CURNEW MEDTECH INNOVATIONS PRIVATE LIMITED SD03Q01

-Mohamed Arshath R (1832033)

Problem Statement:

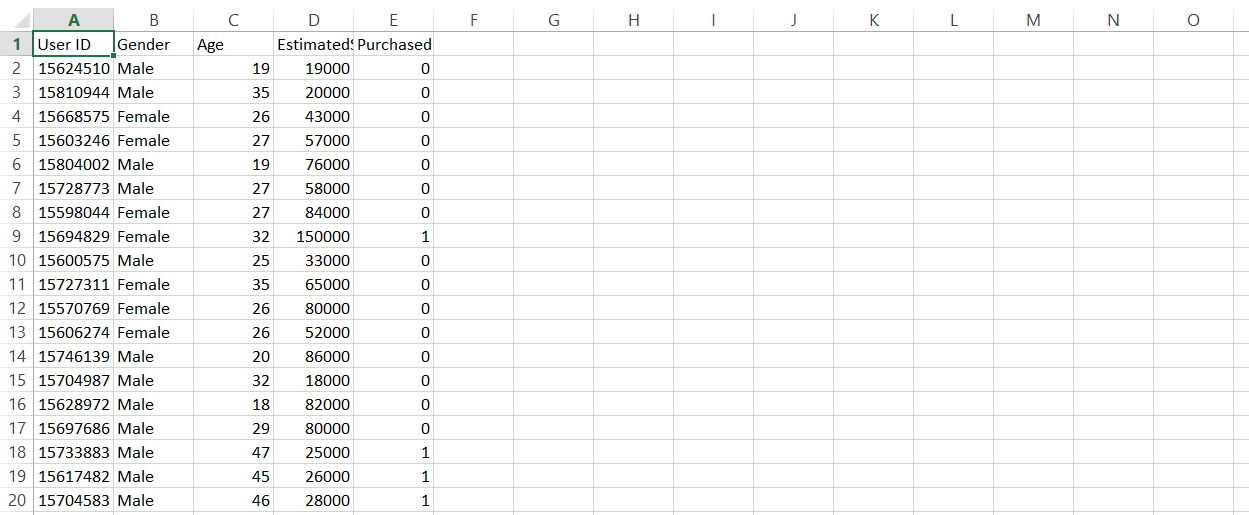
To understand and find the suitable ML model for the given Social\_Network\_Ads.csv dataset, and to retrieve the given output plot.

Abstract:

The given problem deals with the creation of a suitable classification model for the given social network advertisement. The tool used here is Python. The model here is implemented by using Random forest algorithm. For the given dataset, we first explore the data and analyse each and every variable, then split data into train and test sets, define a Random Forest classifier from scratch and achieve the final output plot.

About the Dataset:

The dataset contains some information about all of our users in the social network, including their User ID, Gender, Age, and Estimated Salary. The last column of the dataset is a vector of Booleans describing whether or not each individual ended up clicking on the advertisement (0 = False, 1 = True).

page1image6433088page1image6433280page1image6434048page1image6433472page1image6434432

1

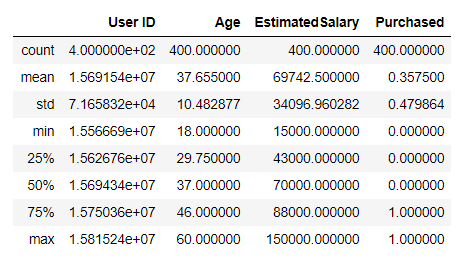
Exploratory Data Analysis:

First we read the dataset from the system. Then we extract X and Y variables. The X variable consists of Age and Estimated Salary, whereas the Y variable consists of the Purchase column, which states whether the customer has purchased or not. In order to understand the data better first we describe the data, which gives the summary statistics of each and every variable.

Code:

data.describe()

Output:

page2image6310144page2image6320512page2image6320320

Here the detailed summary for our data is given with count mean, std, min, max, etc. for all the variables in the dataset.

Code:

data.isnull().sum()

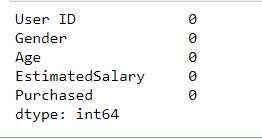
Output:

Now we check for null values in the dataset. More the null values lesser the accuracy. Here we can see that there are no null values. So we need not pre-process the data.

Code:

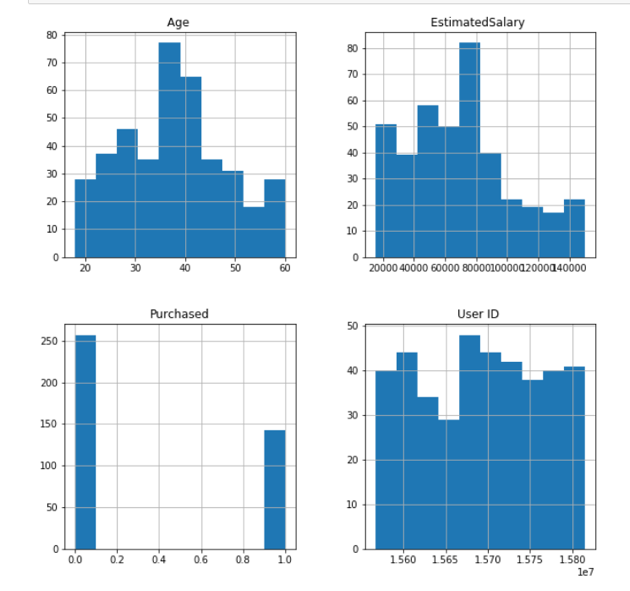
data.hist(figsize=(10,10))

plt.show()

page2image6332480page2image6332096page2image6333056

2

Output:

page3image6486272

The histograms of each variable clearly shows their range and how they have influenced the data. Age falls between 20-60. Estimated Salary falls between 20k-1L. Purchased is the target variable which consists of 0s and 1s. User IDs are scattered throughout the data.

Finding the best ML model:

Code:

lr = LR(C = 0.2, max\_iter = 1000)  
nb = NB()  
dt = tree.DecisionTreeClassifier(criterion='entropy') rf = RFC(max\_depth=5, random\_state=0)  
svm = SVC(probability=True)  
knn = KNN(n\_neighbors = 5, metric = 'minkowski', p = 2) models = []  
models.append(('LR', lr))  
models.append(('NB', nb))  
models.append(('DT', dt))  
models.append(('RF', rf))  
models.append(('SVM', svm))  
models.append(('KNN', knn))  
results = []

page3image6487232page3image6486080

3

names = []

scoring = 'accuracy'

X\_train,X\_test,y\_train,y\_test = TTS(X, y, test\_size = 0.3, random\_state = 0)

for name, model in models:

kfold = model\_selection.KFold(n\_splits = 10)

cv\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring=scoring)

results.append(cv\_results)

names.append(name)

msg = "%s : %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

fig = plt.figure()

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

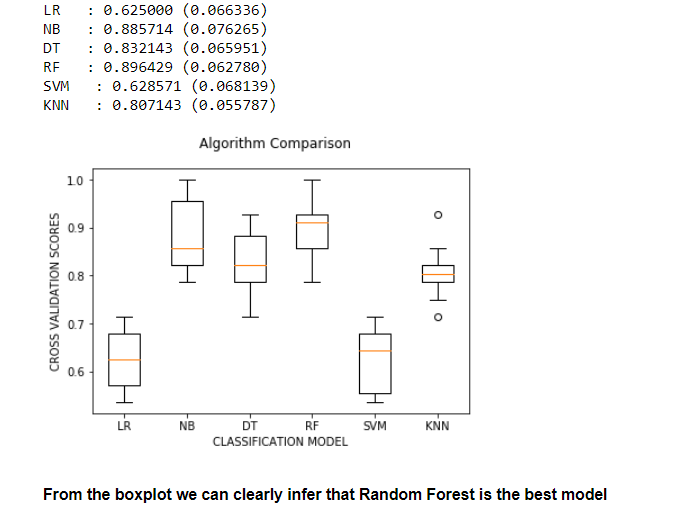
plt.xlabel("CLASSIFICATION MODEL")

plt.ylabel("CROSS VALIDATION SCORES")

ax.set\_xticklabels(names)

plt.show()

Output:

page4image6588288

4

Here we have built a code and a boxplot that gives the accuracies of all the models, from which we found that Random forest is the best model that suits our dataset with an accuracy of 89%. So we will now build the random forest classifier from scratch.

Implementing the Random Forest Classifier:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. Here we have done the same that is we constructed a decision tree first and then we call the function into the random forest The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

Code:

def entropy(y):

hist = np.bincount(y)

ps = hist / len(y)

return -np.sum([p \* np.log2(p) for p in ps if p > 0])

class Node:

def \_\_init\_\_(self, feature=None, threshold=None, left=None, right=None, \*, value=None):

self.feature = feature

self.threshold = threshold

self.left = left

self.right = right

self.value = value

def is\_leaf\_node(self):

return self.value is not None

class DecisionTree:

page5image6590976page5image6591168

5

def \_\_init\_\_(self, min\_samples\_split=2, max\_depth=100, n\_feats=None):

self.min\_samples\_split = min\_samples\_split self.max\_depth = max\_depth  
self.n\_feats = n\_feats  
self.root = None

def fit(self, X, y):

self.n\_feats = X.shape[1] if not self.n\_feats else min(self.n\_feats, X.shape[1])

self.root = self.\_grow\_tree(X, y)

def predict(self, X):

return np.array([self.\_traverse\_tree(x, self.root) for x in

X])

def \_grow\_tree(self, X, y, depth=0):

n\_samples, n\_features = X.shape

n\_labels = len(np.unique(y))

# stopping criteria

if (depth >= self.max\_depth

or n\_labels == 1

or n\_samples < self.min\_samples\_split):

leaf\_value = self.\_most\_common\_label(y)

return Node(value=leaf\_value)

feat\_idxs = np.random.choice(n\_features, self.n\_feats, replace=False)

# greedily select the best split according to information

gain

best\_feat, best\_thresh = self.\_best\_criteria(X, y, feat\_idxs)

# grow the children that result from the split

left\_idxs, right\_idxs = self.\_split(X[:, best\_feat], best\_thresh)

left = self.\_grow\_tree(X[left\_idxs, :], y[left\_idxs], depth+1)

right = self.\_grow\_tree(X[right\_idxs, :], y[right\_idxs], depth+1)

return Node(best\_feat, best\_thresh, left, right) def \_best\_criteria(self, X, y, feat\_idxs):

best\_gain = -1

6

threshold)

X\_column = X[:, feat\_idx]

thresholds = np.unique(X\_column)

for threshold in thresholds:

gain = self.\_information\_gain(y, X\_column,

if gain > best\_gain:

best\_gain = gain

split\_idx = feat\_idx

split

parent\_entropy = entropy(y)  
# generate split  
left\_idxs, right\_idxs = self.\_split(X\_column, split\_thresh) if len(left\_idxs) == 0 or len(right\_idxs) == 0:

return 0  
# compute the weighted avg. of the loss for the children n = len(y)  
n\_l, n\_r = len(left\_idxs), len(right\_idxs)  
e\_l, e\_r = entropy(y[left\_idxs]), entropy(y[right\_idxs]) child\_entropy = (n\_l / n) \* e\_l + (n\_r / n) \* e\_r  
# information gain is difference in loss before vs. after

ig = parent\_entropy - child\_entropy

split\_idx, split\_thresh = None, None

for feat\_idx in feat\_idxs:

split\_thresh = threshold

return split\_idx, split\_thresh

def \_information\_gain(self, y, X\_column, split\_thresh): # parent loss

return ig

def \_split(self, X\_column, split\_thresh):

left\_idxs = np.argwhere(X\_column <= split\_thresh).flatten() right\_idxs = np.argwhere(X\_column > split\_thresh).flatten() return left\_idxs, right\_idxs

def \_traverse\_tree(self, x, node):

if node.is\_leaf\_node():

return node.value

7

if x[node.feature] <= node.threshold:  
return self.\_traverse\_tree(x, node.left)

return self.\_traverse\_tree(x, node.right) def \_most\_common\_label(self, y):

counter = Counter(y)  
most\_common = counter.most\_common(1)[0][0] return most\_common

import numpy as np

from collections import Counter

def bootstrap\_sample(X, y):

n\_samples = X.shape[0]  
idxs = np.random.choice(n\_samples, n\_samples, replace=True) return X[idxs], y[idxs]

def most\_common\_label(y):  
counter = Counter(y)  
most\_common = counter.most\_common(1)[0][0] return most\_common

class RandomForest:  
def \_\_init\_\_(self, n\_trees=10, min\_samples\_split=2,

max\_depth=100, n\_feats=None):

self.n\_trees = n\_trees

self.min\_samples\_split = min\_samples\_split self.max\_depth = max\_depth  
self.n\_feats = n\_feats  
self.trees = []

def fit(self, X, y):

self.trees = []

for \_ in range(self.n\_trees):

tree = DecisionTree(min\_samples\_split=self.min\_samples\_split,

max\_depth=self.max\_depth, n\_feats=self.n\_feats) X\_samp, y\_samp = bootstrap\_sample(X, y) tree.fit(X\_samp, y\_samp)  
self.trees.append(tree)

8

def predict(self, X):

tree\_preds = np.array([tree.predict(X) for tree in self.trees])

tree\_preds = np.swapaxes(tree\_preds, 0, 1)

y\_pred = [most\_common\_label(tree\_pred) for tree\_pred in tree\_preds]

return np.array(y\_pred)

import numpy as np

import pandas as pd  
from sklearn import datasets  
from sklearn.model\_selection import train\_test\_split def accuracy(y\_true, y\_pred):

accuracy = np.sum(y\_true == y\_pred) / len(y\_true)

return accuracy

data = pd.read\_csv(r'C:\Users\Selva Vignesh M\Desktop\Social\_Network\_Ads.csv')

X = data.iloc[:, [2, 3]].values

y = data.iloc[:, 4].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=123)

clf = RandomForest(n\_trees=3, max\_depth=10) clf.fit(X\_train, y\_train)  
y\_pred = clf.predict(X\_test)  
acc = accuracy(y\_test, y\_pred)

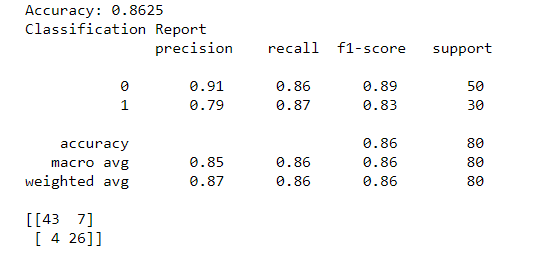
print ("Accuracy:", acc)  
from sklearn.metrics import confusion\_matrix,classification\_report

print('Classification Report \n', classification\_report(y\_test, y\_pred))

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred) print(cm)

9

Output:

page10image6574592

The accuracy we achieved from the random forest model which is built from scratch is 86% which is a good accuracy. Using built-in functions will surely improve the accuracy of the model. Now comes the classification report which displays the precision, recall, F1 and support scores for the model.

* •  Precision Score for class 0 (negative) is 0.91 and for class 1 (positive) is 0.84, indicating the preciseness of the model which is so accurate.
* •  Recall value for class 0 is 0.76 and class 1 is 0.87, which describes the amount up – to which the model can predict the output.
* •  As the precision and recall values are not similar, there are dominance in classes. From the confusion matrix we can see that there are 43 true positives, 7 false positives, 4

false negatives and 26 true negative values.

Built-in Random Forest model:

In order to obtain the given output plot we need to build once again the random forest model using built-in functions as scratch model did not achieve the exact plot.

Code:

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

page10image6574784page10image6574976

10

Model Visualization:

page11image6577472

from matplotlib.colors import ListedColormap

from matplotlib import pyplot as plt

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop =

X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

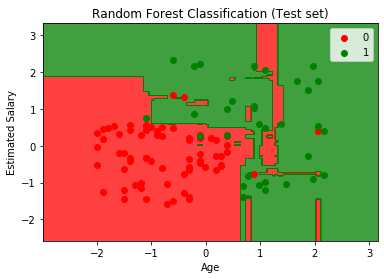
plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],  
c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Random Forest Classification (Test set)') plt.xlabel('Age')  
plt.ylabel('Estimated Salary')  
plt.legend()

plt.show()

Output:

Finally we have achieved the given output plot for the random forest model with and accuracy of 86%. From the graph we can infer that most of the data points are predicted correctly and rarely a few data points are misinterpreted. But we can see that there are some

page11image6577664

11

red spots in the green region which infers that the classifier needs more essential attributes to perform well and boost up the classifier.

Conclusion:

But while there is always room for improvement, we can be satisfied with this model as our final product. Our accuracy is high, but not so high that we need to be suspicious of any over fitting. We can safely say that an increase in both Age and Estimated Salary will lead to a higher probability of clicking the advertisement. As new users sign-up for the website, we can use this model to quickly determine whether or not to expose them to this particular ad or choose another that is more relevant to their profile.

page12image6579968

12