**Census Income**

**Problem Definition:**

This data was extracted from the [1994 Census bureau database](http://www.census.gov/en.html) by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1) && (HRSWK>0)). **The prediction task is to determine whether a person makes over $50K a year.**

**Description of fnlwgt (final weight)**

The weights on the Current Population Survey (CPS) files are controlled to independent estimates of the civilian non-institutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:

A single cell estimate of the population 16+ for each state.

1. Controls for Hispanic Origin by age and sex.
2. Controls by Race, age and sex.

We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

We need to predict the who is earning the more then 50 K based on these independent variables

**Feature variables:**

* Age
* Workclass
* Fnlwgt
* Education
* Education\_num
* Marital\_status
* Occupation
* Relationship
* Race
* Sex
* Capital\_gain
* Capital\_loss
* Hours\_per\_week
* Native\_country

**Target Variable**

* Income

**#Importing the Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import sklearn

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix,classification\_report

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from scipy.stats import zscore

import warnings

warnings.filterwarnings('ignore')

**#Loading the dataset**

df= pd.read\_csv('census\_income.csv')

df.head()

Application

Description automatically generated with medium confidence

**Data Analysis and EDA**

df.shape

(32560, 15)

Dataset contains the 32560 records and 15 variables

df.dtypes

Age int64

Workclass object

Fnlwgt int64

Education object

Education\_num int64

Marital\_status object

Occupation object

Relationship object

Race object

Sex object

Capital\_gain int64

Capital\_loss int64

Hours\_per\_week int64

Native\_country object

Income object

Data types of the variables is integer and object types

#we using the Heat map to check the null values in dataset

plt.figure(figsize=[10,10])

sns.heatmap(df.isnull(), cbar=False)

plt.title('Null values')

plt.show()

Shape

Description automatically generated

There is no null values in the dataset

**Univariant analysis:**

print(df['Income'].value\_counts())

sns.countplot(df['Income'])

plt.show()

Chart, bar chart

Description automatically generated

We clearly see that classification of target variables data is imbalanced so we need to treat

**Multivariant analysis:**

fig=plt.figure(figsize=(15,5))

sns.countplot(x='Education', hue='Income', data=df)

plt.show()

Chart, bar chart

Description automatically generated

We clearly see that highest number of bachelors degree holders only make the more the 50K income per annum

fig=plt.figure(figsize=(20,5))

sns.countplot(x='Occupation', hue='Income', data=df)

plt.show()

A picture containing chart

Description automatically generated

Who have occupation Ex-Manager have highest number of people making the 50K income

**Encoding the Categorical variables :**

We are encoding the categorical variables using label encoder

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

#df\_train = pd.get\_dummies(df\_train, columns=['Profile'])

df['Workclass'] = le.fit\_transform(df['Workclass'])

df['Education'] = le.fit\_transform(df['Education'])

df['Marital\_status'] = le.fit\_transform(df['Marital\_status'])

df['Occupation'] = le.fit\_transform(df['Occupation'])

df['Relationship'] = le.fit\_transform(df['Relationship'])

df['Race'] = le.fit\_transform(df['Race'])

df['Sex'] = le.fit\_transform(df['Sex'])

df['Native\_country'] = le.fit\_transform(df['Native\_country'])

df['Income'] = le.fit\_transform(df['Income'])

we are using the to check the co-relation heat map

corr\_hmap=df.corr()

plt.figure(figsize=(15,10))

sns.heatmap(corr\_hmap, annot=True)

plt.show()

Timeline, treemap chart

Description automatically generated

Observation:

1. Education\_num and income have a positive co-relation 34%

2. relationship with income have negative co-relation  -25%

3. Hour\_per\_week have positive relation with income 23%

4. Occuption and work class positive co-relation of 26%

#to display the corelation with target variables

corr\_matrix=df.corr()

corr\_matrix['Income'].sort\_values(ascending = False)

A picture containing text, receipt

Description automatically generated

Observation:

1. Education number have positive relationship with target varibles income 33%

2. Education number, age, Hours\_per\_week,Capital\_gain,Sex,Capital\_loss,Education,Occupation,Race, Workclass, Native\_country, ALso  have positive reaionship with Income

3. FnlGHT, Merital\_status, relationship negative relationship income

columns =['Age', 'Workclass', 'Fnlwgt', 'Education', 'Education\_num',

       'Marital\_status', 'Occupation', 'Relationship', 'Race', 'Sex',

       'Capital\_gain', 'Capital\_loss', 'Hours\_per\_week', 'Native\_country',

       'Income']

for i in df[columns]:

    plt.figure()

    sns.distplot(df[i])

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

A picture containing logo

Description automatically generated Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated Chart, histogram

Description automatically generated

A picture containing text

Description automatically generated Shape

Description automatically generated

Shape

Description automatically generated Chart

Description automatically generated

Shape

Description automatically generated Icon

Description automatically generated with low confidence

#BOX plot

df.boxplot(figsize=(18,10))

plt.show()

Chart, histogram

Description automatically generated

EDA Concluding Remark.:

There is no null values in the dataset

We clearly see that classification of target variables data is imbalanced so we need to treat

We clearly see that highest number of bachelors degree holders only make the more the 50K income per annum

Who have occupation Ex-Manager have highest number of people making the 50K income

Observation:

1. Education number have positive relationship with target varibles income 33%

2. Education number, age, Hours\_per\_week,Capital\_gain,Sex,Capital\_loss,Education,Occupation,Race, Workclass, Native\_country, ALso  have positive reaionship with Income

3. FnlGHT, Merital\_status, relationship negative relationship income

Pre-Processing Pipeline.

# creating train test splits

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=40)

Treating the imbalanced data using Oversampling method

#import library

from collections import Counter

#import imblearn

from imblearn import under\_sampling, over\_sampling

from imblearn.over\_sampling import SMOTE

counter =Counter(y\_train)

print('Before', counter)

smote = SMOTE()

x\_smote, y\_smote = smote.fit\_resample(x, y)

counter =Counter(y\_smote)

print('after', counter)



Building Machine Learning Models.:

This problem is an classification so we use multiple classification algorithm which we get highest performance we save the model

**#Decision Tree model**

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()

dt.fit(x\_smote,y\_smote)   #over sampled data using here

#dt.fit(x\_train,y\_train)

p=dt.predict(x\_test)

print(accuracy\_score(y\_test,p))

Observation:

1. We getting the Decsison Tree classification model accuracy is :81.29%

2. We getting the Decsison Tree classification model ovsersample balanced data we get accuracy is : 100%

**#Logistic regression Model:**

from sklearn.linear\_model import LogisticRegression

lg=LogisticRegression()

#lg.fit(x\_smote,y\_smote)   #Over sampled data

lg.fit(x\_train,y\_train)

pred =lg.predict(x\_test)

print(pred)

print("Accuracy\_score", accuracy\_score(y\_test,pred))

print(confusion\_matrix(y\_test,pred))

print(classification\_report(y\_test, pred))

Table

Description automatically generated

Graphical user interface

Description automatically generated

#Grid Search

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

from sklearn.model\_selection import GridSearchCV

clf = LogisticRegression()

grid\_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10,25]}

grid\_clf\_acc = GridSearchCV(clf, param\_grid = grid\_values,scoring = 'recall')

grid\_clf\_acc.fit(x\_train, y\_train)

#Predict values based on new parameters

y\_pred\_acc = grid\_clf\_acc.predict(x\_test)

# New Model Evaluation metrics

print('Accuracy Score : ' + str(accuracy\_score(y\_test,y\_pred\_acc)))

print('Precision Score : ' + str(precision\_score(y\_test,y\_pred\_acc)))

print('Recall Score : ' + str(recall\_score(y\_test,y\_pred\_acc)))

print('F1 Score : ' + str(f1\_score(y\_test,y\_pred\_acc)))

#Logistic Regression (Grid Search) Confusion matrix

confusion\_matrix(y\_test,y\_pred\_acc)

Text, letter

Description automatically generated

**Random Forest model:**

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n\_estimators=100)

#rf.fit(x\_smote, y\_smote)  #balanced data

rf.fit(x\_train, y\_train)

rf\_pred =rf.predict(x\_test)

print('accuracy score', rf\_pred)

print(accuracy\_score(y\_test,rf\_pred))

print(confusion\_matrix(y\_test,rf\_pred))

print(classification\_report(y\_test,rf\_pred))

Table

Description automatically generated

Observation:

1. we are getting the RandomForest model accuracy is 85% using imbalanced data

2. we are getting the RandomForest model accuracy is 100% using balanced data

**SVM model**

#importing the svm model

from sklearn.svm import SVC

svclassifier = SVC(kernel='linear')

svclassifier.fit(x\_train, y\_train)

#svclassifier.fit(x\_smote, y\_smote)  #balanced data

sv\_pred = svclassifier.predict(x\_test)

print(accuracy\_score(y\_test,sv\_pred))

print(confusion\_matrix(y\_test,sv\_pred))

print(classification\_report(y\_test,sv\_pred))

Table

Description automatically generated

 Concluding Remarks.

We Getting the highest accuracy score in decision tree algorithm so we are saving the

**Saving the model:**

We Getting the highest accuracy score in decision tree algorithm so we are saving the model

We are using the pickle method store the model

#importing the library

from sklearn.externals import joblib

# Save the model as a pickle in a file

joblib.dump(DT, 'DT.pkl')

# Load the model from the file

DT\_joblib = joblib.load('DT.pkl')

# Use the loaded model to make predictions

#DT\_joblib.predict(x\_test)