Project #2

a)

Aim of the Project: Learn Classification Models

b) Brief description about Software or Tool

We have implemented the 3 models in Python 2.

We have used the following packages in our program -

Scikit-learn	To create models of Naive Bayes, Decision Trees and Multi-Layer Perceptron, and to get metrics of TPR/FPR and calculate AUC scores
mglearn and graphviz	To create graphs and trees images for Multi-Layer perceptron and Decision Trees
Plotly, seaborn and cufflinks	To plot ROC curves
Numpy	To perform all array operations
Matplotlib	To plot the graph of Feature Importances and Accuracy of each models
Pandas	Load data and perform One-hot encoding of Categorical Features

c)

Conduct the classification using each of those methods
We did the following steps -

- We did List-wise deletion to remove null values of '?'.
- We did One-hot encoding of categorical columns into separate features with binary values.
- Separated independent columns into X matrix, and dependent column into Y.
- Create a separate training and test set from X and Y.
- Using scikit-learn create models for each Naive Bayes models, Decision tree models and Multi-Layer Perceptron model.
- For each model compared the Y_test and Y_pred data to get accuracy score.
- For Decision trees, to prevent overfitting, we reduced the Depth of the Tree to 4.
- For Multi-Layer perceptron, to prevent overfitting, we reduced the number of Hidden Layers to 100.
- Plot in graph the accuracy score of each models used above, to determine the Best classifier.
- Calculated the FPR (False positive rate) & TPR (True Positive Rate) for each class in each model.
- Plotted the ROC curve (Receiver Operating Characteristic) and calculated AUC (Area Under Curve) values for each model and for each class.

Best classifier from each method

In terms of accuracy (percentage of correct classifications) best classifier is – Multi-Layer Perceptron.

```
In [71]: plt.figure(figsize=(10,5))
         plt.ylabel('Accuracies')
         plt.plot(X_plot,Y_plot)

Out[71]: [<matplotlib.lines.Line2D at 0x7fcff9909810>]
```

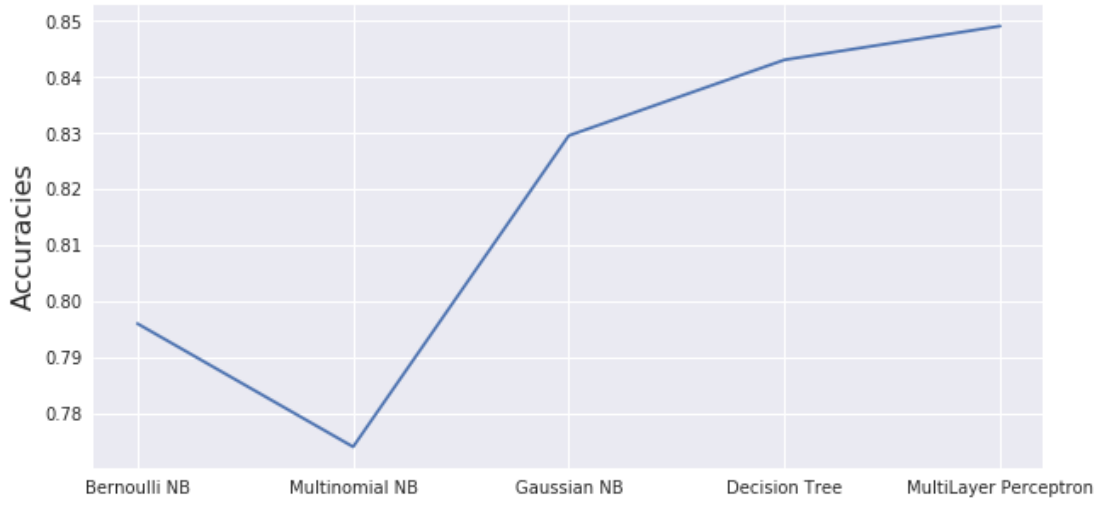


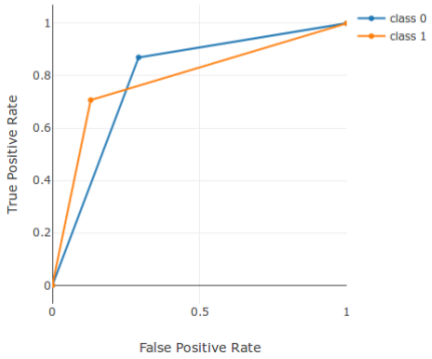
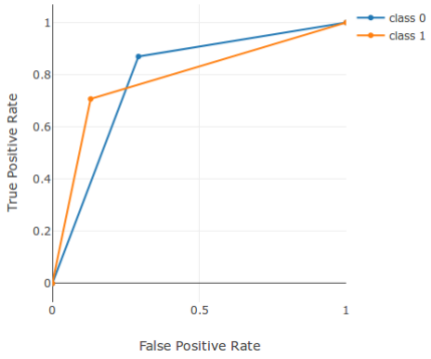
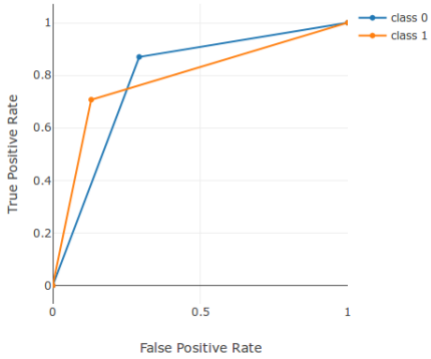
Fig: Comparing accuracy of models

ROC curves & AUC

While looking at ROC curve and AUC values, we found that best classifier is Naive Bayes.

Reason - Overall accuracy is based on one specific cutpoint, while ROC tries all of the cutpoint and plots the sensitivity and specificity. So when we compare the overall accuracy, we are comparing the accuracy based on some cutpoint. The overall accuracy varies from different cutpoint.

MultiLayer Perceptron	Decision Tree	Naive Bayes

		
AUC = 0.623836	AUC = 0.738833	AUC = 0.788415

FIND PROFILE OF PERSON WHO IS LIKELY TO MAKE OVER $50K

Naive Bayes formula -
P(A|B) = P(A) . P (B|A) / P(B)

P(X | Income >=50K) = P(X) . P(Income >= 50K | X) / P(Income >= 50K)

Where X is the profile of person.

Now, for each feature we will find the highest probability for the Profile of person who is likely to make over 50K.

<u>Feature 1 (Age)</u>

Probability of person being less than 40 age, given he earns more than 50K

P(Age < 40 | Income >= 50K) $= \frac{P(age<40).P(Income\geq 50K/Age<40)}{P(Income\geq 50K)}$

= 0.3938

Probability of person being greater than 40 age, given he earns more than 50K

P(Age >= 40 | Income >= 50K) $= \frac{P(age\geq 40).P(Income\geq 50K/Age\geq 40)}{P(Income\geq 50K)}$

= 0.6403

Therefore, person is likely to be **greater than age 40**, given he is earning more than 50K.

Feature 2 (Work class)

Similarly highest probability among Work classes is below,

P(Work class = Private | Income >= 50K) =

$$\frac{P(Workclass=Private).P(Income\geq 50K/WorkClass=Private)}{P(Income\geq 50K)}$$

= 0.6329

Therefore, person is likely to be **Work class = private**, given he is earning more than 50K.

Feature 3 (Education)

Similarly highest probability among Education is below,

P(Education = Bachelors | Income >= 50K) =

$$\frac{P(Education=Bachelors).P(Income\geq 50K/Education=Bachelors)}{P(Income\geq 50K)}$$

= 0.2832

Therefore, person is likely to be **Education = Bachelors**, given he is earning more than 50K.

Feature 4 (Years of Education)

Similarly highest probability among Years of Education (YoE) is below,

P(YoE >= 10 | Income >= 50K) =

$$\frac{P(YoE\geq 10).P(Income\geq 50K/YoE\geq 10)}{P(Income\geq 50K)}$$

= 0.7552

Therefore, person is likely to be **Year Of Education >= 10**, given he is earning more than 50K.

Feature 5 (Marital status)

Similarly highest probability among Marital Status (MS) is below,

P(MS = Married-civ-spouse | Income >= 50K) =

$$\frac{P(MS=Married-civ-spouse).P(Income\geq 50K/MS=Married-civ-spouse)}{P(Income\geq 50K)}$$

= 0.8534

Therefore, person is likely to be **Marital Status = Married-civ-spouse**, given he is earning more than 50K.

Feature 6 (Occupation)

Similarly highest probability among Occupation (O) is below,

P(O = Exec-managerial | Income >= 50K) =

$$\frac{P(O=Exec-managerial).P(Income\geq 50K/O=Exec-managerial)}{P(Income\geq 50K)}$$

= 0.25098

Therefore, person is likely to be **Occupation = Exec-managerial**, given he is earning more than 50K.

Feature 7 (Relationship)

Similarly highest probability among Relationship (R) is below,

P(R = Husband | Income >= 50K) =

$$\frac{P(R=Husband).P(Income\geq 50K/R=Husband)}{P(Income\geq 50K)}$$

= 0.7547

Therefore, person is likely to be **Relationship = Husband**, given he is earning more than 50K.

Feature 8 (Gender)

Similarly highest probability among Gender (G) is below,

P(G = Male | Income >= 50K) =

$$\frac{P(G=Male).P(Income\geq 50K/G=Male)}{P(Income\geq 50K)}$$

= 0.8496

Therefore, person is likely to be **Gender = Male**, given he is earning more than 50K.

Feature 9 (Capital Gain)

Similarly highest probability among Capital Gain (CG) is below,

P(CG = 0 | Income >= 50K) =

$$\frac{P(CG=0).P(Income \geq 50K / CG=0)}{P(Income \geq 50K)}$$

= 0.7861

Therefore, person is likely to have **Capital Gain = 0**, given he is earning more than 50K.

Feature 10 (Capital Loss)

Similarly highest probability among Capital Loss (CL) is below,

P(CL = 0 | Income >= 50K) =

$$\frac{P(CL=0).P(Income \geq 50K / CL=0)}{P(Income \geq 50K)}$$

= 0.9014

Therefore, person is likely to have **Capital Loss = 0**, given he is earning more than 50K.

Feature 11 (Hours per week)

Similarly highest probability among Hours per Week (HPW) is below,

P(40 <= HPW <= 60 | Income >= 50K) =

$$\frac{P(40 \leq HPW \leq 60).P(Income \geq 50K / 40 \leq HPW \leq 60)}{P(Income \geq 50K)}$$

= 0.8543

Therefore, person is likely to work **Hours per week is greater than 40 and less than 60**, given he is earning more than 50K.

PROFILE CONCLUSION

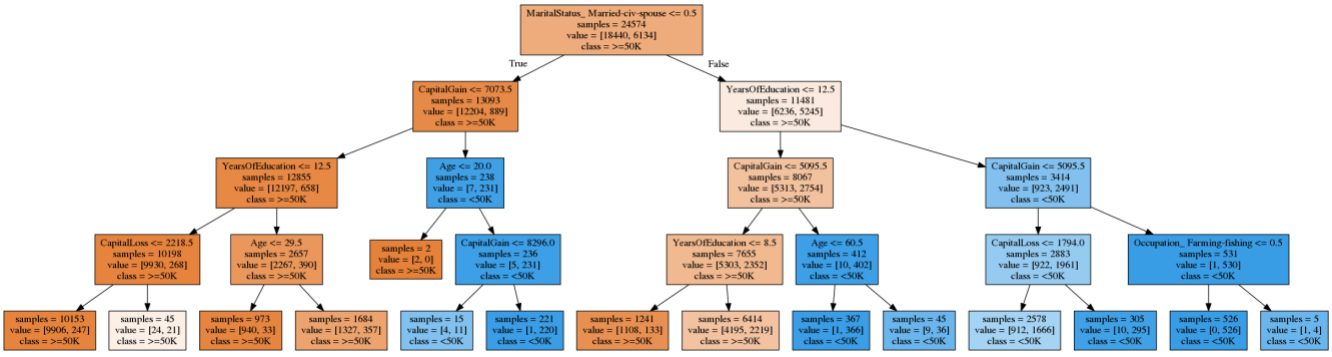
Therefore, the profile of person who is likely to make over $50k is :

- Age >= 40
- Work class = Private
- Education = Bachelors
- Years of education >= 10
- Marital status = married-civ-spouse
- Occupation = Exec-managerial
- Relationship = Husband
- Gender = Male
- Capital Gain = 0
- Capital Loss = 0
- 40 <= Hours per week <= 60

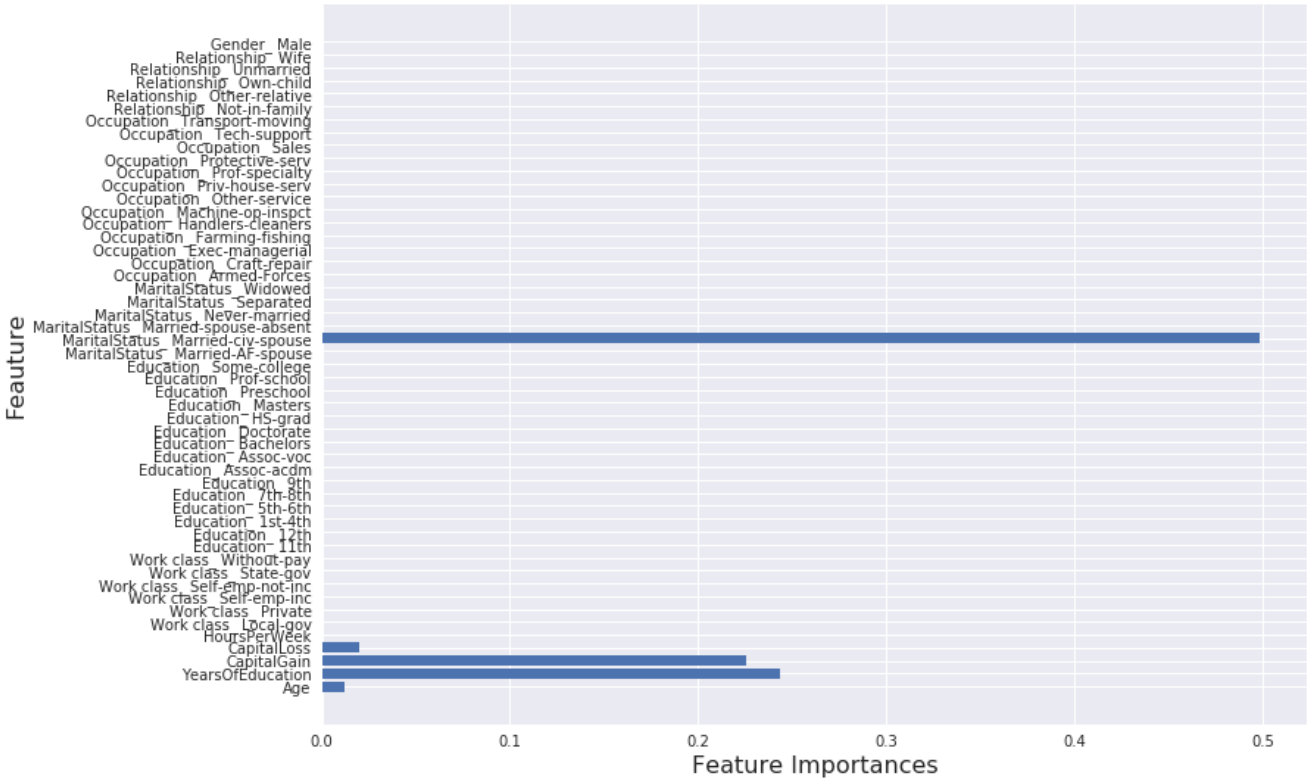
d)

Major GUI snapshot

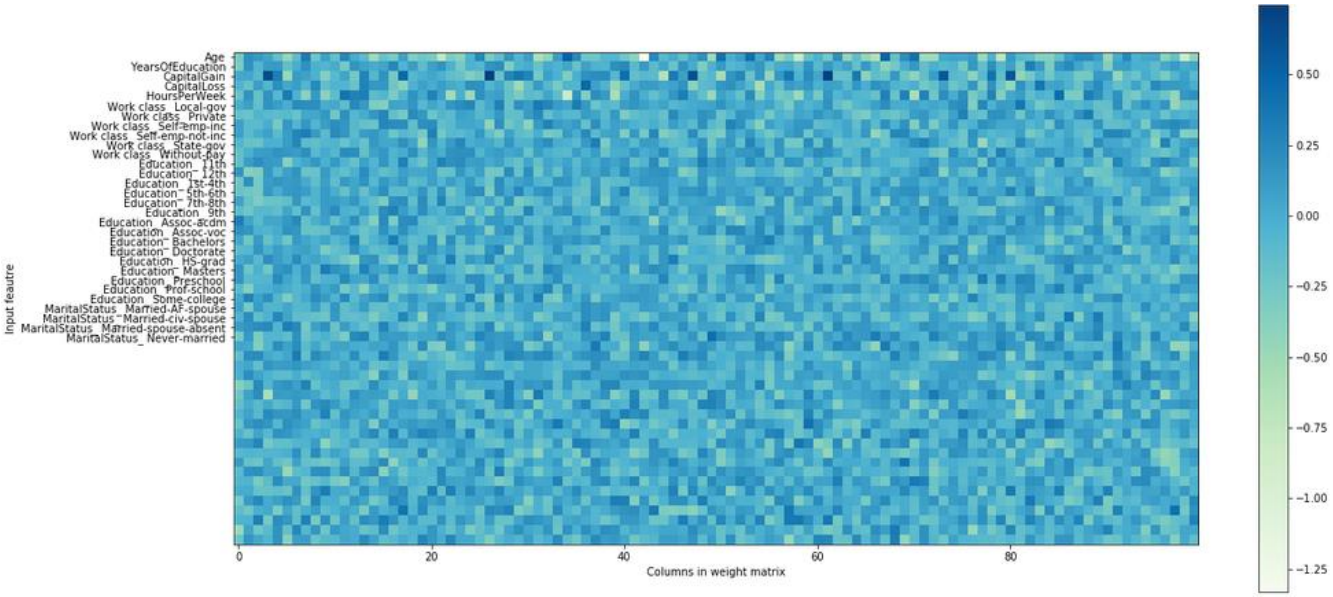
Decision tree created



Important features used for dividing the decision tree nodes, showing Marital status = ‘Married-civ-spouse’ as the main feature, and CapitalGain, CapitalLoss and Age as the main feature for divisions and contributing to Income.



Important feature weights as created in the Multi-layer perceptron, show the columns Age, Years of Education, Capital Gain, Capital Loss, and Hours Per Week as the important features contributing towards income.



Python 2 Code script

```
import pandas as pd
import numpy as np

from sklearn.cross_validation import train_test_split

from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc

from sklearn.tree import DecisionTreeClassifier

import graphviz
from sklearn.tree import export_graphviz

import matplotlib.pyplot as plt

import mglearn
import graphviz

from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

import seaborn as sns
sns.set(style="darkgrid")
from IPython.display import display
params = {'legend.fontsize': 16,
'legend.handlelength': 2,
'figure.figsize': (12,9),
'axes.titlesize': 16,
'axes.labelsize': 16
}
plt.rcParams.update(params)


import plotly
import plotly.offline as py
from plotly.offline import init_notebook_mode, download_plotlyjs
import plotly.graph_objs as go
import cufflinks as cf
init_notebook_mode(connected=True)
# plotly.tools.set_credentials_file(username='vkrishnamani', api_key='uTN0DvhXNYXtzrrmwwpG')
```

```
cf.set_config_file(offline=True, world_readable=True, theme='pearl')



raw_data = pd.read_csv("Dataset_Copy.csv")

# Create One hot features for the Categorical columns in Dataset
raw_data = pd.get_dummies(raw_data, columns = ['Work class', 'Education',
'MaritalStatus','Occupation','Relationship','Gender'], drop_first=True)


# Separate the Dependent Column
y = raw_data.Income

# Separate the Independent Columns, by dropping Dependent Column
raw_data = raw_data.drop(['Income'], axis=1)


# Final Input and Label datasets using numpy arrays
X = np.array(raw_data)
Y = np.array(y)


# Split data into Train & Test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=17)


# Arrays to store the Accuracy for each Classification Model
X_plot = np.array(["Bernoulli NB","Multinomial NB", "Gaussian NB", "Decision Tree", "MultiLayer
Perceptron"])
Y_plot = np.array([])



## Bernoulli Naive Bayes Classifier with Binarize

BernNB = BernoulliNB(binarize=0.1)
BernNB.fit(X_train,Y_train)

### Training accuracy
Y_expect = Y_train
Y_pred = BernNB.predict(X_train)
print accuracy_score(Y_expect,Y_pred)

### Testing accuracy
Y_expect = Y_test
```

```
Y_pred = BernNB.predict(X_test)
acc = accuracy_score(Y_expect,Y_pred)
print acc

Y_plot = np.append(Y_plot,acc)




## Multinomial Naive Bayes Classifier

MultiNB = MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
MultiNB.fit(X_train,Y_train)

### Training accuracy
Y_expect = Y_train
Y_pred = MultiNB.predict(X_train)

print accuracy_score(Y_expect,Y_pred)

### Testing accuracy
Y_expect = Y_test
Y_pred = MultiNB.predict(X_test)

acc = accuracy_score(Y_expect,Y_pred)
print acc

Y_plot = np.append(Y_plot,acc)




## Gaussian Naive Bayes Classifier

GausNB = GaussianNB()
GausNB.fit(X_train,Y_train)

### Training accuracy
Y_expect = Y_train
Y_pred = GausNB.predict(X_train)

print accuracy_score(Y_expect,Y_pred)
```

```
### Testing accuracy
Y_expect = Y_test
Y_pred = GausNB.predict(X_test)

acc = accuracy_score(Y_expect,Y_pred)
print acc

Y_plot = np.append(Y_plot,acc)




## Decision tree classifier with Overfitting
tree = DecisionTreeClassifier(random_state = 0)
tree.fit(X_train,Y_train)

### Training accuracy
print tree.score(X_train,Y_train)

### Test accuracy
print tree.score(X_test,Y_test)


features = list(raw_data.columns)
export_graphviz(tree, out_file='census_overfit.dot',class_names=['>=50K','<50K'],
feature_names=features,
impurity=False, filled=True)




## Decision tree classifier without Overfitting (i.e. less Depth of tree)
tree = DecisionTreeClassifier(max_depth = 4, random_state = 0)
tree.fit(X_train,Y_train)

### Training accuracy
print tree.score(X_train,Y_train)

### Test accuracy
acc = tree.score(X_test,Y_test)
print acc

Y_plot = np.append(Y_plot,acc)
```

```
features = list(raw_data.columns)
export_graphviz(tree, out_file='census.dot',class_names=['>=50K','<50K'], feature_names=features,
impurity=False, filled=True)



### Find important features used in the decision tree
print "Feature importances {0}".format(tree.feature_importances_)

n_features = len(features)

plt.barh(range(n_features), tree.feature_importances_,align='center')
plt.yticks(np.arange(n_features), features)
plt.xlabel("Feature Importances")
plt.ylabel("Feauture")

fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 10
fig_size[1] = 15
plt.rcParams["figure.figsize"] = fig_size

plt.show()







# Multi-Layer Perceptron

mlp = MLPClassifier(random_state = 42)
mlp.fit(X_train, Y_train)


### Training accuracy
print mlp.score(X_train,Y_train)

### Testing accuracy
print mlp.score(X_test,Y_test)




### Feature scaling and lesser hidden layers to prevent Overfitting
```

```
scaler = StandardScaler()

X_train_scaled = scaler.fit(X_train).transform(X_train)
X_test_scaled = scaler.fit(X_test).transform(X_test)


mlp = MLPClassifier(max_iter = 1000, random_state = 42, hidden_layer_sizes=(100,))
mlp.fit(X_train_scaled, Y_train)

### Training accuracy
print mlp.score(X_train_scaled,Y_train)

### Testing accuracy
acc = mlp.score(X_test_scaled,Y_test)
print acc

Y_plot = np.append(Y_plot,acc)

## Check the weights of the perceptron
features = list(raw_data.columns)

plt.figure(figsize=(20,10))
plt.imshow(mlp.coefs_[0], interpolation='None', cmap = 'GnBu')
plt.yticks(range(30), features)
plt.xlabel("Columns in weight matrix")
plt.ylabel("Input feautre")
plt.colorbar()




# Comparison of accuracies

plt.figure(figsize=(10,5))
plt.ylabel('Accuracies')
plt.plot(X_plot,Y_plot)
```

```
# ROC curves - Multilayer Perceptron
Y_pred = mlp.predict(X_test)
y_onehot_test = pd.get_dummies(Y_test, columns = ['Income'])
y_onehot_pred = pd.get_dummies(Y_pred, columns = ['Income'])
y_onehot_test = np.array(y_onehot_test)
y_onehot_pred = np.array(y_onehot_pred)
stats_list = []
for i in range(2):
# Calculate ROC Curve
fpr, tpr, _ = roc_curve(y_onehot_test[:, i], y_onehot_pred[:, i])
# Calculate area under the curve
roc_auc = [auc(fpr, tpr)] * len(fpr)
classes = [i] * len(fpr)
stats_list += zip(fpr, tpr, roc_auc, classes)
stats = pd.DataFrame(stats_list, columns=['fpr', 'tpr', 'auc', 'class'])

data = []
for key, grp in stats.groupby(['class']):
trace = go.Scatter(x = grp.fpr,
y = grp.tpr,
name = 'class %d' % key)
data.append(trace)
# Edit the layout
layout = dict(title = 'Receiver Operating Characterstic',
xaxis = dict(title = 'False Positive Rate',
range = [0, 1]),
yaxis = dict(title = 'True Positive Rate'),
width=500,
height=500,
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-line')

stats.groupby('class').mean().auc









# ROC curves - Decision Trees
Y_pred = tree.predict(X_test)
y_onehot_test = pd.get_dummies(Y_test, columns = ['Income'])
```

```
y_onehot_pred = pd.get_dummies(Y_pred, columns = ['Income'])
y_onehot_test = np.array(y_onehot_test)
y_onehot_pred = np.array(y_onehot_pred)
stats_list = []
for i in range(2):
# Calculate ROC Curve
fpr, tpr, _ = roc_curve(y_onehot_test[:, i], y_onehot_pred[:, i])
# Calculate area under the curve
roc_auc = [auc(fpr, tpr)] * len(fpr)
classes = [i] * len(fpr)
stats_list += zip(fpr, tpr, roc_auc, classes)
stats = pd.DataFrame(stats_list, columns=['fpr', 'tpr', 'auc', 'class'])

data = []
for key, grp in stats.groupby(['class']):
trace = go.Scatter(x = grp.fpr,
y = grp.tpr,
name = 'class %d' % key)
data.append(trace)
# Edit the layout
layout = dict(title = 'Receiver Operating Characterstic',
xaxis = dict(title = 'False Positive Rate',
range = [0, 1]),
yaxis = dict(title = 'True Positive Rate'),
width=500,
height=500,
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-line')

stats.groupby('class').mean().auc




# ROC curves - Naive Bayes

Y_pred = GausNB.predict(X_test)
y_onehot_test = pd.get_dummies(Y_test, columns = ['Income'])
y_onehot_pred = pd.get_dummies(Y_pred, columns = ['Income'])
y_onehot_test = np.array(y_onehot_test)
y_onehot_pred = np.array(y_onehot_pred)
```

```
stats_list = []
for i in range(2):
# Calculate ROC Curve
fpr, tpr, _ = roc_curve(y_onehot_test[:, i], y_onehot_pred[:, i])
# Calculate area under the curve
roc_auc = [auc(fpr, tpr)] * len(fpr)
classes = [i] * len(fpr)
stats_list += zip(fpr, tpr, roc_auc, classes)
stats = pd.DataFrame(stats_list, columns=['fpr', 'tpr', 'auc', 'class'])

data = []
for key, grp in stats.groupby(['class']):
trace = go.Scatter(x = grp.fpr,
y = grp.tpr,
name = 'class %d' % key)
data.append(trace)
# Edit the layout
layout = dict(title = 'Receiver Operating Characterstic',
xaxis = dict(title = 'False Positive Rate',
range = [0, 1]),
yaxis = dict(title = 'True Positive Rate'),
width=500,
height=500,
)

fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-line')

stats.groupby('class').mean().auc
```