Introduction

For this final project on Supervised Machine Learning, we will be analyzing the personal bank loans and determining what features are associated with people earning a personal bank loan. We will be using the Adaboost algorithm on selected features to identify the best hyperparameters to use and see how accurate our model predicts on test data.

The data we'll be using is the Bank_Personal_Loan_Modelling dataset that's publicly avaiable on kaggle.com

About the dataset (data source form kaggle: https://www.kaggle.com/datasets/zohrehtofighizavareh/bank-personal-loan)

This dataset contains the information of more than 5000 customers, based on the points that each customer has earned, a loan is offered to them. The features are:

Age: Customer's age in completed years **Experience:** Years of professional experience **Income:** Annual income of the customer **Zip code:** home address Zip code **Family:** Family size of customer **CCAvg:** Spending on credit cards per month **Education:** Education level (Undergraduate=1, Graduate= 2, Advanced=3) **Mortgage:** Value of house mortgage if any **Personalloan:** Did this customer accept the personal loan offered in the last campaign? **Securityaccount:** Does the customer have a securities account**with this bank?**Cd_account: **Does the customer have a certificate of deposit (CD) account with this bank?**Online: **Does the customer use internet banking facilities?**Creditcard:__ Does the customer use a credit card issued by Universal Bank?

Project Overview

1. Load data, Exploratory Data Analysis (EDA), and cleaning
2. Prepare dateset for model training
3. Training the model using Adaboost and determining best hyperparameters
4. Predictions and results
5. Discussion/Conclusion

1. Load data, Exploratory Data Analysis (EDA), and cleaning

In [1]:

```
##Load packages

import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib.colors import Normalize
import matplotlib.pyplot as plt
import math
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
##from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import roc_curve
from sklearn.metrics import RocCurveDisplay
```

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay
```

In [2]:

```
df= pd.read_csv('Bank_Personal_Loan_Modelling.csv', sep= ',')
df.head(5)
```

Out[2]:

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

In [3]:

```
## Check for Null values
print(df.isnull().sum())

## Check infomation of the data types
print(df.dtypes)
```

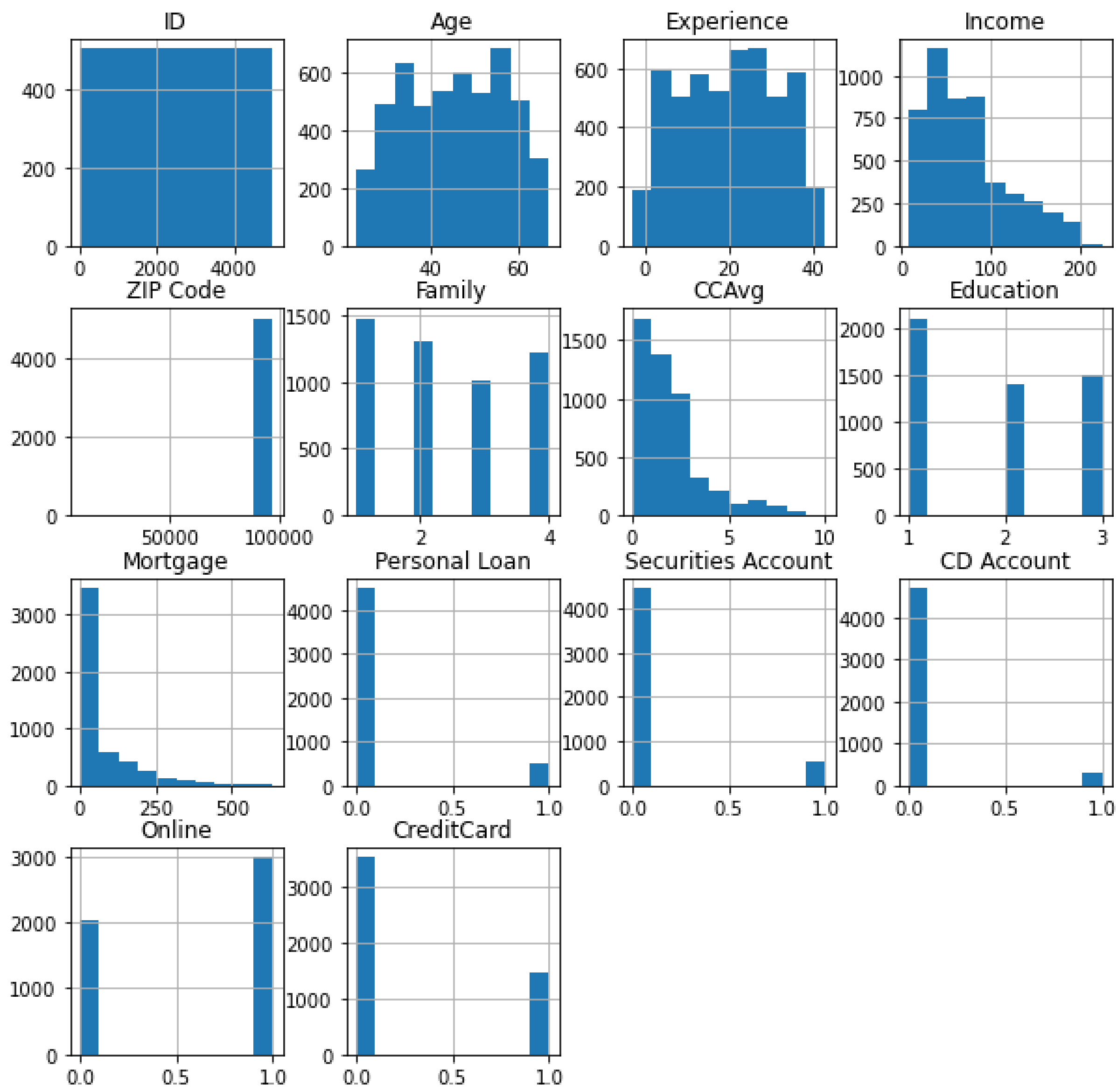
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
ID                      int64
Age                     int64
Experience              int64
Income                  int64
ZIP Code                int64
Family                  int64
CCAvg                 float64
Education               int64
Mortgage                int64
Personal Loan           int64
Securities Account      int64
CD Account              int64
Online                  int64
CreditCard              int64
dtype: object
```

In [4]:

```
##Plot and visulaize histogram of each column in the dataframe

dfhistall= df.hist(bins=10, figsize= (10,10))
print(dfhistall)
```

```
[[<AxesSubplot:title={'center':'ID'}>
```

```
 <AxesSubplot:title={'center':'Age'}>
 <AxesSubplot:title={'center':'Experience'}>
 <AxesSubplot:title={'center':'Income'}>]
[<AxesSubplot:title={'center':'ZIP Code'}>
 <AxesSubplot:title={'center':'Family'}>
 <AxesSubplot:title={'center':'CCAvg'}>
 <AxesSubplot:title={'center':'Education'}>]
[<AxesSubplot:title={'center':'Mortgage'}>
 <AxesSubplot:title={'center':'Personal Loan'}>
 <AxesSubplot:title={'center':'Securities Account'}>
 <AxesSubplot:title={'center':'CD Account'}>]
[<AxesSubplot:title={'center':'Online'}>
 <AxesSubplot:title={'center':'CreditCard'}> <AxesSubplot:>
 <AxesSubplot:>]]
```



In [5]:

```
## For this dataset, it would not make since to have negative values in any columns
## (besides the label values  for Adaboost classfier using -1 and 1)
## For example, there can not be negative age, experience, income or family
## For this reason we'll check for any number in any columns that are below 0 and remove t

## Now lets check the number of unique values for each column
for c in list(df.columns):
    print("Num of unique values for", c,"=", df[c].nunique(), "& presence of neg =", (df[c
```

```
Num of unique values for ID = 5000 & presence of neg = False
Num of unique values for Age = 45 & presence of neg = False
Num of unique values for Experience = 47 & presence of neg = True
Num of unique values for Income = 162 & presence of neg = False
```

```
Num of unique values for ZIP Code = 467 & presence of neg = False
Num of unique values for Family = 4 & presence of neg = False
Num of unique values for CCAvg = 108 & presence of neg = False
Num of unique values for Education = 3 & presence of neg = False
Num of unique values for Mortgage = 347 & presence of neg = False
Num of unique values for Personal Loan = 2 & presence of neg = False
Num of unique values for Securities Account = 2 & presence of neg = False
Num of unique values for CD Account = 2 & presence of neg = False
Num of unique values for Online = 2 & presence of neg = False
Num of unique values for CreditCard = 2 & presence of neg = False
```

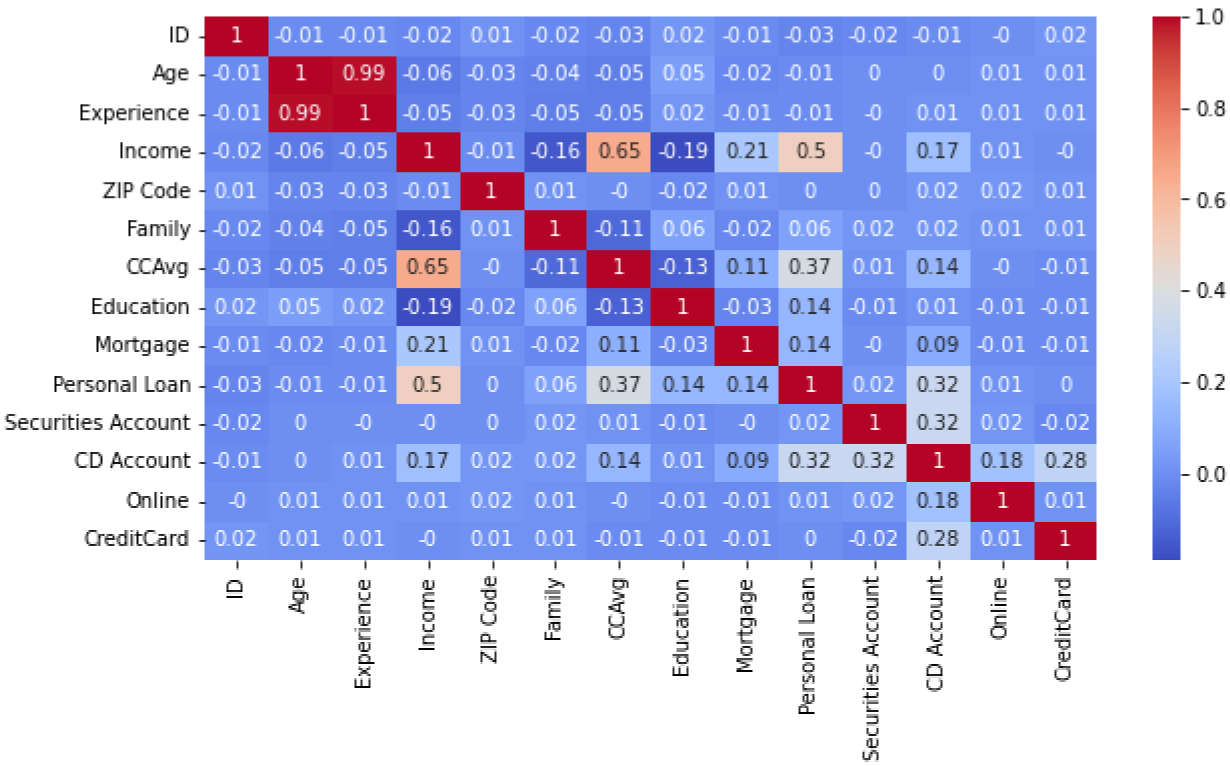
In [6]:

```
## Clean data
## Drop rows with any negative number

df2= df.drop(df[ df['Experience'] <0].index)
print("Presence of neg for Experience=", (df2['Experience']<0).any())
```

```
Presence of neg for Experience= False
```

In [7]:

```
## Let's check the correlation of the columns
plt.rcParams["figure.figsize"]=10,5
mat= df2.corr().round(2)
sns.heatmap(mat, annot=True,
            cmap="coolwarm")
plt.show()
```

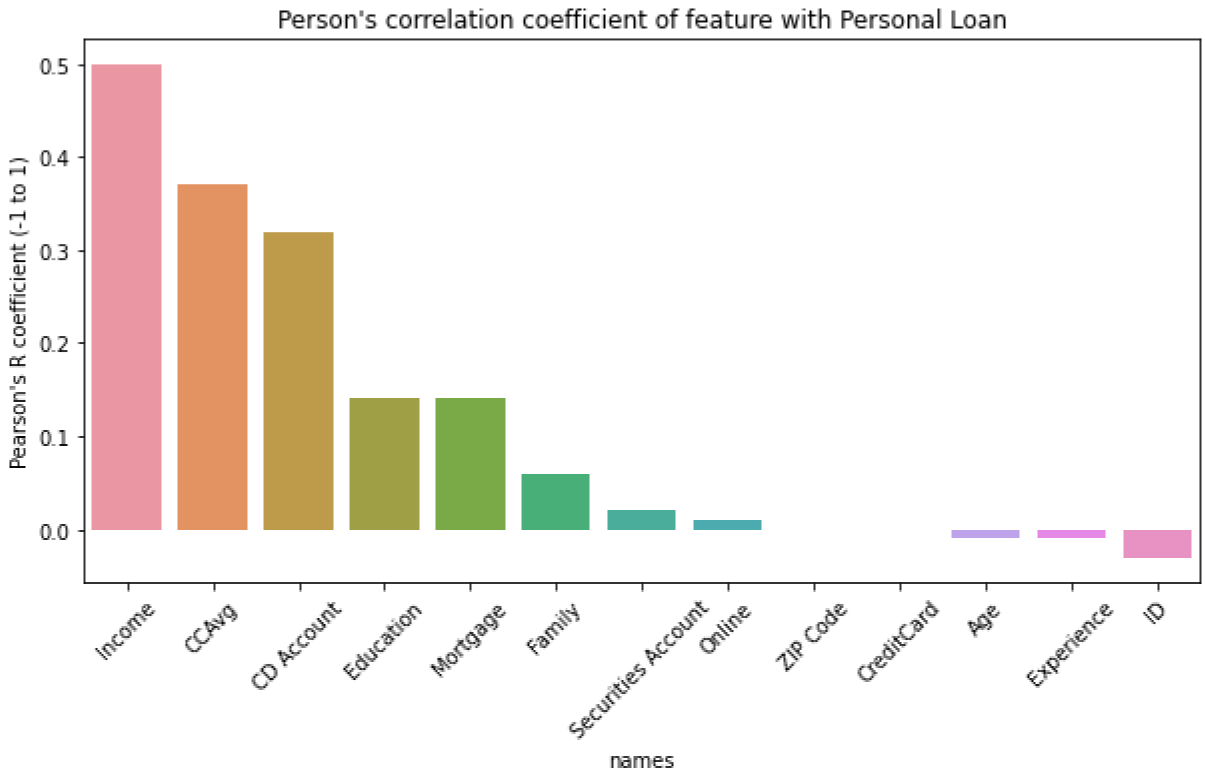


In [8]:

```
## Order and plot the coorelation matrix pearsons r for each feature
## and relation with our label to predict, personal loan

newmat= mat.sort_values(['Personal Loan'], ascending= False)
perloan= newmat['Personal Loan']

##newdataframe for plotting. remove personal load in row
perloan2= perloan.rename_axis("names").reset_index()
perloan2= perloan2[perloan2['Personal Loan'] <1]
print(perloan2)

##plot
```

```
featsbar= perloan2['names']
loanbar= perloan2['Personal Loan']
sns.set_palette('Set2')
sns.barplot(x=featsbar, y=loanbar)
plt.xticks(rotation=45)
plt.ylabel("Pearson's R coefficient (-1 to 1)")
plt.title("Person's correlation coefficient of feature with Personal Loan")
plt.show()
```

```
                 names  Personal Loan
1               Income           0.50
2                CCAvg           0.37
3           CD Account           0.32
4            Education           0.14
5             Mortgage           0.14
6               Family           0.06
7   Securities Account           0.02
8               Online           0.01
9             ZIP Code           0.00
10          CreditCard           0.00
11                 Age          -0.01
12          Experience          -0.01
13                  ID          -0.03
```



In [9]:

```
## Finally, let's drop the columns we won't be using in modeling and predictions.
## From our data we'll remove Zip Code, CreditCard, Age Experience and ID because of the 
## Zip code and ID also have many unique values and are not continuous or binary variable
## makes sense to remove for that reason as well

unnec_feats= ['ZIP Code', 'CreditCard', 'Age', 'Experience', 'ID']
df3= df2.drop(unnec_feats, axis=1)

## Check end of dataframe
df3.tail(5)
```

Out[9]:

	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online
4995	40	1	1.9	3	0	0	0	0	1

	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online
4996	15	4	0.4	1	85	0	0	0	1
4997	24	2	0.3	3	0	0	0	0	0
4998	49	3	0.5	2	0	0	0	0	1
4999	83	3	0.8	1	0	0	0	0	1

2. Prepare dateset for model training

In [10]:

```
## Create X and y from the cleaned dataframe
dfcleaned= df3
X=dfcleaned.loc[:, dfcleaned.columns !='Personal Loan']
y=dfcleaned.loc[:, dfcleaned.columns =='Personal Loan']

X_train, X_test, y_train, y_test= train_test_split(X,y,
                                  random_state=1234,
                                  test_size=0.25,
                                  shuffle=True)

## Check to see if dimensions for train and test are correct (75% and 25%)
print("Train X rows:",X_train.shape[0],
      "| Test X rows:", X_test.shape[0],
      "| Train y rows:", y_train.shape[0],
      "| Test y rows:", y_test.shape[0])
```

```
Train X rows: 3711 | Test X rows: 1237 | Train y rows: 3711 | Test y rows: 1237
```

3. Training the model using Adaboost and determining best hyperparameters

In [11]:

```
## Quick view of X and y train data
## They need to be in the correct shapes (2d and 1d array) for modeling)
xtrain= np.array(X_train)
ytrain= y_train.values.flatten()
xtest= np.array(X_test)
ytest= y_test.values.flatten()

print(xtrain)
print(ytrain)
```

```
[[141.    2.    4.9 ...   0.    0.    0. ]
 [ 14.    4.    0.4 ...   0.    0.    1. ]
 [ 83.    1.    2.8 ...   0.    0.    0. ]
 ...
 [180.    1.    1.7 ...   0.    0.    1. ]
 [152.    3.    3.3 ...   0.    0.    1. ]
 [ 63.    1.    1.6 ...   0.    0.    1. ]]
[1 0 0 ... 0 1 0]
```

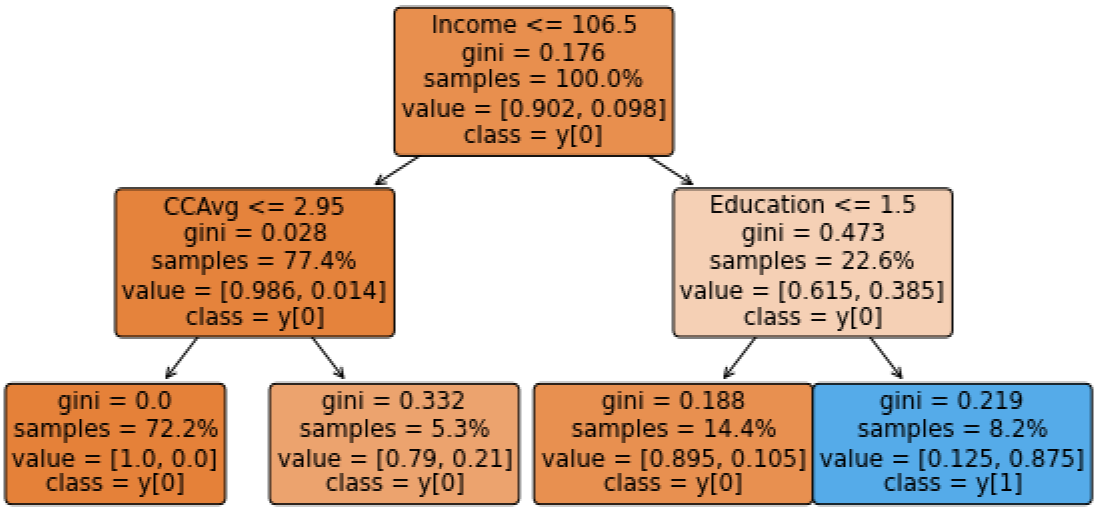
In [13]:

```
## before we go into the next step of grid searching using the adaboost model, lets visua
## The decision tree at depth one will be used as out wek base classifier to grow our stum

treeclf= tree.DecisionTreeClassifier(max_depth=2, random_state=1234)
treeclf= treeclf.fit(xtrain, ytrain)
tree.plot_tree(treeclf, proportion=True,
               class_names=True,
               rounded= True,
```

```
                 feature_names= X_train.columns,
                 filled= True)
```

Out[13]:

```
[Text(279.0, 226.5, 'Income <= 106.5\ngini = 0.176\nsamples = 100.0%\nvalue = [0.902, 0.09
8]\nclass = y[0]'),
 Text(139.5, 135.9, 'CCAvg <= 2.95\ngini = 0.028\nsamples = 77.4%\nvalue = [0.986, 0.014]
\nclass = y[0]'),
 Text(69.75, 45.29999999999998, 'gini = 0.0\nsamples = 72.2%\nvalue = [1.0, 0.0]\nclass =
y[0]'),
 Text(209.25, 45.29999999999998, 'gini = 0.332\nsamples = 5.3%\nvalue = [0.79, 0.21]\nclas
s = y[0]'),
 Text(418.5, 135.9, 'Education <= 1.5\ngini = 0.473\nsamples = 22.6%\nvalue = [0.615, 0.38
5]\nclass = y[0]'),
 Text(348.75, 45.29999999999998, 'gini = 0.188\nsamples = 14.4%\nvalue = [0.895, 0.105]\nc
lass = y[0]'),
 Text(488.25, 45.29999999999998, 'gini = 0.219\nsamples = 8.2%\nvalue = [0.125, 0.875]\ncl
ass = y[1]')]
```



In [14]:

```
## Define our model for Checking multiple hyperparameters to using GridSearchCV to idenfi
adamodel= AdaBoostClassifier(base_estimator= DecisionTreeClassifier(max_depth=2, random_st
                             random_state= 1234)

## Define decitionaries of parameters to use in the grid search
params= {'n_estimators': list(range(25,200+25,25)),
        'learning_rate': list(np.arange(0.100, 1+0.100, 0.100))}

## Grid search fit
grid= GridSearchCV(cv=3,
                   estimator= adamodel,
                   param_grid= params)
#ada_grid= grid.fit(xtrain, ytrain)
##ada_grid
grid.fit(xtrain, ytrain)
```

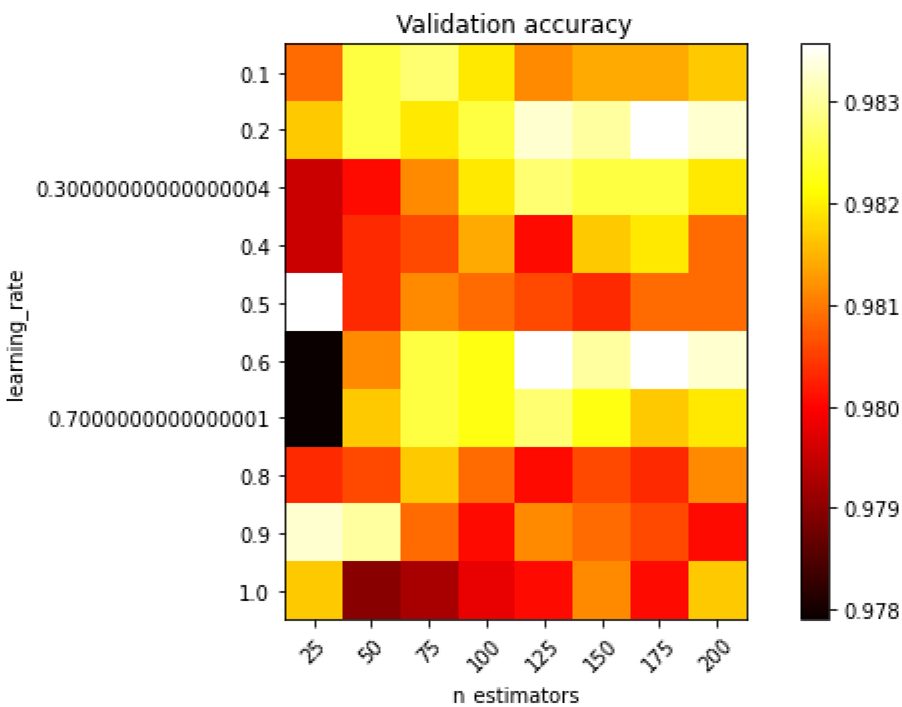
Out[14]:

```
GridSearchCV(cv=3,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=
2,
                                                                                 random_sta
te=1234),
                                          random_state=1234),
             param_grid={'learning_rate': [0.1, 0.2, 0.30000000000000004, 0.4,
                                           0.5, 0.6, 0.7000000000000001, 0.8,
                                           0.9, 1.0],
                         'n_estimators': [25, 50, 75, 100, 125, 150, 175, 200]})
```

In [28]:

```
## Plot for visualizing gridseach of n_estimators and learning_rate
```

```
## (Code repurposed from class week 6 assignment)

def plotSearchGrid(grid):

    scores = [x for x in grid.cv_results_["mean_test_score"]]
    scores = np.array(scores).reshape(len(grid.param_grid["learning_rate"]),len(grid.param

    plt.figure(figsize=(10, 5))
    plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
    plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot) ##cmap="gist_earth")
    plt.xlabel('n_estimators')
    plt.ylabel('learning_rate')
    plt.colorbar()
    plt.xticks(np.arange(len(grid.param_grid["n_estimators"])), grid.param_grid["n_estimat
    plt.yticks(np.arange(len(grid.param_grid["learning_rate"])), grid.param_grid["learning
    plt.title('Validation accuracy')
    plt.show()

plotSearchGrid(grid)
```



In [30]:

```
## What are the best parameters to use from our grid search?

print("Best parameters are:", grid.best_params_)
print("Best accuracy is:", round(grid.best_score_,4))
```

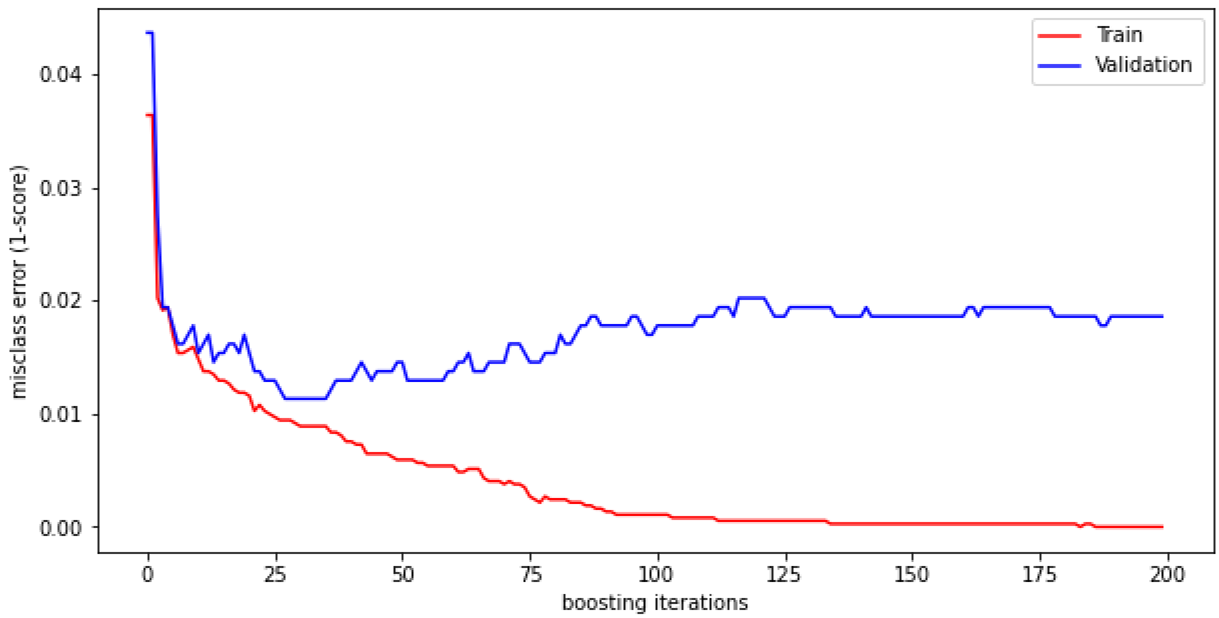
```
Best parameters are: {'learning_rate': 0.2, 'n_estimators': 175}
Best accuracy is: 0.9836
```

4. Predictions and Results

In [31]:

```
## Use adaboost with best n_estimators and learning_rate
clf= AdaBoostClassifier(n_estimators= 200,
                        base_estimator= DecisionTreeClassifier(max_depth=2),
                        learning_rate= .3,
                        random_state= 1234)
clf.fit(xtrain, ytrain)

check_train= clf.staged_score(xtrain, ytrain)
check_test= clf.staged_score(xtest, ytest)
```

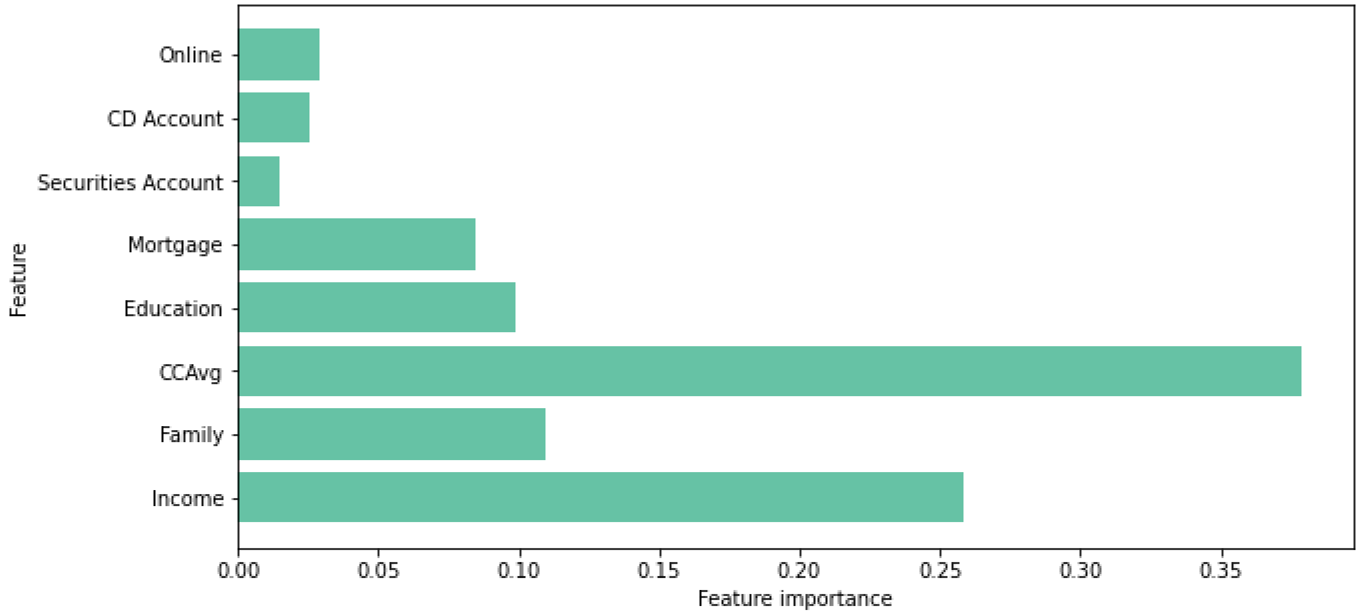
```
##get list of misclass score from generator function
miscl_train= [1-sc for sc in check_train]
miscl_test= [1-sc for sc in check_test]

x= range(200)
plt.plot(x, miscl_train, color= 'r',label= 'Train')
plt.plot(x, miscl_test, color='b', label= 'Validation')
plt.xlabel('boosting iterations')
plt.ylabel('misclass error (1-score)')
##plt.title()
plt.legend()
plt.show()
```



In [32]:

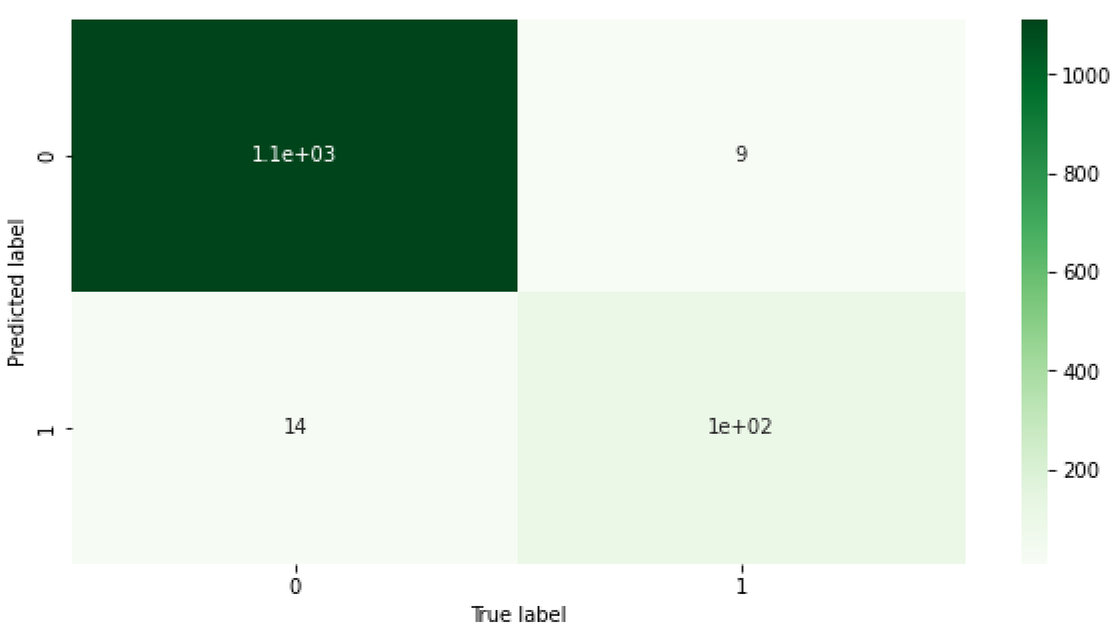
```
##check for important features by rank
impfeats= clf.feature_importances_
print(X_train.head(0))
print(impfeats)

##plot important features
## the feature importance is the amount of information gain determined by the
## average feature importance from our base Decision Tree Classifier

nfeats= X_train.shape[1]
plt.figure(figsize=(10,5))
plt.barh(range(nfeats), clf.feature_importances_, align='center')
plt.yticks(np.arange(nfeats), X_train.columns.values)
plt.xlabel('Feature importance')
plt.ylabel('Feature')
plt.show()

## Income is the feature with the most information gain in our adaboost model
## followed by Credit cards per month (CCavg)
```

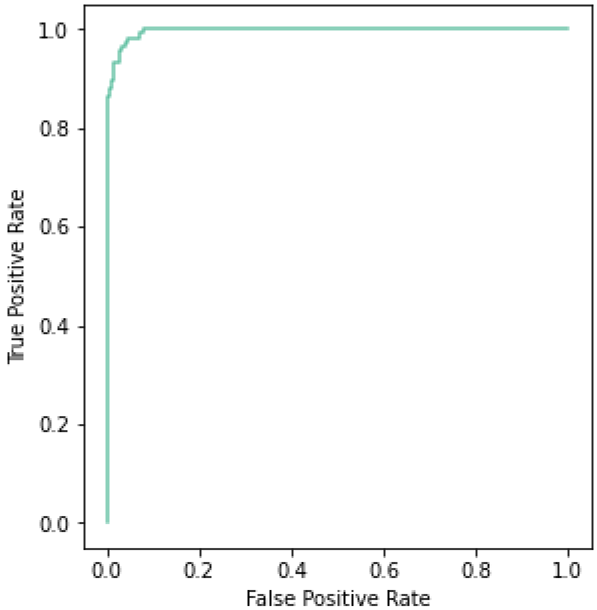
```
Empty DataFrame
Columns: [Income, Family, CCAvg, Education, Mortgage, Securities Account, CD Account, Onli
ne]
Index: []
[0.25845661 0.10946993 0.37837335 0.09926493 0.08463702 0.01486591
 0.02561827 0.02931398]
```

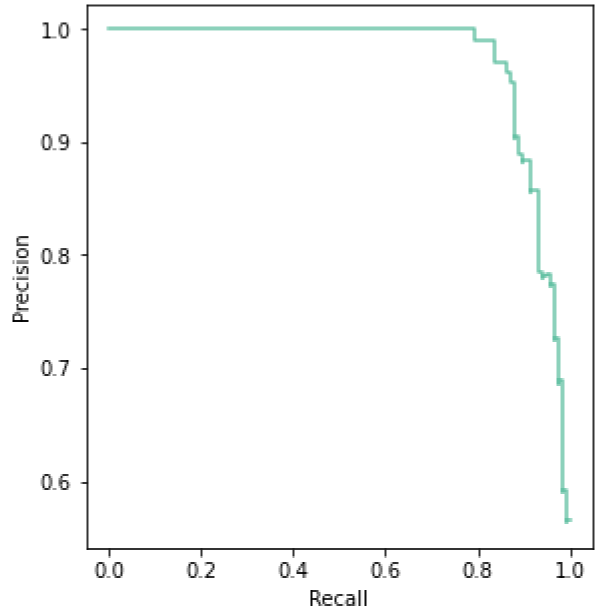


In [33]:

```
## Use a confusion Matrix to visualze performance including metrics such as true positve
y_pred= clf.predict(xtest)
confmat= confusion_matrix(ytest, y_pred)
sns.heatmap(confmat, annot= True, cmap="Greens")
plt.xlabel("True label")
plt.ylabel("Predicted label")

##trupos and precision
y_score= clf.decision_function(xtest)
falsepos, truepos, _ = roc_curve(y_test, y_score, pos_label=clf.classes_[1])
roc= RocCurveDisplay(fpr= falsepos, tpr= truepos)

prec, rec, _ = precision_recall_curve(y_test, y_score, pos_label=clf.classes_[1])
prec_rec= PrecisionRecallDisplay(precision= prec, recall= rec)

##plot both in same chart
figure, (p1, p2)= plt.subplots(1, 2, figsize= (10,5))

roc.plot(p1)
prec_rec.plot(p2)
plt.show()
```







5. Discussion/Conclusion

In this project, we analyzed the personal bank loan data using the adaboost model. Through exploratory data analysis, we identified no missing values in the dataset. We determined that all features should be postive values with some features that are binary class while some are continuous numeric. Some variables we removed because the correlation is very low with earning of a personal bank loan (such as age and experience) while others such as zip code and ID do not make since to include since they are more like identifiers or nominal (categorical) data. The adaboost model is a simple yet powerful machine learning tool for making predictions using the training and testing datasets and I was able to determine the most importance features (CCavg and income) that are associated with an individual earning a personal loan. It would be interesting to compare the accuracy of predictions with other ensemble methods such as gradient boost and randomforest. Support vector machines may also produce high prediction accuracy in less learning time although for dataset with as relatively low features and moderate-sized observations, a logitregression or Naivebayes model may do just as well to make accurate predictions and classifications.