Personal Loan Prediction

April 15, 2024

1 KNN Classifier

1.1 Author - Dev

Importing all the needed modules :

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn import preprocessing, neighbors, model_selection, metrics
```

1.1.1 Data Reading

[2]:
```
df = pd.read_csv("/Users/devmarwah/Downloads/UniversalBank.csv")
```

Printing head and shape of the data to have a look at it :

[3]:
```
df.head()
```

[3]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	\
0	1	25	1	49	91107	4	1.6	1	0	
1	2	45	19	34	90089	3	1.5	1	0	
2	3	39	15	11	94720	1	1.0	1	0	
3	4	35	9	100	94112	1	2.7	2	0	
4	5	35	8	45	91330	4	1.0	2	0	

	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	0	1	0	0	0
1	0	1	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

[4]:
```
df.shape
```

[4]:
```
(5000, 14)
```

1.1.2 Data Preparation

We will be dropping ID and ZIP code since they are irrelavent to the target variable **Personal Loan**

[5]:
```
df.drop(["ID","ZIP Code"], axis=1,inplace = True)
```

[6]:
```
df.head()
```

[6]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	\
0	25	1	49	4	1.6	1	0	0	
1	45	19	34	3	1.5	1	0	0	
2	39	15	11	1	1.0	1	0	0	
3	35	9	100	1	2.7	2	0	0	
4	35	8	45	4	1.0	2	0	0	

	Securities Account	CD Account	Online	CreditCard
0	1	0	0	0
1	1	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	1

Here, the factor education has more than 2 categories (3). Therefore, we need to convert it into dummy variables for Knn algorithm to be able to work the right way.

[7]:
```
df.dtypes
```

[7]:
```
Age                    int64
Experience             int64
Income                 int64
Family                 int64
CCAvg                float64
Education              int64
Mortgage               int64
Personal Loan          int64
Securities Account     int64
CD Account             int64
Online                 int64
CreditCard             int64
dtype: object
```

We need to be converting Education to a cateorical variable to convert it into dummy variables.

[8]:
```
df["Education"] = df["Education"].astype("category")
df["Education"].dtype
```

[8]:
```
CategoricalDtype(categories=[1, 2, 3], ordered=False)
```

[9]:
```
df.dtypes
```

[9]:
```
Age                     int64
Experience              int64
Income                  int64
Family                  int64
CCAvg                 float64
Education            category
Mortgage                int64
Personal Loan           int64
Securities Account      int64
CD Account              int64
Online                  int64
CreditCard              int64
dtype: object
```

[10]:
```
df=pd.get_dummies(df,"Education").astype("int")
```

Verifying dummies by displaying head of the dataframe :

[11]:
```
df.head()
```

[11]:

	Age	Experience	Income	Family	CCAvg	Mortgage	Personal Loan	\
0	25	1	49	4	1	0	0	
1	45	19	34	3	1	0	0	
2	39	15	11	1	1	0	0	
3	35	9	100	1	2	0	0	
4	35	8	45	4	1	0	0	

	Securities Account	CD Account	Online	CreditCard	Education_1	\
0	1	0	0	0	1	
1	1	0	0	0	1	
2	0	0	0	0	1	
3	0	0	0	0	0	
4	0	0	0	1	0	

	Education_2	Education_3
0	0	0
1	0	0
2	0	0
3	1	0
4	1	0

We can observe above that the factor “Education” has been converted to three dummy variables.

Normalizing Data

[12]:
```
df_norm = pd.DataFrame(preprocessing.StandardScaler().fit_transform(df.iloc[:
↪,np.r_[0:6,7:11]]))
```

[13]:
```
df.iloc[:,np.r_[0:6,7:11]] = df_norm
```

[14]:
```
df.head()
```

[14]:

	Age	Experience	Income	Family	CCAvg	Mortgage	\
0	-1.774417	-1.666078	-0.538229	1.397414	-0.295024	-0.555524	
1	-0.029524	-0.096330	-0.864109	0.525991	-0.295024	-0.555524	
2	-0.552992	-0.445163	-1.363793	-1.216855	-0.295024	-0.555524	
3	-0.901970	-0.968413	0.569765	-1.216855	0.279176	-0.555524	
4	-0.901970	-1.055621	-0.625130	1.397414	-0.295024	-0.555524	

	Personal Loan	Securities Account	CD Account	Online	CreditCard	\
0	0	2.928915	-0.25354	-1.216618	-0.645314	
1	0	2.928915	-0.25354	-1.216618	-0.645314	
2	0	-0.341423	-0.25354	-1.216618	-0.645314	
3	0	-0.341423	-0.25354	-1.216618	-0.645314	
4	0	-0.341423	-0.25354	-1.216618	1.549632	

	Education_1	Education_2	Education_3
0	1	0	0
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0

1.1.3 Model Construction

Splitting data into training and testing data :

[15]:
```
x = np.array(df.drop("Personal Loan",axis=1))
y = np.array(df["Personal Loan"])
t=model_selection.train_test_split
x_train,x_test, y_train ,y_test = t(x,y,test_size=0.2,random_state=2)
```

1.2 K-means

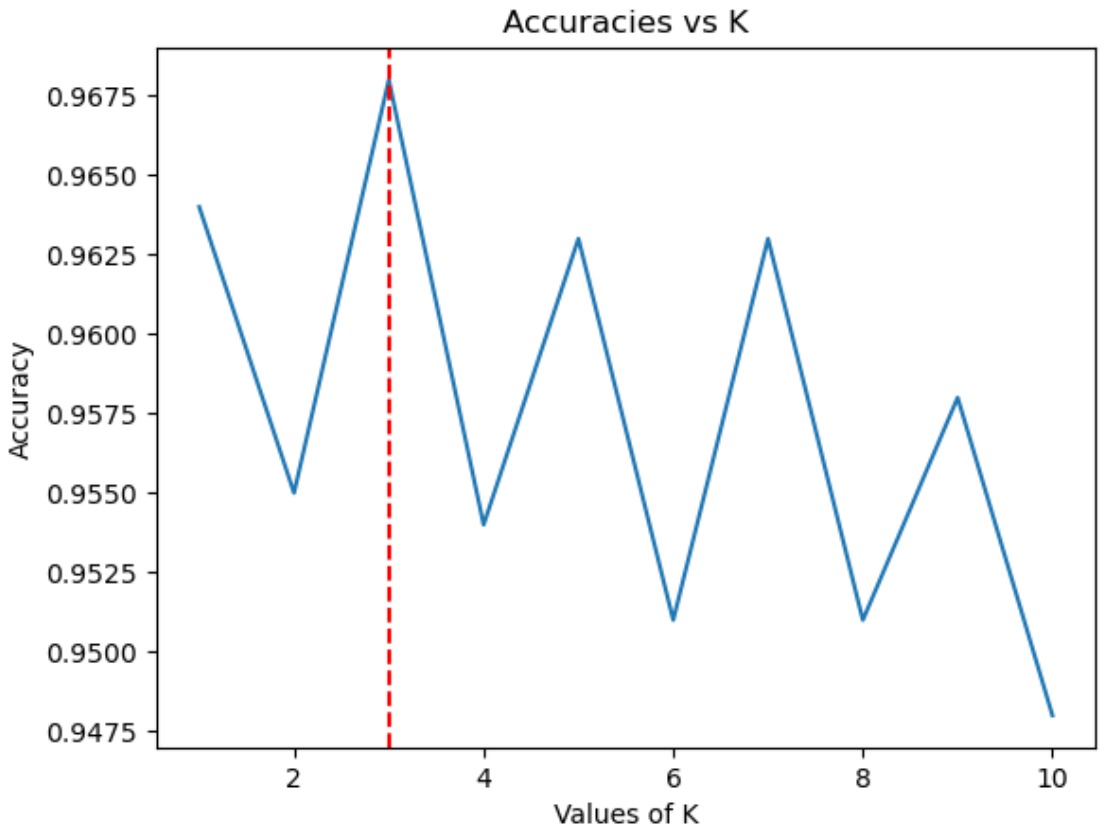
Training model for a range of k to find the optimal value of k

[16]:
```
# Setting a range of values for k
k_values = range(1,11)
# Making empty list for accuracies
accuracies = []
# Training the model
for k in k_values:
    clf=neighbors.KNeighborsClassifier(n_neighbors = k)
    clf.fit(x_train,y_train)
    accuracy = clf.score(x_test,y_test)
    accuracies.append(accuracy)
```

Plotting accuracies vs k values to get the optimal value of k

[17]:
```
plt.plot(k_values,accuracies)
plt.title("Accuracies vs K")
plt.xlabel('Values of K')
plt.ylabel("Accuracy")
plt.axvline(x=3,color="r",linestyle="--")
```

[17]:
```
<matplotlib.lines.Line2D at 0x13e4c35d0>
```



Highest accuracy is for the value of k so we will train our model for k=3

[18]:
```
clf=neighbors.KNeighborsClassifier(n_neighbors=3)
clf.fit(x_train,y_train)
```

[18]:
```
KNeighborsClassifier(n_neighbors=3)
```

Finding accuracy of the model

[19]:
```
print("Accuracy :",clf.score(x_test,y_test))
```

```
Accuracy : 0.968
```

Hence, our Knn model is 96.8% accurate

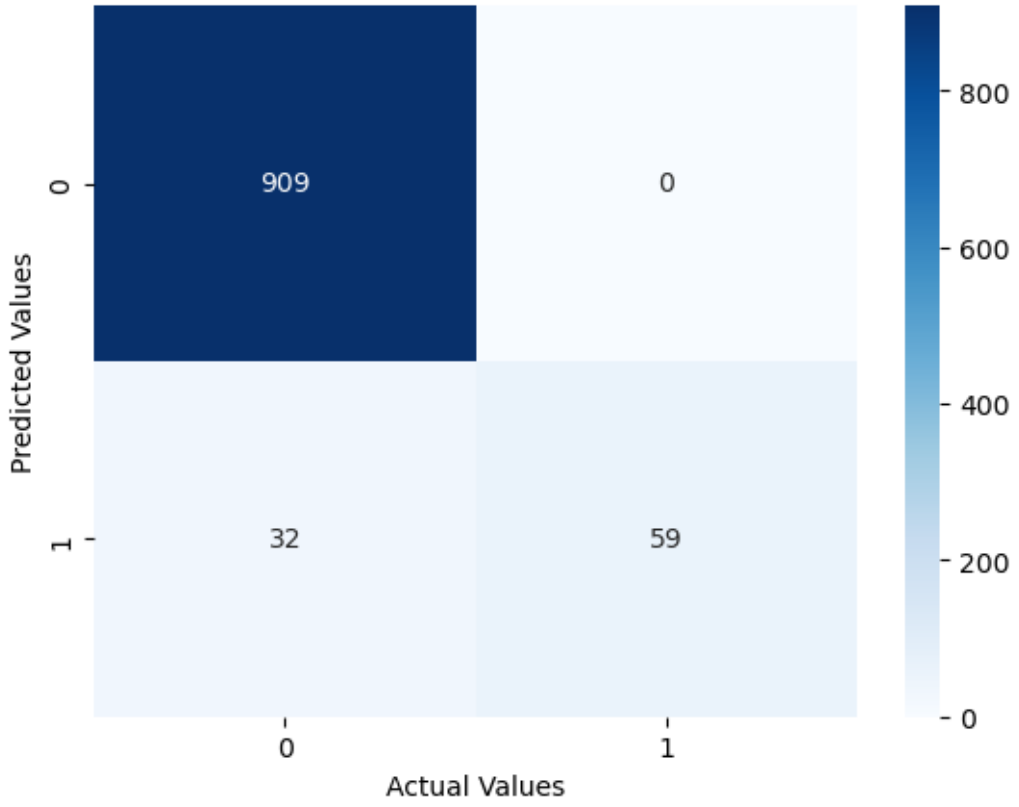
Making predictions using this model

[20]:
```
p = clf.predict(x_test)
```

Making confusion matrix of final results :

[21]:
```
cm = metrics.confusion_matrix(y_test,p)
plt.figure()
sb.
  ↪heatmap(cm,cmap="Blues",xticklabels=["0","1"],yticklabels=["0","1"],annot=True,fmt="d")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
```

[21]:
```
Text(50.722222222222214, 0.5, 'Predicted Values')
```



1.2.1 Decision Trees

[22]:
```
from sklearn.tree import DecisionTreeClassifier , plot_tree
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train,y_train)
```

[22]:
```
DecisionTreeClassifier(random_state=42)
```

Scoring Decision tree model on testing data

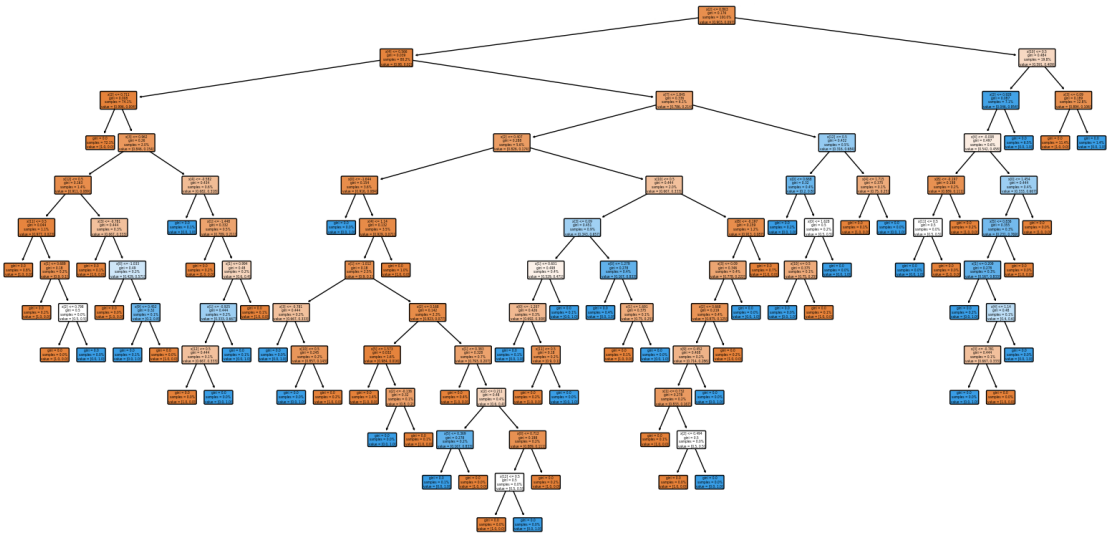
[23]:
```
model.score(x_test,y_test)
```

[23]:
```
0.985
```

Hence, our decision tree model provide us with a better accuracy of 98.5%

Visualizing Decision Trees

[24]:
```
plt.figure(figsize=(20,10))
plot_tree(model,filled=True,rounded=True,proportion=True)
plt.show();
```



1.2.2 Random Forest

Now, we will try the Random Forest as our final model to classify our data.

[25]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=200,random_state=42)
model.fit(x_train,y_train)
```

[25]:
```
RandomForestClassifier(n_estimators=200, random_state=42)
```

Scoring Random Forest Model on testing dataset.

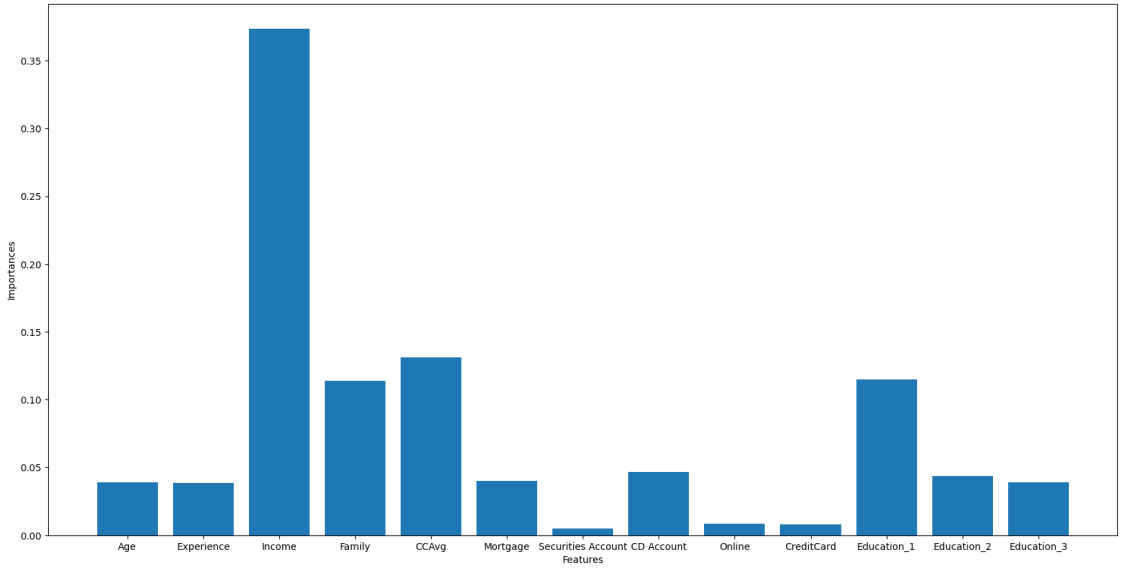
[26]:
```
model.score(x_test,y_test)
```

[26]:
```
0.988
```

Random Forest Model gives us the highest accuracy of 98.8 %

Checking for variable importance :

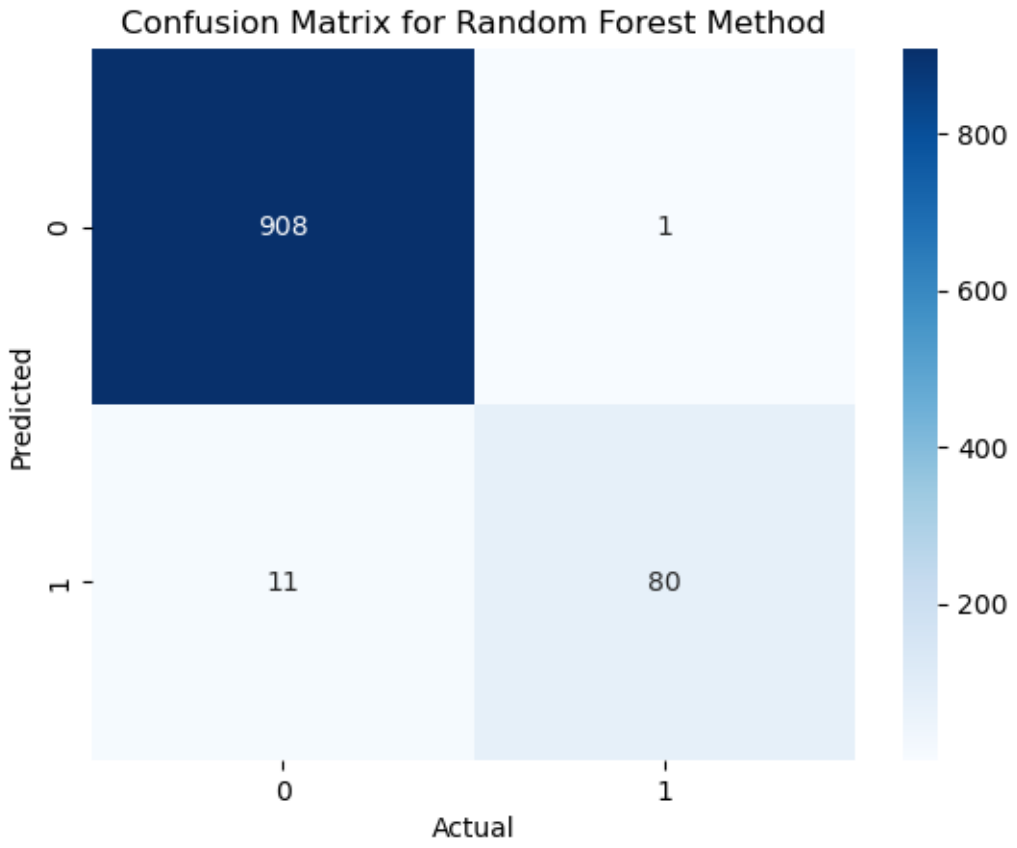
[27]:
```
l=list(df.columns)
l.remove("Personal Loan")
importances = model.feature_importances_
index=range(len(importances))
plt.figure(figsize=(20,10))
plt.bar(index,importances)
plt.xlabel("Features")
plt.ylabel("Importances")
plt.xticks(index,l)
plt.show()
```



Plotting confusion Matrix:

[28]:
```
p=model.predict(x_test)
cm=metrics.confusion_matrix(y_test,p)
sb.heatmap(cm,cmap="Blues",annot=True,fmt="d")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Confusion Matrix for Random Forest Method")
```

[28]:
```
Text(0.5, 1.0, 'Confusion Matrix for Random Forest Method')
```


Confusion Matrix for Random Forest Method
908
1
11
80
0
1
Predicted
Actual
800
600
400
200