Random Forest Classifier - Income Classsification

June 30, 2022

1 Random Forest Classifier

Data-set `income_evaluation.csv` was extracted from the 1994 Census bureau database of USA. The prediction task is to determine whether a person makes over $50K a year or not.

Features:

1. age: continuous
2. workclass: categorical
3. fnlwgt: continuous
4. education: categorical
5. education-num: continuous
6. marital-status: categorical
7. occupation: categorical
8. relationship: categorical
9. race: categorical
10. sex: categorical
11. capital-gain: continuous
12. capital-loss: continuous
13. hours-per-week: continuous
14. native-country: categorical
15. income: target

2 Import required packages

[ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report
```

3 Load dataset

[ ]:
```
df = pd.read_csv('../data/income_evaluation.csv')
```

4 Exploratory Data Analysis

Explore the data to gain insights about the data.

4.1 View dimensions of dataset

[ ]:
```
df.shape
```

We can see that there are 32561 instances and 15 attributes in the data set.

4.2 Preview the dataset

[ ]:
```
df.head()
```

4.3 Rename column names

We can see that the dataset does not have proper column names. The column names contain underscore. We should give proper names to the columns. I will do it as follows:-

[ ]:
```
df.columns
```

[ ]:
```
df.columns = [i.replace('-','_').strip() for i in df.columns]
```

[ ]:
```
df.columns
```

4.4 View summary of dataset

[ ]:
```
df.info()
```

Findings

- We can see that the dataset contains 9 character variables and 6 numerical variables.
- There are no missing values in the dataset.

4.5 Check the data types of columns

- The above `df.info()` command gives us the number of filled values along with the data types of columns.
- If we simply want to check the data type of a particular column, we can use the following command.

```
[ ]: df.dtypes
```

4.6 View statistical properties of dataset

```
[ ]: df.describe()
```

- The above `df.describe()` command presents statistical properties in vertical form.
- If we want to view the statistical properties in horizontal form, we should run the following command.

```
[ ]: df.describe().T
```

```
[ ]: df.describe(include='all')
```

4.7 Check for missing values

- In Python missing data is represented by two values:
  - **None** : None is a Python singleton object that is often used for missing data in Python code.
  - **NaN** : NaN is an acronym for Not a Number. It is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation.
- There are different methods in place on how to detect missing values.

Pandas `isnull()` and `notnull()` functions

- Pandas offers two functions to test for missing values - **isnull()** and **notnull()**.
- These are simple functions that return a boolean value indicating whether the passed in argument value is in fact missing data.

Below, I will list some useful commands to deal with missing values.

Useful commands to detect missing values

- **df.isnull()**

The above command checks whether each cell in a dataframe contains missing values or not. If the cell contains missing value, it returns True otherwise it returns False.

- **df.isnull().sum()**

The above command returns total number of missing values in each column in the dataframe.

- **df.isnull().sum().sum()**

It returns total number of missing values in the dataframe.

- **df.isnull().mean()**

It returns percentage of missing values in each column in the dataframe.

- **df.isnull().any()**

It checks which column has null values and which has not. The columns which has null values returns TRUE and FALSE otherwise.

- **df.isnull().any().any()**

It returns a boolean value indicating whether the dataframe has missing values or not. If dataframe contains missing values it returns TRUE and FALSE otherwise.

- **df.isnull().values.any()**

It checks whether a particular column has missing values or not. If the column contains missing values, then it returns TRUE otherwise FALSE.

- **df.isnull().values.sum()**

It returns the total number of missing values in the dataframe.

[ ]:
```
# check for missing values
df.isnull().sum()
```

Interpretation

We can see that there are no missing values in the dataset.

4.7.1 Check with assert statement

- We must confirm that our dataset has no missing values.
- We can write an **Assert statement** to verify this.
- We can use an assert statement to programmatically check that no missing, unexpected 0 or negative values are present.
- This gives us confidence that our code is running properly.
- **Assert statement** will return nothing if the value being tested is true and will throw an AssertionError if the value is false.
- Asserts
  - assert 1 == 1 (return Nothing if the value is True)
  - assert 1 == 2 (return AssertionError if the value is False)

[ ]:
```
#assert that there are no missing values in the dataframe

assert pd.notnull(df).all().all()
```

Interpretation

- The above command does not throw any error. Hence, it is confirmed that there are no missing or negative values in the dataset.
- All the values are greater than or equal to zero excluding character values.

4.8 Explore Categorical Variables

[ ]:
```
categorical = ['workclass', 'education', 'marital_status',
               'occupation', 'relationship', 'race', 'sex',
               'native_country']
```

[ ]:
```
df[categorical].head()
```

4.8.1 Frequency distribution of categorical variables

Now, we will check the frequency distribution of categorical variables.

[ ]:
```
for var in categorical:
    print(df[var].value_counts(),'\n')
```

4.8.2 Percentage of frequency distribution of values

[ ]:
```
for var in categorical:
     print(df[var].value_counts(normalize=True),'\n')
```

Findings

- Now, we can see that there are several variables like `workclass`, `occupation` and `native_country` which contain missing values.
- Generally, the missing values are coded as `NaN` and python will detect them with the usual command of `df.isnull().sum()`.
- But, in this case the missing values are coded as `?`. Pandas fails to detect these as missing values because it does not consider `?` as missing values.
- So, we have to replace `?` with `NaN` so that Python can detect these missing values.
- We will explore these variables and replace `?` with `NaN`.

4.8.3 Explore target variable

[ ]:
```
# check for missing values
df['income'].isnull().sum()
```

We can see that there are no missing values in the `income` target variable.

[ ]:
```
# view number of unique values
df['income'].nunique()
```

There are 2 unique values in the `income` variable.

```
[ ]: # view the unique values
     df['income'].unique()
```

The two unique values are `<=50K` and `>50K`.

```
[ ]: # view the frequency distribution of values
     df['income'].value_counts()
```

```
[ ]: # view percentage of frequency distribution of values
     df['income'].value_counts(normalize=True)
```

```
[ ]: # visualize frequency distribution of income variable

     f,ax=plt.subplots(1,2,figsize=(18,8))

     ax[0] = df['income'].value_counts().plot.pie(explode=[0,0],autopct='%1.
      ↪1f%%',ax=ax[0])
     ax[0].set_title('Income Share')


     #f, ax = plt.subplots(figsize=(6, 8))
     ax[1] = sns.countplot(x="income", data=df, palette="Set1")
     ax[1].set_title("Frequency distribution of income variable")

     plt.show()
```

4.8.4 Visualize income wrt sex variable

```
[ ]: f, ax = plt.subplots(figsize=(10, 8))
     ax = sns.countplot(x="income", hue="sex", data=df, palette="Set1")
     ax.set_title("Frequency distribution of income variable wrt sex")
     plt.show()
```

Interpretation

- We can see that males make more money than females in both the income categories.

4.8.5 Visualize income wrt race

```
[ ]: f, ax = plt.subplots(figsize=(10, 8))
     ax = sns.countplot(x="income", hue="race", data=df, palette="Set1")
     ax.set_title("Frequency distribution of income variable wrt race")
     plt.show()
```

Interpretation

- We can see that whites make more money than non-whites in both the income categories.

4.8.6 Explore workclass variable

[ ]:
```
# check number of unique labels
df.workclass.nunique()
```

[ ]:
```
# view the unique labels
df.workclass.unique()
```

[ ]:
```
# view frequency distribution of values
df.workclass.value_counts()
```

We can see that there are 1836 values encoded as `?` in workclass variable. I will replace these `?` with `NaN`.

[ ]:
```
# replace '?' values in workclass variable with `NaN`
df['workclass'].replace(' ?', np.NaN, inplace=True)
```

[ ]:
```
# again check the frequency distribution of values in workclass variable
df.workclass.value_counts()
```

- Now, we can see that there are no values encoded as `?` in the workclass variable.
- We will adopt similar approach with `occupation` and `native_country` column.

4.8.7 Visualize workclass variable

[ ]:
```
f, ax = plt.subplots(figsize=(10, 6))
ax = df.workclass.value_counts().plot(kind="bar", color="green")
ax.set_title("Frequency distribution of workclass variable")
ax.set_xticklabels(df.workclass.value_counts().index, rotation=90)
plt.show()
```

Interpretation

- We can see that there are lot more private workers than other category of workers.

4.8.8 Visualize workclass variable wrt income variable

[ ]:
```
f, ax = plt.subplots(figsize=(12, 8))
ax = sns.countplot(x="workclass", hue="income", data=df, palette="Set1")
ax.set_title("Frequency distribution of workclass variable wrt income")
ax.legend(loc='upper right')
plt.show()
```

Interpretation - We can see that workers make less than equal to 50k in most of the working categories. - But this trend is more appealing in Private `workclass` category.

4.8.9 Visualize workclass variable wrt sex variable

[ ]:
```
f, ax = plt.subplots(figsize=(12, 8))
ax = sns.countplot(x="workclass", hue="sex", data=df, palette="Set1")
ax.set_title("Frequency distribution of workclass variable wrt sex")
ax.legend(loc='upper right')
plt.show()
```

Interpretation - We can see that there are more male workers than female workers in all the working category. - The trend is more appealing in Private sector.

4.8.10 Explore occupation variable

[ ]:
```
# check number of unique labels
df.occupation.nunique()
```

[ ]:
```
# view unique labels
df.occupation.unique()
```

[ ]:
```
# view frequency distribution of values
df.occupation.value_counts()
```

We can see that there are 1843 values encoded as `?` in occupation variable. I will replace these `?` with `NaN`.

[ ]:
```
# replace '?' values in occupation variable with `NaN`
df['occupation'].replace(' ?', np.NaN, inplace=True)
```

[ ]:
```
# again check the frequency distribution of values
df.occupation.value_counts()
```

[ ]:
```
# visualize frequency distribution of `occupation` variable
f, ax = plt.subplots(figsize=(12, 8))
ax = sns.countplot(x="occupation", data=df, palette="Set1")
ax.set_title("Frequency distribution of occupation variable")
ax.set_xticklabels(df.occupation.value_counts().index, rotation=90)
plt.show()
```

4.8.11 Explore native_country variable

[ ]:
```
# check number of unique labels
df.native_country.nunique()
```

[ ]:
```
# view unique labels
df.native_country.unique()
```

[ ]:
```
# check frequency distribution of values
df.native_country.value_counts()
```

We can see that there are 583 values encoded as ? in native_country variable. I will replace these ? with NaN.

```
[ ]: # replace '?' values in native_country variable with `NaN`
df['native_country'].replace(' ?', np.NaN, inplace=True)
```

```
[ ]: # again check the frequency distribution of values
df.native_country.value_counts()
```

```
[ ]: # visualize frequency distribution of `native_country` variable
f, ax = plt.subplots(figsize=(16, 12))
ax = sns.countplot(x="native_country", data=df, palette="Set1")
ax.set_title("Frequency distribution of native_country variable")
ax.set_xticklabels(df.native_country.value_counts().index, rotation=90)
plt.show()
```

We can see that `United-States` dominate amongst the `native_country` variables.

4.8.12 Check missing values in categorical variables

```
[ ]: df[categorical].isnull().sum()
```

Now, we can see that `workclass`, `occupation` and `native_country` variable contains missing values.

4.8.13 Number of labels: Cardinality

- The number of labels within a categorical variable is known as **cardinality**.
- A high number of labels within a variable is known as **high cardinality**.
- High cardinality may pose some serious problems in the machine learning model. So, we will check for high cardinality.

```
[ ]: # check for cardinality in categorical variables
for var in categorical:
    print(var, ' contains ', len(df[var].unique()), ' labels')
```

We can see that `native_country` column contains relatively large number of labels as compared to other columns.

4.9 Explore Numerical Variables

```
[ ]: numerical = ['age', 'fnlwgt', 'education_num', 'capital_gain',
             'capital_loss', 'hours_per_week']
```

4.9.1 Preview the numerical variables

```
[ ]: df[numerical].head()
```

4.9.2 Check missing values in numerical variables

[ ]:
```
df[numerical].isnull().sum()
```

We can see that there are no missing values in the numerical variables.

4.9.3 Explore `age` variable

[ ]:
```
# View the distribution of `age` variable
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
ax = sns.distplot(x, bins=10, color='blue')
ax.set_title("Distribution of age variable")
plt.show()
```

We can see that `age` is slightly positively skewed.

[ ]:
```
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
x = pd.Series(x, name="Age variable")
ax = sns.kdeplot(x, shade=True, color='red')
ax.set_title("Distribution of age variable")
plt.show()
```

4.9.4 Detect outliers in `age` variable with boxplot

[ ]:
```
f, ax = plt.subplots(figsize=(10,8))
x = df['age']
ax = sns.boxplot(x)
ax.set_title("Visualize outliers in age variable")
plt.show()
```

We can see that there are lots of outliers in `age` variable.

4.9.5 Explore relationship between `age` and `income` variables

[ ]:
```
f, ax = plt.subplots(figsize=(10, 8))
ax = sns.boxplot(x="income", y="age", data=df)
ax.set_title("Visualize income wrt age variable")
plt.show()
```

Interpretation

- As expected, younger people make less money as compared to senior people.

4.9.6 Visualize income wrt age and sex variable

[ ]:
```
f, ax = plt.subplots(figsize=(10, 8))
ax = sns.boxplot(x="income", y="age", hue="sex", data=df)
ax.set_title("Visualize income wrt age and sex variable")
ax.legend(loc='upper right')
plt.show()
```

[ ]:
```
plt.figure(figsize=(8,6))
ax = sns.catplot(x="income", y="age", col="sex", data=df, kind="box", height=8,␣
 ↪aspect=1)
plt.show()
```

Interpretation

- Senior people make more money than younger people.

4.9.7 Visualize relationship between race and age

[ ]:
```
plt.figure(figsize=(12,8))
sns.boxplot(x ='race', y="age", data = df)
plt.title("Visualize age wrt race")
plt.show()
```

Interpretation

- Whites are more older than other groups of people.

4.9.8 Find out the correlations

[ ]:
```
df.corr()
```

[ ]:
```
# plot correlation heatmap to find out correlations
df.corr().style.format("{:.4}").background_gradient(cmap=plt.
 ↪get_cmap('coolwarm'), axis=1)
```

Interpretation - We can see that there is no strong correlation between variables.

5 Feature Engineering

- **Feature Engineering** is the process of transforming raw data into useful features that help us to understand our model better and increase its predictive power.
- We will carry out feature engineering on different types of variables.
- First, we will display the categorical and numerical variables in training set separately.

5.1 Display categorical variables in training set

[ ]:
```
categorical
```

5.2 Display numerical variables in training set

[ ]:
```
numerical
```

5.3 Engineering missing values in categorical variables

5.3.1 Create feature vector and target variable

[ ]:
```
X = df.drop(['income'], axis=1)
y = df['income']
```

[ ]:
```
# print percentage of missing values in the categorical variables in training␣
↪set
X[categorical].isnull().sum()
```

[ ]:
```
X['workclass'].mode()[0]
```

[ ]:
```
# impute missing categorical variables with most frequent value

X['workclass'].fillna(X['workclass'].mode()[0], inplace=True)
X['occupation'].fillna(X['occupation'].mode()[0], inplace=True)
X['native_country'].fillna(X['native_country'].mode()[0], inplace=True)
```

[ ]:
```
categorical
```

[ ]:
```
# check missing values in categorical variables in X_train
X[categorical].isnull().sum()
```

As a final check, I will check for missing values in X

[ ]:
```
# check missing values in X_train
X.isnull().sum()
```

[ ]:
```
X.head()
```

We can see that there are no missing values in X.

5.4 Encode categorical variables

[ ]:
```
# preview categorical variables in X_train
df[categorical].head()
```

```
[ ]: # encode categorical variables with one-hot encoding
ohe = OneHotEncoder()
ohe.fit(df[categorical])
```

```
[ ]: enc_df = pd.DataFrame(ohe.transform(df[categorical]).toarray(), columns=ohe.
      ↪get_feature_names())
```

```
[ ]: enc_df
```

```
[ ]: numerical
```

```
[ ]: # merge with main df[numerical] on key values
X = X[numerical].join(enc_df)
```

```
[ ]: X.head()
```

```
[ ]: X.columns
```

```
[ ]: X.shape
```

We can see that from the initial 15 columns, we now have 108 columns in data set.

```
[ ]: ## Split data into separate training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
      ↪random_state = 569)
```

```
[ ]: # check the shape of X_train and X_test
X_train.shape, X_test.shape
```

6 Model Training & Evalution

6.1 Train Random Forest model

```
[ ]: # instantiate the classifier
rfc = RandomForestClassifier(random_state=1, n_estimators=10)
```

```
[ ]: # fit the model
rfc.fit(X_train, y_train)
```

We have build the Random Forest Classifier model with default parameter of `n_estimators = 100`. So, we have used 100 decision-trees to build the model.

6.2 Model Evalution

```
[ ]: # Predict the Test set results
y_pred = rfc.predict(X_test)
```

```
[ ]: y_pred[:10]
```

6.2.1 Accuracy

```
[ ]: accuracy_score(y_test, y_pred) *100
```

Here, **y__test** are the true class labels and **y__pred** are the predicted class labels in the test-set.

6.2.2 Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called **Type I error.**

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called **Type II error.**

These four outcomes are summarized in a confusion matrix given below.

```
[ ]: # Print the Confusion Matrix and slice it into four pieces
     cm = confusion_matrix(y_test, y_pred)
     print('Confusion matrix\n', cm)
```

```
[ ]: # visualize confusion matrix with seaborn heatmap

     cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual␣
      ↪Negative:0'],
                                      index=['Predict Positive:1', 'Predict Negative:
      ↪0'])

     sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

6.2.3 Classification Report

Classification report is another way to evaluate the classification model performance. It displays the **precision**, **recall**, **f1** and **support** scores for the model.

[ ]:
```python
from sklearn.metrics import f1_score, precision_score, recall_score
```

[ ]:
```python
print(classification_report(y_test, y_pred))
```

7 Exercise

Increase/decrease the number of decision-trees and see its effect on 1. Accuracy 2. Type 1 error 3. Type 2 Error 4. F1-score

Note your results in a table in following format:

No. of Trees	Accuracy	Type 1 Error (FP)	Type 2 Error (FN)	F1-score
10				
50				
100				
150				
200				