***BINARY SALARY CLASSIFICATION USING MACHINE LEARNING***

Prepared By :-

Rivanshu Goyal 17UCS132

Mohit Jindal 17UCS091

Rohan Jauhari 17UCS133

Suman Kumar Choudhary 17UCC058

1. Abstract :-

Central governments of every country take periodic census. Their motive is to make inferences from the population and predict the standard of living, human development index etc. In this project, we will be analyzing such a population data.

We will be given a population census dataset. On it we will apply Machine learning algorithms to predict if the annual income of a person is more or less than $50k. Some useful things can be known if the government knows the income such as his taxation status, if he is above or below the poverty line, if he is eligible for various government schemes etc.

Annual income of a person depends on various factors/attributes. We have considered 14 attributes. We have a dataset of 48000 people. We will use classification analysis to predict the annual income of a person with given attributes.

This is a supervised learning based dataset on which we will apply classification algorithms.

2. Dataset Used :-

The Data Set we Used for the analysis is [Census Income Data Set](https://archive.ics.uci.edu/ml/datasets/Census+Income) we took this dataset from the website <https://archive.ics.uci.edu/>.

This training dataset consists of 48842 instances and of 14 Attributes. The features represent the contributing factors to the annual income of a person and the training set is labelled with the income.

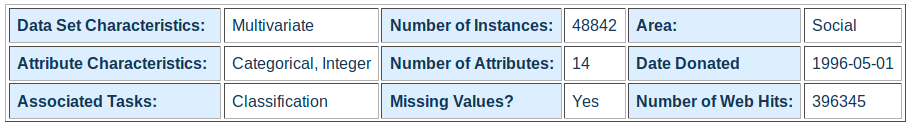


Table 1: Dataset Used.

2.1 Attributes Considered :-

1. **Age**: Age of the person
2. **Workclass**: Working class of a person from the following categories- Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
3. **Fnlwgt**: Weight associated to each record(not to be used for model training)continuous.
4. **Education:** The type of education a person has taken like Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
5. **Education-num**: Integer indexed to categories of education. Its continuous.
6. **Marital-status**: Whether a person is married or not. It contain categories as follows Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
7. **Occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspect, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
8. **Relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
9. **Race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black. People of certain races do only specific jobs that have a particular return.
10. **Sex**: Male are thought to earn more than females. Female/Male.
11. **Capital-gain**: Income from other sources - continuous.
12. **Capital-loss**: Loss due to other income sources - continuous.
13. **Hours-per-week**: A usual trend in comparison of income of people in the same work field is that a person who works more hours, earns more.
14. **Native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad & Tobago, Peru, Hong, Holland-Netherlands. Blacks, Asians, hispanic often do blue collar jobs that have less return where as europeans, indians americans do white collar jobs that pays more.
15. **Income:** It is a class attribute classifying the income of the person in two categories namely, “less than equal to 50k” and “greater than equal to 50k”.

3 Preprocessing

3.1 Libraries Required:-

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sns**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **sklearn.ensemble** **import** RandomForestClassifier

**from** **sklearn.metrics** **import** accuracy\_score

**from** **sklearn.metrics** **import** confusion\_matrix

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

**from** **sklearn.model\_selection** **import** RandomizedSearchCV

1. Pandas Library - It is a data manipulation library in python.It offers data structures and operations for manipulating numerical tables and time series.
2. Numpy Library - Numpy is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.
3. Seaborn Library - Seaborn is a data visualization library based on matplotlib used to plot graphs from data.
4. Scikit-Learn (sklearn) Library - *Scikit*-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in *Python*.
5. Matplotlib - It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits

3.2 Reading The File:-

First, we get the dataset using the pandas ***read\_csv*** function. In this dataset, columns were not assigned any Labels , so they were passed as an argument to the read\_csv function:

col\_names = ['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship','race',

'sex','capital-gain','capital-loss','hours-per-week','native-country','income']

df\_train = pd.read\_csv(r"https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",names = col\_names)

df\_test = pd.read\_csv(r"https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.test",names = col\_names)

3.3 Getting count of instances in Train and Test Dataset

To find the number of instances and features present in both the training and test set we use:

print(df\_train.shape)

print(df\_test.shape)

OUTPUT

(32561, 15) #In Training set

(16282, 15) #In Test set

3.4 Removing unwanted row in the test

Using the head function, we display the top 5 records from the dataset.

df\_test.head()

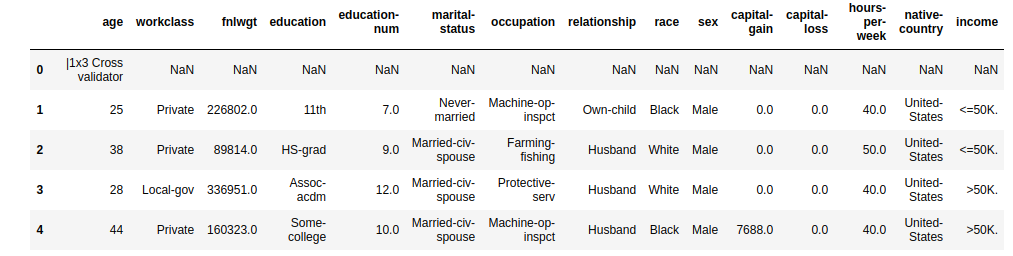


Table 2: This is the first 5 records from the dataset.

As we can see the first row in the test set contains a record full of NaN values,which we don’t require, so we have to drop this row.

df\_test=df\_test[1:]

df\_test.head()

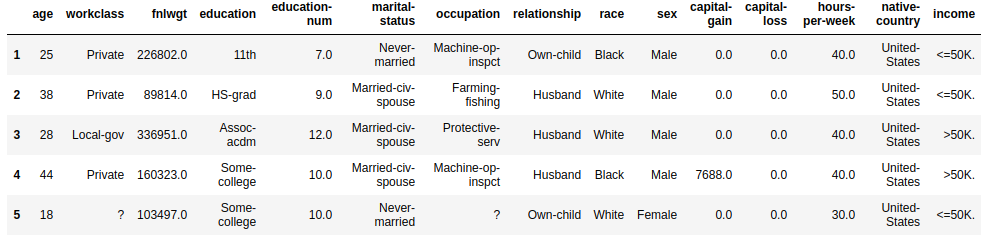


Table 3: This is the first 5 records from the dataset after removing unwanted row

3.5 Resetting the Indexes

Now as the row with index 0 has been dropped so we have to reset the indexes for the test set.

df\_test.reset\_index(drop=**True**,inplace=**True**)

df\_test.head()

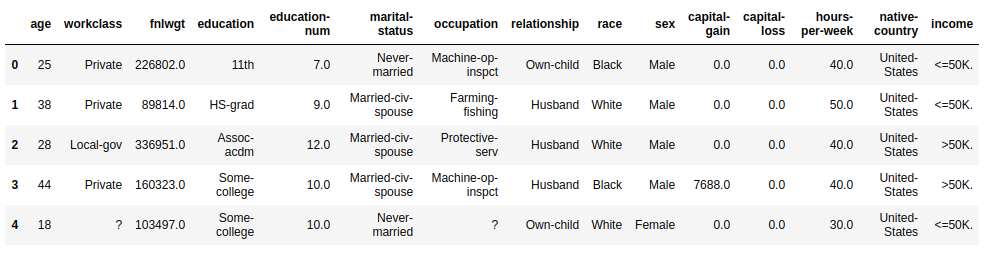


Table 4: This is the first 5 records from the dataset after indexes been reset.

3.6 Handling the Missing values

As we can see in the above table that there are some rows which contains “ ?”.These are the missing values, so we replace those “?” values with NaN values in both training and test set. This is done so as python can determine those values as missing. Also the test set’s class attribute has error in its class labels which were also handled here:

df\_train.replace(to\_replace = ' ?',value=np.nan,inplace = **True**)

df\_test.replace(to\_replace = ' ?',value=np.nan,inplace = **True**)

df\_test.replace(to\_replace = ' <=50K.',value=' <=50K',inplace = **True**)

df\_test.replace(to\_replace = ' >50K.',value=' >50K',inplace = **True**)

Now finding the sum of NaN values in all the attributes in Training set.

df\_train.isnull().sum()

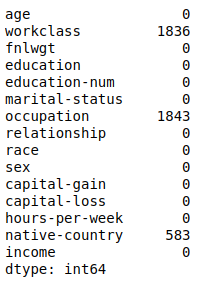


Table 5: Showing the sum of no. of null values for each attribute in training set

Now finding the sum of NaN values in all the attributes in Test set.

df\_test.isnull().sum()

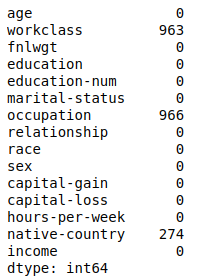


Table 6: Showing the sum of no. of null values for each attribute in test set.

3.7 Dropping the rows which contains at least one attribute having NaN values

As the Dataset is very large, so Instead of providing the categorical attributes with the most frequent value which would not be certainly accurate, It is better to drop the rows which contains at least one attribute having NaN value:

df\_train.dropna(inplace=**True**)

df\_test.dropna(inplace=**True**)

Now verifying if the records were dropped perfectly:

df\_train.isnull().sum()

df\_test.isnull().sum()

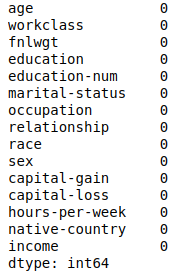


Table 7 Showing the sum of no. of null values for each attribute after removing.

(This table will be the same for both training and test set)

3.8 Checking the number of instances in training set

After doing some changes above in the training set now we are checking the number of instances left in the training set.

df\_train.shape

Output:

(30162, 15) #for training set

4 Random Sampling

The following command was used to check the distribution of instances between two classes:

df\_train[df\_train.income == ' <=50K'].describe()

Output:

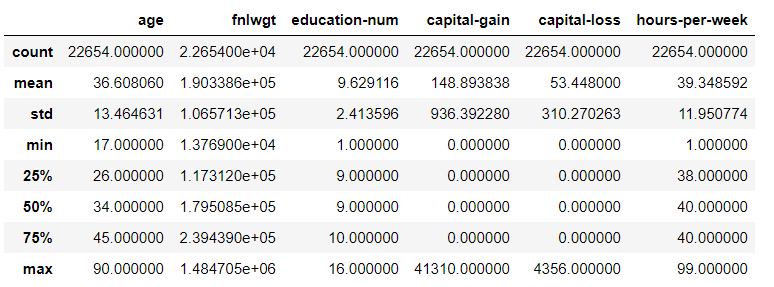


Table 8: This is the to check the description of instances.

As we can see here the dataset contains most of the instances for people having income less than or equal to 50K. So for a better model training, we take a random sample from the instances of records of classes of <=50K and concatenate it with the remaining one.

random\_sample1 = df\_train[df\_train.income == ' <=50K'].sample(n = 8000,replace = **False**,random\_state = 0)

random\_sample2 = df\_train[df\_train.income == ' >50K'].copy()

new\_train = pd.concat([random\_sample1,random\_sample2])

Hence the Problem of imbalance has been resolved. But since the data is concatenated we might shuffle it as well:

new\_train = new\_train.sample(frac = 1).reset\_index(drop=**True**)

new\_test = df\_test.copy()

The test set was given an alias just for better readability of the code.

new\_train.head()

Output of the shuffled Sample:

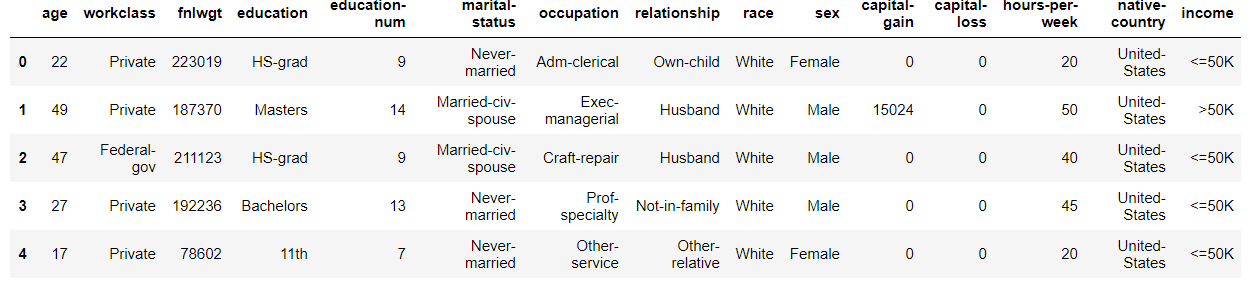


Table 9: This is the first 5 records from the dataset after shuffling.

4.1 Analyzing the Data Types

The data type of each attribute can be viewed using ***.dtypes***. As we can see the data contains lot of attributes with data type as object which is the categorical attribute.

new\_train.dtypes

Specifying data type of the attribute

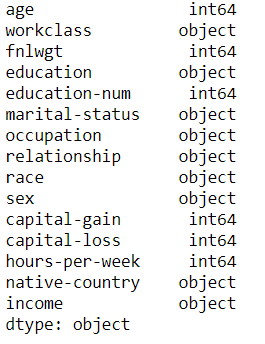


Table 10: Showing the types of each attribute.

4.1.1 We analyze different types of attributes by separating them as follows:

num\_attributes = new\_train.select\_dtypes(include=['int64'])

num\_attributes.hist(figsize=(10,10))

Plotting histogram of int64 data type

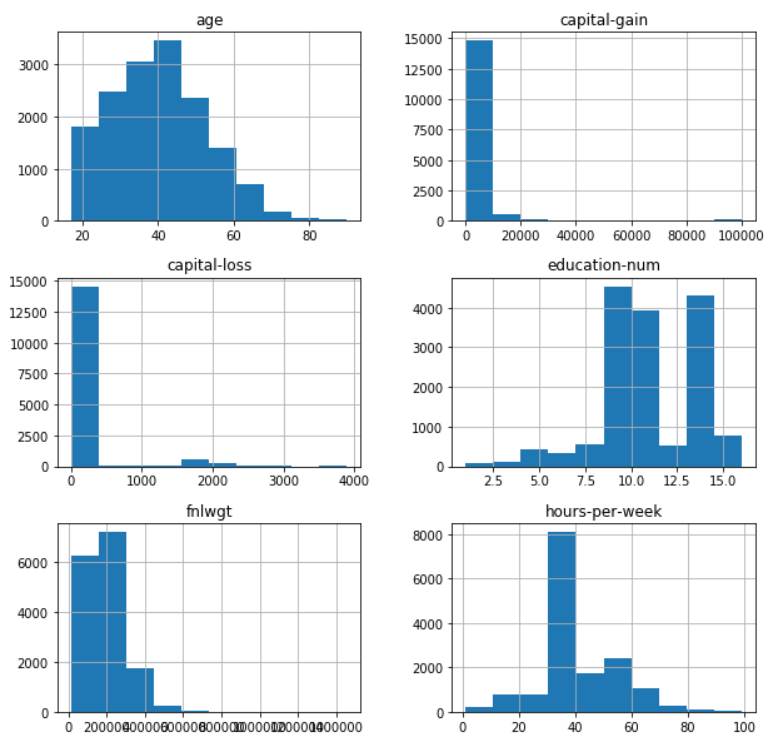


Figure 1: Showing the histogram of int64 data type attributes.

As we can observe from the histograms that data set contains mostly people of age between 20-50, and there aren’t many people with capital gain or loss. Also the amount of education - num is mostly between the 8-15.

4.1.2 The Categorical Attributes:

cat\_attributes = new\_train.select\_dtypes(include=['object'])

sns.countplot(y='education', hue='income', data = cat\_attributes)

Here, we are plotting the annual incomes against the no of years of education.

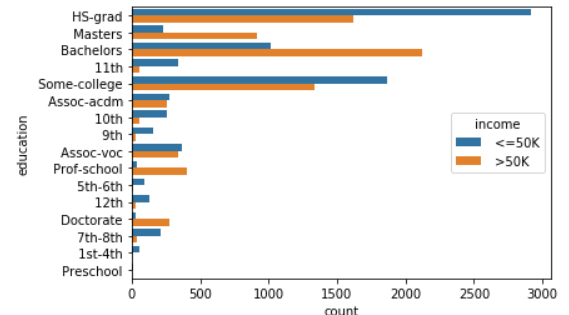


Figure 2: Plot for the annual income against the no. of years.

As we can deduce from the above graph, people with income >50K are mostly Bachelors and that with <=50K are mostly HS-graduates.

4.2 Encoding the Categorical Attributes

The categorical attributes need to be mapped with certain numerical numbers so that we can train a model on it. Also the Test Set was encoded too, so that predictions can be made.

This was done using the ***LabelEncoder*** class in the sklearn.

**from** **sklearn.preprocessing** **import** LabelEncoder

labels = list(cat\_attributes.columns)

**for** i **in** labels:

le = LabelEncoder()

le.fit(new\_train[i])

new\_train[i] = le.transform(new\_train[i])

new\_test[i] = le.transform(new\_test[i])

new\_train.head()

The mapped values are as follows for the Training and Test Set respectively.

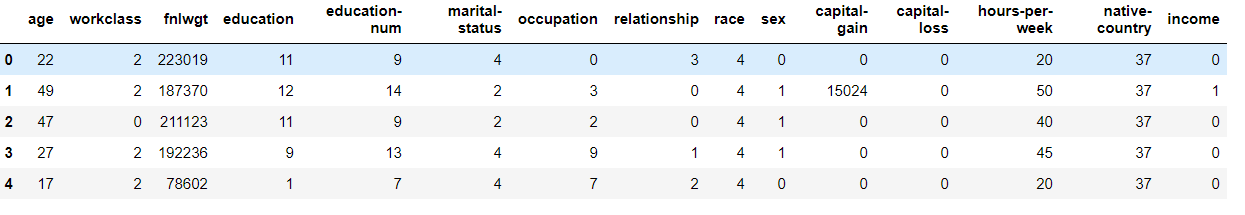


Table 11: First 5 records from the dataset after encoding in training set.

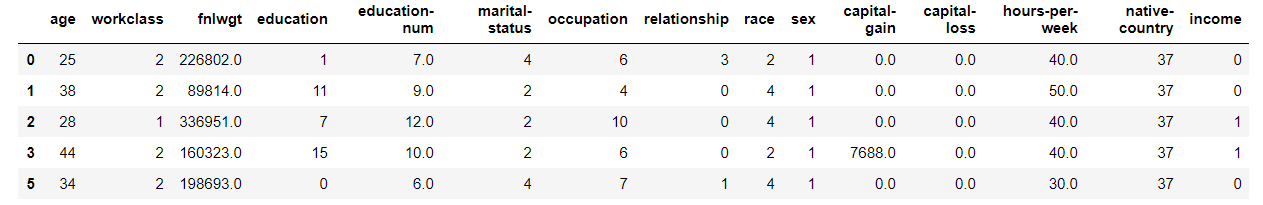


Table 12: First 5 records from the dataset after encoding in test set.

4.3 Splitting The Training and Test Set

The Training and Test set were split into two parts each to separate the attributes from the class attribute. But before that we plot a heatmap of the attributes so that we can find the correlation between them and if any are highly correlated we can drop them so as to increase the accuracy and efficiency of our model:

X\_train = new\_train.drop(['income','fnlwgt','education'], axis =1)

Y\_train = new\_train['income']

X\_test = new\_test.drop(['income','fnlwgt','education'], axis =1)

Y\_test = new\_test['income']

The education column was dropped as we already had a column containing the numerical value associated with education of a person. The fnlwgt column was also dropped as it was related to the weight of each record.

sns.heatmap(new\_train.corr())

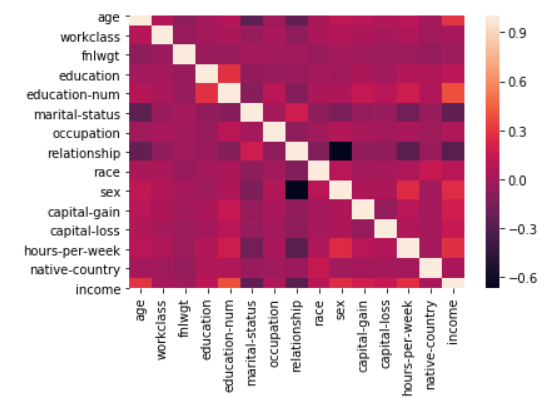


Figure 3: Heatmap formed from our features

The Heatmap clearly shows that attributes are not highly correlated to each other so we cannot drop any other attribute.

5 MODEL TRAINING

5.1 Applying Random Forest

Random forest is a collection of decision trees which return mean of all the decision tree’s accuracy as the prediction. It is also known as a bagging method. We chose this model because decision trees are good at handling categorical data and random forest further enhances the capabilities of Decision Trees.

**from** **sklearn.ensemble** **import** RandomForestClassifier

model\_1 = RandomForestClassifier(n\_estimators=100,bootstrap=**True**,random\_state=0)

model\_1.fit(X\_train,Y\_train)

pred\_randfor = model\_1.predict(X\_test)

print(accuracy\_score(pred\_randfor, Y\_test.values))

Accuracy = 0.8070385126162019

After applying random forest model, we get 80.7% (approx) accuracy.

Confusion Matrix

Confusion matrix here is used to know how many people were predicted to earn more than 50k vs how many actually earn and vice versa. We use the matplotlib and seaborn library to plot the confusion matrix as follows:

**from** **sklearn.metrics** **import** confusion\_matrix

**import** **matplotlib.pyplot** **as** **plt**

cfm = confusion\_matrix(pred\_randfor, Y\_test.values)

sns.heatmap(cfm, annot=**True**)

plt.xlabel('Predicted classes')

plt.ylabel('Actual classes')

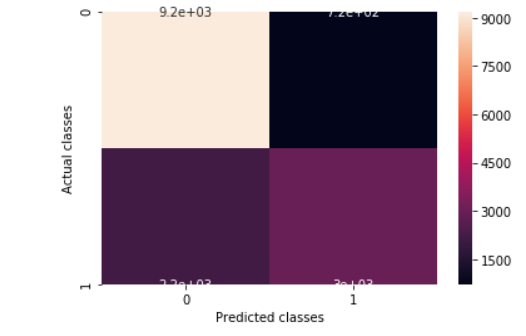


Figure 4: Confusion matrix for Actual classes and Predicted classes

To get the exact numerals of the values in confusion matrix we do the following:

confusion\_matrix(pred\_randfor, Y\_test.values)

OutPut:

array([[9172, 718],

[2188, 2982]], dtype=int64)

5.2 Applying Cross Validation

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

scores = cross\_val\_score(model\_1, X\_train,

Y\_train, cv=5)

print(np.mean(scores))

Mean=0.8168045249021187

Here we are using cross validation technique to train and test model by dividing dataset. We will divide the dataset in 5 parts and use one part as testing and other 4 parts for training. Cross validation is better than train\_test\_split() or holdout method because, here we have more data for training than in holdout method.

We then get the mean of all these 5 possibilities.

5.3 Applying RandomSearch

Random Search is applied to get the hyper parameters to be used to increase the accuracy of our model. It tries to apply different parameter combinations to our model to train on the provided dataset and returns the parameters that gives the best accuracy.

Random grid is hyper parameter for random forest that has options for various .

Randomized search is the algorithm to select best hyperparameter given in random grid for maximum accuracy.

n\_estimators = [int(x) **for** x **in** np.linspace(start = 200, stop = 2000, num = 10)]

*# Number of features to consider at every split*

max\_features = ['auto', 'sqrt']

*# Maximum number of levels in tree*

max\_depth = [int(x) **for** x **in** np.linspace(10, 110, num = 11)]

max\_depth.append(**None**)

min\_samples\_split = [2, 5, 10]

min\_samples\_leaf = [1, 2, 4]

bootstrap = [**True**, **False**]

random\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap}

**from** **sklearn.model\_selection** **import** RandomizedSearchCV

rf\_random = RandomizedSearchCV(estimator = model\_1, param\_distributions = random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)

*# Fit the random search model*

rf\_random.fit(X\_train, Y\_train)

rf\_random.best\_score\_

Output=0.8309259736909982

After applying random forest model, we get 83.1% (approx) accuracy.

6 Inferences

The data set was imbalanced so we performed the Random sampling to make a balanced data set on which our model would train. We plot the heat map of the attributes and found that no two attributes were highly correlated hence we couldn’t perform any Feature selection except dropping the education.

Since the data set included a lot of categorical data and Decision trees are good at handling it, so we trained our model based on Decision Tree classifier and to getter better results, we used bagging of Decision Trees known as Random Forest. Initially we got an accuracy of 80.7% and then used cross validation to improve the accuracy which led us to an increment of 1%. It is due to the fact that cross validation trains the model again and again on the same dataset using different pairs of X\_train and Y\_train.

Then to further improve our accuracy we used Random Search which trains our model on different combinations of Hyper parameters and gives us the best accuracy that could be achieved using the best combination of Hyperparameters. This resulted in achieving an accuracy of about 83%.

So, at last we were able to classify the salaries of the test data and the classification accuracy was **83%** which was achieved through Random Search.