A red and black logo

Description automatically generatedA purple and white logo

Description automatically generated

**Bsc. (Hons) Artificial Intelligene & Data Science**

**Academic Year :** 2023/2024

**Semester :** 2

**Module Number :** CM2604

**Module Title :** Machine Learning

**Assessment Method Coursework :** Individual

**Deadline (time and date) :** 29th March 23.59 hr

**Submission Assessment :** Dropbox in the Module Study Area in CampusMoodle.

**Word Limit :** N/A

**Module Co-ordinator :** Mr. Sahan Priyanayana

**Student IIT ID :** 20211111

**Student RGU ID :** 2237949

**Student Name :** Manith Ratnayake

# Contents

[Contents 2](#_Toc162657036)

[List of Tables 2](#_Toc162657037)

[List of Figures 2](#_Toc162657038)

[1. Abstarct 3](#_Toc162657039)

[2. Corpus Preparation 3](#_Toc162657040)

[3. Implementation 4](#_Toc162657041)

[4. Experiments 4](#_Toc162657042)

[5. Discussions 4](#_Toc162657043)

[6. Limitations 5](#_Toc162657044)

[7. Further Enchancements 5](#_Toc162657045)

[8. Appendix 5](#_Toc162657046)

[9. References 6](#_Toc162657047)

# List of Tables

No table of figures entries found.

# List of Figures

[Figure 1 5](#_Toc162664026)

[Figure 2 5](#_Toc162664027)

[Figure 3 5](#_Toc162664028)

[Figure 4 6](#_Toc162664029)

[Figure 5 6](#_Toc162664030)

[Figure 6 6](#_Toc162664031)

# Abstarct

The dataset is used to complete the project is available on UCI. Adult dataset analysis is a supervised binary classification problem. The aim of the project is to predict whether a person makes over 50k a year

Github link: **https://github.com/Manith-Ratnayake/Adult\_Income\_Census\_UCI**

# Corpus Preparation

Data set has two files

1. Adult data
2. Adult test

For training purposes “Adult.data” file is used and for testing purposes “Adult.test” file is used. For missing values in columns for both files the most frequently occurred value is added to the to the missing value.

Feature enginnering has taken in the data set

1. Education column value has replace ['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th','10th', '11th', '12th'] to 'school'
2. education column value has replace ('HS-grad') to (‘high school')
3. education column value has replace (['Assoc-voc', 'Assoc-acdm', 'Prof-school', 'Some-college']) to ('higher')
4. education column value has replace ('Bachelors') to ('undergrad')
5. education column value has replace('Masters') to ('grad')
6. education column value has replace ('Doctorate') to ('doc')
7. marital-status column value has replace (['Married-civ-spouse', 'Married-AF-spouse']) to ('married')
8. marital-status value has replace (['Never-married'] to ('not-married')
9. marital-status column value has replace (['Divorced', 'Separated','Widowed', 'Married-spouse-absent'] to ('other')

Approximately 2/3 of all data is used to training phase and for testing 1/3 is used.

Data has been encoded using one hot encoding method.

# Implementation

For predicting the income whether it exceeds 50k or not two machine learning models are used

1. Naïve Bays
2. Randomforest

For naïve bays model the basic available model in sci kit learn is used and also parameter tunned sci-kit model is used.

Also for randomforest model the basic available model in sci kit learn is used and also parameter tunned sci-kit model is used.

# Experiments

Naïve bayes default model evaluation :

A screenshot of a computer

Description automatically generated

Figure 1

Naïve bayes parameters tuned model evaluation :

A screenshot of a computer code

Description automatically generated

Figure 2

Random forest default model evaluation :

A screenshot of a computer

Description automatically generated

Figure 3

Random forest parameters tuned model evaluation :

A screenshot of a computer

Description automatically generated

Figure 4

Naïve bayes experimentail model parameters:

A computer code with text

Description automatically generated with medium confidence

Figure 5

Randomforest experimentail model parameters

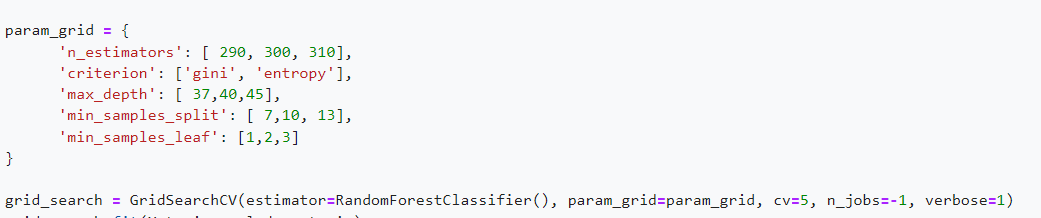


Figure 6

# Discussions

After model training finsihed each model is evaluated. Different evaluation methods are used

1. Accuracy
2. Precision
3. Recall
4. F1 score
5. Confustion matrix
6. ROC curve
7. AUR curve
8. PRC curve
9. Learning Curve

are provided. Using the parameters tuned for each model type is wise decision due to it has the best parameters for the case study.

The most optimal machine learning algorithm to predict is randomforest tuned model.

**Brief Comparision**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naïve Bays Default | Naïve Bays Parameters Tuned | Random Forest Default | Randomforest Parameters tuned |
| Accuracy | 0.48 | 0.62 | 0.84 | 0.86 |
| Precision | 0.3074 | 0.38 | 0.8425 | 0.8584 |
| Recall | 0.9584 | 0.921 | 0.8485 | 0.8641 |
| F1 score | 0.4656 | 0.54 | 0.8442 | 0.8585 |
| AUC | 0.71 | 0.84 | 0.8991 | 0.9167 |

# Limitations

Significant limitaions are not found.

# Further Enchancements

The current prediction is used only two machine learning models. More results can be experimented and tested using different machine learning models. Logistic regression, Ada boost, XGboost and KNN are some model examples.

# Appendix

The model is developed using google colab and the notebook is downloaded and code is pasted below as a text.

# from google.colab import drive

# drive.mount('/content/drive')

"""#\*\*1. Libraries\*\*"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# import warnings

# warnings.filterwarnings('ignore')

"""#\*\*2. File Reading\*\*"""

# Location

# adult\_file\_data\_path = '/content/drive/My Drive/Machine Learning/adult.data'

# adult\_file\_test\_path = '/content/drive/My Drive/Machine Learning/adult.test'

adult\_file\_data\_path = 'adult.data'

adult\_file\_test\_path = 'adult.test'

# Reading

adult\_data = pd.read\_csv(adult\_file\_data\_path, delimiter=',', header=None)

adult\_test = pd.read\_csv(adult\_file\_test\_path, delimiter=',', header=None, skiprows=1)

# Columns

column\_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',

                'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',

                'hours-per-week', 'native-country', 'income']

adult\_data.columns = column\_names

adult\_test.columns = column\_names

"""#\*\*3. Data Set Overview\*\*"""

# Checking the first few rows

adult\_data.head()

adult\_data.info()

# Statistics

adult\_data.describe().T

# No of Rows and columns in adult data

adult\_data.shape

# No of Rows and columns adult test

adult\_test.shape

"""#\*\*4. Column Overview\*\*"""

adult\_data.columns

adult\_data.nunique()

# Type

adult\_data.dtypes

"""###\*\*Empty Columns\*\*"""

adult\_data['occupation'].value\_counts()

adult\_data['native-country'].value\_counts()

adult\_data['workclass'].value\_counts()

"""###\*\*Not Empty Columns\*\*"""

# Distribution of the target variable

adult\_data['capital-gain'].value\_counts()

# Gender

adult\_data['sex'].value\_counts()

# Education

adult\_data['education'].value\_counts()

adult\_data['education-num'].value\_counts()

# Distribution of the target variable

adult\_data['income'].value\_counts()

"""#\*\*Missing Values\*\*"""

# Null columns filled with most frequently ocuured value in that column

adult\_data['workclass'] = adult\_data['workclass'].replace('?', 'Private')

adult\_data['occupation'] = adult\_data['occupation'].replace('?', 'Prof-specialty')

adult\_data['native-country'] = adult\_data['native-country'].replace('?', 'United-States')

# Same for Testing Data Set

adult\_test['workclass'] = adult\_test['workclass'].replace('?', 'Private')

adult\_test['occupation'] = adult\_test['occupation'].replace('?', 'Prof-specialty')

adult\_test['native-country'] = adult\_test['native-country'].replace('?', 'United-States')

"""#\*\*Feature Engineering\*\*"""

# Education Category

adult\_data.education= adult\_data.education.replace(['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th','10th', '11th', '12th'], 'school')

adult\_data.education = adult\_data.education.replace('HS-grad', 'high school')

adult\_data.education = adult\_data.education.replace(['Assoc-voc', 'Assoc-acdm', 'Prof-school', 'Some-college'], 'higher')

adult\_data.education = adult\_data.education.replace('Bachelors', 'undergrad')

adult\_data.education = adult\_data.education.replace('Masters', 'grad')

adult\_data.education = adult\_data.education.replace('Doctorate', 'doc')

# Same for Testing Data Set

adult\_test.education= adult\_test.education.replace(['Preschool', '1st-4th', '5th-6th', '7th-8th', '9th','10th', '11th', '12th'], 'school')

adult\_test.education = adult\_test.education.replace('HS-grad', 'high school')

adult\_test.education = adult\_test.education.replace(['Assoc-voc', 'Assoc-acdm', 'Prof-school', 'Some-college'], 'higher')

adult\_test.education = adult\_test.education.replace('Bachelors', 'undergrad')

adult\_test.education = adult\_test.education.replace('Masters', 'grad')

adult\_test.education = adult\_test.education.replace('Doctorate', 'doc')

# Martial status

adult\_data['marital-status']= adult\_data['marital-status'].replace(['Married-civ-spouse', 'Married-AF-spouse'], 'married')

adult\_data['marital-status'] = adult\_data['marital-status'].replace(['Never-married'], 'not-married')

adult\_data['marital-status'] = adult\_data['marital-status'].replace(['Divorced', 'Separated','Widowed',

                                                'Married-spouse-absent'], 'other')

# Same for Testing Data Set

adult\_test['marital-status'] = adult\_test['marital-status'].replace(['Married-civ-spouse', 'Married-AF-spouse'], 'married')

adult\_test['marital-status'] = adult\_test['marital-status'].replace(['Never-married'], 'not-married')

adult\_test['marital-status'] = adult\_test['marital-status'].replace(['Divorced', 'Separated','Widowed',

                                                'Married-spouse-absent'], 'other')

"""#\*\*Explotary Data Analysis (EDA)\*\*"""

sns.countplot(x='income', palette='coolwarm', hue='age', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='workclass', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='education', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='marital-status', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='occupation', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='relationship', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='race', data=adult\_data)

sns.countplot(x='income', palette='coolwarm', hue='sex', data=adult\_data)

sns.countplot(x='income', palette='husl', hue='capital-gain', data=adult\_data)

sns.countplot(x='income', palette='husl', hue='capital-loss', data=adult\_data)

sns.countplot(x='income', palette='husl', hue='native-country', data=adult\_data)

adult\_data.corr(numeric\_only=True)

plt.figure(figsize=(15,12))

sns.heatmap(adult\_data.corr(numeric\_only=True), annot=True, fmt=".2f", cmap='magma')

adult\_data.hist(figsize=(12,12), layout=(3,3), sharex=False);

adult\_data.plot(kind='box', figsize=(12,12), layout=(3,3), sharex=False, subplots=True);

"""#\*\*Training Testing Data Set\*\*"""

# Training Income Column

adult\_data['income'] = adult\_data['income'].str.strip().replace({'<=50K': 0, '>50K': 1})

# Testing Income Column

adult\_test['income'] = adult\_test['income'].str.strip().replace({'<=50K.': 0, '>50K.': 1})

# Train Set

X\_train = adult\_data.drop('income', axis=1)

y\_train = adult\_data['income']

# Test Set

X\_test = adult\_test.drop('income', axis=1)

y\_test = adult\_test['income']

"""#\*\*Encoding\*\*"""

X\_train\_encoded = pd.get\_dummies(X\_train)

X\_test\_encoded = pd.get\_dummies(X\_test)

# Confirming both datasets have the same set of columns after encoding

X\_train\_encoded, X\_test\_encoded = X\_train\_encoded.align(X\_test\_encoded, join='left', axis=1, fill\_value=0)

"""#\*\*Feature Scaling\*\*"""

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train\_encoded)

X\_test\_scaled = scaler.transform(X\_test\_encoded)

"""#\*\*Model Input Variables\*\*"""

X\_train\_scaled

X\_test\_scaled

y\_train

y\_test

"""#\*\*Naive Bayes\*\*

###\*\*Naive Bayes Basic Model\*\*

"""

from sklearn.naive\_bayes import GaussianNB

naive\_base\_basic\_model = GaussianNB()

naive\_base\_basic\_model.fit(X\_train\_scaled, y\_train)

naive\_base\_y\_predictions = naive\_base\_basic\_model.predict(X\_test\_scaled)

"""###\*\*Naive Bayes Basic Model Evaluation\*\*

######\*\*Classification Report\*\*

"""

from sklearn.metrics import accuracy\_score, classification\_report, precision\_score, recall\_score, f1\_score, roc\_auc\_score

print(classification\_report(y\_test,naive\_base\_y\_predictions))

print("Accuracy : ", accuracy\_score(y\_test,naive\_base\_y\_predictions),"\n")

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

positive\_label = 1

precision = precision\_score(y\_test, naive\_base\_y\_predictions, pos\_label=positive\_label)

recall = recall\_score(y\_test, naive\_base\_y\_predictions, pos\_label=positive\_label)

f1 = f1\_score(y\_test, naive\_base\_y\_predictions, pos\_label=positive\_label)

y\_prob = naive\_base\_basic\_model.predict\_proba(X\_test\_scaled)[:, 1]  # Probabilities for the positive class

naive\_bayes\_auc = roc\_auc\_score(y\_test, y\_prob)  # Now correctly using probabilities

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"AUC: {naive\_bayes\_auc:.4f}\n")

from sklearn.metrics import classification\_report

report\_dict = classification\_report(y\_test, naive\_base\_y\_predictions, output\_dict=True)

report\_df = pd.DataFrame(report\_dict).transpose()

# Removing accuracy column

report\_df.drop(['accuracy'], inplace=True)

report\_df.plot(kind='bar', figsize=(12, 8))

plt.title('Classification Report')

plt.xlabel('Classes and Averages')

plt.ylabel('Scores')

plt.xticks(rotation=45)

plt.legend(loc='upper left')

plt.grid(axis='y', linestyle='--')

plt.show()

"""######\*\*Confusion Matix\*\*"""

from sklearn.metrics import confusion\_matrix

naive\_bayes\_basic\_model\_confussion\_matrix = confusion\_matrix(y\_test, naive\_base\_y\_predictions)

sns.heatmap(naive\_bayes\_basic\_model\_confussion\_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix')

plt.show()

"""######\*\*ROC Curve\*\*"""

y\_prob = naive\_base\_basic\_model.predict\_proba(X\_test\_scaled)[:, 1]

from sklearn.metrics import roc\_curve

# For binary classification, get the probability of the positive class

fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)

plt.plot(fpr, tpr, label='ROC Curve')

plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.show()

# AUC using probabilities

naive\_bayes\_basic\_model\_auc\_prob = roc\_auc\_score(y\_test, y\_prob)

print(f"AUC (Using probabilities): {naive\_bayes\_basic\_model\_auc\_prob:.4f}")

"""######\*\*Precision-Recall Curve\*\*"""

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import auc

precision, recall, \_ = precision\_recall\_curve(y\_test, y\_prob)

naive\_bayes\_basic\_model\_auc\_precision\_recall = auc(recall, precision)

plt.plot(recall, precision, marker='.')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.show()

print(f"AUC Precision-Recall: {naive\_bayes\_basic\_model\_auc\_precision\_recall:.4f}")

"""######\*\*Learning Curve\*\*"""

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(naive\_base\_basic\_model, X\_train\_scaled, y\_train, cv=5, scoring='accuracy', n\_jobs=-1, train\_sizes=np.linspace(.1, 1.0, 5))

# Calculate mean and standard deviation for training set scores

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

# Calculate mean and standard deviation for test set scores

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

plt.fill\_between(train\_sizes, train\_mean - train\_std, train\_mean + train\_std, color="#DDDDDD")

plt.fill\_between(train\_sizes, test\_mean - test\_std, test\_mean + test\_std, color="#DDDDDD")

plt.plot(train\_sizes, train\_mean, label="Training score")

plt.plot(train\_sizes, test\_mean, label="Cross-validation score")

plt.title("Learning Curve")

plt.xlabel("Training Set Size"), plt.ylabel("Accuracy Score"), plt.legend(loc="best")

plt.tight\_layout()

plt.show()

"""######\*\*CV score\*\*"""

from sklearn.model\_selection import cross\_val\_score

cv\_scores = cross\_val\_score(naive\_base\_basic\_model, X\_train\_scaled, y\_train, cv=5, scoring='accuracy')

print(f"CV Accuracy Scores: {cv\_scores}")

print(f"CV Accuracy Mean: {cv\_scores.mean():.4f}")

"""###\*\*Naive Bayes Experimental Model\*\*

"""

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import make\_scorer

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import GridSearchCV

gnb = GaussianNB()

param\_grid = {

    'var\_smoothing': np.logspace(0,-9, num=100)

}

# Define multiple scoring metrics

scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc\_auc']

# Initialize the GridSearchCV object with multiple scorers

grid\_search = GridSearchCV(estimator=gnb, param\_grid=param\_grid, cv=5, scoring=scoring, refit='accuracy')

# Fit to the training data

grid\_search.fit(X\_train\_scaled, y\_train)

# Results for each metric

results = grid\_search.cv\_results\_

for scorer in scoring:

    print(f"Best '{scorer}' score: {results['mean\_test\_' + scorer][grid\_search.best\_index\_]}")

best\_model = grid\_search.best\_estimator\_

# Predictions

naive\_bayes\_experimental\_model\_y\_prediction = best\_model.predict(X\_test\_scaled)

# Probability scores for the positive class

naive\_bayes\_experimental\_model\_y\_probability = best\_model.predict\_proba(X\_test\_scaled)[:, 1]

"""###\*\*Naive Bayes Experimental Model Evaluation\*\*

######\*\*Classification Report\*\*

"""

from sklearn.metrics import confusion\_matrix, roc\_auc\_score, precision\_recall\_curve

report = classification\_report(y\_test, naive\_bayes\_experimental\_model\_y\_prediction)

print(report)

accuracy = accuracy\_score(y\_test, naive\_bayes\_experimental\_model\_y\_prediction)

print("Accuracy:", accuracy)

from sklearn.metrics import classification\_report

report = classification\_report(y\_test, naive\_bayes\_experimental\_model\_y\_prediction, output\_dict=True)

df\_report = pd.DataFrame(report).transpose()

# Dropping the 'support' column

df\_report.drop('support', axis=1, inplace=True, errors='ignore')

# Reset index to get the classes as a column

df\_report.reset\_index(inplace=True)

# Melt the DataFrame to have proper columns for seaborn

df\_melt = df\_report.melt(id\_vars="index", var\_name="Metric", value\_name="Score")

# Plot

plt.figure(figsize=(10, 6))

sns.barplot(x='Score', y='index', hue='Metric', data=df\_melt)

plt.xlabel('Score')

plt.ylabel('Class')

plt.title('Classification Report')

plt.legend(title='Metric', title\_fontsize='13', loc='lower right')

plt.tight\_layout()

plt.show()

"""######\*\*Confussion Matrix\*\*"""

naive\_bayes\_experimental\_model\_confussion\_matrix = confusion\_matrix(y\_test, naive\_bayes\_experimental\_model\_y\_prediction)

# Plot the confusion matrix

sns.heatmap(naive\_bayes\_experimental\_model\_confussion\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix of Optimized Naive Bayes Model')

plt.show()

"""######\*\*ROC Curve\*\*"""

# ROC Curve

from sklearn.metrics import roc\_curve

fpr, tpr, thresholds = roc\_curve(y\_test, naive\_bayes\_experimental\_model\_y\_probability)

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, label='ROC Curve (area = %0.2f)' % roc\_auc\_score(y\_test, naive\_bayes\_experimental\_model\_y\_probability))

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

"""######\*\*Precision Recall Curve (PRC)\*\*"""

# Precision-Recall Curve

precision, recall, thresholds = precision\_recall\_curve(y\_test, naive\_bayes\_experimental\_model\_y\_probability)

plt.figure(figsize=(8, 6))

plt.plot(recall, precision, label='Precision-Recall curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend(loc="lower left")

plt.show()

"""######\*\*Learning Curve\*\*"""

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, validation\_scores = learning\_curve(

    estimator = GaussianNB(var\_smoothing=grid\_search.best\_params\_['var\_smoothing']),

    X = X\_train\_scaled,

    y = y\_train,

    train\_sizes = np.linspace(0.1, 1.0, 10),

    cv = 5,

    scoring = 'accuracy'  # "accuracy" or "precision" or "F1" or "recall"

)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

validation\_scores\_mean = np.mean(validation\_scores, axis=1)

validation\_scores\_std = np.std(validation\_scores, axis=1)

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

                 train\_scores\_mean + train\_scores\_std, color='r', alpha=0.1)

plt.fill\_between(train\_sizes, validation\_scores\_mean - validation\_scores\_std,

                 validation\_scores\_mean + validation\_scores\_std, color='g', alpha=0.1)

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color='r', label='Training score')

plt.plot(train\_sizes, validation\_scores\_mean, 'o-', color='g', label='Validation score')

plt.title('Learning Curve')

plt.xlabel('Training set size')

plt.ylabel('Accuracy Score')

plt.legend(loc='best')

plt.show()

"""#\*\*Random Forest\*\*

###\*\*Random Forest Basic Model\*\*

"""

from sklearn.ensemble import RandomForestClassifier

random\_forest\_basic\_model = RandomForestClassifier()

random\_forest\_basic\_model.fit(X\_train\_scaled,y\_train)

random\_forest\_basic\_model.score(X\_test\_scaled,y\_test)

random\_forest\_basic\_model\_y\_prediction = random\_forest\_basic\_model.predict(X\_test\_scaled)

importances = dict(zip(X\_train.columns, random\_forest\_basic\_model.feature\_importances\_))

importances\_sorted = {k: v for k, v in sorted(importances.items(), key=lambda item: item[1], reverse=True)}

importances\_sorted

"""###\*\*Random Forest Basic Model Evaluation\*\*

######\*\*Classification Report\*\*

"""

from sklearn.metrics import classification\_report, accuracy\_score

# Assuming 'random\_forest\_basic\_model\_y\_prediction' are the predictions from your model

# and 'y\_test' are the true labels

# Generate the classification report

report = classification\_report(y\_test, random\_forest\_basic\_model\_y\_prediction)

# Print the classification report

print("Classification Report:\n", report)

# Calculate and print the overall accuracy

accuracy = accuracy\_score(y\_test, random\_forest\_basic\_model\_y\_prediction)

print("Overall Accuracy:", accuracy)

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score

# Calculate weighted precision, recall, and F1-score

precision = precision\_score(y\_test, random\_forest\_basic\_model\_y\_prediction, average='weighted')

recall = recall\_score(y\_test, random\_forest\_basic\_model\_y\_prediction, average='weighted')

f1 = f1\_score(y\_test, random\_forest\_basic\_model\_y\_prediction, average='weighted')

# Make sure to use X\_test\_scaled for consistency with training

y\_prob = random\_forest\_basic\_model.predict\_proba(X\_test\_scaled)[:, 1]  # Get probabilities for the positive class

# Calculate ROC AUC

roc\_auc = roc\_auc\_score(y\_test, y\_prob)

# Print out the metrics

print(f"Weighted Precision: {precision:.4f}")

print(f"Weighted Recall: {recall:.4f}")

print(f"Weighted F1-Score: {f1:.4f}")

print(f"ROC AUC: {roc\_auc:.4f}")

from sklearn.metrics import classification\_report

# Assuming y\_pred\_forest are model predictions and y\_test are the true labels

report\_dict = classification\_report(y\_test, random\_forest\_basic\_model\_y\_prediction, output\_dict=True)

df\_report = pd.DataFrame(report\_dict).transpose()

# Dropping the 'support' column

df\_report.drop(columns='support', errors='ignore', inplace=True)

# Dropping 'accuracy' row

df\_report.drop(index='accuracy', errors='ignore', inplace=True)

# Plotting with pandas (which uses Matplotlib)

ax = df\_report.plot(kind='bar', figsize=(10, 7))

ax.set\_title('Classification Report')

ax.set\_xlabel('Class')

ax.set\_ylabel('Score')

ax.set\_xticklabels(df\_report.index, rotation=45, ha='right')

plt.tight\_layout()

plt.show()

"""######\*\*Confusion Matrix\*\*"""

from sklearn.metrics import confusion\_matrix

random\_forest\_basic\_model\_confussion\_matrix = confusion\_matrix(y\_test, random\_forest\_basic\_model\_y\_prediction)

random\_forest\_basic\_model\_confussion\_matrix

# Plot the confusion matrix

sns.heatmap(random\_forest\_basic\_model\_confussion\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix of Optimized Naive Bayes Model')

plt.show()

"""######\*\*ROC Curve\*\*"""

from sklearn.metrics import roc\_auc\_score, roc\_curve

# Target should be binary before using roc\_auc\_score

if len(np.unique(y\_test)) == 2:

    random\_forest\_basic\_model\_y\_probability = random\_forest\_basic\_model.predict\_proba(X\_test\_scaled)[:, 1]  # probability estimates for the positive class

    roc\_auc = roc\_auc\_score(y\_test, random\_forest\_basic\_model\_y\_probability)

    fpr, tpr, thresholds = roc\_curve(y\_test, random\_forest\_basic\_model\_y\_probability)

    plt.figure()

    plt.plot(fpr, tpr, label=f'ROC curve (area = {roc\_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title('ROC Curve')

    plt.legend(loc="lower right")

    plt.show()

"""######\*\*Precision-Recall Curve (PRC)\*\*"""

from sklearn.metrics import precision\_recall\_curve, auc

precision, recall, thresholds = precision\_recall\_curve(y\_test, random\_forest\_basic\_model\_y\_probability)

pr\_auc = auc(recall, precision)

plt.figure()

plt.plot(recall, precision, label=f'PR curve (area = {pr\_auc:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend(loc="upper right")

plt.show()

"""######\*\*Learning Curve\*\*"""

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(random\_forest\_basic\_model, X\_train\_scaled, y\_train, cv=5)

train\_scores\_mean = np.mean(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

plt.figure()

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

plt.xlabel("Training examples")

plt.ylabel("Score")

plt.legend(loc="best")

plt.show()

"""######\*\*Column Importance\*\*"""

random\_forest\_basic\_model.feature\_importances\_

# Ensuring "random\_forest\_experimental\_model" is an instance of RandomForestClassifier

if isinstance(random\_forest\_basic\_model, RandomForestClassifier):

    importances = dict(zip(X\_train\_encoded.columns, random\_forest\_basic\_model.feature\_importances\_))

    importances\_sorted = {k: v for k, v in sorted(importances.items(), key=lambda item: item[1], reverse=True)}

    for feature, importance in importances\_sorted.items():

        print(f"{feature}: {importance}")

else:

    print("The model does not support feature\_importances\_. Please check the model type.")

"""###\*\*Random Forest Experimental Model\*\*"""

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestClassifier

param\_grid = {

      'n\_estimators': [ 290, 300, 310],

      'criterion': ['gini', 'entropy'],

      'max\_depth': [ 37,40,45],

      'min\_samples\_split': [ 7,10, 13],

      'min\_samples\_leaf': [1,2,3]

}

grid\_search = GridSearchCV(estimator=RandomForestClassifier(), param\_grid=param\_grid, cv=5, n\_jobs=-1, verbose=1)

grid\_search.fit(X\_train\_scaled, y\_train)

print("Best parameters: ", grid\_search.best\_params\_)

print("Best score: ", grid\_search.best\_score\_)

# Best estimator found by the grid search

random\_forest\_experimental\_model = grid\_search.best\_estimator\_

# Predictions on test set

random\_forest\_best\_y\_prediction = random\_forest\_experimental\_model.predict(X\_test\_scaled)

"""###\*\*Random Forest Experimental Model Evaluation\*\*

######\*\*Classification Report\*\*

"""

from sklearn.metrics import classification\_report, accuracy\_score

# Generate the classification report

report = classification\_report(y\_test, random\_forest\_best\_y\_prediction)

print("Classification Report:\n", report)

# Calculate and print the accuracy

accuracy = accuracy\_score(y\_test, random\_forest\_best\_y\_prediction)

print("Accuracy:", accuracy)

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score

weighted\_precision = precision\_score(y\_test, random\_forest\_best\_y\_prediction, average='weighted')

weighted\_recall = recall\_score(y\_test, random\_forest\_best\_y\_prediction, average='weighted')

weighted\_f1 = f1\_score(y\_test, random\_forest\_best\_y\_prediction, average='weighted')

y\_prob = random\_forest\_experimental\_model.predict\_proba(X\_test\_scaled)[:, 1]  # Replace X\_test\_scaled with your test features

roc\_auc = roc\_auc\_score(y\_test, y\_prob)

print(f"Weighted Average Precision: {weighted\_precision:.4f}")

print(f"Weighted Average Recall: {weighted\_recall:.4f}")

print(f"Weighted Average F1-Score: {weighted\_f1:.4f}")

print(f"ROC AUC: {roc\_auc:.4f}")

# Classification report into a dictionary, then to a DataFrame conversion

report\_dict = classification\_report(y\_test, random\_forest\_best\_y\_prediction, output\_dict=True)

report\_df = pd.DataFrame(report\_dict).transpose()

# # Drop 'support' column and 'accuracy' row if present

report\_df.drop(columns='support', errors='ignore', inplace=True)

report\_df.drop(index='accuracy', errors='ignore', inplace=True)

# Reset the index to use in plotting

report\_df.reset\_index(inplace=True)

report\_df.rename(columns={'index': 'Metric'}, inplace=True)

# Melt the DataFrame for seaborn plotting

melted\_df = report\_df.melt(id\_vars="Metric", var\_name="Measures", value\_name="Score")

# Plot the classification report

plt.figure(figsize=(12, 8))

sns.barplot(x='Measures', y='Score', hue='Metric', data=melted\_df)

plt.title('Classification Metrics for RandomForest Experimental Model')

plt.xticks(rotation=45)

plt.ylabel('Score')

plt.xlabel('Metrics')

plt.tight\_layout()

plt.show()

"""######\*\*Confusion Matrix\*\*"""

from sklearn.metrics import confusion\_matrix

# Generate the confusion matrix using true labels and predictions

random\_forest\_experimental\_confussion\_matrix = confusion\_matrix(y\_test, random\_forest\_best\_y\_prediction)

# Plot the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(random\_forest\_experimental\_confussion\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix for Best Random Forest Model')

plt.show()

"""######\*\*ROC Curve\*\*"""

from sklearn.metrics import roc\_curve, auc

# Probabilities

random\_forest\_best\_y\_probability = grid\_search.best\_estimator\_.predict\_proba(X\_test\_scaled)[:, 1]

# ROC curve and AUC

fpr, tpr, thresholds = roc\_curve(y\_test, random\_forest\_best\_y\_probability)

roc\_auc = auc(fpr, tpr)

# Plotting

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

"""######\*\*Precision-Recall Curve (PRC)\*\*"""

from sklearn.metrics import precision\_recall\_curve

from sklearn.metrics import average\_precision\_score

precision, recall, \_ = precision\_recall\_curve(y\_test, random\_forest\_best\_y\_probability)

ap\_score = average\_precision\_score(y\_test, random\_forest\_best\_y\_probability)

plt.figure(figsize=(8, 6))

plt.plot(recall, precision, marker='.', label=f'Precision-Recall curve (AP = {ap\_score:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve')

plt.legend()

plt.grid(True)

plt.show()

"""######\*\*Learning Curve\*\*"""

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(

    estimator=grid\_search.best\_estimator\_, # Best model

    X=X\_train\_scaled,

    y=y\_train,

    train\_sizes=np.linspace(0.1, 1.0, 10), # Training set sizes

    cv=5, # Cross-validation splitting strategy

    n\_jobs=-1,

    scoring='accuracy' # different scoring if needed("")

)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

plt.figure(figsize=(8, 6))

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

                 train\_scores\_mean + train\_scores\_std, alpha=0.1, color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

                 test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r", label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g", label="Cross-validation score")

plt.title("Learning Curve")

plt.xlabel("Training Examples")

plt.ylabel("Score")

plt.legend(loc="best")

plt.grid(True)

plt.show()

"""######\*\*Column Importance\*\*"""

random\_forest\_experimental\_model.feature\_importances\_

# Ensuring "random\_forest\_experimental\_model" is an instance of RandomForestClassifier

if isinstance(random\_forest\_experimental\_model, RandomForestClassifier):

    importances = dict(zip(X\_train\_encoded.columns, random\_forest\_experimental\_model.feature\_importances\_))

    importances\_sorted = {k: v for k, v in sorted(importances.items(), key=lambda item: item[1], reverse=True)}

    for feature, importance in importances\_sorted.items():

        print(f"{feature}: {importance}")

else:

    print("The model does not support feature\_importances\_. Please check the model type.")

# References

1. SPOTLESS TECH, 2021. *Census income dataset analysis using Python | income prediction machine learning model using XGBoost*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=WULwst0vW8g [Accessed 29 Mar 2024].
2. NEURALNINE, 2023. *Income prediction machine learning project in python*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=dhoKFqhVJu0 [Accessed 29 Mar 2024].
3. FREECODECAMP.ORG, 2024. *Machine Learning in 2024 – Beginner’s Course*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=bmmQA8A-yUA [Accessed 29 Mar 2024].
4. MADAN, A., 2021. *Adult Sensus Income Kaggle Dataset Analysis | Kaggle | Who earns more than $50K/year ??* [online]. Youtube. Available from: https://www.youtube.com/watch?v=reVAGcwOxH8 [Accessed 29 Mar 2024].
5. MACHINE MANTRA, 2020. *Census income dataset python | UCI Adult dataset (part 2)| how to build a model in machine learning*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=kEPYo45n51M [Accessed 29 Mar 2024].
6. MACHINE MANTRA, 2020b. *Census income dataset analysis with python | UCI data set download*. [online]. Youtube. Available from: https://www.youtube.com/watch?v=a3Tqg0q\_LeI [Accessed 29 Mar 2024].