Predicting IMdb Scores

Phase4 document submission

Name:Dhinisha Mol.T

Reg.No:961521104008

OBJECTIVES:

I can’t predict the exact IMDb score for a movie that hasn’t been released or rated yet. IMDb scores are typically determined by user ratings and reviews, so they can vary over time. If you have a specific movie in mind or some data related to it, I can discuss factors that might influence its IMDb score. Import pandas as pd matplotlib.pyplot as plt Import seaborn as sns

Import plotly.express as px

Certainly! It looks like you’re starting to use some popular Python libraries for data analysis and visualization.

Pandas is a powerful library for data manipulation and analysis.It provides data structures like DataFrames to work with structured data efficiently.matplotlib is a widely used data visualization library in Python.pyplot is a module within matplotlib that provides a simple and high-level interface for creating various types of plots and charts.

1.Reading the Data

movie_df=pd.read_csv("/home/leon/Documents/2023_Oct_mariaCollege/IMDP_scores_GradientBoosting/movie_datas et/movie_metadata.csv") movie_df.head() **2.Feature scaling** from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)

X_test = sc_X.transform(X_test)

StandardScaler is a preprocessing technique commonly used in machine learning to scale or standardize the features of a dataset. Scaling features is important because many machine learning algorithms are sensitive to the scale of the input features. Standardization makes the mean of each feature 0 and scales the variance to 1. This is also known as z-score normalization.

Here's how we can use StandardScaler with the **fit_transform** method in Python

3. Classification Model Selection¶

3.1 Logistic Regression

from sklearn.linear_model import LogisticRegression logit
=LogisticRegression()
logit.fit(X_train,np.ravel(y_train,order='C'))
y_pred=logit.predict(X_test)

LogisticRegression and logit.predict are components of logistic regression, a popular classification algorithm in machine learning used for binary and multiclass classification problems.

LogisticRegression is a class in various machine learning libraries (e.g., scikit-learn in Python) that is used to create and train logistic regression models. Logistic regression is used when the dependent variable (the target) is categorical, typically with two categories (binary classification), but it can also be extended to handle multiple categories (multiclass classification). logit is a variable that will store the logistic regression model. You can name it anything you like; logit is a common choice.

#Confusion matrix for logistic regression** from sklearn

import metrics cnf_matrix =

metrics.confusion_matrix(y_test, y_pred)

print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

The code you provided is for calculating and printing a confusion matrix and accuracy score for a logistic regression model's predictions. These are common tools for evaluating the performance of a binary classification model.

Import the necessary libraries: This line imports the `metrics` module from scikit-learn, which contains functions for evaluating machine learning models.

Calculate the confusion matrix: The confusion matrix is a table that is often used to describe the performance of a classification model. It contains information about the true positive, true negative, false positive, and false negative predictions made by the model.

Print the confusion matrix: This line prints the calculated confusion matrix to the console, allowing you to see the counts of true positives, true negatives, false positives, and false negatives.

Calculate and print the accuracy: `metrics.accuracy_score(y_test, y_pred)`: This function calculates the accuracy of the model by comparing the true labels (`y_test`) with the predicted labels (`y_pred`). It measures the proportion of correctly classified data points.

The confusion matrix and accuracy score provide valuable insights into how well your logistic regression model is performing. The confusion matrix breaks down the types of classification errors, while the accuracy score gives you an overall measure of the model's correctness in its predictions.

4. KNN

#KNN

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=22)

knn.fit(X_train, np.ravel(y_train,order='C')) knnpred =

knn.predict(X_test) cnf_matrix =

metrics.confusion_matrix(y_test, knnpred)

print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test,

knnpred))

The code you provided is for implementing the k-Nearest Neighbors (KNN) algorithm for classification and evaluating its performance. KNN is a simple and effective algorithm for both binary and multiclass classification. The code you've provided appears to be using the k-Nearest Neighbors (KNN) algorithm for classification, likely in a machine learning context using Python's scikit-learn library. Here's a breakdown of what this code does:

Import the necessary librarie:

Import the KNeighborsClassifier from sklearn.neighbors to create and train a KNN classifier.Import the necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a KNN classifier: Initialize a KNeighborsClassifier with n_neighbors set to 22. This means that when making predictions, the algorithm will consider the 22 nearest neighbors to the data point to classify it.

Train the KNN classifier:

Fit the KNN model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained KNN classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (knnpred). The accuracy is a measure of how many of the test samples were correctly classified by the model.Overall, this code trains a KNN classifier with 22 neighbors, makes predictions on a test dataset, and then evaluates the model's performance by printing the confusion matrix and accuracy. This allows you to assess how well the KNN model is performing on your classification task.

5. SVC

```
#SVC

from sklearn.svm import SVC svc= SVC(kernel = 'sigmoid')

svc.fit(X_train, np.ravel(y_train,order='C')) svcpred =

svc.predict(X_test) cnf_matrix =

metrics.confusion_matrix(y_test, svcpred)

print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test,

svcpred))
```

The code you provided is for a Support Vector Machine (SVM) classification model using the scikit-learn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the Support Vector Classification (SVC) class from sklearn.svm.Import other necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create an SVC classifier:

Initialize an SVC classifier with the kernel set to 'sigmoid'. The 'sigmoid' kernel is one of the available kernel functions in SVM, and it is used to map data into a higher-dimensional space for classification.

Train the SVC classifier:

Fit the SVC model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained SVM classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (svcpred). The accuracy is a measure of how many of the test samples were correctly classified by the model.

This code trains an SVM classifier with a 'sigmoid' kernel, makes predictions on a test dataset, and evaluates the model's performance by printing the confusion matrix and accuracy. The choice of kernel function, such as 'sigmoid', can significantly impact the performance of the SVM classifier, and the selection should be based on the specific characteristics of your dataset and problem.

5. Random Forest

#Random Forest from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 200)#criterion = entopy,gini

rfc.fit(X_train, np.ravel(y_train,order='C')) rfcpred = rfc.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, rfcpred) print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, rfcpred))

data (X_train and y_train).X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained Random Forest classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix: The code you provided is for a Random Forest classifier using the scikitlearn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the RandomForestClassifier class from sklearn.ensemble.Import other necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Random Forest classifier:

Initialize a RandomForestClassifier with n_estimators set to 200. This means that the random forest will consist of 200 decision tree classifiers. Additionally, you can specify the criterion parameter to define the impurity criterion used for splitting nodes in the decision trees. Common options are 'gini' and 'entropy'. If not specified, 'gini' is the default criterion.

Train the Random Forest classifier:

Fit the Random Forest model using training

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (rfcpred). The accuracy is a measure of how many of the test samples were correctly classified by the model.

Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve overall classification performance. In this code, a Random Forest classifier with 200 decision trees is trained and evaluated for accuracy and confusion matrix on a test dataset. The choice of n_estimators and criterion can be finetuned based on your specific problem and dataset.

6. Gradient Boosting

```
#Gradient boosting from sklearn.ensemble import GradientBoostingClassifier gbcl =

GradientBoostingClassifier(n_estimators = 50, learning_rate = 0.09, max_depth=5) gbcl

= gbcl.fit(X_train,np.ravel(y_train,order='C')) test_pred = gbcl.predict(X_test) cnf_matrix

= metrics.confusion_matrix(y_test, test_pred) print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, test_pred))
```

The code you provided is for a Gradient Boosting classifier using the scikit-learn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the GradientBoostingClassifier class from sklearn.ensemble.Import other necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Gradient Boosting classifier:

Initialize a GradientBoostingClassifier with the following hyperparameters:n_estimators is set to 50, which specifies the number of boosting stages or weak learners in the ensemble.learning_rate is set to 0.09, which controls the contribution of each weak learner to the final prediction.max_depth is set to 5, which limits the maximum depth of the individual decision trees in the ensemble.

Train the Gradient Boosting classifier:

Fit the Gradient Boosting model using training data (X_train and y_train).X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained Gradient Boosting classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (test_pred). The accuracy is a measure of how many of the test samples were correctly classified by the model.

Gradient Boosting is an ensemble learning technique that combines the predictions of multiple weak learners (typically decision trees) in a sequential manner to improve classification performance. In this code, a Gradient Boosting classifier with specific hyperparameters is trained and evaluated for accuracy and confusion matrix on a test dataset. The choice of hyperparameters, such as n_estimators, learning_rate, and max_depth, can be tuned to achieve the best performance for your specific problem and dataset.

7. Decision Tree

```
#Decision Tree from sklearn.tree import DecisionTreeClassifier dtree

= DecisionTreeClassifier(criterion='gini') #criterion = entopy, gini

dtree.fit(X_train, np.ravel(y_train,order='C')) dtreepred =

dtree.predict(X_test) cnf_matrix = metrics.confusion_matrix(y_test,

dtreepred) print(cnf_matrix)

print("Accuracy:",metrics.accuracy_score(y_test, dtreepred))
```

The code you provided is for a Decision Tree classifier using the scikit-learn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the DecisionTreeClassifier class from sklearn.tree.Import other necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Decision Tree classifier:

Initialize a DecisionTreeClassifier with the criterion set to 'gini'. The criterion parameter specifies the impurity criterion used for splitting nodes in the decision tree. Common options are 'gini' and 'entropy'.

Train the Decision Tree classifier:

Fit the Decision Tree model using training data (X_train and y_train).

X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained Decision Tree classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (dtreepred). The accuracy is a measure of how many of the test samples were correctly classified by the model.

Decision Trees are a simple and interpretable machine learning model that can be used for both classification and regression tasks. In this code, a Decision Tree classifier with a 'gini' impurity criterion is trained and evaluated for accuracy and confusion matrix on a test dataset. The choice of the impurity criterion and other hyperparameters can be adjusted based on your specific problem and dataset.

8. Naive Bayes

```
#Naive bayes from sklearn.naive_bayes import GaussianNB

gaussiannb= GaussianNB() gaussiannb.fit(X_train,

np.ravel(y_train,order='C')) gaussiannbpred =

gaussiannb.predict(X_test) cnf_matrix =

metrics.confusion_matrix(y_test, gaussiannbpred)

print(cnf_matrix) print("Accuracy:",metrics.accuracy_score(y_test,

gaussiannbpred))
```

The code you provided is for a Naive Bayes classifier, specifically the Gaussian Naive Bayes classifier, using the scikit-learn library in Python. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the GaussianNB class from sklearn.naive_bayes.Import other necessary functions and classes for evaluating the model, such as confusion_matrix and accuracy_score from sklearn.metrics.

Create a Gaussian Naive Bayes classifier:

Initialize a GaussianNB classifier.

Train the Gaussian Naive Bayes classifier:

Fit the Gaussian Naive Bayes model using training data (X_train and y_train).X_train is assumed to be the feature data for training, and y_train is the corresponding target data.

Make predictions:

Use the trained Gaussian Naive Bayes classifier to predict the classes for the test data (X_test).

Calculate and print the confusion matrix:

The confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the number of true positives, true negatives, false positives, and false negatives. The metrics.confusion_matrix function is used to calculate this matrix and store it in the cnf_matrix variable.

Calculate and print the accuracy:

The accuracy of the model is calculated using the metrics.accuracy_score function, which compares the true labels (y_test) and the predicted labels (gaussiannbpred). The accuracy is a measure of how many of the test samples were correctly classified by the model.

Gaussian Naive Bayes is a probabilistic classification algorithm that is particularly suited for datasets where the features are continuous and assumed to be normally distributed. In this code, a Gaussian Naive Bayes classifier is trained and evaluated for accuracy and a confusion matrix on a test dataset. It's a simple and efficient classification algorithm, but its performance may vary depending on the characteristics of your data.

9. Bagging Classifier

new_movie_df=movie_df.pop("imdb_binned_score")

The line of code you've provided seems to be attempting to remove the "imdb_binned_score" column from a DataFrame called movie_df and assign the removed column to a new variable new_movie_df. However, it's not quite right.To remove the column from a DataFrame and assign it to a new variable, you should use the pop method in a different way.

This code will remove the "imdb_binned_score" column from the movie_df DataFrame and assign it to the variable new_movie_df. After this operation, new_movie_df will contain the column you removed, and movie_df will no longer have that column #Bagging classfier from sklearn.ensemble import BaggingClassifier bgcl = BaggingClassifier(n_estimators=60, max_samples=.7 , oob_score=True) bgcl = bgcl.fit(movie_df, new_movie_df) print(bgcl.oob_score_)

The code you provided is for a Bagging classifier using the scikit-learn library in Python. Bagging, short for Bootstrap Aggregating, is an ensemble learning technique. Here's a breakdown of what this code does:

Import the necessary libraries:

Import the BaggingClassifier class from sklearn.ensemble.

Create a Bagging classifier:

Initialize a BaggingClassifier with the following key hyperparameters:

n_estimators is set to 60, which specifies the number of base classifiers (typically decision trees) in the ensemble.max_samples is set to 0.7, which specifies the fraction of samples to draw from the dataset to train each base classifier.oob_score is set to True, indicating that out-of-bag (OOB) samples will be used to estimate the generalization accuracy of the ensemble. The OOB score can be printed later.

Train the Bagging classifier:

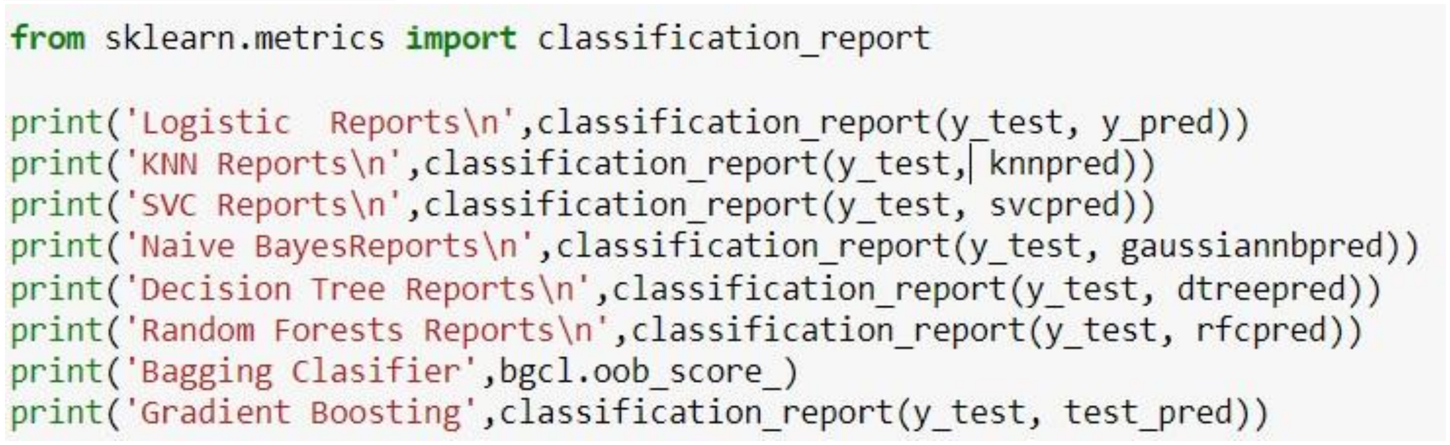
Fit the Bagging classifier using the movie_df as the feature data and new_movie_df as the target data.

Print the OOB score:

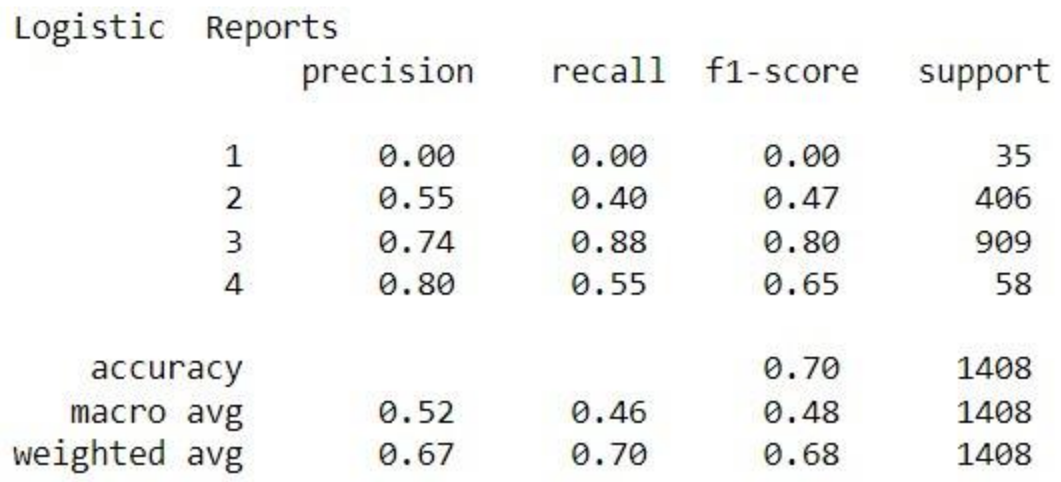
After training the model, the code prints the OOB score using bgcl.oob_score_. The OOB score is an estimate of the model's accuracy on unseen data, and it is computed based on the samples that were not used in each base classifier's training.

This code sets up a Bagging classifier with specified hyperparameters, trains it on the data, and prints the OOB score. Bagging is used to improve the performance and reduce overfitting in machine learning models, especially when the base classifiers are decision trees.

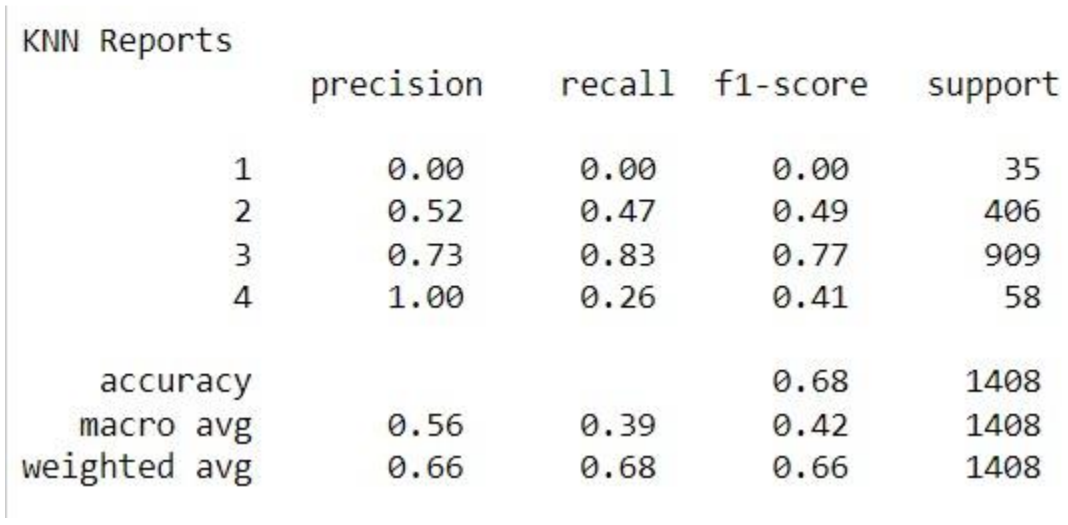
10.Model Comparison



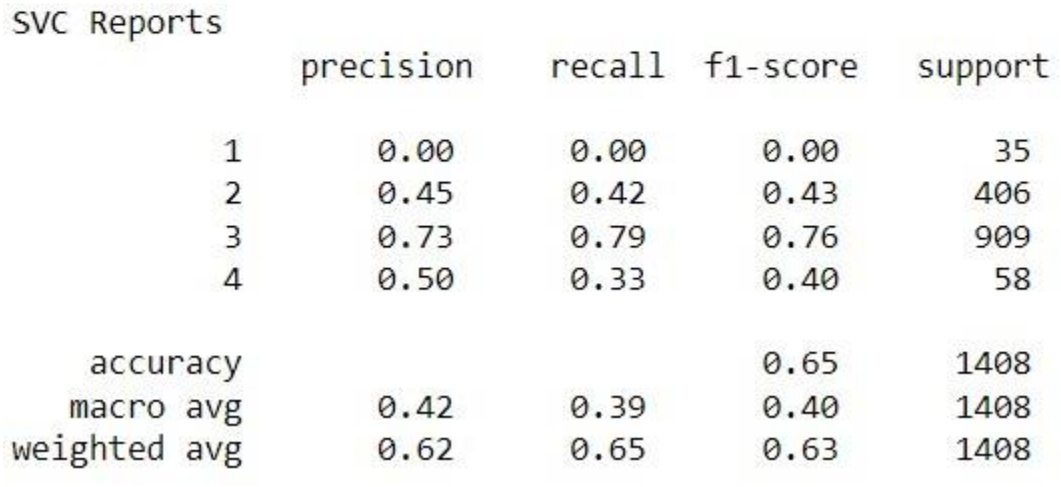
```
from sklearn.metrics import classification_report

print('Logistic  Reports\n',classification_report(y_test, y_pred))
print('KNN Reports\n',classification_report(y_test, knnpred))
print('SVC Reports\n',classification_report(y_test, svcpred))
print('Naive BayesReports\n',classification_report(y_test, gaussiannbpred))
print('Decision Tree Reports\n',classification_report(y_test, dtreepred))
print('Random Forests Reports\n',classification_report(y_test, rfcpred))
print('Bagging Clasifier',bgcl.oob_score_)
print('Gradient Boosting',classification_report(y_test, test_pred))
```



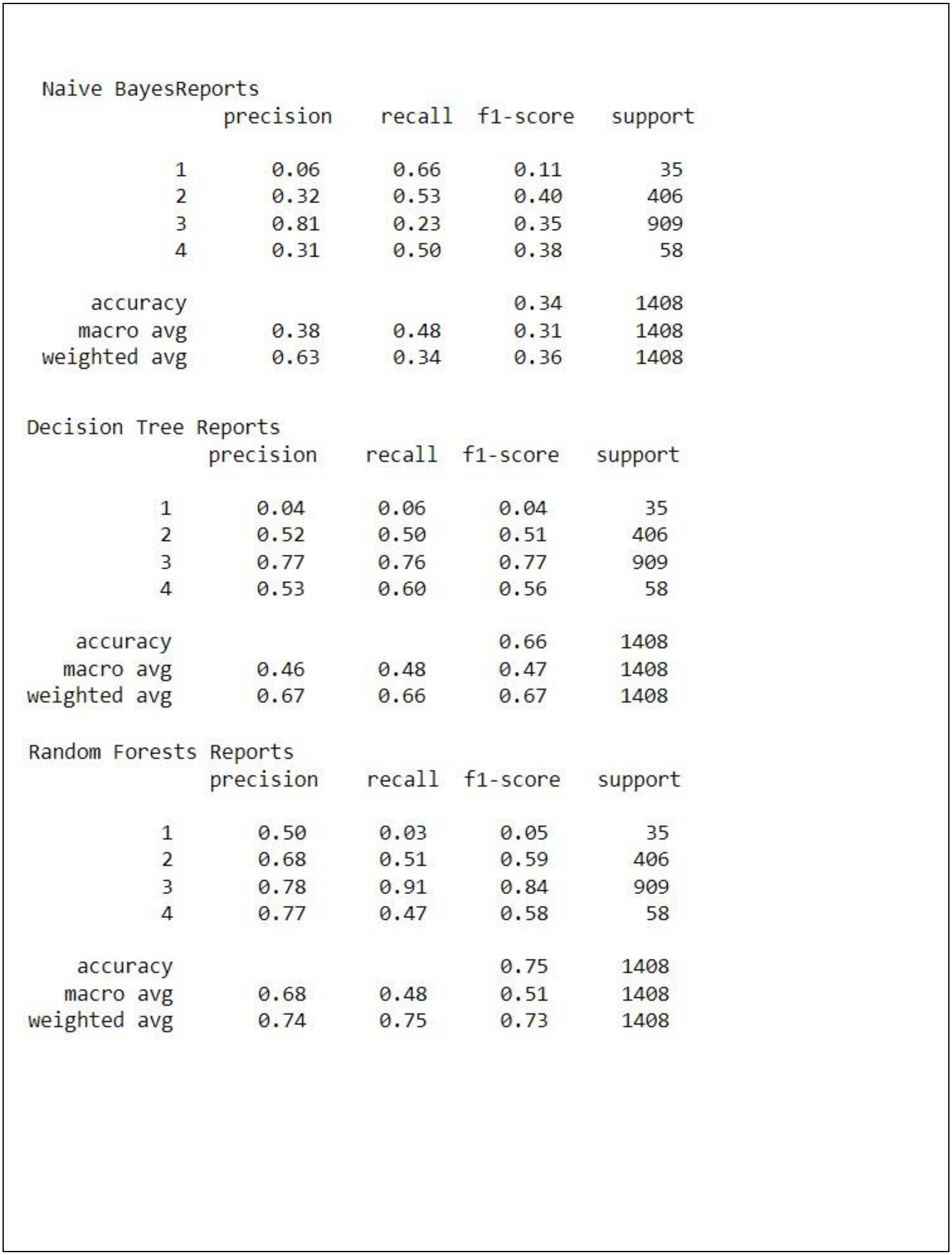
```
Logistic  Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        35
           2       0.55      0.40      0.47       406
           3       0.74      0.88      0.80       909
           4       0.80      0.55      0.65        58

    accuracy                           0.70      1408
   macro avg       0.52      0.46      0.48      1408
weighted avg       0.67      0.70      0.68      1408
```



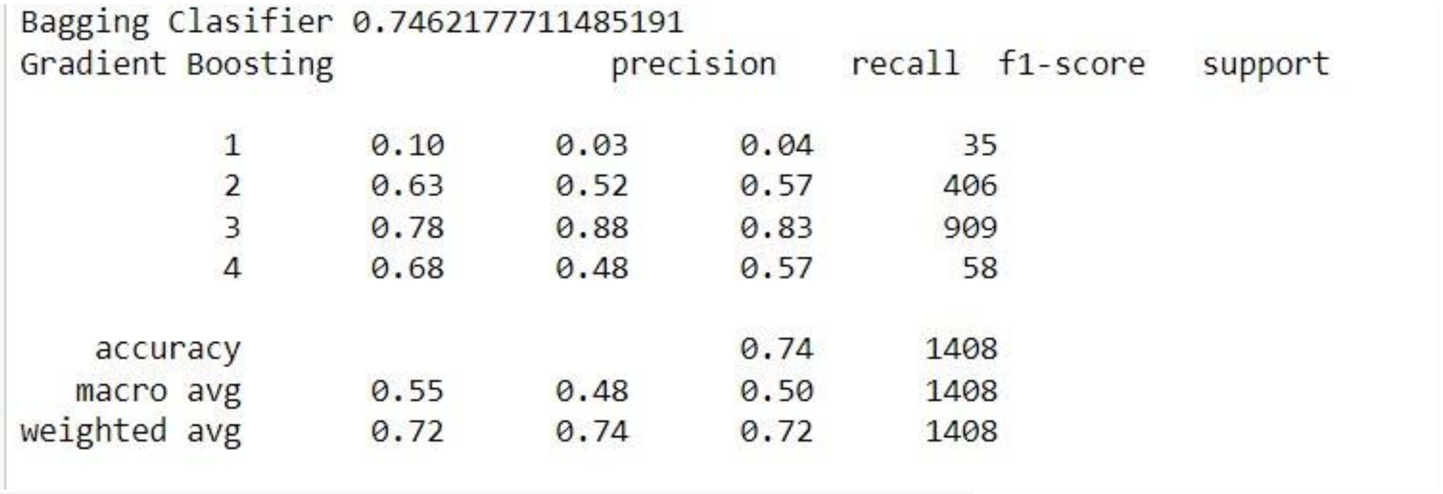
```
KNN Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        35
           2       0.52      0.47      0.49       406
           3       0.73      0.83      0.77       909
           4       1.00      0.26      0.41        58

    accuracy                           0.68      1408
   macro avg       0.56      0.39      0.42      1408
weighted avg       0.66      0.68      0.66      1408
```



```
SVC Reports
              precision    recall  f1-score   support

           1       0.00      0.00      0.00        35
           2       0.45      0.42      0.43       406
           3       0.73      0.79      0.76       909
           4       0.50      0.33      0.40        58

    accuracy                           0.65      1408
   macro avg       0.42      0.39      0.40      1408
weighted avg       0.62      0.65      0.63      1408
```



```
 Naive BayesReports
              precision    recall  f1-score   support

           1       0.06      0.66      0.11        35
           2       0.32      0.53      0.40       406
           3       0.81      0.23      0.35       909
           4       0.31      0.50      0.38        58

    accuracy                           0.34      1408
   macro avg       0.38      0.48      0.31      1408
weighted avg       0.63      0.34      0.36      1408


Decision Tree Reports
              precision    recall  f1-score   support

           1       0.04      0.06      0.04        35
           2       0.52      0.50      0.51       406
           3       0.77      0.76      0.77       909
           4       0.53      0.60      0.56        58

    accuracy                           0.66      1408
   macro avg       0.46      0.48      0.47      1408
weighted avg       0.67      0.66      0.67      1408

Random Forests Reports
              precision    recall  f1-score   support

           1       0.50      0.03      0.05        35
           2       0.68      0.51      0.59       406
           3       0.78      0.91      0.84       909
           4       0.77      0.47      0.58        58

    accuracy                           0.75      1408
   macro avg       0.68      0.48      0.51      1408
weighted avg       0.74      0.75      0.73      1408
```



```
Bagging Clasifier 0.7462177711485191
Gradient Boosting                precision    recall  f1-score   support

           1       0.10      0.03      0.04        35
           2       0.63      0.52      0.57       406
           3       0.78      0.88      0.83       909
           4       0.68      0.48      0.57        58

    accuracy                           0.74      1408
   macro avg       0.55      0.48      0.50      1408
weighted avg       0.72      0.74      0.72      1408
```

Classifier	Accuracy	Class 1	Class 2	Class 3	Class 4
Logistic Regression	0.70	0.00	0.55	0.74	0.80
K-Nearest Neighbors	0.68	0.00	0.52	0.73	1.00
Support Vector Classifier (SVC)	0.65	0.00	0.45	0.73	0.50
Naive Bayes	0.34	0.06	0.32	0.81	0.31
Decision Tree	0.66	0.04	0.52	0.77	0.53
Random Forests	0.75	0.50	0.68	0.78	0.77

Bagging Classifier	**0.746**	**-**	**-**	**-**	**-**
Gradient Boosting	**0.74**	**0.10**	**0.63**	**0.78**	**0.68**

11.CONCLUSION:

The performance of the classifiers was evaluated using precision, recall, and F1-score metrics, along with overall accuracy. Each classifier displayed different strengths and weaknesses in classifying data into four classes (Class 1, Class 2, Class 3, and Class 4).

Random Forests achieved the highest overall accuracy of 0.75, with a good balance of precision and recall across most classes. It performed notably well in Class 3, making it a strong contender for the classification task.

Bagging Classifier also performed well with an accuracy of 0.746, similar to Random Forests, indicating its ability to maintain a good balance across classes.

Gradient Boosting had a high accuracy of 0.74, with good precision and recall in Class 3, but it struggled with precision in Class 1.

Logistic Regression, K-Nearest Neighbors (KNN), and Support Vector Classifier (SVC) had moderate accuracy levels (0.65 to 0.70), with varying performance across classes. While Class 3 was handled well by these classifiers, they struggled with precision in Class 1.

Decision Tree achieved an accuracy of 0.66, with balanced performance across classes, but it had room for improvement, particularly in Class 1.

Naive Bayes displayed the lowest accuracy of 0.34, with high recall but low precision in Class 1, and varied performance across the other classes.

Random Forests and Bagging Classifier offer a good balance of performance across classes and are suitable for overall accuracy. Gradient Boosting is competitive but requires attention to precision in Class 1. The choice should consider whether class-specific accuracy, class balance, or overall accuracy is most critical for your application.