IEE 598: DATA SCIENCE FOR SYSTEM INFORMATICS:

ASSIGNMENT 3

HETUL VARAIYA

1211306106

Topics covered in this assignment:

- Kernel Ridge Regression
- Linear Regression
- Random Forest Regression
- Training and Testing Split
- SVM
- Regularized SVM
- Logistic Regression
- precision/recall
- f1-score
- confusion matrix
- ROC Curve
- Area Under the curve
- Cross Validation using GridSearchCV

Problem 1: Kernel regression

Kernel method can be a powerful tool for spatial dataset or images. For this example, we would like to explore the use of Kernel Ridge Regression for estimating a 2D surface.

The true function is given in the Jupyter notebook as $z = x\exp(-x^2 - y^2)$. The goal is to use (x, y) to predict z.

1. Please start with the linear regression method and report the training and testing accuracy. Please visualize the final prediction in an image.

2. Please use the Kernel Ridge Regression and report the training and testing accuracy. Please visualize the final prediction in an image. Please use cross-validation to select the best tuning parameters.

3. (Bonus 10 points) Please use the Random Forest Regressor and report the training and testing accuracy. Please visualize the final prediction in an image. Please use cross-validation to select the best tuning parameters. From the three models that you have tried, which one works the best and why?

- Creating the function:

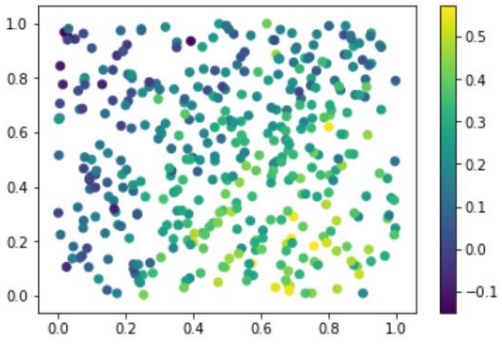
Kernel Ridge Regression

Let's first simulate a 2D nonlinear functions as follow. The goal of this task is to predict the true function by observing few samples from it.

In [21]:
```
import numpy as np

def create2Dsurface(x,y):
    z = x*np.exp(-x**2-y**2)
    return z
```

- Creating the Training Dataset and visualizing it:

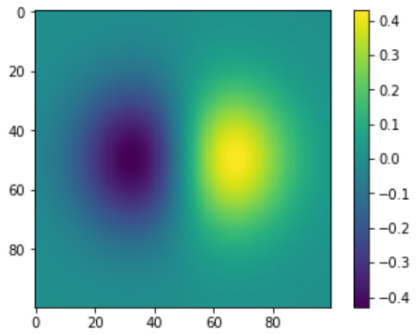
In [32]:
```
import matplotlib.pyplot as plt

n = 400
x = np.random.rand(n)
y = np.random.rand(n)
Ztrue = create2Dsurface(x,y)
y_train = Ztrue+np.random.randn(n)*0.1
X_train = np.concatenate((x.flatten()[:,np.newaxis],y.flatten()[:,np.newaxis]),axis=1)
#print(y_train.shape)
#print(X_train.shape)
plt.scatter(X_train[:,0],X_train[:,1],c=y_train)
plt.colorbar()
plt.show()
```



- Create Testing Data:

We will create our Testing dataset based on the 2D meshgrid and visulaize the image of this 2D surface

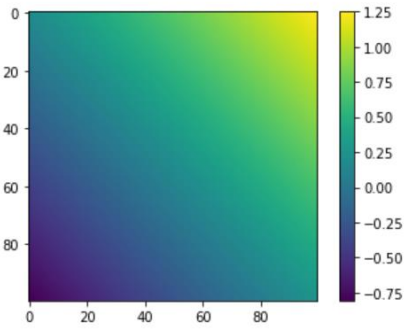
In [111]:
```
ntest = 100
ti = np.linspace(-2.0, 2.0, ntest)
x1test, x2test = np.meshgrid(ti, ti)

X_test = np.concatenate((x1test.flatten()[:,np.newaxis],x2test.flatten()[:,np.newaxis]),axis=1)
y_test = create2Dsurface(x1test,x2test).flatten()
plt.imshow(y_test.reshape((ntest,ntest)))
plt.colorbar()
plt.show()
```



- Linear Regression on the Dataset with the Visualization and Mean Square Error:

In [132]:
```
import sklearn.linear_model as skl_lm
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
regr = skl_lm.LinearRegression()
fm = regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
print('Mean Squared Error is ',mean_squared_error(y_test,y_pred))
#x1  = X_test.as_matrix()
print(X_test.shape)
plt.imshow(y_pred.reshape((ntest,ntest)))
plt.colorbar()
plt.show()
```

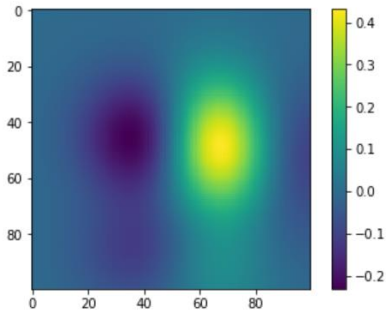
```
Mean Squared Error is  0.20458135840862057
(10000, 2)
```



- Kernel Ridge Regression - Fitting the best parameters, visualizing the result, Mean Square Error:

In [148]:
```
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import GridSearchCV
alpha1 = [0.01,0.1,1,10]
gamma1= [0.1,1,10]
param_grid = { 'alpha': alpha1, 'gamma' : gamma1}
clf3 = KernelRidge(kernel='rbf')
kr = GridSearchCV(clf3,param_grid, return_train_score = True, cv=5)
kr.fit(X_train, y_train)
print(kr.best_params_)
y_pred = kr.predict(X_test)
plt.imshow(y_pred.reshape((ntest,ntest)))
plt.colorbar()
plt.show()
print('Mean Squared Error is ',mean_squared_error(y_test,y_pred))
```

```
{'alpha': 0.1, 'gamma': 1}
```

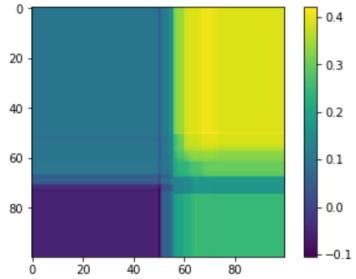



```
Mean Squared Error is  0.004490185149702359
```

- Random Forest Regression:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_moons, make_circles, make_classification
import numpy as np
n_samples=300
X, y = make_classification(n_samples=n_samples,n_features=2, n_redundant=0, n_informative=2,
                           random_state=0, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)
from sklearn.model_selection import GridSearchCV
datasets = [make_moons(n_samples=n_samples,noise=0.3, random_state=0),
            make_circles(n_samples=n_samples,noise=0.2, factor=0.5, random_state=1),linearly_separable]
depthsi = [1,2,4,8,16,32,64]
ntrees = [1,2,4,8,16,32,64]
parameters1 = {'max_depth':depthsi, 'n_estimators':ntrees}
for X, y in datasets:
    # preprocess dataset, split into training and test part
    rfc1 = RandomForestRegressor(max_depth=depthsi,n_estimators = ntrees)
    clf1 = GridSearchCV(rfc1, parameters1,return_train_score=True)
    clf1.fit(X_train, y_train)
    print('best parameters:',clf1.best_params_)
y_pred1= clf1.predict(X_test)
plt.imshow(y_pred1.reshape((ntest,ntest)))
plt.colorbar()
plt.show()
print('Mean Squared Error is ',mean_squared_error(y_test,y_pred1))
```

```
best parameters: {'max_depth': 4, 'n_estimators': 32}
best parameters: {'max_depth': 4, 'n_estimators': 16}
best parameters: {'max_depth': 4, 'n_estimators': 64}
```




```
Mean Squared Error is  0.05165406531838417
```

Model Name	Parameter Range	MSE
Linear Regression	None	0.2045
Random Forest	Depth: {1,2,4,8,16,32,64} No of trees: {1,2,4,8,16,32,64}	0.05165
Kernel Ridge Regression	Alpha: {0.01,0.1,1,10} Gamma: {0.1,1,10}	0.0045

- From the above table is seen that Kernel Ridge regression produces the best model when the alpha value is 0.1 and the gamma value is 1 as it has the least mean square error.
- KRR uses squared error loss combined with l2 regularization.

4. (Bonus 20 points) For Ridge regression problem, the primal problem is given as:

$$\min_w l(w) = \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|_2^2$$

The dual of the problem is given as:

$$\max_\alpha -\lambda^2\|\alpha\|^2 + 2\lambda\alpha^T y - \lambda\alpha^T XX^T\alpha$$

Please solve the analytical solution and gradient descent algorithm for the dual problem and compare its complexity against the primal problem in the following table. When would you prefer to solve the problem in the primal or dual?

	Primal		Dual	
	Time	Space	Time	Space
Gradient Descent	$O(np)$	$O(np)$		
Analytical Solution	$O(np^2 + p^3)$	$O(np + p^2)$		

(4) Ridge regression:

Primal problem

$$\min_{w} J(w) = \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|_2^2$$

Dual problem:

$$\max_{\alpha} -\lambda^2\|\alpha\|^2 + 2\lambda\alpha^T y - \lambda\alpha^T X X^T \alpha$$

	Primal		Dual	
	Time	Space	Time	Space
Gradient Descent	$O(np)$	$O(np)$	~~$O(np^2+n^3)$~~ $O(np)$	~~$O(n^2+np)$~~ $O(n^2)$
Analytical Soln	$O(np^2+p^3)$	$O(np+p^2)$	$O(np^2+n^3)$	$O(n^2+np)$

Ridge regression or least squares regression with squared Euclidean norm regularisation added.

$n \rightarrow$ no. of vectors x_i

$m \rightarrow$ dimension of vector x_i

$y_i \rightarrow$ scalar label.

problem is expressed as finding the weight vector w and scalar bias b where min the objective function

$$\min_{w} \; l(w) = \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|_2^2$$

Eliminating the bias.

finding the derivative with respect to b to zero yields

$$\frac{\partial f(w,b)}{\partial b} = \sum_{i=1}^{n} (x_i^T w + b - y_i) = 0$$

$$b = \bar{y} - \bar{x}^T w$$

& therefore the problem is to find the minimizer of

$$h(w) = f(w, b(w)) = \frac{1}{2}\sum_{i=1}^{n}\left[(x_i - \bar{x})^T w - (y_i - \bar{y})\right]^2 + \frac{\lambda}{2}\|w\|^2$$

vectors have zero mean.

$$h(w) = \frac{1}{2}\sum_{i=1}^{n}(x_i^T w - y_i)^2 + \frac{\lambda}{2}\|w\|^2$$

$$\operatorname{argmin}_{w}\left[\frac{1}{2} w^T (S + \lambda I) w - w^T X y\right]$$

where $S = XX^T$ is the $m \times m$ covariance matrix.

Dual problem:

$$\arg\min_{w,r}\left[\frac{1}{2}\|r\|^2+\frac{\lambda}{2}\|w\|^2\right] \quad \text{Subject to } r=X^Tw-y$$

whose Lagrangian is

$$L(w,r,\alpha)=\frac{1}{2}\|r\|^2+\frac{\lambda}{2}\|w\|^2+\alpha^T(r-X^Tw+y)$$

$$\frac{\partial L}{\partial w}(w,r,\alpha)=\lambda w-X\alpha=0 \;;\; w=\frac{1}{\lambda}X\alpha.$$

$$\frac{\partial L}{\partial r}(w,r,\alpha)=r+\alpha=0, \qquad r=-\alpha$$

$$g(\alpha)=L(w(\alpha),r(\alpha),\alpha)$$

$$=\frac{1}{2}\|\alpha\|^2+\frac{1}{2\lambda}\|X\alpha\|^2+\alpha^T\left(-\alpha-\frac{1}{\lambda}X^TX\alpha+y\right)$$

$$=-\frac{1}{2}\|\alpha\|^2-\frac{1}{2\lambda}\|X\alpha\|^2+\alpha^Ty$$

dual

$$\arg\max_{\alpha}\left[\frac{1}{2}\alpha^T(K+\lambda I)\alpha-\lambda\alpha^Ty\right]$$

where $K=X^TX$ is a $n\times n$ kernel matrix.

The soln is obtained $\alpha=\lambda(K+\lambda I)^{-1}y$

& $w=X(K+\lambda I)^{-1}y$

Primal vs. Dual:

One using the covariance matrix & the other the kernel matrix -

$$w = (S + \lambda I)^{-1} Xy = X(K + \lambda I)^{-1} y$$

$(K + \lambda I)^{-1} = O(n^3)$

$(K + \lambda I) = O(n)$

$K \rightarrow O(n^2 p)$

$X \cdot (K + \lambda I)^{-1} y = O(n^2)$

Time complexity $O(n^2 p + n^3)$

Space.

$X = O(np)$; $K = O(n^2)$ $(XX^T + \lambda I) = O(n^2)$

$(X X^T + \lambda I)^{-1} = O(n^2)$

$(X X^T + \lambda I)^{-1} y = O(n)$

Space complexity is $O(n^2 + np)$

Gradient descent:

L-Lipschitz continuous functions g^* (if g is strongly convex, then many existing algorithms are available & converge with a linear rate.

$$E[P(\alpha^{(t)}) - D(\alpha^*)] \leq \frac{C}{D(t)}$$

where $D(t)$ is a fn which has usually

a linear or quadratic growth (ie. $D(t) \sim O(t)$ or $D(t) \sim O(t^2)$)

Problem 2: Click Through Rate Prediction

The following problem is to predict the click through rate. The dataset description is given as follow:

- Data fields
  - id: ad identifier
  - click: 0/1 for non-click/click
  - hour: format is YYMMDDHH, so 14091123 means 23:00 on Sept. 11, 2014 UTC.
  - C1 -- anonymized categorical variable
  - banner_pos
  - site_id
  - site_domain
  - site_category
  - app_id
  - app_domain
  - app_category
  - device_id
  - device_ip
  - device_model
  - device_type
  - device_conn_type
  - C14-C21 -- anonymized categorical variables

1. Let's Start with SVM. Please use svm.LinearSVC, Let's try to add balanced weight to handle the class-imbalance issue.
   (a) please compute the precision/recall, f1-score, and confusion matrix.
   (b) Please run the algorithm for multiple times and observe the result.
2. Regularized SVM
   (a) Let's try to add penalty, please exlpore the use of the 'l1' and'l2' penalty in Scikit-learn, Please also use cross validation to select the best tuning parameters C.
   (b) please compute the precision/recall, f1-score, and confusion matrix for 'l1' and 'l2' model with the best tuning parameter C.

3. Please also explore using Logistic Regression on this problem and report the result.
   (a) Please plot the ROC curve and compute the area under the ROC curve. (You don't need to explore the use of penalty since the cross validation can be very slow)
   (b) Please plot the precision recall curve and compute the average precision
   (c) Please compute the F1-score and confusion matrix.
4. Please also explore using Random Forest on this problem and report the result.
   (a) Please use cross-validation to select the best tuning parameters
   (b) Please plot the ROC curve and compute the area under the ROC curve.
   (c) Please plot the precision recall curve and compute the average precision
   (d) Please compute the F1-score and confusion matrix
5. (Bonus 15 points) Please try to implement the xgboost library to this dataset.
   (a) Please use cross-validation to select the best tuning parameters
   (b) Please plot the ROC curve and compute the area under the ROC curve.
   (c) Please plot the precision recall curve and compute the average precision
   (d) Please compute the F1-score and confusion matrix

- Read the Data file:

```
In [2]: import pandas as pd
        data = pd.read_csv('C:/Users/hetul/Downloads/Assignment 3/trainsubset.csv')
```

- Overview of how the data looks like:

```
In [3]: data = data.head(10000)
        print(data.head(5))
```

```
                     id  click      hour    C1  banner_pos   site_id  \
0   10000941815109427 3      0  14102100  1005           0  1fbe01fe
1  10001693491178637 15      0  14102100  1005           0  1fbe01fe
2  10003719042151194 86      0  14102100  1005           0  1fbe01fe
3  10006407244808383 76      0  14102100  1005           0  1fbe01fe
4  10006790564170420 96      0  14102100  1005           1  fe8cc448

  site_domain site_category    app_id app_domain ...  device_type  \
0    f3845767      28905ebd  ecad2386   7801e8d9 ...            1
1    f3845767      28905ebd  ecad2386   7801e8d9 ...            1
2    f3845767      28905ebd  ecad2386   7801e8d9 ...            1
3    f3845767      28905ebd  ecad2386   7801e8d9 ...            1
4    9166c161      0569f928  ecad2386   7801e8d9 ...            1

  device_conn_type    C14  C15  C16   C17  C18  C19     C20  C21
0               2  15706  320   50  1722    0   35      -1   79
1               0  15704  320   50  1722    0   35  100084   79
2               0  15704  320   50  1722    0   35  100084   79
3               0  15706  320   50  1722    0   35  100084   79
4               0  18993  320   50  2161    0   35      -1  157

[5 rows x 24 columns]
```

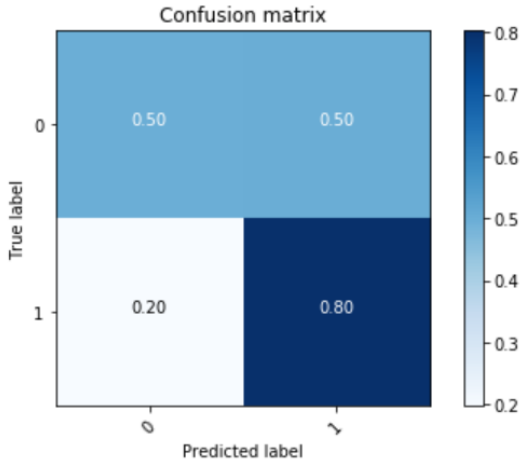
- Splitting the training and the testing data:

```
In [14]: selected_columns = ['C1','site_domain','app_id','app_domain','site_category','banner_pos','device_type',
                             'device_conn_type','C14','C15','C16','C17','C18','C19']
         # Please put your code here to define the data matrix X and y

         sel1 = data[selected_columns]
         sel2 = pd.get_dummies(sel1)
         X = sel2
         y = data[data.columns[1:2]]

In [5]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
        #print(X_train)
```

- Using SVM(svm.LinearSVC) with balanced weights to handle the class – imbalance issue:
- precision/recall, f1-score, and confusion matrix:

  Running 1st Time:

```
              precision    recall  f1-score   support

           0       0.93      0.50      0.65      2753
           1       0.24      0.80      0.37       547

avg / total       0.81      0.55      0.60      3300

0.2255178643448741
Normalized confusion matrix
[[0.4990919  0.5009081 ]
 [0.19744059 0.80255941]]
```

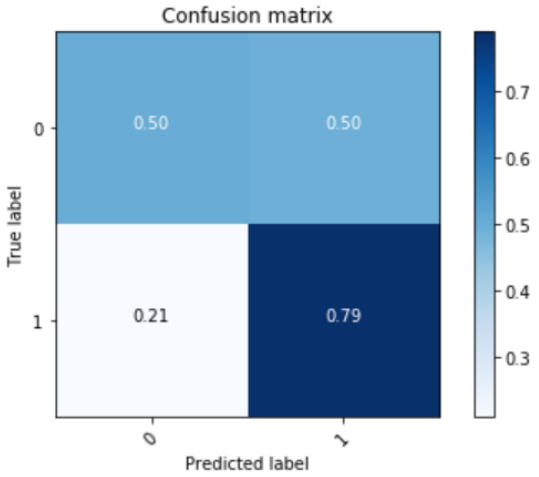


Running the algorithm multiple times has no effect on the accuracy, F-1 score and the confusion matrix. The average of all the values has little or no variance after multiple runs.

Running 2nd Time:

	precision	recall	f1-score	support
0	0.92	0.50	0.65	2753
1	0.24	0.79	0.37	547
avg / total	0.81	0.55	0.60	3300

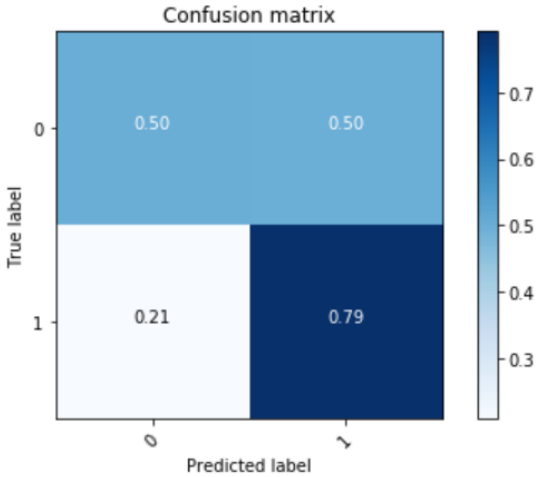
```
0.21818747510899097
Normalized confusion matrix
[[0.50236106 0.49763894]
 [0.21023766 0.78976234]]
```



Running 3rd Time:

	precision	recall	f1-score	support
0	0.92	0.50	0.65	2753
1	0.24	0.79	0.37	547
avg / total	0.81	0.55	0.60	3300

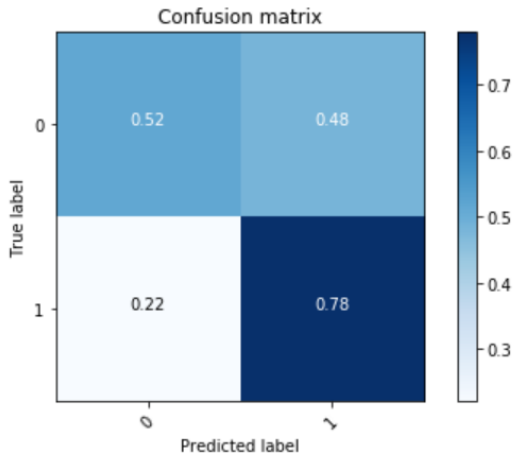
```
0.2193064832230781
Normalized confusion matrix
[[0.50199782 0.49800218]
 [0.20840951 0.79159049]]
```



- Regularized SVM:

  L1 as the penalty for regularization

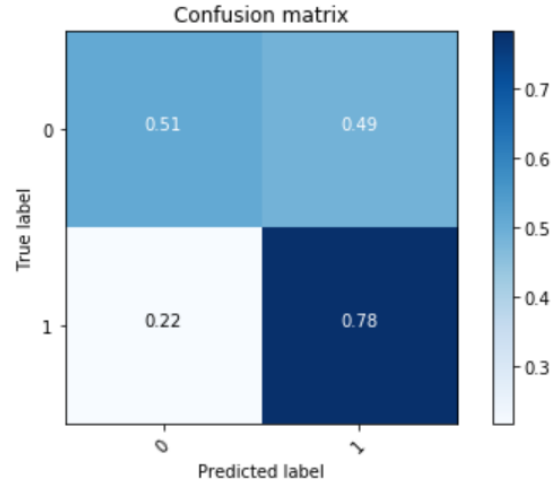
```
best parameters for l1: {'C': 0.01} ,testing accuracy for l1 : 0.5593939393939394
             precision    recall  f1-score   support

          0       0.92      0.52      0.66      2753
          1       0.24      0.78      0.37       547

avg / total       0.81      0.56      0.61      3300

[[1420 1333]
 [ 121  426]]
Normalized confusion matrix
[[0.51580094 0.48419906]
 [0.22120658 0.77879342]]
```



- Regularized SVM:

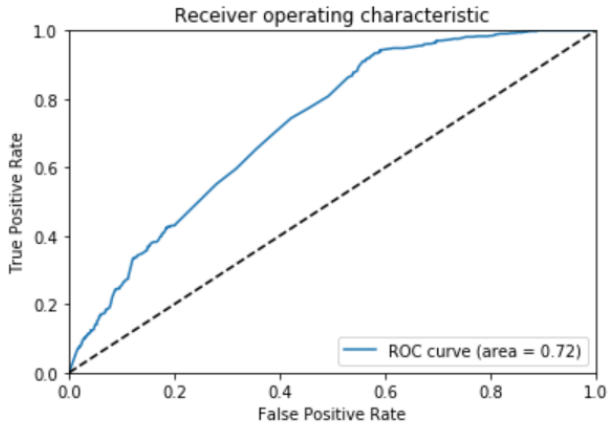
  L2 as the penalty for regularization

```
best parameters for l2: {'C': 1} ,testing accuracy for l2: 0.5563636363636364
             precision    recall  f1-score   support

          0       0.92      0.51      0.66      2753
          1       0.24      0.78      0.37       547

avg / total       0.81      0.56      0.61      3300

Normalized confusion matrix
[[0.51144206 0.48855794]
 [0.21755027 0.78244973]]
```

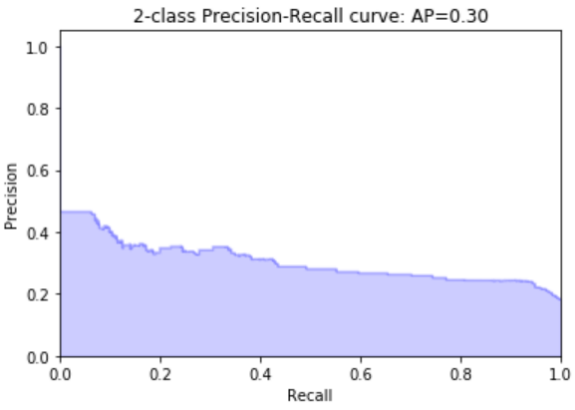


- Logistic Regression:

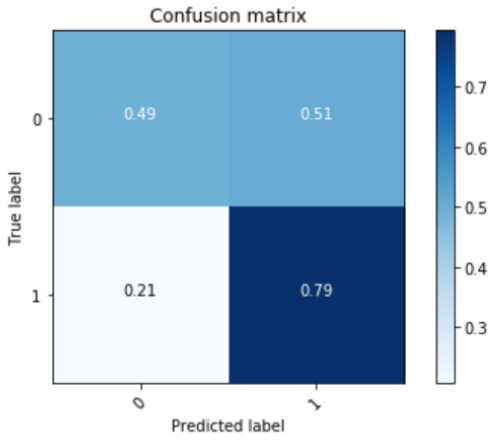
```
Area under the ROC curve : 0.723658
```




```
             precision    recall  f1-score   support

          0       0.92      0.51      0.66      2753
          1       0.24      0.78      0.37       547

avg / total       0.81      0.56      0.61      3300

Average precision-recall score: 0.30
[[1408 1345]
 [ 119  428]]
```

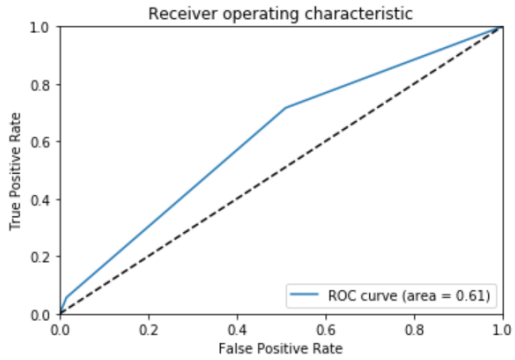



```
Normalized confusion matrix
[[0.49328006 0.50671994]
 [0.20658135 0.79341865]]
```



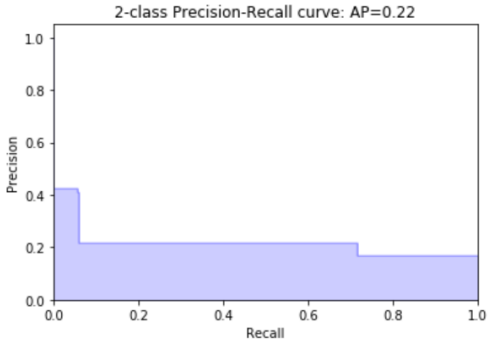
- Random Forest:

```
best parameters: {'max_depth': 2, 'max_features': 0.1, 'n_estimators': 1} ,testing accuracy: 0.5272727272727272
Area under the ROC curve : 0.612110
```



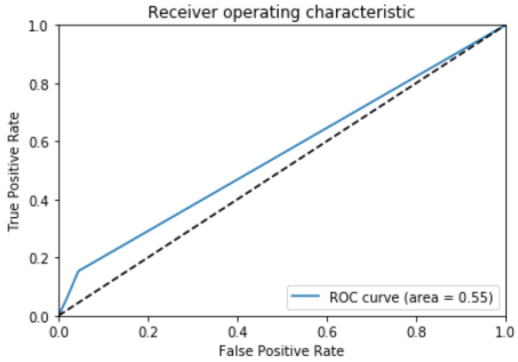
	precision	recall	f1-score	support
0	0.89	0.25	0.39	2753
1	0.18	0.84	0.30	547
avg / total	0.77	0.35	0.37	3300

```
Average precision-recall score: 0.22
Confusion Matrix
[[1348 1405]
 [ 155  392]]
```



- Gradient Boosting Tree:

```
              precision    recall  f1-score   support

           0       0.90      0.49      0.63      2753
           1       0.22      0.72      0.33       547

 avg / total       0.78      0.53      0.58      3300


best parameters: {'max_depth': 2, 'max_features': 0.1, 'n_estimators': 2} ,testing accuracy: 0.8284848484848485
Area under the ROC curve : 0.554373
```




```
Average precision-recall score: 0.20
Confusion Matrix
[[2700   53]
 [ 513   34]]
```