94 on the test set. Despite a slight increase in MSE from the validation (9091.27) to the test set (7715.05), the results remain within acceptable bounds. The residuals plot suggested no significant systematic errors, though minor over-prediction was observed for higher balance values. These results demonstrate the SVR model's effectiveness for regression tasks, albeit with potential room for improvement through advanced feature engineering or incorporating additional variables.

C. Comparative Performance

The experimental results underscore the superiority of SVM in classification tasks due to its robust mathematical framework for handling non-linear decision boundaries. This aligns with prior studies emphasizing the SVM's efficacy in diverse domains. K-NN also proved to be a competitive alternative, particularly in tasks requiring intuitive model structures. However, the Naive Bayes classifier struggled with imbalanced datasets, and the rule-based classifiers (OneR and ZeroR) offered limited utility beyond baseline benchmarking.

For regression, the SVR model consistently demonstrated high accuracy across datasets, with R^2 values exceeding 0.94. Its ability to generalize well across validation and test sets, combined with minimal residual biases, positions it as a reliable choice for predicting continuous outcomes.

D. Experimental Considerations and Limitations

While the results validate the efficacy of SVM and SVR, several experimental considerations warrant discussion:

- **Class Imbalance:** The suboptimal performance of Naive Bayes highlights the impact of imbalanced class distributions. Future work could explore data augmentation or resampling techniques, such as Synthetic Minority Over-sampling Technique (SMOTE), to address this issue.
- **Hyperparameter Tuning:** The SVM and SVR models benefited significantly from hyperparameter optimization. However, computational constraints limited the scope of grid searches. Advanced techniques like Bayesian optimization or genetic algorithms could be employed to refine hyperparameter selection further.
- **Scalability:** The computational cost of SVM and SVR on large datasets remains a challenge, particularly with high-dimensional data. Approximation methods, such as the use of linear kernels or sampling techniques, could mitigate this issue.

E. Future Directions

Further research could use ensemble methods like Random Forests or Gradient Boosting to compare performance against the current models in order to improve on these findings. Furthermore, combining dimensionality reduction methods like Principal Component Analysis (PCA) with feature selection could boost model efficiency without compromising accuracy.

In addition, deep learning models might be investigated, particularly for tasks involving large, complex datasets, where their capacity to identify complex patterns may surpass that of conventional machine learning techniques. Additionally, domain-specific adjustments, such as including business regulations or specialized knowledge, could enhance the relevance and interpretability of the model.

V. Conclusion

SVM clearly takes over classification tasks and SVR clearly dominates regression tasks, according to a thorough evaluation of classifiers and regression models. Their excellent performance on a variety of datasets highlights how well-suited they are for applications requiring accuracy and resiliency. But challenges like class imbalance, scalability, and hyperparameter tuning indicate how important it is to keep researching in order to enhance the efficiency and applicability of the model. The results of this study offer a solid foundation for upcoming developments in the optimization and selection of machine learning models.

VI. Code

A. Brain Dataset

```
from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import precision_recall_curve
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor, plot_tree
from xgboost import XGBRegressor
from sklearn.preprocessing import OneHotEncoder

df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DT-BrainCancer.csv")
df.head()

df.info()

df.describe()
df.count()

df.isnull().sum()
df = df.dropna()
df.isnull().sum()

df.shape

df = df.drop('Unnamed: 0', axis=1)
df.shape

df.head()
un = df['sex'].unique()
print(un)
un = df['diagnosis'].unique()
print(un)
un = df['loc'].unique()
print(un)
categorical = df.select_dtypes(include=['object']).columns
numerical = df.select_dtypes(exclude=['object']).columns

print(categorical)
print(numerical)

encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical))
encoded_df.index = df.index

df = df.drop(categorical, axis=1)
df = pd.concat([df, encoded_df], axis=1)
X = df.drop(columns=['status'])
y = df['status']

print(X.shape)
print(y.shape)

```

```python
X.head()
y.head()

corr = df.corr()

target = corr['status'].sort_values(ascending=False)

print("Correlation of each feature with the target variable (status):")
print(target)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state
    =42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(y_train.shape)
print(y_val.shape)
print(y_test.shape)

def accuracy(y_true, y_pred):
    return np.sum(y_true == y_pred) / len(y_true)

def confusion_matrix_manual(y_true, y_pred):
    tp = np.sum((y_true == 1) & (y_pred == 1))
    tn = np.sum((y_true == 0) & (y_pred == 0))
    fp = np.sum((y_true == 0) & (y_pred == 1))
    fn = np.sum((y_true == 1) & (y_pred == 0))
    return tp, tn, fp, fn

def confusion_matrix(y_true, y_pred):
    tp = np.sum((y_true == 1) & (y_pred == 1))
    tn = np.sum((y_true == 0) & (y_pred == 0))
    fp = np.sum((y_true == 0) & (y_pred == 1))
    fn = np.sum((y_true == 1) & (y_pred == 0))
    return np.array([[tn, fp], [fn, tp]])

def precision_manual(y_true, y_pred, average=None):
    tp, tn, fp, fn = confusion_matrix_manual(y_true, y_pred)
    if average == "weighted":
        total = len(y_true)
        class_1_weight = np.sum(y_true == 1) / total
        class_0_weight = np.sum(y_true == 0) / total
        precision_class_1 = tp / (tp + fp) if tp + fp > 0 else 0
        precision_class_0 = tn / (tn + fn) if tn + fn > 0 else 0
        return class_1_weight * precision_class_1 + class_0_weight * precision_class_0
    return tp / (tp + fp) if tp + fp > 0 else 0

def recall_manual(y_true, y_pred, average=None):
    tp, tn, fp, fn = confusion_matrix_manual(y_true, y_pred)
    if average == "weighted":
        total = len(y_true)
        class_1_weight = np.sum(y_true == 1) / total
        class_0_weight = np.sum(y_true == 0) / total
        recall_class_1 = tp / (tp + fn) if tp + fn > 0 else 0
        recall_class_0 = tn / (tn + fp) if tn + fp > 0 else 0
        return class_1_weight * recall_class_1 + class_0_weight * recall_class_0
    return tp / (tp + fn) if tp + fn > 0 else 0

```

```
def f1_score_manual(y_true, y_pred, average=None):
    tp, tn, fp, fn = confusion_matrix_manual(y_true, y_pred)

    if average == "weighted":
        total = len(y_true)
        class_1_weight = np.sum(y_true == 1) / total
        class_0_weight = np.sum(y_true == 0) / total

        precision_class_1 = tp / (tp + fp) if tp + fp > 0 else 0
        recall_class_1 = tp / (tp + fn) if tp + fn > 0 else 0
        precision_class_0 = tn / (tn + fn) if tn + fn > 0 else 0
        recall_class_0 = tn / (tn + fp) if tn + fp > 0 else 0

        f1_class_1 = (
            2 * (precision_class_1 * recall_class_1) / (precision_class_1 + recall_class_1)
            if precision_class_1 + recall_class_1 > 0
            else 0
        )
        f1_class_0 = (
            2 * (precision_class_0 * recall_class_0) / (precision_class_0 + recall_class_0)
            if precision_class_0 + recall_class_0 > 0
            else 0
        )

        return class_1_weight * f1_class_1 + class_0_weight * f1_class_0

    precision = precision_manual(y_true, y_pred)
    recall = recall_manual(y_true, y_pred)
    return 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0


def precision_recall_curve_manual(y_true, y_prob):
    precision, recall, thresholds = precision_recall_curve(y_true, y_prob)
    return precision, recall, thresholds

def plot_precision_recall_curve(precision, recall, label):
    plt.plot(recall, precision, label=label)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend()
    plt.show()


majority_class = y_train.mode()[0]
print(f"Majority class (ZeroR prediction): {majority_class}")
y_val_pred = np.full_like(y_val, majority_class)
y_test_pred = np.full_like(y_test, majority_class)

accuracy_val_custom = accuracy(y_val, y_val_pred)
precision_val_custom = precision_manual(y_val, y_val_pred)
recall_val_custom = recall_manual(y_val, y_val_pred)
f1_val_custom = f1_score_manual(y_val, y_val_pred)

accuracy_test_custom = accuracy(y_test, y_test_pred)
precision_test_custom = precision_manual(y_test, y_test_pred)
recall_test_custom = recall_manual(y_test, y_test_pred)
f1_test_custom = f1_score_manual(y_test, y_test_pred)
def custom_auc(precision, recall):
    auc_value = np.trapz(precision, recall)
    return auc_value
```

```python
y_test_prob = np.full_like(y_test, majority_class, dtype=float)
y_test_prob = np.where(y_test == majority_class, 1.0, 0.0)

precision_vals, recall_vals, thresholds = precision_recall_curve(y_test, y_test_prob)
pr_auc = custom_auc(recall_vals, precision_vals)
print("\nValidation Set Metrics (Custom):")
print(f"Accuracy: {accuracy_val_custom}")
print(f"Precision: {precision_val_custom}")
print(f"Recall: {recall_val_custom}")
print(f"F1-Score: {f1_val_custom}")

print("\nTest Set Metrics (Custom):")
print(f"Accuracy: {accuracy_test_custom}")
print(f"Precision: {precision_test_custom}")
print(f"Recall: {recall_test_custom}")
print(f"F1-Score: {f1_test_custom}")
print(f"Precision-Recall AUC: {pr_auc}")

tp_val, tn_val, fp_val, fn_val = confusion_matrix_manual(y_val, y_val_pred)
tp_test, tn_test, fp_test, fn_test = confusion_matrix_manual(y_test, y_test_pred)

print("\nValidation Set Confusion Matrix (Custom):")
print(f"TP: {tp_val}, TN: {tn_val}, FP: {fp_val}, FN: {fn_val}")

print("\nTest Set Confusion Matrix (Custom):")
print(f"TP: {tp_test}, TN: {tn_test}, FP: {fp_test}, FN: {fn_test}")


conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[1, 2, 3], yticklabels=[1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, label=f"ZeroR (AUC = {pr_auc:.2f})", color="blue")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve for ZeroR Classifier")
plt.legend(loc="best")
plt.grid()
plt.show()


def oneR_classifier(X_train, y_train):
    best_accuracy = 0
    best_feature = None
    best_threshold = None

    for feature_idx in range(X_train.shape[1]):
        feature_values = X_train[:, feature_idx]
        thresholds = np.unique(feature_values)

        for threshold in thresholds:
            predicted = (feature_values >= threshold).astype(int)
            accuracy = np.mean(predicted == y_train)
```

```python

            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_feature = feature_idx
                best_threshold = threshold

    best_feature_values = X_train[:, best_feature]
    predictions = (best_feature_values >= best_threshold).astype(int)

    return predictions, best_feature, best_threshold


y_train_pred, best_feature, best_threshold = oneR_classifier(X_train, y_train)
best_feature_values_val = X_val[:, best_feature]
y_val_pred = (best_feature_values_val >= best_threshold).astype(int)


accuracy_val = accuracy(y_val, y_val_pred)
conf_matrix_val = confusion_matrix_manual(y_val, y_val_pred)
precision_val = precision_manual(y_val, y_val_pred)
recall_val = recall_manual(y_val, y_val_pred)
f1_val = f1_score_manual(y_val, y_val_pred)

print("\nValidation Set Metrics:")
print(f"Accuracy: {accuracy_val}")
print(f"Confusion Matrix (TP, TN, FP, FN): {conf_matrix_val}")
print(f"Precision: {precision_val}")
print(f"Recall: {recall_val}")
print(f"F1-Score: {f1_val}")

y_val_prob = y_val_pred
precision_vals, recall_vals, _ = precision_recall_curve_manual(y_val, y_val_prob)
plot_precision_recall_curve(precision_vals, recall_vals, label="OneR Classifier (Validation)")

best_feature_values_test = X_test[:, best_feature]
y_test_pred = (best_feature_values_test >= best_threshold).astype(int)

accuracy_test = accuracy(y_test, y_test_pred)
conf_matrix_test = confusion_matrix_manual(y_test, y_test_pred)
precision_test = precision_manual(y_test, y_test_pred)
recall_test = recall_manual(y_test, y_test_pred)
f1_test = f1_score_manual(y_test, y_test_pred)

print("\nTest Set Metrics:")
print(f"Accuracy: {accuracy_test}")
print(f"Confusion Matrix (TP, TN, FP, FN): {conf_matrix_test}")
print(f"Precision: {precision_test}")
print(f"Recall: {recall_test}")
print(f"F1-Score: {f1_test}")


conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[1, 2, 3], yticklabels=[1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

```

```
y_test_prob = y_test_pred
precision_vals, recall_test, _ = precision_recall_curve_manual(y_test, y_test_prob)
plot_precision_recall_curve(precision_vals, recall_test, label="OneR Classifier (Test)")

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)


knn.fit(X_train, y_train)
y_val_pred = knn.predict(X_val)

accuracy_val = accuracy(y_val, y_val_pred)
precision_val = precision_manual(y_val, y_val_pred, average='weighted')
recall_val = recall_manual(y_val, y_val_pred, average='weighted')
f1_val = f1_score_manual(y_val, y_val_pred, average='weighted')


print(f"Validation Accuracy: {accuracy_val}")
print(f"Validation Precision: {precision_val}")
print(f"Validation Recall: {recall_val}")
print(f"Validation F1 Score: {f1_val}")
y_test_pred = knn.predict(X_test)
accuracy_test = accuracy(y_test, y_test_pred)
precision_test = precision_manual(y_test, y_test_pred, average='weighted')
recall_test = recall_manual(y_test, y_test_pred, average='weighted')
f1_test = f1_score_manual(y_test, y_test_pred, average='weighted')
print(f"Test Accuracy: {accuracy_test}")
print(f"Test Precision: {precision_test}")
print(f"Test Recall: {recall_test}")
print(f"Test F1 Score: {f1_test}")


conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)


plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[1, 2, 3], yticklabels
    =[1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


from sklearn.metrics import roc_curve, auc


fpr, tpr, thresholds = roc_curve(y_test, knn.predict_proba(X_test)[:, 1], pos_label=1)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC curve (area = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()


param_grid = {
```

```
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Parameters from Grid Search:")
print(grid_search.best_params_)
print(f"Best Accuracy from Grid Search: {grid_search.best_score_}")

best_knn = grid_search.best_estimator_

y_test_pred_best = best_knn.predict(X_test)

accuracy_best = accuracy(y_test, y_test_pred_best)
precision_best = precision_manual(y_test, y_test_pred_best, average='weighted')
recall_best = recall_manual(y_test, y_test_pred_best, average='weighted')
f1_best = f1_score_manual(y_test, y_test_pred_best, average='weighted')

print(f"Test Accuracy (Best K-NN): {accuracy_best}")
print(f"Test Precision (Best K-NN): {precision_best}")
print(f"Test Recall (Best K-NN): {recall_best}")
print(f"Test F1 Score (Best K-NN): {f1_best}")


k_values = [3, 5, 7, 9, 11]
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracies.append(accuracy(y_test, y_pred))

plt.figure(figsize=(8, 6))
plt.plot(k_values, accuracies, marker='o', color='green')
plt.title('Accuracy vs. Number of Neighbors (K)')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

k_values = [3, 5, 7, 9, 11]
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_val_pred = knn.predict(X_val)
    accuracies.append(accuracy(y_val, y_val_pred))

plt.figure(figsize=(8, 6))
plt.plot(k_values, accuracies, marker='o', color='green')
plt.title('Accuracy vs. Number of Neighbors (K)')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```

```python
from sklearn.naive_bayes import GaussianNB


nb = GaussianNB()
nb.fit(X_train, y_train)
y_val_pred = nb.predict(X_val)
y_test_pred = nb.predict(X_test)
y_test_pred = nb.predict(X_test)


accuracy_val = accuracy(y_val, y_val_pred)
precision_val = precision_manual(y_val, y_val_pred, average='weighted')
recall_val = recall_manual(y_val, y_val_pred, average='weighted')
f1_val = f1_score_manual(y_val, y_val_pred, average='weighted')



print(f"Validation Accuracy: {accuracy_val}")
print(f"Validation Precision: {precision_val}")
print(f"Validation Recall: {recall_val}")
print(f"Validation F1 Score: {f1_val}")


accuracy_test = accuracy(y_test, y_test_pred)
precision_test = precision_manual(y_test, y_test_pred, average='weighted')
recall_test = recall_manual(y_test, y_test_pred, average='weighted')
f1_test = f1_score_manual(y_test, y_test_pred, average='weighted')


print(f"Test Accuracy: {accuracy_test}")
print(f"Test Precision: {precision_test}")
print(f"Test Recall: {recall_test}")
print(f"Test F1 Score: {f1_test}")

conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(conf_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=[1, 2, 3], yticklabels=[1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()



accuracy_test = accuracy(y_test, y_test_pred)
precision_test = precision_manual(y_test, y_test_pred, average='weighted')
recall_test = recall_manual(y_test, y_test_pred, average='weighted')
f1_test = f1_score_manual(y_test, y_test_pred, average='weighted')

print(f"Test Accuracy (Naive Bayes): {accuracy_test}")
print(f"Test Precision (Naive Bayes): {precision_test}")
print(f"Test Recall (Naive Bayes): {recall_test}")
print(f"Test F1 Score (Naive Bayes): {f1_test}")



svm.fit(X_train, y_train)
y_test_pred_svm = svm.predict(X_test)

```

```python



accuracy_test_svm = accuracy(y_test, y_test_pred_svm)
precision_test_svm = precision_manual(y_test, y_test_pred_svm, average='weighted')
recall_test_svm = recall_manual(y_test, y_test_pred_svm, average='weighted')
f1_test_svm = f1_score_manual(y_test, y_test_pred_svm, average='weighted')

print(f"Test Accuracy (SVM): {accuracy_test_svm}")
print(f"Test Precision (SVM): {precision_test_svm}")
print(f"Test Recall (SVM): {recall_test_svm}")
print(f"Test F1 Score (SVM): {f1_test_svm}")



conf_matrix_svm = confusion_matrix(y_test, y_test_pred_svm)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_svm, annot=True, fmt="d", cmap="Blues", xticklabels=[1, 2, 3],
    yticklabels=[1, 2, 3])
plt.title("SVM Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


y_test_bin = label_binarize(y_test, classes=[1, 2, 3])
y_score_svm = svm.decision_function(X_test)

print(f"Shape of y_score_svm: {y_score_svm.shape}")
print(f"Shape of y_test_bin: {y_test_bin.shape}")



y_score_svm = svm.predict_proba(X_test)[:, 1]

precision, recall, _ = precision_recall_curve(y_test, y_score_svm)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', label='SVM')
plt.title('Precision-Recall Curve for SVM (Binary Classification)')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='best')
plt.show()


y_score_svm = svm.predict_proba(X_test)[:, 1]

precision, recall, _ = precision_recall_curve(y_test, y_score_svm)

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', label='SVM')
plt.title('Precision-Recall Curve for SVM (Binary Classification)')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='best')
plt.show()
```

B. Car Dataset

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.preprocessing import label_binarize
import sklearn
import warnings
%matplotlib inline
sns.set()
warnings.filterwarnings('ignore')
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/car.data")
df.head()
df.columns = ["buying", "maint", "doors", "persons", "lug_boot", "safety", "class values"]

df['doors'] = df['doors'].replace(['5more'], '5')
df['persons'] = df['persons'].replace(['more'], '4')
for col in ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']:
    df[col] = df[col].astype('category')

for col in ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']:
    df[col] = df[col].cat.codes
df = pd.get_dummies(df, columns=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])

df.head()

df['class values'].unique()
df = df.drop(['buying_3', 'maint_3', 'doors_3', 'persons_1', 'lug_boot_2', 'safety_2'], axis=1)
df.head()
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['class values'] = labelencoder.fit_transform(df['class values'])
x = df.drop(["class values"], axis=1)
y = df["class values"]
x.head()
y.unique()
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=42)

class OneRClassifier:
    def __init__(self):
        self.rule = {}
        self.best_feature = None

    def fit(self, X, y):
        best_error = float('inf')

        for feature in X.columns:
            feature_rule = {}
            grouped = pd.DataFrame({'feature': X[feature], 'target': y}).groupby('feature')['
    target'].agg(lambda x: x.value_counts().idxmax())
            feature_rule = grouped.to_dict()

            y_pred = X[feature].map(feature_rule)
            error = sum(y_pred != y)

            if error < best_error:
                best_error = error
                self.rule = feature_rule
                self.best_feature = feature

    def predict(self, X):
```

```
        if not self.best_feature:
            raise ValueError("The model has not been trained yet!")
        return X[self.best_feature].map(self.rule)

def calculate_confusion_matrix(y_true, y_pred, num_classes):
    cm = np.zeros((num_classes, num_classes), dtype=int)
    for true, pred in zip(y_true, y_pred):
        cm[true][pred] += 1
    return cm

def calculate_metrics(y_true, y_pred):
    num_classes = len(np.unique(y_true))
    cm = calculate_confusion_matrix(y_true, y_pred, num_classes)
    total = cm.sum()
    accuracy = np.trace(cm) / total

    precisions = []
    recalls = []
    f1_scores = []
    for i in range(num_classes):
        tp = cm[i, i]
        fp = sum(cm[:, i]) - tp
        fn = sum(cm[i, :]) - tp
        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

        precisions.append(precision)
        recalls.append(recall)
        f1_scores.append(f1)

    avg_precision = np.mean(precisions)
    avg_recall = np.mean(recalls)
    avg_f1 = np.mean(f1_scores)

    return {
        'accuracy': accuracy,
        'confusion_matrix': cm,
        'average_precision': avg_precision,
        'average_recall': avg_recall,
        'average_f1': avg_f1,
    }

model = OneRClassifier()
model.fit(x_train, y_train)

y_train_pred = model.predict(x_train)
y_val_pred = model.predict(x_val)
y_test_pred = model.predict(x_test)

metrics = calculate_metrics(y_val.to_numpy(), y_val_pred.to_numpy())
metrics2 = calculate_metrics(y_test.to_numpy(), y_test_pred.to_numpy())

print("OneR Classifier Validation Metrics:")
print(f"Accuracy: {metrics['accuracy']:.2f}")
print("Confusion Matrix (Custom):")
print(metrics['confusion_matrix'])
print(f"Average Precision: {metrics['average_precision']:.2f}")
print(f"Average Recall: {metrics['average_recall']:.2f}")
print(f"Average F1-Score: {metrics['average_f1']:.2f}")
def plot_confusion_matrix(cm, class_names):
    plt.figure(figsize=(8, 6))
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=class_names, yticklabels=
    class_names)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Class")
    plt.ylabel("True Class")
    plt.show()

class_names = labelencoder.inverse_transform(range(len(np.unique(y_train))))

plot_confusion_matrix(metrics['confusion_matrix'], class_names)
print("OneR Classifier test Metrics:")
print(f"Accuracy: {metrics2['accuracy']:.2f}")
print("Confusion Matrix (Custom):")
print(metrics2['confusion_matrix'])
print(f"Average Precision: {metrics2['average_precision']:.2f}")
print(f"Average Recall: {metrics2['average_recall']:.2f}")
print(f"Average F1-Score: {metrics2['average_f1']:.2f}")
def plot_confusion_matrix(cm, class_names):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=class_names, yticklabels=
    class_names)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Class")
    plt.ylabel("True Class")
    plt.show()

class_names = labelencoder.inverse_transform(range(len(np.unique(y_train))))

plot_confusion_matrix(metrics2['confusion_matrix'], class_names)
majority_class = y_train.mode()[0]
y_val_pred_zeror = [majority_class] * len(y_val)
y_test_pred_zeror = [majority_class] * len(y_test)
def accuracy_score(y_true, y_pred):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    corr_pred = np.sum(y_true == y_pred)

    total_pred = len(y_true)

    accuracy = corr_pred / total_pred

    return accuracy
def confusion_matrix(y_true, y_pred, labels=None):
    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    if labels is None:
        labels = np.unique(np.concatenate([y_true, y_pred]))

    cm = np.zeros((len(labels), len(labels)), dtype=int)

    label_to_index = {label: idx for idx, label in enumerate(labels)}

    for true_label, pred_label in zip(y_true, y_pred):
        true_idx = label_to_index[true_label]
        pred_idx = label_to_index[pred_label]
        cm[true_idx, pred_idx] += 1

    return cm

def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(8, 6))
```

```
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()

    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)
    for i in range(len(labels)):
        for j in range(len(labels)):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > cm.max() / 2 else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
def precision_recall_f1(cm):
    epsilon = 1e-10
    precision = np.diag(cm) / (np.sum(cm, axis=0) + epsilon)
    recall = np.diag(cm) / (np.sum(cm, axis=1) + epsilon)
    f1 = 2 * (precision * recall) / (precision + recall + epsilon)

    accuracy = np.sum(np.diag(cm)) / np.sum(cm)
    f1 = np.nan_to_num(f1)

    return precision, recall, f1, accuracy


def classification_report(y_true, y_pred, labels=None):
    if labels is None:
        labels = np.unique(np.concatenate([y_true, y_pred]))

    cm = confusion_matrix(y_true, y_pred, labels)

    precision, recall, f1, accuracy = precision_recall_f1(cm)

    report = "               precision    recall  f1-score   support\n\n"

    for idx, label in enumerate(labels):
        report += f"{label:>10}       {precision[idx]:.2f}      {recall[idx]:.2f}      {f1[idx]:.2f}       {np.sum(cm[idx]):>3}\n"

    macro_avg_precision = np.mean(precision)
    macro_avg_recall = np.mean(recall)
    macro_avg_f1 = np.mean(f1)
    report += f"\n    accuracy                           {accuracy:.2f}      {len(y_true)}\n"
    report += f"   macro avg       {macro_avg_precision:.2f}      {macro_avg_recall:.2f}      {macro_avg_f1:.2f}      {len(y_true)}\n"

    return report
val_accuracy = accuracy_score(y_val, y_val_pred_zeror)
test_accuracy = accuracy_score(y_test, y_test_pred_zeror)

print(f"ZeroR Validation Accuracy (Custom): {val_accuracy:.2f}")
print(f"ZeroR Test Accuracy (Custom): {test_accuracy:.2f}")
labels = np.unique(y_test)
cm = confusion_matrix(y_test, y_test_pred_zeror, labels=labels)

report = classification_report(y_test, y_test_pred_zeror, labels=labels)
print("\nCustom Classification Report (Test Data):")
print(report)
labels = np.unique(y_val)
```

```python
cm = confusion_matrix(y_val, y_val_pred_zeror, labels=labels)

print("Confusion Matrix:")
print(cm)

plot_confusion_matrix(cm, labels=labels)
labels = np.unique(y_test)
cm = confusion_matrix(y_test, y_test_pred_zeror, labels=labels)

print("Confusion Matrix:")
print(cm)

plot_confusion_matrix(cm, labels=labels)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import  PrecisionRecallDisplay
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 21), 'weights': ['uniform', 'distance'], 'metric': ['
    euclidean', 'manhattan']}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_knn = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

y_test_pred = best_knn.predict(x_test)
y_val_pred = best_knn.predict(x_val)

val_accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy:", val_accuracy)

test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)
cm1 = confusion_matrix(y_val, y_val_pred)
print("Validation Confusion Matrix:")
print(cm1)

cm2 = confusion_matrix(y_test, y_test_pred)
print("Test Confusion Matrix:")
print(cm2)

def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()

    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)

    for i in range(len(labels)):
        for j in range(len(labels)):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > cm.max() / 2 else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
plot_confusion_matrix(cm1, labels=labelencoder.classes_)
```

```python

plot_confusion_matrix(cm2, labels=labelencoder.classes_)

val_report = classification_report(y_val, y_val_pred)
print("Validation Classification Report:")
print(val_report)

report = classification_report(y_test, y_test_pred)
print("Test Classification Report:")
print(report)
y_val_pred_prob = knn.predict_proba(x_val)

for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_val == i, y_val_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
    plt.title(f'Validation Precision-Recall Curve for class {labelencoder.classes_[i]}')

plt.show()
y_test_pred_prob = knn.predict_proba(x_test)

for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_test == i, y_test_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
    plt.title(f'Test Precision-Recall Curve for class {labelencoder.classes_[i]}')

plt.show()
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import PrecisionRecallDisplay
y_test_pred = nb.predict(x_test)
y_val_pred = nb.predict(x_val)
accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", accuracy)
accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy:", accuracy)
cm1 = confusion_matrix(y_test, y_test_pred)
print("Test Confusion Matrix:")
print(cm1)

cm2 = confusion_matrix(y_val, y_val_pred)
print("Validation Confusion Matrix:")
print(cm2)
def plot_confusion_matrix(cm, labels):
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()

    tick_marks = np.arange(len(labels))
    plt.xticks(tick_marks, labels, rotation=45)
    plt.yticks(tick_marks, labels)

    for i in range(len(labels)):
        for j in range(len(labels)):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > cm.max() / 2 else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

```
plot_confusion_matrix(cm1, labels=labelencoder.classes_)
plot_confusion_matrix(cm2, labels=labelencoder.classes_)
test_report = classification_report(y_test, y_test_pred)
print("Test Classification Report:")
print(test_report)
val_report = classification_report(y_val, y_val_pred)
print("Validation Classification Report:")
print(val_report)
y_test_pred_prob = nb.predict_proba(x_test)

for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_test == i, y_test_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
    plt.title(f'Precision-Recall Curve for class {labelencoder.classes_[i]}')

plt.show()

y_val_pred_prob = nb.predict_proba(x_val)

for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_val == i, y_val_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
    plt.title(f'Precision-Recall Curve for class {labelencoder.classes_[i]}')

plt.show()

from sklearn.svm import SVC
from sklearn.metrics import PrecisionRecallDisplay


# svm = SVC(kernel='linear', probability=True)
# svm.fit(x_train, y_train)
from sklearn.model_selection import GridSearchCV
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': [0.001, 0.01, 0.1]
}
grid_search = GridSearchCV(SVC(probability=True), param_grid, cv=5)
grid_search.fit(x_train, y_train)
best_model = grid_search.best_estimator_
# y_val_pred = svm.predict(x_val)
y_val_pred = best_model.predict(x_val)

val_accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy:", val_accuracy)

val_cm = confusion_matrix(y_val, y_val_pred)
print("Validation Confusion Matrix:")
print(val_cm)

plot_confusion_matrix(val_cm, labels=labelencoder.classes_)

val_report = classification_report(y_val, y_val_pred)
print("Validation Classification Report:")
print(val_report)
y_val_pred_prob = best_model.predict_proba(x_val)
for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_val == i, y_val_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
```

```
    plt.title(f'Precision-Recall Curve for class {labelencoder.classes_[i]} (Validation)')

plt.show()

# y_test_pred = svm.predict(x_test)
y_test_pred = best_model.predict(x_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print("Test Accuracy:", test_accuracy)
test_cm = confusion_matrix(y_test, y_test_pred)
print("Test Confusion Matrix:")
print(test_cm)

plot_confusion_matrix(test_cm, labels=labelencoder.classes_)

test_report = classification_report(y_test, y_test_pred)
print("Test Classification Report:")
print(test_report)

y_test_pred_prob = best_model.predict_proba(x_test)
for i in range(len(labelencoder.classes_)):
    precision, recall, _ = precision_recall_curve(y_test == i, y_test_pred_prob[:, i])
    PrecisionRecallDisplay(precision=precision, recall=recall, average_precision=np.mean(
    precision)).plot()
    plt.title(f'Precision-Recall Curve for class {labelencoder.classes_[i]} (Test)')

plt.show()
```

C. Wage Dataset

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DT-Wage.csv')
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DT-Wage.csv')

df.shape

df.info()
un = df['maritl'].unique()
print(un)

un = df['race'].unique()
print(un)

un = df['education'].unique()
print(un)

un = df['region'].unique()
print(un)

un = df['jobclass'].unique()
print(un)

un = df['health'].unique()
```

```python
print(un)

categorical = df.select_dtypes(include=['object']).columns
numerical = df.select_dtypes(exclude=['object']).columns

encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical))
encoded_df.index = df.index

df = df.drop(categorical, axis=1)
df = pd.concat([df, encoded_df], axis=1)

X = df.drop(columns=['wage'])
y = df['wage']
X.head()

X.shape

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
svr = SVR()

param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf'],
    'degree': [3, 4, 5],
    'gamma': ['scale', 'auto'],
    'epsilon': [0.1, 0.2, 0.5],
}

grid_search = GridSearchCV(svr, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)

best_svr = grid_search.best_estimator_
y_pred_val = best_svr.predict(X_val)
def custom_mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def custom_r2(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total)

mse = custom_mse(y_val, y_pred_val)
r2 = custom_r2(y_val, y_pred_val)
print(f"Validation MSE: {mse}")
print(f"Validation R^2: {r2}")


y_pred_test = best_svr.predict(X_test)
test_mse = custom_mse(y_test, y_pred_test)
test_r2 = custom_r2(y_test, y_pred_test)
print(f"Test MSE: {test_mse}")
```

```
print(f"Test R^2: {test_r2}")

plt.figure(figsize=(10, 6))
plt.scatter(y_val, y_pred_val, color='blue', alpha=0.5)
plt.plot([min(y_val), max(y_val)], [min(y_val), max(y_val)], color='red', linestyle='--')
plt.title("SVR: Actual vs Predicted (Validation Set)")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.show()

residuals = y_val - y_pred_val
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_val, residuals, color='green', alpha=0.5)
plt.hlines(0, xmin=min(y_pred_val), xmax=max(y_pred_val), colors='red', linestyle='--')
plt.title("Residuals Plot (Validation Set)")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_test, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.title("SVR: Actual vs Predicted (Test Set)")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.show()

residuals_test = y_test - y_pred_test
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_test, residuals_test, color='green', alpha=0.5)
plt.hlines(0, xmin=min(y_pred_test), xmax=max(y_pred_test), colors='red', linestyle='--')
plt.title("Residuals Plot (Test Set)")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.show()
```

D. Credit Dataset

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DT-Credit.csv')
df.head()
df.shape
df.describe()
df.isnull().sum()
un = df['Own'].unique()
print(un)

un = df['Student'].unique()
print(un)

un = df['Region'].unique()
print(un)

```

```
un = df['Married'].unique()
print(un)

categorical = df.select_dtypes(include=['object']).columns
numerical = df.select_dtypes(exclude=['object']).columns

encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_data = encoder.fit_transform(df[categorical])
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical))
encoded_df.index = df.index

df = df.drop(categorical, axis=1)
df = pd.concat([df, encoded_df], axis=1)
X = df.drop(columns=['Balance'])
y = df['Balance']
X.head()

X.shape

corr = df.corr()
target = corr['Balance'].sort_values(ascending=False)
print("Correlation of each feature with the target variable (Balance):")
print(target)

fig, ax = plt.subplots(1,1, figsize=(12, 12))
df.boxplot('Income', 'Balance', ax=ax)
plt.suptitle('Income vs Balance')
plt.title('')
plt.ylabel('Income')
plt.xticks(rotation=90)
plt.show()

fig, ax = plt.subplots(1,1, figsize=(12, 12))
df.boxplot('Age', 'Balance', ax=ax)
plt.suptitle('Age vs Balance')
plt.title('')
plt.ylabel('Age')
plt.xticks(rotation=90)
plt.show()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state
    =42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

X_train.shape
X_val.shape
X_test.shape

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

param_grid = {
    'C': [0.1, 1, 10, 100,200,300,500,1500,2000],
    'epsilon': [0.01, 0.1, 0.2,0.25,0.3],
    'gamma': ['scale', 'auto']
}

svr = SVR(kernel='rbf')

grid_search = GridSearchCV(estimator=svr, param_grid=param_grid,
```

```
                           cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best hyperparameters found by GridSearchCV:")
print(best_params)

best_svr = grid_search.best_estimator_
y_pred_train = best_svr.predict(X_train)
y_pred_val = best_svr.predict(X_val)
y_pred_test = best_svr.predict(X_test)
def custom_mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def custom_r2(y_true, y_pred):
    ss_total = np.sum((y_true - np.mean(y_true)) ** 2)
    ss_residual = np.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total)

mse_train = custom_mse(y_train, y_pred_train)
mse_val = custom_mse(y_val, y_pred_val)
mse_test = custom_mse(y_test, y_pred_test)

r2_train = custom_r2(y_train, y_pred_train)
r2_val = custom_r2(y_val, y_pred_val)
r2_test = custom_r2(y_test, y_pred_test)

print("SVR Model with GridSearchCV Evaluation:")
print(f"Training MSE: {mse_train}")
print(f"Validation MSE: {mse_val}")
print(f"Test MSE: {mse_test}")
print(f"Training  R  : {r2_train}")
print(f"Validation  R  : {r2_val}")
print(f"Test  R  : {r2_test}")


plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.scatter(y_train, y_pred_train, color='blue', alpha=0.5)
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], color='red', lw=2)
plt.title('Training Set: Predicted vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 2)
plt.scatter(y_val, y_pred_val, color='green', alpha=0.5)
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], color='red', lw=2)
plt.title('Validation Set: Predicted vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 3)
plt.scatter(y_test, y_pred_test, color='orange', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', lw=2)
plt.title('Test Set: Predicted vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted')
```

```
plt.tight_layout()
plt.show()
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.scatter(y_pred_train, y_pred_train - y_train, color='blue', alpha=0.5)
plt.hlines(y=0, xmin=y_pred_train.min(), xmax=y_pred_train.max(), color='red', lw=2)
plt.title('Training Set: Residuals')
plt.xlabel('Predicted')
plt.ylabel('Residuals (Error)')

plt.tight_layout()
plt.show()
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 2)
plt.scatter(y_pred_val, y_pred_val - y_val, color='green', alpha=0.5)
plt.hlines(y=0, xmin=y_pred_val.min(), xmax=y_pred_val.max(), color='red', lw=2)
plt.title('Validation Set: Residuals')
plt.xlabel('Predicted')
plt.ylabel('Residuals (Error)')

plt.tight_layout()
plt.show()
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 3)
plt.scatter(y_pred_test, y_pred_test - y_test, color='orange', alpha=0.5)
plt.hlines(y=0, xmin=y_pred_test.min(), xmax=y_pred_test.max(), color='red', lw=2)
plt.title('Test Set: Residuals')
plt.xlabel('Predicted')
plt.ylabel('Residuals (Error)')

plt.tight_layout()
plt.show()
```