In [1]:

```
import numpy as np
import pandas as pd
from time import time
from IPython.display import display
import matplotlib.pyplot as plt
import visuals as vs


%matplotlib inline


data = pd.read_csv("census.csv")

display(data.head(n=1))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	se
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Mal

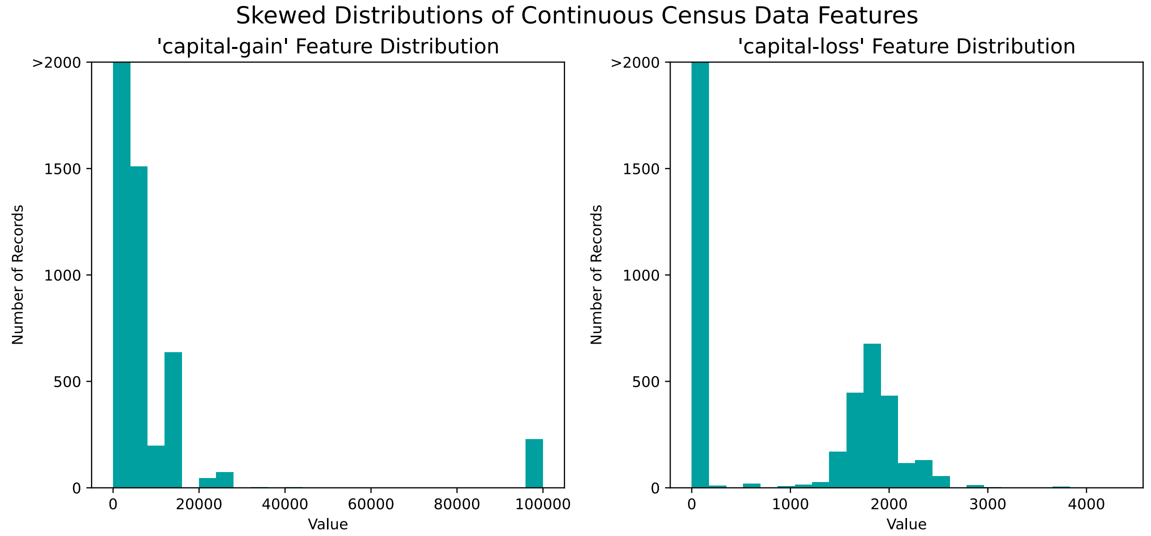
In [2]:

```
n_records = data.shape[0]

n_greater_50k = data[data["income"] == ">50K"].shape[0]

n_at_most_50k = data[data["income"] == "<=50K"].shape[0]

greater_percent = n_greater_50k/n_records*100.0

print("Total number of records: {}".format(n_records))
print("Individuals making more than $50,000: {}".format(n_greater_50k))
print("Individuals making at most $50,000: {}".format(n_at_most_50k))
print("Percentage of individuals making more than $50,000: {:.2f}%".format(greater_percent))
print("Feature valuse for each column:\n",data.columns)
```

```
Total number of records: 45222
Individuals making more than $50,000: 11208
Individuals making at most $50,000: 34014
Percentage of individuals making more than $50,000: 24.78%
Feature valuse for each column:
 Index(['age', 'workclass', 'education_level', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'income'],
      dtype='object')
```

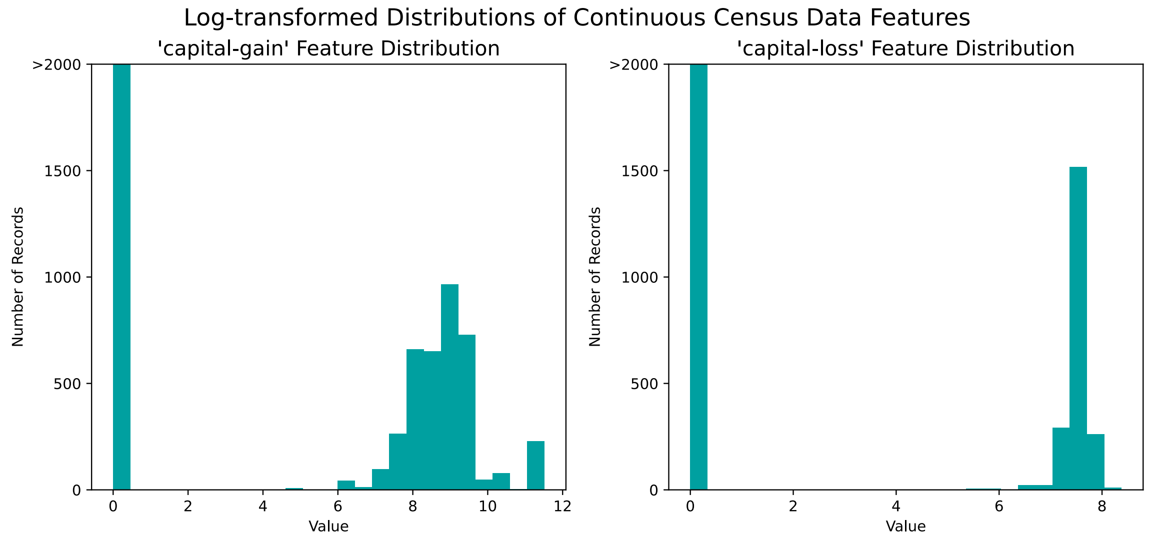
In [3]:

```
income_raw = data['income']
features_raw = data.drop('income', axis = 1)

vs.distribution(data)
```



In [4]:

```
skewed = ['capital-gain', 'capital-loss']
features_log_transformed = pd.DataFrame(data = features_raw)
features_log_transformed[skewed] = features_raw[skewed].apply(lambda x: np.log(x + 1))

vs.distribution(features_log_transformed, transformed = True)
```



In [5]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed[numeric
al])

display(features_log_minmax_transform.head(n = 5))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race
0	0.301370	State-gov	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family	White
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husband	White
2	0.287671	Private	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family	White
3	0.493151	Private	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	0.150685	Private	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife	Black

In [6]:

```
data2 =pd.get_dummies(features_log_minmax_transform)

income = income_raw.replace(["<=50K",">50K"],[0,1])

encoded = list(data2.columns)
print("{} total features ".format(len(encoded)))

print (encoded)
```

```
103 total features
['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'workcl
ass_ Federal-gov', 'workclass_ Local-gov', 'workclass_ Private', 'workclass_ Self-
emp-inc', 'workclass_ Self-emp-not-inc', 'workclass_ State-gov', 'workclass_ Witho
ut-pay', 'education_level_ 10th', 'education_level_ 11th', 'education_level_ 12t
h', 'education_level_ 1st-4th', 'education_level_ 5th-6th', 'education_level_ 7th-
8th', 'education_level_ 9th', 'education_level_ Assoc-acdm', 'education_level_ Ass
oc-voc', 'education_level_ Bachelors', 'education_level_ Doctorate', 'education_le
vel_ HS-grad', 'education_level_ Masters', 'education_level_ Preschool', 'educatio
n_level_ Prof-school', 'education_level_ Some-college', 'marital-status_ Divorce
d', 'marital-status_ Married-AF-spouse', 'marital-status_ Married-civ-spouse', 'ma
rital-status_ Married-spouse-absent', 'marital-status_ Never-married', 'marital-st
atus_ Separated', 'marital-status_ Widowed', 'occupation_ Adm-clerical', 'occupati
on_ Armed-Forces', 'occupation_ Craft-repair', 'occupation_ Exec-managerial', 'occ
upation_ Farming-fishing', 'occupation_ Handlers-cleaners', 'occupation_ Machine-o
p-inspct', 'occupation_ Other-service', 'occupation_ Priv-house-serv', 'occupation
_ Prof-specialty', 'occupation_ Protective-serv', 'occupation_ Sales', 'occupation
_ Tech-support', 'occupation_ Transport-moving', 'relationship_ Husband', 'relatio
nship_ Not-in-family', 'relationship_ Other-relative', 'relationship_ Own-child',
'relationship_ Unmarried', 'relationship_ Wife', 'race_ Amer-Indian-Eskimo', 'race
_ Asian-Pac-Islander', 'race_ Black', 'race_ Other', 'race_ White', 'sex_ Female',
'sex_ Male', 'native-country_ Cambodia', 'native-country_ Canada', 'native-country
_ China', 'native-country_ Columbia', 'native-country_ Cuba', 'native-country_ Dom
inican-Republic', 'native-country_ Ecuador', 'native-country_ El-Salvador', 'nativ
e-country_ England', 'native-country_ France', 'native-country_ Germany', 'native-
country_ Greece', 'native-country_ Guatemala', 'native-country_ Haiti', 'native-co
untry_ Holand-Netherlands', 'native-country_ Honduras', 'native-country_ Hong', 'n
ative-country_ Hungary', 'native-country_ India', 'native-country_ Iran', 'native-
country_ Ireland', 'native-country_ Italy', 'native-country_ Jamaica', 'native-cou
ntry_ Japan', 'native-country_ Laos', 'native-country_ Mexico', 'native-country_ N
icaragua', 'native-country_ Outlying-US(Guam-USVI-etc)', 'native-country_ Peru',
'native-country_ Philippines', 'native-country_ Poland', 'native-country_ Portuga
l', 'native-country_ Puerto-Rico', 'native-country_ Scotland', 'native-country_ So
uth', 'native-country_ Taiwan', 'native-country_ Thailand', 'native-country_ Trina
dad&Tobago', 'native-country_ United-States', 'native-country_ Vietnam', 'native-c
ountry_ Yugoslavia']
```

In [7]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(data2,income,test_size = 0.2, random_state = 0)

print("Training set has {} samples".format(X_train.shape[0]))
print("Testing set has {} samples".format(X_test.shape[0]))
```

```
Training set has 36177 samples
Testing set has 9045 samples
```

In [8]:

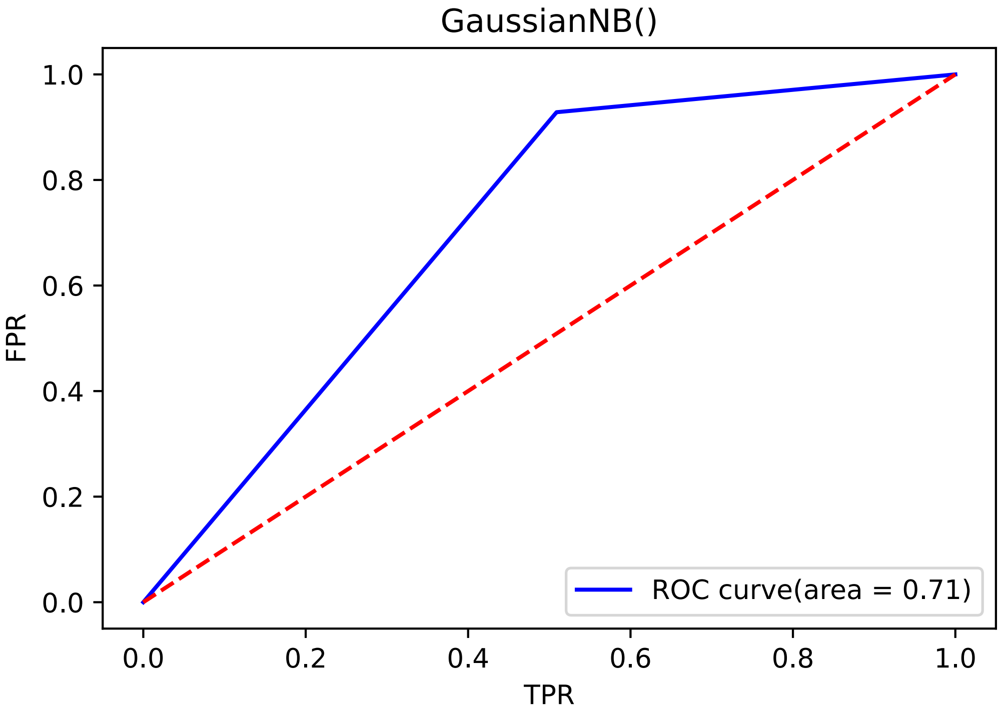
```
print('exercise2:')
TP = np.sum(income)
FP = income.count() - TP
TN = 0
FN = 0
accuracy = float(TP)/(TP+FP)
recall = float(TP)/(TP+FN)
precision = accuracy

beta=0.5
fscore = (1 + beta ** 2)*(precision * recall)/(beta ** 2 *precision + recall)
TPR = float(TP)/(TP + FN)
FPR = float(FP)/(TN + FP)
print("Accuracy score: {:.4f}, F-score: {:.4f}".format(accuracy, fscore))
```

```
exercise2:
Accuracy score: 0.2478, F-score: 0.2917
```

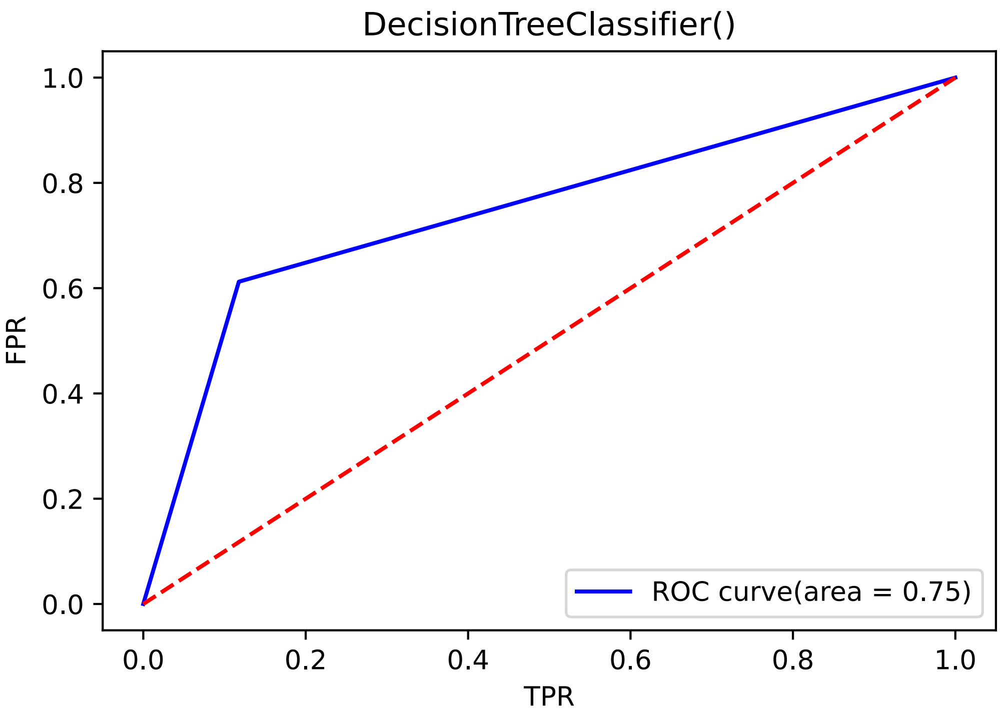
In [9]:

```
from sklearn.metrics import fbeta_score,accuracy_score,confusion_matrix,roc_curve,auc
def train(learner, X_train, Y_train, X_test, Y_test):

    results = {}

    #fit/train
    # start = time()
    learner.fit(X_train,Y_train)
    # end = time()
    # results['train_time'] = end - start

    #predict
    # start = time()
    Y_pred_test = learner.predict(X_test)
    Y_pred_train = learner.predict(X_train)
    # end = time()
    # results['pred_time'] = end-start

    # results['acc_train_score'] = accuracy_score(Y_train,Y_pred_train)

    # results['acc_test_score'] = accuracy_score(Y_test,Y_pred_test)

    # results['train_f_score'] = fbeta_score(Y_train,Y_pred_train,beta=0.5)

    # results['test_f_score'] = fbeta_score(Y_test,Y_pred_test,beta=0.5)

    #draw ROC
    # print(learner.name)
    fpr, tpr, threshold = roc_curve(Y_test,Y_pred_test)
    print('The function {}\'s fpr is {} and ptr is{}'.format(learner,fpr,tpr))
    roc_auc = auc(fpr,tpr)
    plt.figure()
    plt.title(learner)
    plt.plot(fpr,tpr,'b',label = 'ROC curve(area = %.2f)'%roc_auc)
    plt.legend(loc = "lower right")
    plt.plot([0,1],[0,1],'r--',label = 'random')
    plt.xlabel('TPR')
    plt.ylabel('FPR')
    plt.show()

from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
learner = [GaussianNB(),DecisionTreeClassifier(),BaggingClassifier(),AdaBoostClassifier(),RandomForestClassifier(),KNeighborsClassifier(),SVC(),LogisticRegression()]
for i in learner:
    train(i,X_train,Y_train,X_test,Y_test)
```

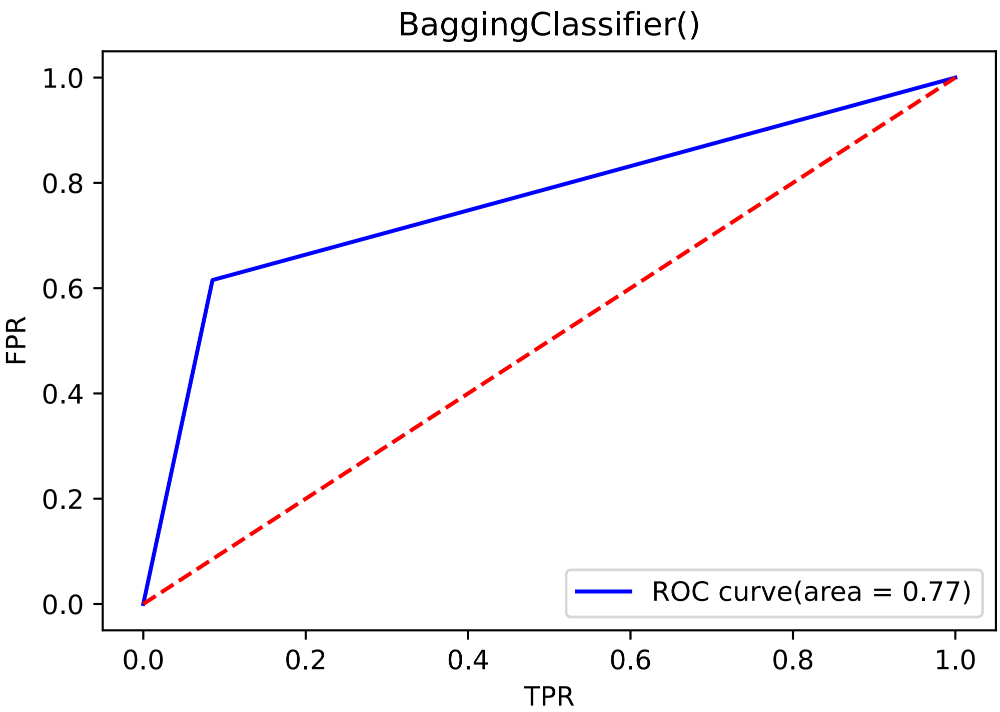
The function GaussianNB()'s fpr is [0.   0.50891813 1.   ] and ptr is [0.   0.92834467 1.   ]



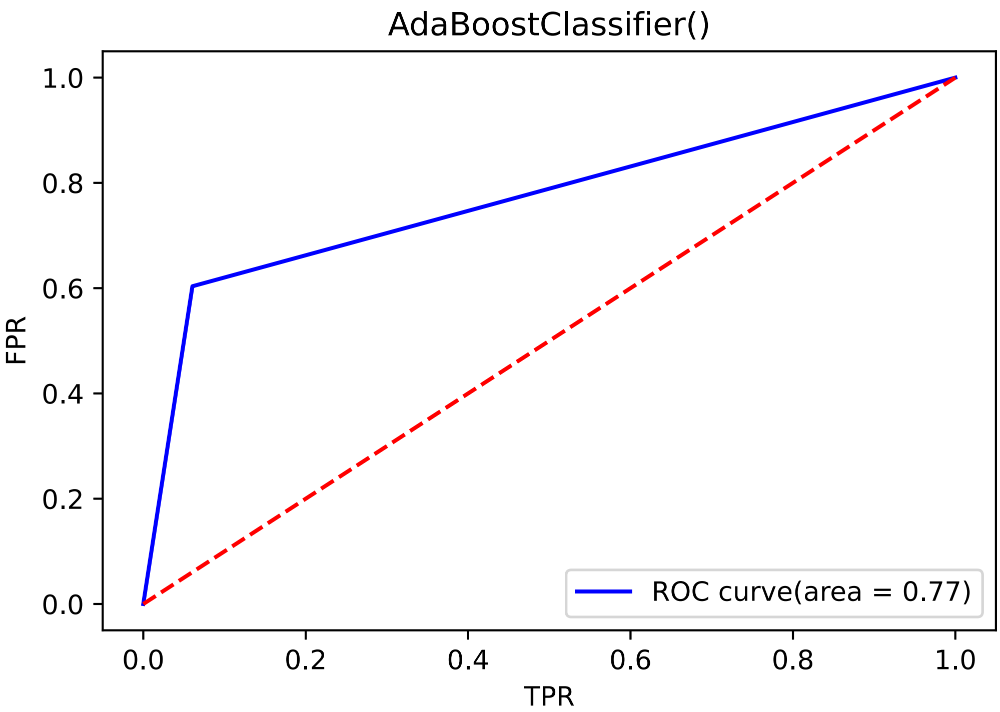
The function DecisionTreeClassifier()'s fpr is [0.   0.11769006 1.   ] and ptr is[0.   0.6122449 1.   ]



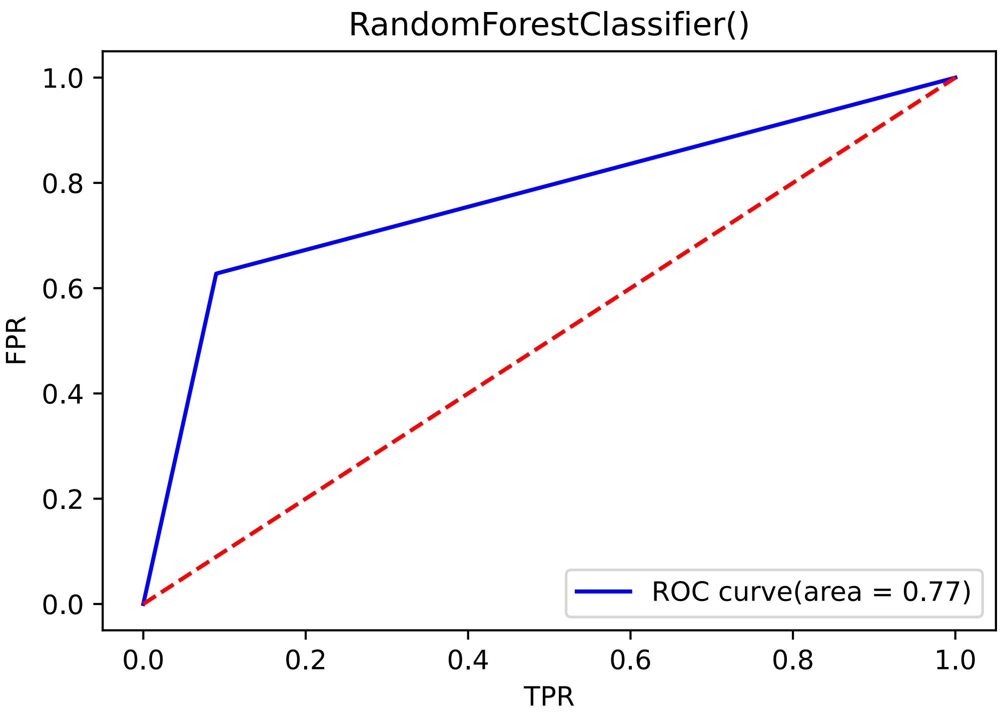

```
The function BaggingClassifier()'s fpr is [0.         0.08523392 1.        ] and p
tr is[0.        0.6154195 1.       ]
```



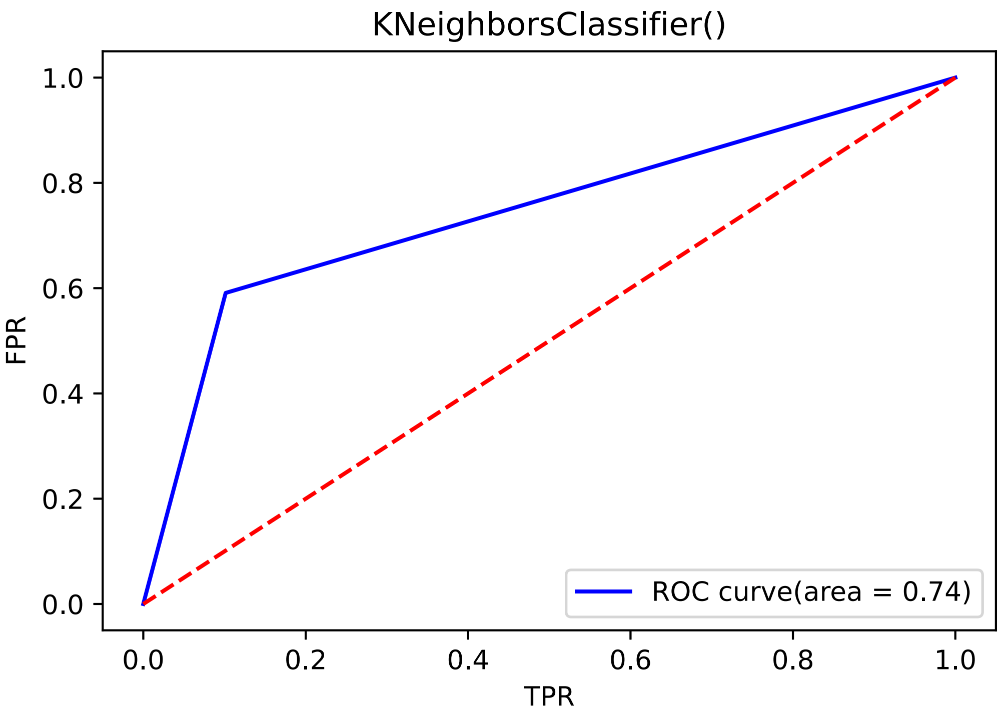

```
The function AdaBoostClassifier()'s fpr is [0.         0.06052632 1.        ] and
ptr is[0.         0.60362812 1.        ]
```



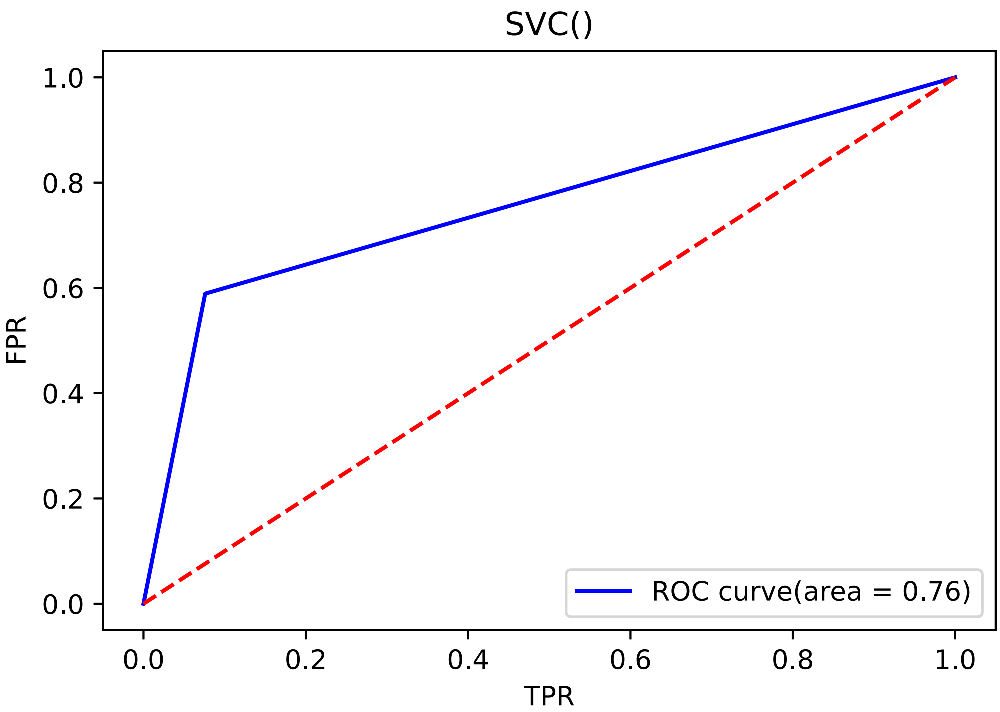

```
The function RandomForestClassifier()'s fpr is [0.         0.08976608 1.        ]
and ptr is[0.         0.6276644 1.        ]
```



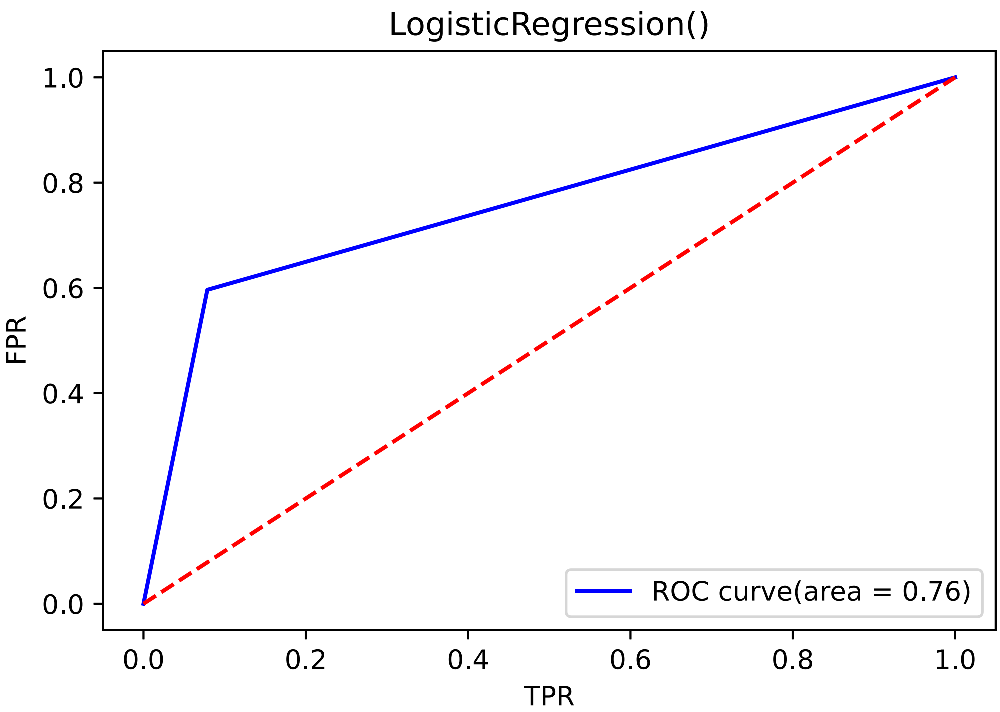

```
The function KNeighborsClassifier()'s fpr is [0.         0.10146199 1.        ] an
d ptr is[0.         0.59092971 1.        ]
```




```
The function SVC()'s fpr is [0.         0.07602339 1.        ] and ptr is[0.
0.58911565 1.        ]
```




```
The function LogisticRegression()'s fpr is [0.         0.07865497 1.        ] and
ptr is[0.         0.59637188 1.        ]
```



In [16]:

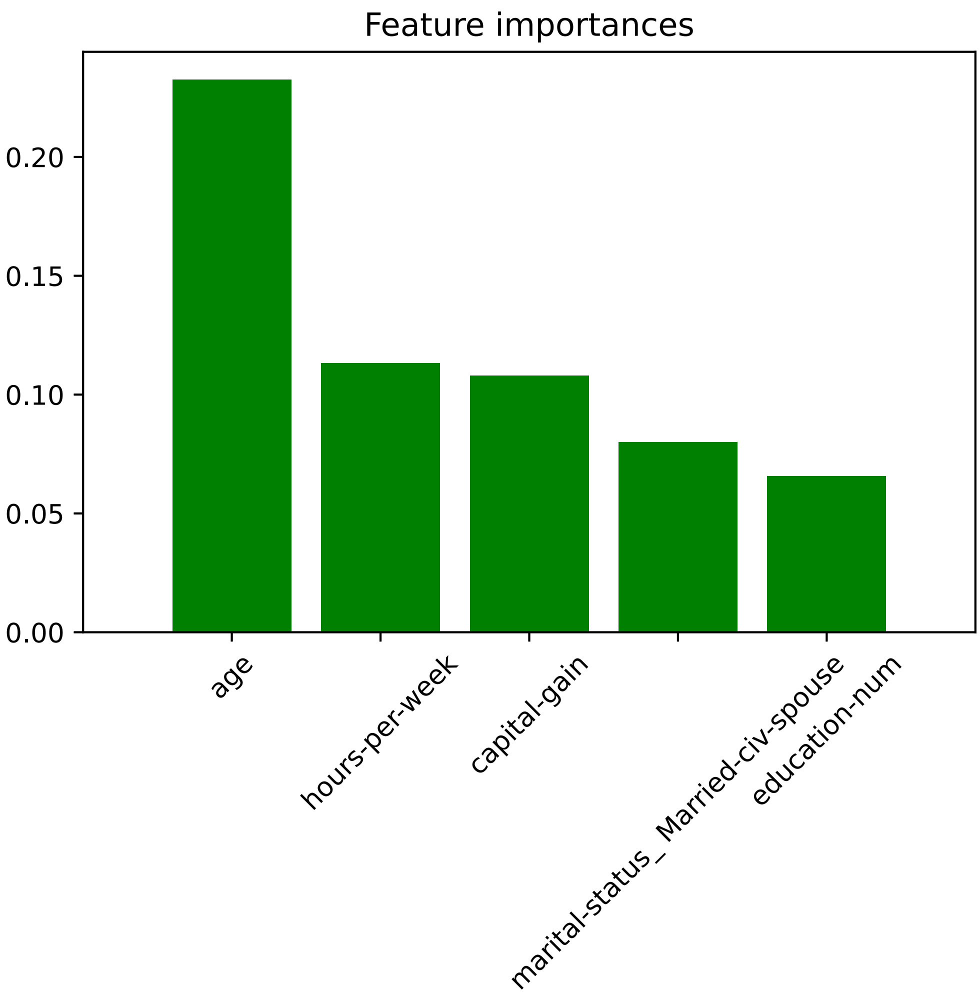
```
RFC = RandomForestClassifier()
RFC.fit(X_train,Y_train)
```

Out[16]:

```
RandomForestClassifier()
```

In [24]:

```
Y_pred_test = RFC.predict(X_test)
Y_pred_train = RFC.predict(X_train)
score = RFC.score(X_test,Y_test)
features = list(X_test.columns)
importances = RFC.feature_importances_
indices = np.argsort(importances)[::-1]
# print top 5 important features
num_features = 5
# num_features = len(importances)

plt.figure()
plt.title("Feature importances")
plt.bar(range(num_features), importances[indices[0:num_features]], color="g", align="center")
plt.xticks(range(num_features), [features[i] for i in indices], rotation='45')
plt.xlim([-1, num_features])
plt.show()
```



Exercise 3:

通过上面方法的 ROC 图我们观察到 Ensemble Methods 有最好的分类结果。

(1) 我们选取了 Decision Trees、AdaBoost、RandomForest 方法并画出了各个方法排名前五的特征，如上图所示。

(2) 土地覆被测绘是地球观测卫星传感器的主要应用之一，它利用遥感和地理空间数据来识别位于目标区域表面的材料和物体。通常，目标材料的类别包括道路，建筑物，河流，湖泊和植被。基于人工神经网络的一些不同的集成学习方法，，带 Boosting 的决策树，随机森林和多分类器系统的自动设计，被提出来有效地识别土地覆盖物。

(3) 随机森林可以处理很高维度的数据，并且不用降维，无需做特征选择，且可以判断特征的重要程度，训练速度较快

(4) 随机森林算法在某些噪音较大的分类或回归上会过拟合，对于不同取值属性的数据，取值划分较多的属性会对随机森林产生更大的影响。

(5) 在这个数据集中，数据的维度比较高，所以使用随机森林算法可以比较快速的完成拟合，在随机森林算法处理完可以从中得出重要特征，更利于分析这个问题得出收入高低的主要影响因素。