Report

Module Name: **Machine Learning**

Module Code: **CM2604**

Module Leader: **Mr. Sahan Priyanayana**

Student Name: **Aadhavan Arkhash Saravanakumar**

IIT ID: **20221213**

RGU ID: **2237045**

GitHub Repository Link:

https://github.com/arkhash0309/ML-Coursework

ROBERT GORDON UNIVERSITY ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

Table of Contents

1. Introduction

This report demonstrated the procedures followed to solve a simple classification problem where a prediction must be made as to whether the income exceeds a benchmark value of $50K/ year based on the provided dataset. Machine Learning technologies must be used as the coursework specification expects the use of Random Forest and Naïve Bayes algorithms to make the predictions. This document contains a detailed report on how the dataset was explored and cleaned before being put into the machine learning algorithms to be trained and tested. This project requires prior knowledge of the following techniques.

1. Python programming
2. Exploratory Data Analysis
3. Data preprocessing and cleaning techniques
4. Feature selection and engineering techniques
5. Machine Learning

2. Corpus Preparation

2.1. Downloading the dataset

The hyperlink provided in the specification led us to the official repository to obtain the dataset to solve the problem. The dataset was downloaded as a zip file from this website. The zip file contained the following five files.

1. adult_data – This file contains the data to be used for training the models.
2. adult.names- This file contains details regarding the features and their descriptions.
3. adult_test- This file contains the data which is to be used for testing the models.
4. Index- This file contains the date in which each of the files were added to the repository.
5. old.adult.names- This is similar to the adult.names file but it is an older version.

2.2.Creating the Data Frame

```
# create a list containing the column names
column_names= ["age","workclass","fnlwgt","education","education.num","marital.status","occupation","relationship","race","sex","capital.gain","capital.loss",
               "hours.per.week","native.country","income"]

# convert the .txt files to .csv
train = pd.read_csv('adult_data.txt', sep=",\s", header=None, names=column_names, engine='python')
test = pd.read_csv('adult_test.txt', sep=",\s", header=None, names=column_names, engine='python')
test['income'].replace(regex=True,inplace=True,to_replace=r'\.',value=r'')

# merge the 2 data frames vertically
df = pd.concat([test,train])
df.reset_index(inplace=True, drop=True) # the index is reset
df
```

To create the Data Frame for the problem, the text files "adult_data" and "adult_test" must be merged vertically. For this, a list is created with all the column names.

Then the .txt files are converted into .csv files using the "read_csv" function of pandas. Within this function, we specify that each column is separated by a comma and a space, and the names of the column are to be retrieved in the list defined prior. The engine is defined as python since we are dealing with python code.

Next, the two .csv files are concatenated using the "concat" function in pandas to ensure they are joined vertically. Finally, the index is reset to start the count from zero. This is done to make sure that the second dataset indexes continue from the first dataset and don't start from zero again.

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	|1x3 Cross validator	None	NaN	None	NaN	None	None	None	None	None	NaN	NaN	NaN	None	None
1	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
2	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
3	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
4	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48838	27	Private	257302.0	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	<=50K
48839	40	Private	154374.0	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
48840	58	Private	151910.0	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	<=50K
48841	22	Private	201490.0	HS-grad	9.0	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	<=50K
48842	52	Self-emp-inc	287927.0	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	>50K

48843 rows × 15 columns

Once this is done, it could be noted that the row with index zero contains a fully null row.

This could be dropped using the code below.

```
df = df.drop(index=0)
df = df.reset_index(drop=True)
df
```

✓ 0.0s Python

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
1	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
2	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
3	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K
4	18	?	103497.0	Some-college	10.0	Never-married	?	Own-child	White	Female	0.0	0.0	30.0	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302.0	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	<=50K
48838	40	Private	154374.0	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
48839	58	Private	151910.0	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	<=50K
48840	22	Private	201490.0	HS-grad	9.0	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	<=50K
48841	52	Self-emp-inc	287927.0	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	>50K

Now, it is visible that the row with index zero has gone off. The data frame has 48842 rows and 15 columns.

Next, to see which ones are categorical columns and which are numerical columns, we could check the data types of each column in the data frame, as shown below.

```
df.dtypes
```

✓ 0.0s

```
age               object
workclass         object
fnlwgt           float64
education         object
education.num    float64
marital.status    object
occupation        object
relationship      object
race              object
sex               object
capital.gain     float64
capital.loss     float64
hours.per.week   float64
native.country    object
income            object
dtype: object
```

So based on the above results, the columns with "object" as the data type are categorical columns whereas the ones with "float64" have numerical values and therefore are numerical columns.

Next, the number of null values in each column is calculated.

```
df.isnull().sum()
✓ 0.0s

age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

This returns that none of the columns have null values in them. However, upon careful observation, it is noticeable that some values in the categorical columns have “?” in them which is irrelevant to the dataset.

Thus, to check which of these columns have the “?” in them, we could use the “Counter” extension from the “collections” library. Following this, we could add all the unique values of the categorical column to a dictionary and print them out.

```python
from collections import Counter

workclass_vals = dict(Counter(df['workclass'])).keys()
nationality_vals = dict(Counter(df['native.country'])).keys()
education_vals = dict(Counter(df['education'])).keys()
maritial_status_vals = dict(Counter(df['marital.status'])).keys()
occupation_vals = dict(Counter(df['occupation'])).keys()
relationship_vals = dict(Counter(df['relationship'])).keys()
race_vals = dict(Counter(df['race'])).keys()
sex_vals = dict(Counter(df['sex'])).keys()

# printing all the values for each column
print("Workclass: ", list(workclass_vals), '\n')
print("Nationality: ", list(nationality_vals), '\n')
print("Education levels: ", list(education_vals), '\n')
print("Maritial Status: ", list(maritial_status_vals), '\n')
print("Occupation: ", list(occupation_vals), '\n')
print("Relationship: ", list(relationship_vals), '\n')
print("Race: ", list(race_vals), '\n')
print("Sex: ", list(sex_vals), '\n')
```

This would allow us to see all the possible values for each categorical column in the data frame as shown below.

```
Workclass:  ['Private', 'Local-gov', '?', 'Self-emp-not-inc', 'Federal-gov', 'State-gov', 'Self-emp-inc', 'Without-pay', 'Never-worked']

Nationality:  ['United-States', '?', 'Peru', 'Guatemala', 'Mexico', 'Dominican-Republic', 'Ireland', 'Germany', 'Philippines', 'Thailand', 'Haiti', 'El-Salvador', 'Puerto-Rico', 'Vietn

Education levels:  ['11th', 'HS-grad', 'Assoc-acdm', 'Some-college', '10th', 'Prof-school', '7th-8th', 'Bachelors', 'Masters', 'Doctorate', '5th-6th', 'Assoc-voc', '9th', '12th', '1st-

Maritial Status:  ['Never-married', 'Married-civ-spouse', 'Widowed', 'Divorced', 'Separated', 'Married-spouse-absent', 'Married-AF-spouse']

Occupation:  ['Machine-op-inspct', 'Farming-fishing', 'Protective-serv', '?', 'Other-service', 'Prof-specialty', 'Craft-repair', 'Adm-clerical', 'Exec-managerial', 'Tech-support', 'Sal

Relationship:  ['Own-child', 'Husband', 'Not-in-family', 'Unmarried', 'Wife', 'Other-relative']

Race:  ['Black', 'White', 'Asian-Pac-Islander', 'Other', 'Amer-Indian-Eskimo']

Sex:  ['Male', 'Female']
```

Now, it is visible to us that the following columns have irrelevant values in them.

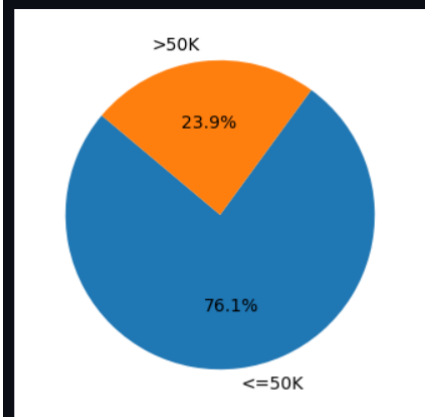
- Work class.
- Nationality
- Occupation

These null values could be handled in different ways; either they could be filled in using Simple Imputation or the rows with these values could be dropped. To come up with a decision for this, let us check the balance of the dataset by drawing a pie chart of the two categories of income: <=50K and >50K.

```
income_counts = df['income'].value_counts() # getting the number of occcurences for each unique value in the "income" column

# a pie chart is plotted to see for any imbalances in the output column
plt.figure(figsize=(4,4))
plt.pie(income_counts, labels=income_counts.index, autopct='%1.1f%%', startangle=140)
```

✓ 0.0s

```
([<matplotlib.patches.Wedge at 0x1a2099e6150>,
  <matplotlib.patches.Wedge at 0x1a2099e3850>],
 [Text(0.13270858417163964, -1.0919653985759616, '<=50K'),
  Text(-0.13270853305305033, 1.0919654047884977, '>50K')],
 [Text(0.07238650045725797, -0.5956174901323427, '76.1%'),
  Text(-0.07238647257439108, 0.5956174935209987, '23.9%')])
```



Now, this pie chart shows us that there is a severe imbalance in the dataset towards one category. This situation could be handled in two ways; we could either drop off the rows containing these question marks, or we could fill them in using Simple Imputation. Simple Imputation is a technique used to fill in null values by using statistical measures such as the mean, mode or median.

In our dataset, if we use the Simple Imputation technique, it could tend to cause a further bias towards one variable as the mean, mode and median would usually point towards the majority class. Therefore, we could drop off the rows containing these irrelevant values.

```python
df[df == '?'] = np.nan # dropping the rows with "?"
df = df.dropna(axis=0)
df
```

[9] Python

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	<=50K
1	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	<=50K
2	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
3	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	>50K
5	34	Private	198693.0	10th	6.0	Never-married	Other-service	Not-in-family	White	Male	0.0	0.0	30.0	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302.0	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	<=50K
48838	40	Private	154374.0	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	>50K
48839	58	Private	151910.0	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	<=50K
48840	22	Private	201490.0	HS-grad	9.0	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	<=50K
48841	52	Self-emp-inc	287927.0	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	>50K

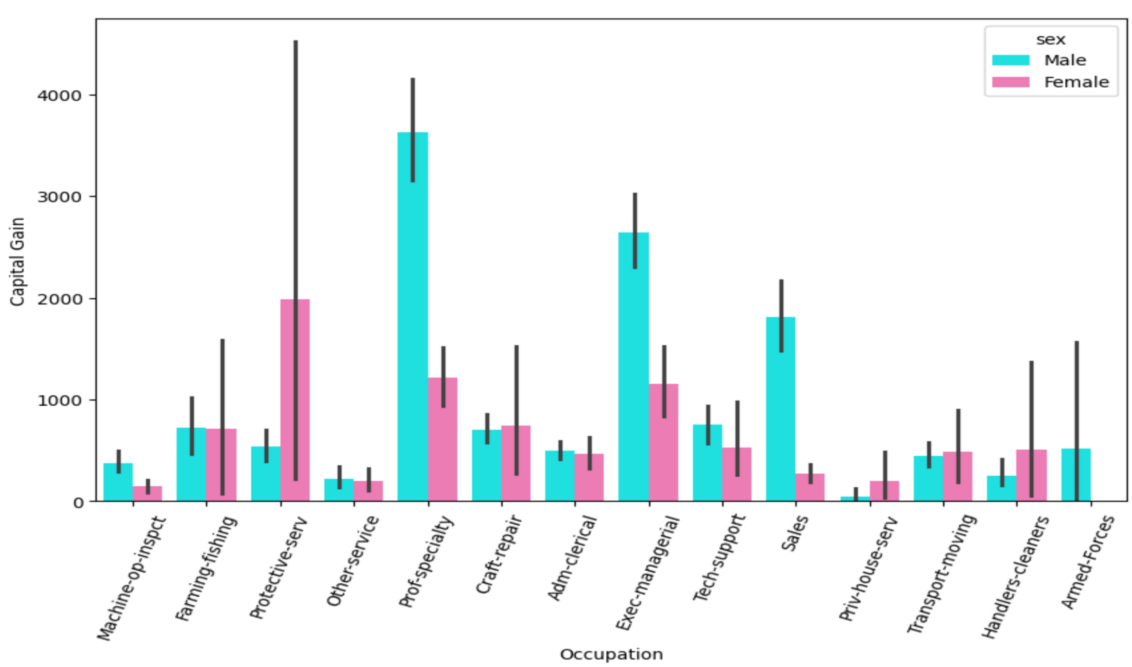
45222 rows × 15 columns

This is how the data frame looks after these rows have been dropped. The number of rows decreases to 45222 but the number of columns remain the same.

2.3.Exploratory Data Analysis

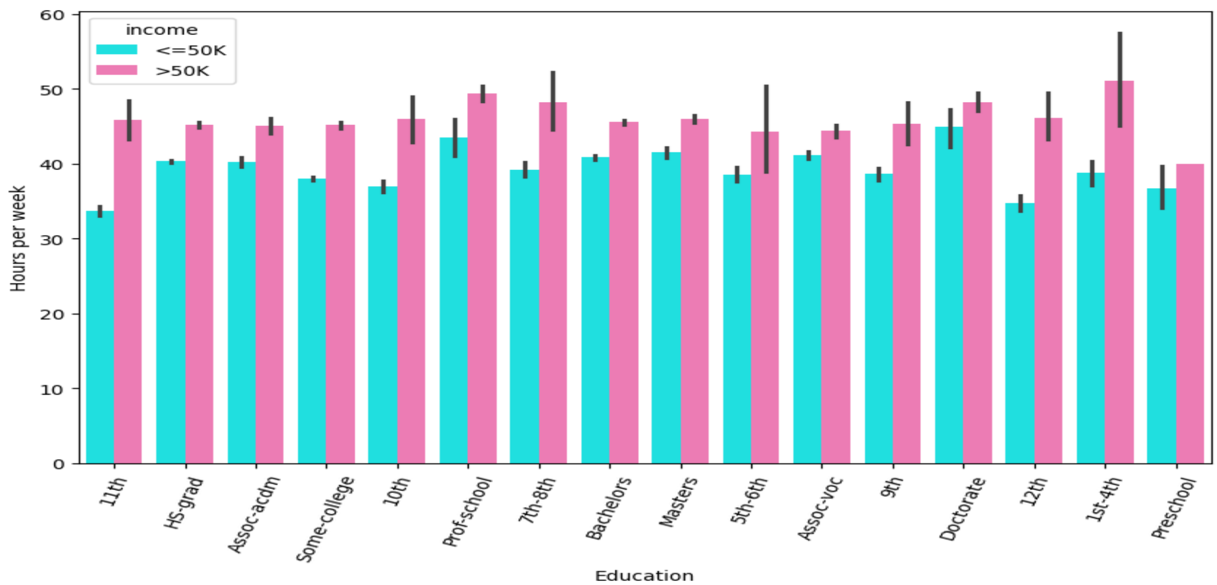
Exploratory Data Analysis is the process of analyzing trends and characteristics of a given dataset so that they could be used for training and testing purposes of the machine learning model. This involves plotting visual graphs, detecting outliers, identifying potential relationships between features, and obtaining statistical measures.

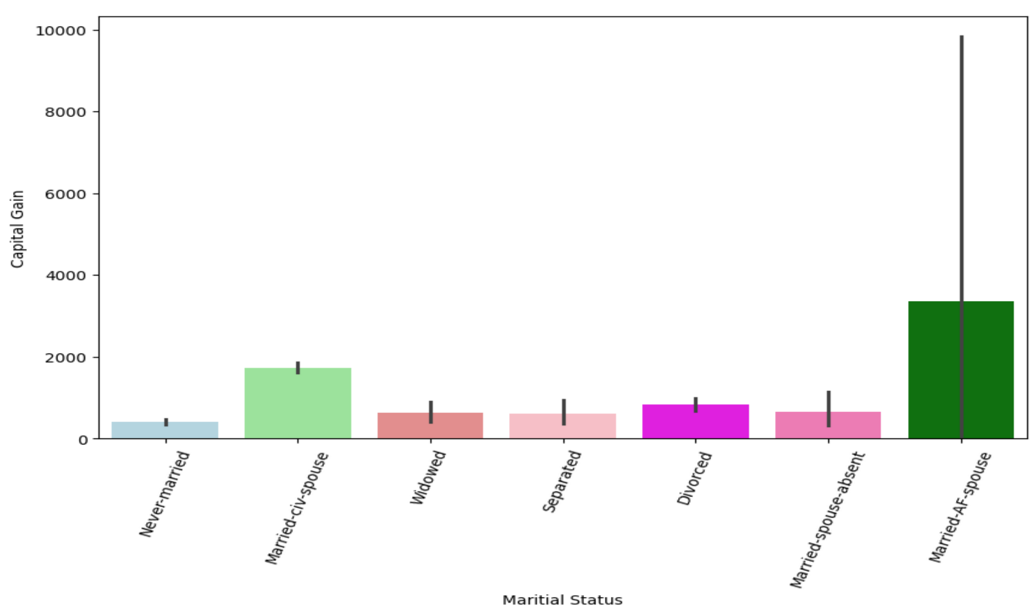
Let us draw a few bar plots to analyze how different features affect our target variable.



The plot above shows how the capital gain changes based on the change in occupation in both males and females. In males, “Professional specialty” has the highest capital gain whereas “Private house servant” has the lowest capital gain. However, in females, “Protective servant” has the highest capital gain and “Machine operation inspector” has the lowest capital gain.

Next, let us draw a bar plot to show the variation of hours per week for the different levels of education for both categories of income.





The above plot shows the variation of capital gain based on the marital status. It could be seen that individuals who were never married tend to have the lowest capital gain. However, Married-AF-spouse individuals have the highest capital gain.

Overall, in summary we could come up with the following conclusions regarding the dataset. The dataset did not have any null values. However, it did have irrelevant values in the form of "?". These had to be removed to prevent a bias being created towards one variable. These were found in the "workclass", "native.country" and "occupation" columns. Once the rows containing these values were dropped, the number of rows in the data frame dropped from 48,842 to 45,222. This shows there were 3,621 rows containing these values. It was also noted that there was a severe imbalance in the dataset towards the <=$50K class.

2.4. Formatting Data Frame

The data frame must be formatted in such a way that it could be used to solve the classification problem. We have two main classification classes in our case study: <=50K and >50K. This depicts that our target column is going to be the "income" column. If a closer look is taken into this column, we can see they have been filled with string values. This will not help in solving our classification problem as it is necessary to convert them into numerical classes.

```
# the values are replaced with the respective new values in the income column
df['income'].replace({'<=50K':0, '>50K':1}, inplace=True)
df
```

As shown in the code above, if the income column value is <=50K, then it is replaced with 0 and if the income column value is >50K, then it is replaced with 1.

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	25	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	0
1	38	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	0
2	28	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	1
3	44	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	1
5	34	Private	198693.0	10th	6.0	Never-married	Other-service	Not-in-family	White	Male	0.0	0.0	30.0	United-States	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302.0	Assoc-acdm	12.0	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	0
48838	40	Private	154374.0	HS-grad	9.0	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	1
48839	58	Private	151910.0	HS-grad	9.0	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	0
48840	22	Private	201490.0	HS-grad	9.0	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	0
48841	52	Self-emp-inc	287927.0	HS-grad	9.0	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	1

45222 rows × 15 columns

This is how the data frame looks after this has been done. The income column has only 1 and 0 in it.

2.5.Feature Selection and Engineering

Feature selection is the process by which the necessary features are selected based on their importance to run the machine learning algorithms. This involves dropping off unnecessary features and normalizing the data frame.

We could say that the column “education.num” could prove to be useless in this classification problem. Thus, we can drop off this column.

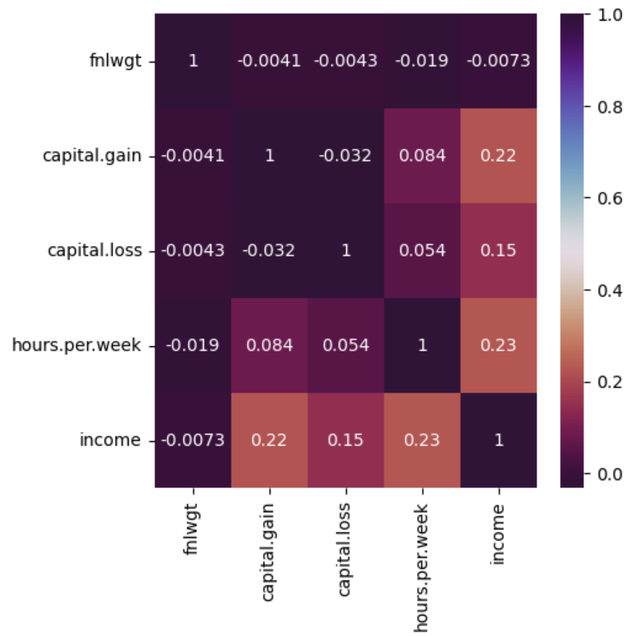
	age	workclass	fnlwgt	education	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	25	Private	226802.0	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	0
1	38	Private	89814.0	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	0
2	28	Local-gov	336951.0	Assoc-acdm	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	1
3	44	Private	160323.0	Some-college	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	1
5	34	Private	198693.0	10th	Never-married	Other-service	Not-in-family	White	Male	0.0	0.0	30.0	United-States	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302.0	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	0
48838	40	Private	154374.0	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	1
48839	58	Private	151910.0	HS-grad	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	0
48840	22	Private	201490.0	HS-grad	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	0
48841	52	Self-emp-inc	287927.0	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	1

45222 rows × 14 columns

It is visible that after this was done, the number of columns in the data frame decreased to 14, whereas the number of rows remains the same.

Next, we could draw a heatmap for all the numerical columns in the data frame. A heatmap helps to analyze the following:

- Easier to identify relationships between numeric variables in the dataset. A very high negative or positive value suggests a strong correlation whereas values close to zero indicate a weak correlation.
- It helps to decide which features to include or exclude in a model.



Next, we can use covariance to find the relationship between continuous variables and a binary variable (variable which takes either zero or one). In our dataset, "income" column is the binary variable.

```
columns = ['age', 'capital.loss', 'capital.gain', 'hours.per.week', 'fnlwgt']

df['income'] = pd.to_numeric(df['income'], errors='coerce')

for column in columns:
    df[column] = pd.to_numeric(df[column], errors='coerce')

    cov_value = np.cov(df['income'], df[column])[0, 1]  # Calculate covariance between income and the current column

    # if the value is greater than zero, it is positive or else it is negative correlation
    if cov_value > 0:
        status = "positive covariance"
    else:
        status = "negative covariance"

    print(column, ":", cov_value, " - ", status)  # Print column name, covariance value, and correlation status
```

As shown in the image above, the continuous variable columns have been added into a list and for each element in the list, the covariance is calculated.

If the covariance value is greater than zero, then it is a positive covariance, which signifies that when the continuous variable increases, the income increases. However, if the covariance is less than zero, then it is a negative covariance, which signifies an inverse relationship between the variables.

```
age : 1.3527934869548128  -  positive covariance
capital.loss : 25.99733648682098  -  positive covariance
capital.gain : 716.3744655913118  -  positive covariance
hours.per.week : 1.1778975840252774  -  positive covariance
fnlwgt : -331.3169579333376  -  negative covariance
```

These are the results produced. This shows that only the "fnlwgt" column has a negative covariance. We could confirm this using the Logistic Regression from the stats model library.

```
import statsmodels.api as sm

X = sm.add_constant(df[['age', 'capital.loss', 'capital.gain', 'hours.per.week', 'fnlwgt']])
y = df['income'] # the target variable is assigned
logistic_model = sm.Logit(y, X) # a logistic regression model instance is created
result = logistic_model.fit() # the model is fitted
print(result.summary()) # print the summary
```

In this code, the target variable has been assigned as the income column. An instance of the logistic regression is created and each of the continuous variable columns is passed as parameters along with the target variable. The model is fitted, and the summary of results is printed to be analyzed.

```
Optimization terminated successfully.
         Current function value: 0.456647
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:                 income   No. Observations:                45222
Model:                          Logit   Df Residuals:                    45216
Method:                           MLE   Df Model:                            5
Date:                Tue, 05 Mar 2024   Pseudo R-squ.:                  0.1845
Time:                        11:56:42   Log-Likelihood:                -20650.
converged:                       True   LL-Null:                       -25322.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -4.9140      0.072    -68.584      0.000      -5.054      -4.774
age                0.0393      0.001     40.639      0.000       0.037       0.041
capital.loss       0.0008   2.53e-05     30.150      0.000       0.001       0.001
capital.gain       0.0003   7.63e-06     42.998      0.000       0.000       0.000
hours.per.week     0.0432      0.001     39.791      0.000       0.041       0.045
fnlwgt          3.816e-07   1.17e-07      3.263      0.001    1.52e-07    6.11e-07
==================================================================================
```

Analyzing these results, it is visible that all the continuous variable columns have an impact on the target column. This could be confirmed as the coefficient is not equal to zero.

However, it is noteworthy that the “fnlwgt” column has a very minimal coefficient value. Since the value is not equal to zero, we know it plays some impact on the target column, but this may seem to be insignificant in this problem scenario.

Thus, we can drop the “fnlwgt” column and this is how the data frame would look after this is done.

	age	workclass	education	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	25	Private	11th	Never-married	Machine-op-inspct	Own-child	Black	Male	0.0	0.0	40.0	United-States	0
1	38	Private	HS-grad	Married-civ-spouse	Farming-fishing	Husband	White	Male	0.0	0.0	50.0	United-States	0
2	28	Local-gov	Assoc-acdm	Married-civ-spouse	Protective-serv	Husband	White	Male	0.0	0.0	40.0	United-States	1
3	44	Private	Some-college	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688.0	0.0	40.0	United-States	1
5	34	Private	10th	Never-married	Other-service	Not-in-family	White	Male	0.0	0.0	30.0	United-States	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Wife	White	Female	0.0	0.0	38.0	United-States	0
48838	40	Private	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0.0	0.0	40.0	United-States	1
48839	58	Private	HS-grad	Widowed	Adm-clerical	Unmarried	White	Female	0.0	0.0	40.0	United-States	0
48840	22	Private	HS-grad	Never-married	Adm-clerical	Own-child	White	Male	0.0	0.0	20.0	United-States	0
48841	52	Self-emp-inc	HS-grad	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024.0	0.0	40.0	United-States	1

45222 rows × 13 columns

We can see the number of columns has reduced from 14 to 13 but the number of rows remains the same.

2.6. Data Preprocessing

Next, we must preprocess the data so that it can be inserted into the machine learning algorithms.

First, we could commence by using One Hot Encoding to create binary columns for each categorical value in a categorical column. For example, if a categorical column had five unique values in it, each of these variables is assigned an individual binary column for them. These columns only take the values True and False.

```
categorical_columns = df.columns[df.dtypes == object].tolist() # the columns are converted into a list
df = pd.get_dummies(df, columns=categorical_columns)
df
```

Here, we convert each categorical column into a list using the “toList()” function. Following this, we create columns for each of them using the function “get_dummies()” of the pandas library.

	age	capital.gain	capital.loss	hours.per.week	income	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	...	native.country_Portugal	native.country_Puer R
0	25	0.0	0.0	40.0	0	False	False	True	False	False	...	False	F
1	38	0.0	0.0	50.0	0	False	False	True	False	False	...	False	F
2	28	0.0	0.0	40.0	1	False	True	False	False	False	...	False	F
3	44	7688.0	0.0	40.0	1	False	False	True	False	False	...	False	F
5	34	0.0	0.0	30.0	0	False	False	True	False	False	...	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	
48837	27	0.0	0.0	38.0	0	False	False	True	False	False	...	False	F
48838	40	0.0	0.0	40.0	1	False	False	True	False	False	...	False	F
48839	58	0.0	0.0	40.0	0	False	False	True	False	False	...	False	F
48840	22	0.0	0.0	20.0	0	False	False	True	False	False	...	False	F
48841	52	15024.0	0.0	40.0	1	False	False	False	True	False	...	False	F

45222 rows × 103 columns

This is how the data frame looks after the One Hot Encoding has been performed. It is visible that each variable in a categorical column has been given a column. This increases the number of columns to 104 but the number of rows remains the same.

Next, let us preprocess the numerical columns. For this, we could Min Max Scaler. This technique is designed especially for numerical columns. This assigns a value between zero and one for each row in the column based on its magnitude.

```
from sklearn.preprocessing import MinMaxScaler

scalable_columns = ['age','capital.gain','capital.loss','hours.per.week'] #columns with numerical values
min_max_scaler = MinMaxScaler() # creating an instance of the Min Max Scaler
scaled_columns = min_max_scaler.fit_transform(df[scalable_columns])

# now the values could be assigned back to the respective column in the DataFrame
df['age']=scaled_columns[:,0]
df['capital.gain']=scaled_columns[:,1]
df['capital.loss']=scaled_columns[:,2]
df['hours.per.week']=scaled_columns[:,3]

df
```

The Min Max Scaler is imported from the preprocessing extension of the scikit learn library. The scalable numerical columns are defined, and an instance of the Min Max Scaler is created. Following this, the columns are scaled by fitting the scalable columns inside the Min Max Scaler. Finally, the values are assigned back to their respective columns.

	age	capital.gain	capital.loss	hours.per.week	income	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	...	native.country_Portugal	native.country_
0	0.109589	0.000000	0.0	0.397959	0	False	False	True	False	False	...	False	
1	0.287671	0.000000	0.0	0.500000	0	False	False	True	False	False	...	False	
2	0.150685	0.000000	0.0	0.397959	1	False	True	False	False	False	...	False	
3	0.369863	0.076881	0.0	0.397959	1	False	False	True	False	False	...	False	
5	0.232877	0.000000	0.0	0.295918	0	False	False	True	False	False	...	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	
48837	0.136986	0.000000	0.0	0.377551	0	False	False	True	False	False	...	False	
48838	0.315068	0.000000	0.0	0.397959	1	False	False	True	False	False	...	False	
48839	0.561644	0.000000	0.0	0.397959	0	False	False	True	False	False	...	False	
48840	0.068493	0.000000	0.0	0.193878	0	False	False	True	False	False	...	False	
48841	0.479452	0.150242	0.0	0.397959	1	False	False	False	True	False	...	False	

45222 rows × 103 columns

This is how the data frame looks after the Min Max Scaler has been applied. It could be seen in columns such as “age” and “capital.gain” that the values are between zero and one throughout. However, it is also noteworthy that the dimensions (rows and columns) of the data frame remain the same. Min Max Scaler does not affect the dimensions of the data frame unlike One Hot Encoder.

Now, our data frame has been fully preprocessed and is ready to be inserted into the machine learning algorithms to be trained.

3. Solution Methodology

3.1. Naïve Bayes Classifier Model

The Naïve Bayes model is a Machine Learning algorithm which works on the principles of the Bayes’ theorem. It is a simple algorithm which makes predictions based on probabilities.

The Bayes theorem derives the probability of the occurrence of an event based on prior knowledge about the occurrence of the event and its conditions.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE

gaussian_model = GaussianNB() # creating an instance of the Gaussian Naive Bayes model

x = df.drop('income', axis=1) # the input features are defined
y = df['income'] # the output feature (income) is defined

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split is taken
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

gaussian = gaussian_model.fit(x_train, y_train) # the model is trained
prediction_gaussian = gaussian.predict(x_test) # the model is used to predict the output

accuracy_gaussian = accuracy_score(y_test, prediction_gaussian) # the accuracy of the model is calculated
print("Accuracy of the model is: ", accuracy_gaussian) # the accuracy is printed

print(classification_report(y_test, prediction_gaussian)) # the classification report is printed

confusion_matrix(y_test, prediction_gaussian) # the confusion matrix is printed
```

First, the Gaussian Naïve Bayes model is imported from the naïve bayes extension of the scikit-learn library. Along with this, other specifications such as the train test split, metrics such as accuracy score, classification report and confusion metrics are imported.

Next, an instance of the model is created, and the input and output features are defined. Following this, the train and test split are defined. Twenty percent of the dataset has been allocated for the testing purpose. The default reproducibility value of 42 has been specified to ensure consistency over several executions. Synthetics Minority Oversampling Technique has been used to handle the imbalance of the dataset. This technique creates additional samples of the minority to class to populate the said minority class to create a balance in the dataset. This technique has been used on the training component as it would be futile on the testing component.

Following this, the model is trained on the training component and its performance is put to the test by making predictions on the testing component. Finally, the accuracy score, classification report and the confusion matrix are returned as outputs.

The model produced an accuracy score of 60.95%.

3.2.Random Forest Classifier Model

The Random Forest model is a type of ensemble machine learning algorithm from the Sci-kit learn library which could be used to solve both classification and regression problems. It runs on the back of multiple decision trees during the training phase and finally the prediction of each decision tree is merged to provide the best possible result.

Due to the usage of several decision trees, the overfitting of the model can be limited to a certain extent.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, r2_score, classification_report, confusion_matrix

rf_model = RandomForestClassifier(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1) # the input features are defined
y = df['income'] # the output feature (income) is defined

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42) # 70% training split is taken

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data
predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated
print(f"Accuracy is {accuracy_rf}") # the accuracy is printed


r2_val_rf = r2_score(y_test, predictions_rf) # the r2 score is calculated
print(f"r2 score is {r2_val_rf}") # the r2 score is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed
```

This code snippet shows how the initial Random Forest classifier model is built. The model is imported from the Sci-kit learn library along with other necessary features such as the training testing split and evaluation metrics such as the accuracy score, classification report and confusion matrix.

An instance of the model is created with the reproducibility seed set to 42, and the input and output features are defined. As in the Naïve Bayes model, the input feature would be all the features of the data frame except the income. The output feature would be the income as that is the feature which has to be predicted.

Following this, the dataset is split into its training and testing components. For this instance, a split of 0.3 is used. This means that 70% of the dataset is used for training and 30% is used for testing. A reproducibility seed of 42 has been set to obtain consistent results across multiple runs.

Next, the model is trained by passing the training components of the input and output features. Then, the model makes its predictions on the testing component of the input features.

Then, the metrics are returned as outputs. This model produced an accuracy score of 82.77%.

We could try to improve the performance of the model by tuning the hyperparameters. First, let us try and change the train test split of the model.

```
rf_model = RandomForestClassifier(random_state=42)

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data
predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated
print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed
```

In this model, a 20% testing split has been defined. This is the only change which has been made from the previous model. This model returned an accuracy score of 82.786%. This is a minor but negligible increase in the accuracy score, so we could stick to either of the splits.

In Random Forest models, there are other hyperparameters which could be tuned to obtain a more optimal model.

1. Minimum samples split- This is the minimum number of samples required to split an internal node when a decision tree is being generated.
2. Maximum depth of decision trees- This is the maximum depth of each tree in the Random Forest model.

```
rf_model = RandomForestClassifier(min_samples_split=10, max_depth=10, random_state=42)

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data
predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated
print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed
```

In this mode, a minimum sample split of 10 is assigned, and the maximum depth of the decision trees is also set at 10. Apart from this, the training sample size of 80% is used and no other changes are made.

This model produced an accuracy of 77.30% which is comparatively lower by a significant margin.

Now, we could try to increase these parameters individually and see how they affect the performance of the model.

```
rf_model = RandomForestClassifier(min_samples_split=20, max_depth=10, random_state=42)

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data
predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated
print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed
```

In this instance of the model, the minimum samples split has been increased to 20 from the original value of 10. No other changes have been made in terms of the hyperparameters, model or the dataset.

Making this change caused the accuracy score of the model to decrease slightly to 77.28%. Thus, we could revert to the original value.

```
rf_model = RandomForestClassifier(min_samples_split=10, max_depth=20, random_state=42)

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data
predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated
print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed
```

This instance of the model assigns 20 as the maximum depth of the decision trees. The other hyperparameters have been reverted to their initial values. This change causes the accuracy score to increase significantly to 81.88%.

3.3.Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

lr = lr_model.fit(x_train, y_train) # the model is trained using the training data
predictions_lr = lr.predict(x_test) # the model is tested using the testing inputs

accuracy_lr = accuracy_score(y_test, predictions_lr) # the accuracy score is calculated
print(f"Accuracy is {accuracy_lr}") # the accuracy is printed

print(classification_report(y_test, predictions_lr)) # the classification report is printed

confusion_matrix(y_test, predictions_lr) # the confusion matrix is printed
```
✓ 4.0s

The logistic regression library is imported from the linear model extension of the scikit-learn library. An instance of the logistic regression model is created, and the input and output features are defined.

Following this, the train and test split is specified; 20% of the whole data set is to be used for training purposes. The Synthetic Minority Oversampling Technique is used on the training data to handle the imbalance in the data set and oversample the minority class to obtain a balance.

Following this, the model is trained on the input data and is tested with the testing input component. The accuracy score, classification report and the confusion matrix are returned.

The model ended up with an accuracy of 81.50%.

3.4.XGBoost Classifier Model

```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

xgb = xgb_model.fit(x_train, y_train) # the model is trained using the training data
predictions_xgb = xgb.predict(x_test) # the model is tested using the testing inputs

accuracy_xgb = accuracy_score(y_test, predictions_xgb) # the accuracy score is calculated
print(f"Accuracy is {accuracy_xgb}") # the accuracy is printed

print(classification_report(y_test, predictions_xgb)) # the classification report is printed

confusion_matrix(y_test, predictions_xgb) # the confusion matrix is printed
```

The XGB Classifier model is imported from the xgboost library. An instance of the model is created, and the random state has been set to the default value of 42, to ensure consistency in the performance of the model.

Next, the input and output features of the dataset are defined, and the train test split is provided; 20% of the data set is provided for the testing purposes. Following this, Synthetic Minority Oversampling Technique is used to oversample the minority class of the data set to handle the imbalance.

Now the model is trained using the training component and is tested on the testing input component. Finally, the accuracy score, classification report and the confusion matrix are returned as outputs.

The model produces an accuracy score of 84.99%, which proves to be the highest of all the models implemented.

4. Model Analysis and Evaluation

Analyzing the performance of the models allows us to come up with the best decision as to which model could suit our purpose the best based on the metrics.

The below table analyzes the performance of each of the models created.

Name of Model	Train component	Hyperparameters	Accuracy Score
Naïve Bayes	20%	N/A	60.95%
Random Forest Classifier	30%	N/A	82.77%
Random Forest Classifier	20%	N/A	82.79%
Random Forest Classifier	20%	Minimum samples split- 10. Maximum depth of decision trees- 10.	77.30%
Random Forest Classifier	20%	Minimum samples split- 20.	77.28%

		Maximum depth of decision trees- 10.	
Random Forest Classifier	20%	Minimum samples split- 10. Maximum depth of decision trees- 20.	81.88%
Logistic Regression	20%	N/A	81.47%
XGBoost Classifier	20%	N/A	85.27%

We can define the efficiency and the predictability of our models by using techniques such as the ROC AUC score and plotting these values.

AUC is the area under the ROC curve. If a completely perfect classifier model is built, it would provide an AUC of 1.0. This shows that it has a 0% false negative rate and a 100% true positive rate. The standard random classifier produces an AUC of 0.5.

ROC curve is used to represent the variation between the true positive rate and the false negative rate.

We could plot an ROC curve and analyze the area under this curve for each of the models.

```
from sklearn.metrics import roc_curve, roc_auc_score, auc

x = df.drop('income', axis=1)
y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 70% training split
x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

gnb = GaussianNB() # creating an instance of the Gaussian Naive Bayes model
rf = RandomForestClassifier(random_state=42) # creating an instance of the Random Forest model
lr = LogisticRegression(random_state=42) # creating an instance of the Logistic Regression model
xgb = XGBClassifier(random_state=42) # creating an instance of the XGBoost model

gnb.fit(x_train, y_train) # the Gaussian Naive Bayes model is trained
rf.fit(x_train, y_train) # the Random Forest model is trained
lr.fit(x_train, y_train) # the Logistic Regression model is trained
xgb.fit(x_train, y_train) # the XGBoost model is trained

prediction_gaussian = gnb.predict_proba(x_test)[:,1] # the Gaussian Naive Bayes model is used to predict the output
predictions_rf = rf.predict_proba(x_test)[:,1] # the Random Forest model is used to predict the output
predictions_lr = lr.predict_proba(x_test)[:,1] # the Logistic Regression model is used to predict the output
predictions_xgb = xgb.predict_proba(x_test)[:,1] # the XGBoost model is used to predict the output
```

The necessary extensions are imported for this purpose. The input and output features are defined, and the training and testing split is specified. A testing size of 0.2 is provided in this scenario.

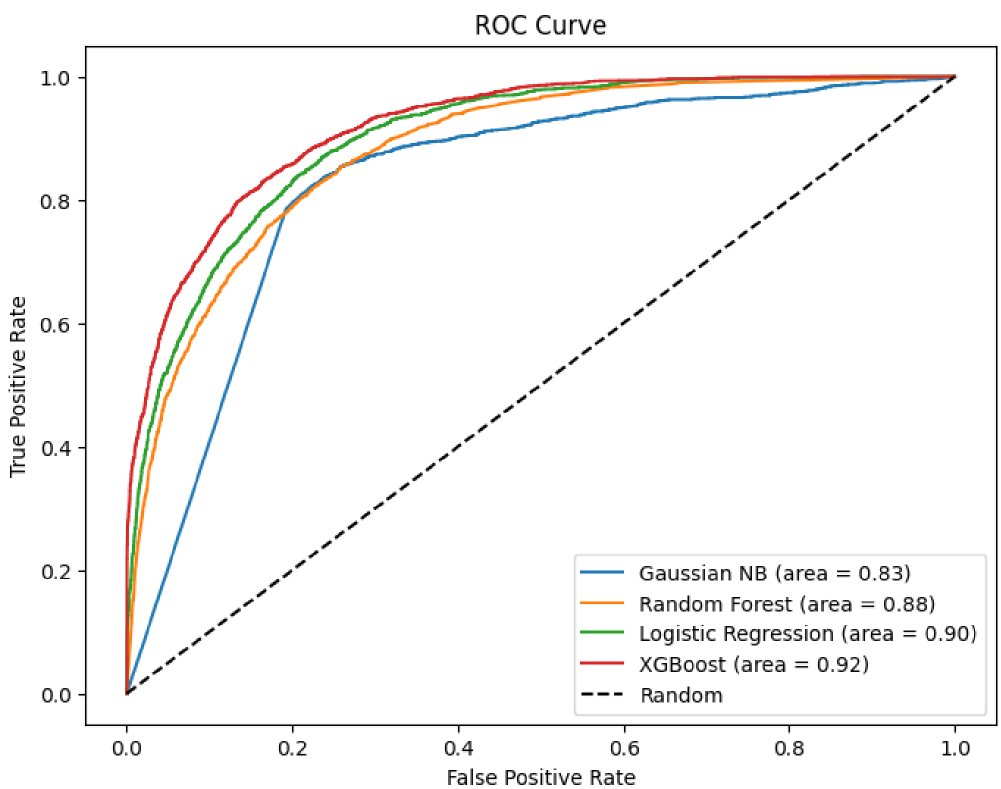
Synthetic Minority Oversampling Technique is used to oversample the minority class. Following this, instances of the Gaussian Naïve Byes, Random Forest Classifier, Logistic Regressor and XGBoost Classifier are created. Next, these models are trained using the training component. Following this, the models are made to produce outputs on the testing component of the dataset.

Following this, the ROC AUC score is calculated using the inbuilt function of scikit-learn and is printed into the console.

```
# lets us check the scores now
gaussian_roc_auc = roc_auc_score(y_test, prediction_gaussian)
print(f"ROC AUC score of the Gaussian NB model is {gaussian_roc_auc}")

rf_roc_auc = roc_auc_score(y_test, predictions_rf)
print(f"ROC AUC score of the Random Forest model is {rf_roc_auc}")

lr_roc_auc = roc_auc_score(y_test, predictions_lr)
print(f"ROC AUC score of the Logistic Regression model is {lr_roc_auc}")

xgb_roc_auc = roc_auc_score(y_test, predictions_xgb)
print(f"ROC AUC score of the XGBoost model is {xgb_roc_auc}")
```

Finally, the ROC curve is plotted once again using the inbuilt function of the scikit-learn library and the axes labels, legend and the title of the plot is specified.

```
fpr_gnb, tpr_gnb, _ = roc_curve(y_test, prediction_gaussian)
fpr_rf, tpr_rf, _ = roc_curve(y_test, predictions_rf)
fpr_lr, tpr_lr, _ = roc_curve(y_test, predictions_lr)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, predictions_xgb)

plt.figure(figsize=(8,6))
plt.plot(fpr_gnb, tpr_gnb, label=f"Gaussian NB (area = {gaussian_roc_auc:.2f})")
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (area = {rf_roc_auc:.2f})")
plt.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (area = {lr_roc_auc:.2f})")
plt.plot(fpr_xgb, tpr_xgb, label=f"XGBoost (area = {xgb_roc_auc:.2f})")

plt.plot([0,1],[0,1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.title('ROC Curve')
```



This is the final ROC AUC curve. The dotted black line shows the Random Classifier which has an AUC of 0.5. The other lines represent each of the four models implemented. Based on the legend of the graph, we can conclude that the XGBoost classifier has the greatest area under the curve whereas the Gaussian Naïve Bayes has the lowest area under the curve. Thus, we can rank the machine learning models in ascending performance as below.

1. Gaussian Naïve Bayes
2. Random Forest Classifier
3. Logistic Regression Model
4. XGBoost Classifier

5. References

1. Scikit-learn (2018). *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.
2. scikit-learn.org. (n.d.). *sklearn.naive_bayes.GaussianNB — scikit-learn 0.22.1 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.
3. scikit-learn (2014). *sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.
4. xgboost.readthedocs.io. (n.d.). *XGBoost Documentation — xgboost 1.5.0 documentation*. [online] Available at: https://xgboost.readthedocs.io/.
5. Sarang Narkhede (2018). *Understanding Logistic Regression*. [online] Medium. Available at: https://towardsdatascience.com/understanding-logistic-regression-9b02c2aec102.

6. Source Code

6.1. Exploratory Data Analysis

This notebook is to perform the Exploratory Data Analysis.

First we have to import the necessary libraries.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

Next we have to merge the 2 datasets vertically after the column names have been added.

```
# create a list containing the column names

column_names=
["age","workclass","fnlwgt","education","education.num","marital.status","occupation","rela
tionship","race","sex","capital.gain","capital.loss",

            "hours.per.week","native.country","income"]

# convert the .txt files to .csv

train = pd.read_csv('dataset/adult_data.txt', sep=",\s", header=None, names=column_names,
engine='python')

test = pd.read_csv('dataset/adult_test.txt', sep=",\s", header=None, names=column_names,
engine='python')

test['income'].replace(regex=True,inplace=True,to_replace=r'\.',value=r'')


# merge the 2 data frames vertically

df = pd.concat([test,train])

df.reset_index(inplace=True, drop=True) # the index is reset

df
```

We can see that the 0th row has irrelevant values.

```
df = df.drop(index=0)

df = df.reset_index(drop=True)
```

df

Let us see the data type of each column in the dataset.

df.dtypes

Next let us see the number of null values in each column.

df.isnull().sum()

Upon careful observation, it could be seen that though the code provides that there are no null values, some of the values have been filled with just "?".

Let us check which of these columns have the "?" in them. This could be done by checking the possible values in each categorical column.

```
from collections import Counter

workclass_vals = dict(Counter(df['workclass'])).keys()

nationality_vals = dict(Counter(df['native.country'])).keys()

education_vals = dict(Counter(df['education'])).keys()

maritial_status_vals = dict(Counter(df['marital.status'])).keys()

occupation_vals = dict(Counter(df['occupation'])).keys()

relationship_vals = dict(Counter(df['relationship'])).keys()

race_vals = dict(Counter(df['race'])).keys()

sex_vals = dict(Counter(df['sex'])).keys()


# printing all the values for each column

print("Workclass: ", list(workclass_vals), '\n')

print("Nationality: ", list(nationality_vals), '\n')

print("Education levels: ", list(education_vals), '\n')

print("Maritial Status: ", list(maritial_status_vals), '\n')

print("Occupation: ", list(occupation_vals), '\n')
```

print("Relationship: ", list(relationship_vals), '\n')

print("Race: ", list(race_vals), '\n')

print("Sex: ", list(sex_vals), '\n')

Let us see the balance of the dataset for the 2 classes to see how these null values could be handled.

income_counts = df['income'].value_counts() # getting the number of occcurences for each unique value in the "income" column

Define custom colors for the pie chart slices

custom_colors = ['hotpink', 'lightgreen']

a pie chart is plotted to see for any imbalances in the output column

plt.figure(figsize=(4,4))

plt.pie(income_counts, labels=income_counts.index, autopct='%1.1f%%', startangle=140, colors=custom_colors)

This displays an imbalance in the dataset as there is 76.1% occurrence of one category while the other category has an occurence of only 23.9%.

Let us drop the rows which have a "?" in them.

df[df == '?'] = np.nan # dropping the rows with "?"

df = df.dropna(axis=0)

df

Next to analyze the distribution and make visualizations, a few plots could be drawn based on the dataset.

plotting a figure to see the change of capital gain with the change in occupation for each sex.

Define a custom color palette

color_palette = {"Male": "cyan", "Female": "hotpink"}

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

sns.barplot(x="occupation", y="capital.gain", data=df, hue="sex", palette=color_palette)

```
plt.xlabel("Occupation")
plt.ylabel("Capital Gain")
plt.xticks(rotation=70)

# plotting a figure of education against the hours per week based on the income.
# Define a custom color palette
color_palette = {"<=50K": "cyan", ">50K": "hotpink"}
plt.figure(figsize=(10,6))
sns.barplot(x="education", y="hours.per.week", data=df, hue="income",
palette=color_palette)
plt.xlabel("Education")
plt.ylabel("Hours per week")
plt.xticks(rotation=70)

# plotting a figure of maritial status against income.
# Define custom colors for the pie chart slices
custom_colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightpink', 'magenta', "hotpink",
"green"]
plt.figure(figsize=(10,6))
sns.barplot(x="marital.status", y="capital.gain", data=df, palette=custom_colors)
plt.xlabel("Maritial Status")
plt.ylabel("Capital Gain")
plt.xticks(rotation=70)
```

This provides an overall summary of our dataset. We can note the below points regarding the dataset.

* The dataset had 3,621 null values in the form "?" in the "workclass", "native.country" and "occupation" columns which were dropped.

* There is an imbalance in the dataset where more than 75% of the outcome is <=50K.

* A few more bar plots were plotted expecting additional information regarding the dataset and possible relationships between features.

6.2.Final Prediction Notebook

This notebook will be used to run the ML algorithms.

First let us import the necessary libraries.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

Next we have to merge the 2 datasets vertically after the column names have been added.

```
# create a list containing the column names

column_names=
["age","workclass","fnlwgt","education","education.num","marital.status","occupation","relation
ship","race","sex","capital.gain","capital.loss","hours.per.week","native.country","income"]


# convert the .txt files to .csv

train = pd.read_csv('dataset/adult_data.txt', sep=",\s", header=None, names=column_names,
engine='python')

test = pd.read_csv('dataset/adult_test.txt', sep=",\s", header=None, names=column_names,
engine='python')

test['income'].replace(regex=True,inplace=True,to_replace=r'\.',value=r'')


# merge the 2 data frames vertically

df = pd.concat([test,train])

df.reset_index(inplace=True, drop=True) # the index is reset

df
```

We can see that the 0th row has irrelevant values.

df = df.drop(index=0)

df = df.reset_index(drop=True)

df

Data Preprocessing

Let us check the dataset for null values.

df.isnull().sum()

Upon careful observation, it could be seen that though the code provides that there are no null values, some of the values in the columns with data type have been filled with just "?".

df[df == '?'] = np.nan # dropping the rows with "?"

df = df.dropna(axis=0)

df

The classification problem requires us to predict whether the income exceeds $50,000 per year. So if the income value is '<=50K', then the value is replaced with 0, else if the income value is '>50K', then the value is replaced with 1.

the values are replaced with the respective new values in the income column

df['income'].replace({'<=50K':0, '>50K':1}, inplace=True)

df

Feature Selection

Now we have to check which features of the dataset exactly contribute to the prediction.

It could be noted that the column "education.num" could prove to be useless in this particular classification problem as it is redunadant with the "education" column. So this column could be dropped.

df = df.drop('education.num', axis=1) # the column is dropped

df

numeric_df = df.select_dtypes(include='number') # only the numeric columns have to be selected

corrmat = numeric_df.corr() # the correlation matrix is calculated

plt.figure(figsize=(5,5)) # the size of the figure is specified

sns.heatmap(corrmat, annot=True, cmap='twilight_shifted_r') # the heatmap is plotted

plt.show()

Covariance is used for identifying the relationship between a binary variable (income) and a continuous variable.

columns = ['age', 'capital.loss', 'capital.gain', 'hours.per.week', 'fnlwgt']

df['income'] = pd.to_numeric(df['income'], errors='coerce')

for column in columns:

   df[column] = pd.to_numeric(df[column], errors='coerce')

   cov_value = np.cov(df['income'], df[column])[0, 1] # Calculate covariance between income and the current column

   # if the value is greater than zero, it is positive or else it is negative correlation

   if cov_value > 0:

      status = "positive covariance"

   else:

      status = "negative covariance"

   print(column, ":", cov_value, " - ", status) # Print column name, covariance value, and correlation status

Through this we can see that only the "fnlwgt" column has a negative correlation with the income column.

Let us confirm this statement with the use of logistic regression.

import statsmodels.api as sm

X = sm.add_constant(df[['age', 'capital.loss', 'capital.gain', 'hours.per.week', 'fnlwgt']])

y = df['income'] # the target variable is assigned

logistic_model = sm.Logit(y, X) # a logistic regression model instance is created

result = logistic_model.fit() # the model is fitted print(result.summary()) # print the summary

Using this report, it could be seen that the all the columns have an impact on the "income" column. However, when analysing the "fnlwgt" column, we know it plays some impact on the "income" column as the coefficient is not zero. However, this impact seems to be insignificant when considering in a practical sitution.

Thus, due to its lack of impact, the "fnlwgt" column could be dropped.

df = df.drop('fnlwgt', axis=1)

df

Next we have to perform One Hot Encoding to create binary columns for each categorical value.

categorical_columns = df.columns[df.dtypes == object].tolist() # the columns are converted into a list

df = pd.get_dummies(df, columns=categorical_columns)

df

Min Max Scaler could be used to provide a value between 0 and 1 for all the numerical columns.

from sklearn.preprocessing import MinMaxScaler

scalable_columns = ['age','capital.gain','capital.loss','hours.per.week'] #columns with numerical values

min_max_scaler = MinMaxScaler() # creating an instance of the Min Max Scaler

scaled_columns = min_max_scaler.fit_transform(df[scalable_columns])

now the values could be assigned back to the respective column in the DataFrame

df['age']=scaled_columns[:,0]

df['capital.gain']=scaled_columns[:,1]

df['capital.loss']=scaled_columns[:,2]

df['hours.per.week']=scaled_columns[:,3]

df

This is how the final data frame would look like after the preprocessing techniques have been carried out.

Machine Learning Algorithms

1. Naïve Bayes Classifier model

```
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from imblearn.over_sampling import SMOTE

gaussian_model = GaussianNB() # creating an instance of the Gaussian Naive Bayes model

x = df.drop('income', axis=1) # the input features are defined

y = df['income'] # the output feature (income) is defined

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split is taken

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

gaussian = gaussian_model.fit(x_train, y_train) # the model is trained

prediction_gaussian = gaussian.predict(x_test) # the model is used to predict the output

accuracy_gaussian = accuracy_score(y_test, prediction_gaussian) # the accuracy of the model is calculated

print("Accuracy of the model is: ", accuracy_gaussian) # the accuracy is printed

print(classification_report(y_test, prediction_gaussian)) # the classification report is printed

confusion_matrix(y_test, prediction_gaussian) # the confusion matrix is printed
```

2. Random Forest Classifier model

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
```

from sklearn.metrics import accuracy_score,classification_report, confusion_matrix

rf_model = RandomForestClassifier(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1) # the input features are defined

y = df['income'] # the output feature (income) is defined

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42) # 70% training split is taken

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data

predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated

print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed

Let us try to increase the accuracy by tuning the hyperparameters.

Changing the training and testing split

rf_model = RandomForestClassifier(random_state=42)

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data

predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated

print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed

Adding parameters such as minimal samples to split an internal node and the maximum depth of the decision trees.

rf_model = RandomForestClassifier(min_samples_split=10, max_depth=10, random_state=42)

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data

predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated

print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed

Tuning the minimum number of samples to split an internal node.

rf_model = RandomForestClassifier(min_samples_split=20, max_depth=10, random_state=42)

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data

predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated

print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed

Increasing the number of samples to split an internal node reduces the accuracy of the model. Thus, we can revert to the original value.

Tuning the maximum depth of the decision trees.

rf_model = RandomForestClassifier(min_samples_split=10, max_depth=20, random_state=42)

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

rf = rf_model.fit(x_train, y_train) # the model is trained using the training data

predictions_rf = rf.predict(x_test) # the model is tested using the testing inputs

accuracy_rf = accuracy_score(y_test, predictions_rf) # the accuracy score is calculated

print(f"Accuracy is {accuracy_rf}") # the accuracy is printed

print(classification_report(y_test, predictions_rf)) # the classification report is printed

confusion_matrix(y_test, predictions_rf) # the confusion matrix is printed

By analyzing the classification report, we can come up with the following conclusions.

* In general, the precision, recall and the f1-score are higher for class zero than for class one.

* However, since the macro average and the weighted average are relatively close, it depicts a balance in the performance between classes.

Thus we can say the model is not overfitting or underfitting.

Further Improvements

3. Logistic Regression model.

from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

lr = lr_model.fit(x_train, y_train) # the model is trained using the training data

predictions_lr = lr.predict(x_test) # the model is tested using the testing inputs

accuracy_lr = accuracy_score(y_test, predictions_lr) # the accuracy score is calculated

print(f"Accuracy is {accuracy_lr}") # the accuracy is printed

print(classification_report(y_test, predictions_lr)) # the classification report is printed

confusion_matrix(y_test, predictions_lr) # the confusion matrix is printed

4. XGBoost Classifier

from xgboost import XGBClassifier

xgb_model = XGBClassifier(random_state=42) # an instance of the model is created

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 80% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

xgb = xgb_model.fit(x_train, y_train) # the model is trained using the training data

predictions_xgb = xgb.predict(x_test) # the model is tested using the testing inputs

accuracy_xgb = accuracy_score(y_test, predictions_xgb) # the accuracy score is calculated

print(f"Accuracy is {accuracy_xgb}") # the accuracy is printed print(classification_report(y_test, predictions_xgb)) # the classification report is printed

confusion_matrix(y_test, predictions_xgb) # the confusion matrix is printed

Thus, it could be seen that models such as XGBoost classifier could prove to be better and more efficient.

Now let us plot an ROC AUC Curve to evaluate the performance of the models.

This curve plots the recall (true positive value) against the false positive value.

from sklearn.metrics import roc_curve, roc_auc_score, auc

x = df.drop('income', axis=1)

y = df['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42) # 70% training split

x_train, y_train = SMOTE().fit_resample(x_train, y_train) # the SMOTE technique is used to balance the data by oversampling the minority class

gnb = GaussianNB() # creating an instance of the Gaussian Naive Bayes model

rf = RandomForestClassifier(random_state=42) # creating an instance of the Random Forest model

lr = LogisticRegression(random_state=42) # creating an instance of the Logistic Regression model

xgb = XGBClassifier(random_state=42) # creating an instance of the XGBoost model

gnb.fit(x_train, y_train) # the Gaussian Naive Bayes model is trained

rf.fit(x_train, y_train) # the Random Forest model is trained

lr.fit(x_train, y_train) # the Logistic Regression model is trained

xgb.fit(x_train, y_train) # the XGBoost model is trained

prediction_gaussian = gnb.predict_proba(x_test)[:,1] # the Gaussian Naive Bayes model is used to predict the output

predictions_rf = rf.predict_proba(x_test)[:,1] # the Random Forest model is used to predict the output

predictions_lr = lr.predict_proba(x_test)[:,1] # the Logistic Regression model is used to predict the output

```
predictions_xgb = xgb.predict_proba(x_test)[:,1] # the XGBoost model is used to predict the output

# lets us check the scores now

gaussian_roc_auc = roc_auc_score(y_test, prediction_gaussian)

print(f"ROC AUC score of the Gaussian NB model is {gaussian_roc_auc}")

rf_roc_auc = roc_auc_score(y_test, predictions_rf)

print(f"ROC AUC score of the Random Forest model is {rf_roc_auc}")

lr_roc_auc = roc_auc_score(y_test, predictions_lr)

print(f"ROC AUC score of the Logistic Regression model is {lr_roc_auc}")

xgb_roc_auc = roc_auc_score(y_test, predictions_xgb)

print(f"ROC AUC score of the XGBoost model is {xgb_roc_auc}")

fpr_gnb, tpr_gnb, _ = roc_curve(y_test, prediction_gaussian)

fpr_rf, tpr_rf, _ = roc_curve(y_test, predictions_rf)

fpr_lr, tpr_lr, _ = roc_curve(y_test, predictions_lr)

fpr_xgb, tpr_xgb, _ = roc_curve(y_test, predictions_xgb)

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

plt.plot(fpr_gnb, tpr_gnb, label=f"Gaussian NB (area = {gaussian_roc_auc:.2f})")

plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (area = {rf_roc_auc:.2f})")

plt.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (area = {lr_roc_auc:.2f})")

plt.plot(fpr_xgb, tpr_xgb, label=f"XGBoost (area = {xgb_roc_auc:.2f})")

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

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc='lower right')

plt.title('ROC Curve')
```

Thus using the above plot, we can rank the models based on best efficiency as below.

1. XGBoost classifier
2. Logistic Regression
3. Random Forest Classifier
4. Gaussian Naive Bayes