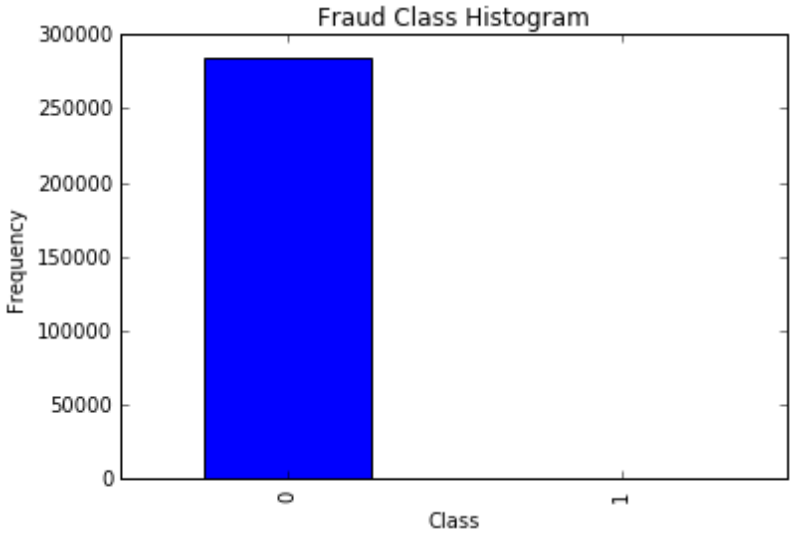
```
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import numpy as np
   ...:
   ...: data = pd.read_csv('C:\Users\Dev\Downloads\creditcard.csv')
   ...: data.head()
   ...:
   ...: #checking the target classes
   ...: count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
   ...: count_classes.plot(kind = 'bar')
   ...: plt.title("Fraud Class Histogram")
   ...: plt.xlabel("Class")
   ...: plt.ylabel("Frequency")
   ...:
Out[1]: <matplotlib.text.Text at 0xb5ce048>
```



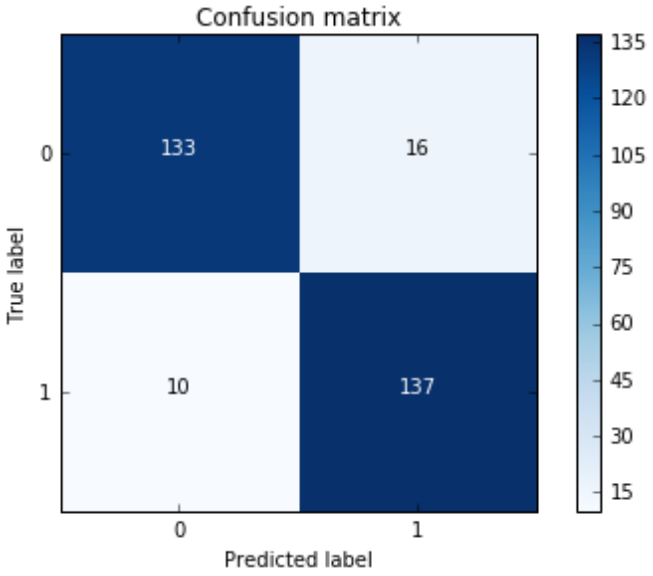

```
In [2]: from sklearn.preprocessing import StandardScaler
   ...:
   ...: data['normAmount'] = StandardScaler().fit_transform(data['Amount'].reshape(-1, 1))
   ...: data = data.drop(['Time','Amount'],axis=1)
   ...: data.head()
   ...:
   ...: X = data.ix[:, data.columns != 'Class']
   ...: y = data.ix[:, data.columns == 'Class']
   ...:
   ...: # Number of data points in the minority class
   ...: number_records_fraud = len(data[data.Class == 1])
   ...: fraud_indices = np.array(data[data.Class == 1].index)
   ...:
   ...: # Picking the indices of the normal classes
   ...: normal_indices = data[data.Class == 0].index
   ...:

In [3]: # Out of the indices we picked, randomly select "x" number (number_records_fraud)
   ...: random_normal_indices = np.random.choice(normal_indices, number_records_fraud, replace = False)
   ...: random_normal_indices = np.array(random_normal_indices)
   ...:
   ...: # Appending the 2 indices
   ...: under_sample_indices = np.concatenate([fraud_indices,random_normal_indices])
   ...:
   ...: # Under sample dataset
   ...: under_sample_data = data.iloc[under_sample_indices,:]
```

```
   ...:
   ...: X_undersample = under_sample_data.ix[:, under_sample_data.columns != 'Class']
   ...: y_undersample = under_sample_data.ix[:, under_sample_data.columns == 'Class']
   ...:

In [4]: print("Total number of transactions in resampled data: ", len(under_sample_data))
('Total number of transactions in resampled data: ', 984)

In [5]: from sklearn.cross_validation import train_test_split
   ...:
   ...: # Whole dataset
   ...: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 0)
   ...:
   ...: print("Number transactions train dataset: ", len(X_train))
   ...: print("Number transactions test dataset: ", len(X_test))
   ...: print("Total number of transactions: ", len(X_train)+len(X_test))
   ...:
   ...: # Undersampled dataset
   ...: X_train_undersample, X_test_undersample, y_train_undersample, y_test_undersample = train_test_split(X_undersample
   ...:                                                                                                     ,y_undersample
   ...:                                                                                                     ,test_size = 0.3
   ...:                                                                                                     ,random_state = 0)
   ...: print("")
   ...: print("Number transactions train dataset: ", len(X_train_undersample))
   ...: print("Number transactions test dataset: ", len(X_test_undersample))
   ...: print("Total number of transactions: ", len(X_train_undersample)+len(X_test_undersample))
   ...:
C:\Users\Dev\Anaconda2\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning: This module was deprecated in
version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the
interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
('Number transactions train dataset: ', 199364)
('Number transactions test dataset: ', 85443)
('Total number of transactions: ', 284807)

('Number transactions train dataset: ', 688)
('Number transactions test dataset: ', 296)
('Total number of transactions: ', 984)

In [6]: from sklearn.linear_model import LogisticRegression
   ...: from sklearn.cross_validation import KFold, cross_val_score
   ...: from sklearn.metrics import
confusion_matrix,precision_recall_curve,auc,roc_auc_score,roc_curve,recall_score,classification_report
   ...:
   ...: def printing_Kfold_scores(x_train_data,y_train_data):
   ...:     fold = KFold(len(y_train_data),5,shuffle=False)
   ...:
   ...:     # Different C parameters
   ...:     c_param_range = [0.01,0.1,1,10,100]
   ...:
   ...:     results_table = pd.DataFrame(index = range(len(c_param_range),2), columns = ['C_parameter','Mean recall score'])
   ...:     results_table['C_parameter'] = c_param_range
   ...:
   ...:     # the k-fold will give 2 lists: train_indices = indices[0], test_indices = indices[1]
   ...:     j = 0
   ...:     for c_param in c_param_range:
   ...:         print('-------------------------------------------')
   ...:         print('C parameter: ', c_param)
   ...:         print('-------------------------------------------')
   ...:         print('')
   ...:
   ...:         recall_accs = []
   ...:         for iteration, indices in enumerate(fold,start=1):
   ...:
   ...:             # Call the logistic regression model with a certain C parameter
   ...:             lr = LogisticRegression(C = c_param, penalty = 'l1')
   ...:
   ...:             # Use the training data to fit the model. In this case, we use the portion of the fold to train the model
   ...:             # with indices[0]. We then predict on the portion assigned as the 'test cross validation' with indices[1]
   ...:             lr.fit(x_train_data.iloc[indices[0],:],y_train_data.iloc[indices[0],:].values.ravel())
```

```
...:
...:             # Predict values using the test indices in the training data
...:             y_pred_undersample = lr.predict(x_train_data.iloc[indices[1],:].values)
...:
...:             # Calculate the recall score and append it to a list for recall scores representing the current c_parameter
...:             recall_acc = recall_score(y_train_data.iloc[indices[1],:].values,y_pred_undersample)
...:             recall_accs.append(recall_acc)
...:             print('Iteration ', iteration,': recall score = ', recall_acc)
...:
...:         # The mean value of those recall scores is the metric we want to save and get hold of.
...:         results_table.ix[j,'Mean recall score'] = np.mean(recall_accs)
...:         j += 1
...:         print('')
...:         print('Mean recall score ', np.mean(recall_accs))
...:         print('')
...:
...:     best_c = results_table.loc[results_table['Mean recall score'].idxmax()]['C_parameter']
...:
...:     # Finally, we can check which C parameter is the best amongst the chosen.
...:     print('*********************************************************************************')
...:     print('Best model to choose from cross validation is with C parameter = ', best_c)
...:     print('*********************************************************************************')
...:
...:     return best_c
...:
...: best_c = printing_Kfold_scores(X_train_undersample,y_train_undersample)
...:
-------------------------------------------
('C parameter: ', 0.01)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.9452054794520548)
('Iteration ', 2, ': recall score = ', 0.9178082191780822)
('Iteration ', 3, ': recall score = ', 1.0)
('Iteration ', 4, ': recall score = ', 0.95945945945945943)
('Iteration ', 5, ': recall score = ', 0.96969696969696972)

('Mean recall score ', 0.95843402555731316)

-------------------------------------------
('C parameter: ', 0.1)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.83561643835616439)
('Iteration ', 2, ': recall score = ', 0.86301369863013699)
('Iteration ', 3, ': recall score = ', 0.9152542372881356)
('Iteration ', 4, ': recall score = ', 0.93243243243243246)
('Iteration ', 5, ': recall score = ', 0.89393939393939392)

('Mean recall score ', 0.88805124012925263)

-------------------------------------------
('C parameter: ', 1)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.84931506849315064)
('Iteration ', 2, ': recall score = ', 0.8904109589041096)
('Iteration ', 3, ': recall score = ', 0.9491525423728813)
('Iteration ', 4, ': recall score = ', 0.94594594594594594)
('Iteration ', 5, ': recall score = ', 0.90909090909090906)

('Mean recall score ', 0.90878308496139937)

-------------------------------------------
('C parameter: ', 10)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.86301369863013699)
('Iteration ', 2, ': recall score = ', 0.8904109589041096)
('Iteration ', 3, ': recall score = ', 0.98305084745762716)
```

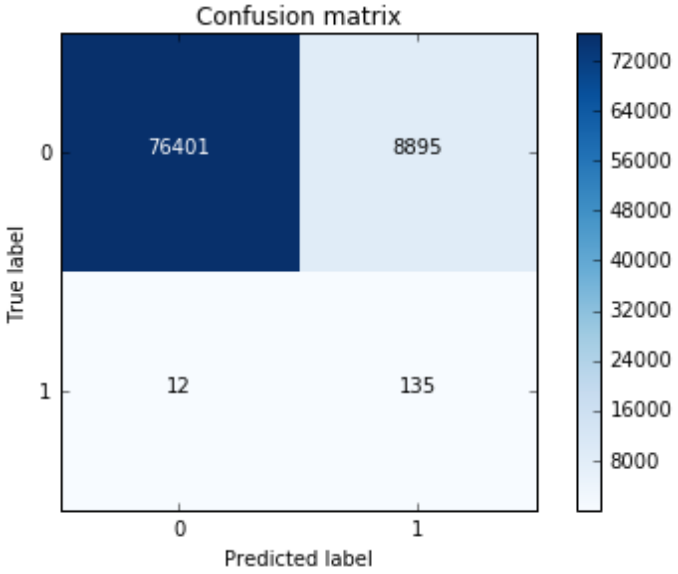
```
('Iteration ', 4, ': recall score = ', 0.93243243243243246)
('Iteration ', 5, ': recall score = ', 0.90909090909090906)

('Mean recall score ', 0.9155997693030431)

-------------------------------------------
('C parameter: ', 100)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.86301369863013699)
('Iteration ', 2, ': recall score = ', 0.90410958904109584)
('Iteration ', 3, ': recall score = ', 0.98305084745762716)
('Iteration ', 4, ': recall score = ', 0.94594594594594594)
('Iteration ', 5, ': recall score = ', 0.90909090909090906)

('Mean recall score ', 0.92104219803314302)

*********************************************************************************
('Best model to choose from cross validation is with C parameter = ', 0.01)
*********************************************************************************

In [7]: import itertools
   ...:
   ...: def plot_confusion_matrix(cm, classes,
   ...:                           normalize=False,
   ...:                           title='Confusion matrix',
   ...:                           cmap=plt.cm.Blues):
   ...:     """
   ...:     This function prints and plots the confusion matrix.
   ...:     Normalization can be applied by setting `normalize=True`.
   ...:     """
   ...:     plt.imshow(cm, interpolation='nearest', cmap=cmap)
   ...:     plt.title(title)
   ...:     plt.colorbar()
   ...:     tick_marks = np.arange(len(classes))
   ...:     plt.xticks(tick_marks, classes, rotation=0)
   ...:     plt.yticks(tick_marks, classes)
   ...:
   ...:     if normalize:
   ...:         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
   ...:         #print("Normalized confusion matrix")
   ...:     else:
   ...:         1#print('Confusion matrix, without normalization')
   ...:
   ...:     #print(cm)
   ...:
   ...:     thresh = cm.max() / 2.
   ...:     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
   ...:         plt.text(j, i, cm[i, j],
   ...:                  horizontalalignment="center",
   ...:                  color="white" if cm[i, j] > thresh else "black")
   ...:
   ...:     plt.tight_layout()
   ...:     plt.ylabel('True label')
   ...:     plt.xlabel('Predicted label')
   ...:

In [8]: # Use this C_parameter to build the final model with the whole training dataset and predict the classes in the test
   ...: # dataset
   ...: lr = LogisticRegression(C = best_c, penalty = 'l1')
   ...: lr.fit(X_train_undersample,y_train_undersample.values.ravel())
   ...: y_pred_undersample = lr.predict(X_test_undersample.values)
   ...:
   ...: # Compute confusion matrix
   ...: cnf_matrix = confusion_matrix(y_test_undersample,y_pred_undersample)
   ...: np.set_printoptions(precision=2)
   ...:
   ...: print("Recall metric in the testing dataset: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))
   ...:
   ...: # Plot non-normalized confusion matrix
```

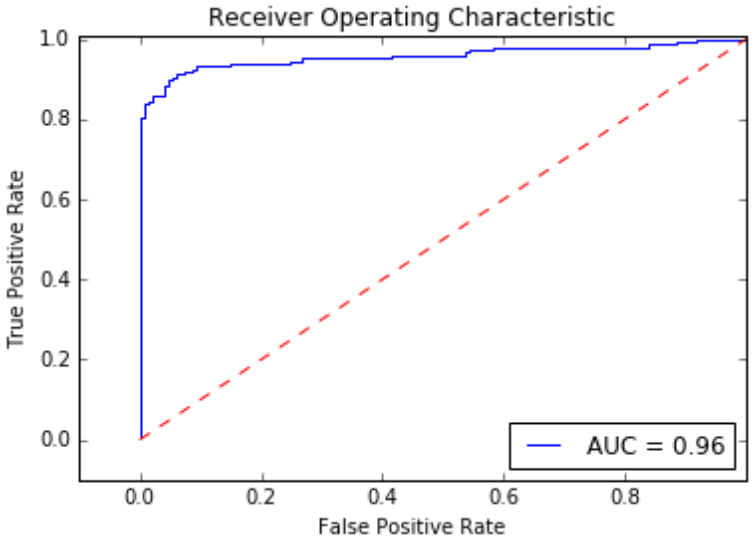
```
   ...: class_names = [0,1]
   ...: plt.figure()
   ...: plot_confusion_matrix(cnf_matrix
   ...:                       , classes=class_names
   ...:                       , title='Confusion matrix')
   ...: plt.show()
   ...:
('Recall metric in the testing dataset: ', 0)
```




```
In [9]: # Use this C_parameter to build the final model with the whole training dataset and predict the classes in the test
   ...: # dataset
   ...: lr = LogisticRegression(C = best_c, penalty = 'l1')
   ...: lr.fit(X_train_undersample,y_train_undersample.values.ravel())
   ...: y_pred = lr.predict(X_test.values)
   ...:
   ...: # Compute confusion matrix
   ...: cnf_matrix = confusion_matrix(y_test,y_pred)
   ...: np.set_printoptions(precision=2)
   ...:
   ...: print("Recall metric in the testing dataset: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))
   ...:
   ...: # Plot non-normalized confusion matrix
   ...: class_names = [0,1]
   ...: plt.figure()
   ...: plot_confusion_matrix(cnf_matrix
   ...:                       , classes=class_names
   ...:                       , title='Confusion matrix')
   ...: plt.show()
   ...:
   ...: # ROC CURVE
   ...: lr = LogisticRegression(C = best_c, penalty = 'l1')
   ...: y_pred_undersample_score =
lr.fit(X_train_undersample,y_train_undersample.values.ravel()).decision_function(X_test_undersample.values)
   ...:
   ...: fpr, tpr, thresholds = roc_curve(y_test_undersample.values.ravel(),y_pred_undersample_score)
   ...: roc_auc = auc(fpr,tpr)
   ...:
   ...: # Plot ROC
   ...: plt.title('Receiver Operating Characteristic')
   ...: plt.plot(fpr, tpr, 'b',label='AUC = %0.2f'% roc_auc)
   ...: plt.legend(loc='lower right')
   ...: plt.plot([0,1],[0,1],'r--')
   ...: plt.xlim([-0.1,1.0])
   ...: plt.ylim([-0.1,1.01])
   ...: plt.ylabel('True Positive Rate')
   ...: plt.xlabel('False Positive Rate')
   ...: plt.show()
   ...:
   ...: best_c = printing_Kfold_scores(X_train,y_train)
```

...:
('Recall metric in the testing dataset: ', 0)





('C parameter: ', 0.01)

('Iteration ', 1, ': recall score = ', 0.4925373134328358)
('Iteration ', 2, ': recall score = ', 0.60273972602739723)
('Iteration ', 3, ': recall score = ', 0.6833333333333335)
('Iteration ', 4, ': recall score = ', 0.56923076923076921)
('Iteration ', 5, ': recall score = ', 0.45000000000000001)

('Mean recall score ', 0.5595682284048672)

('C parameter: ', 0.1)

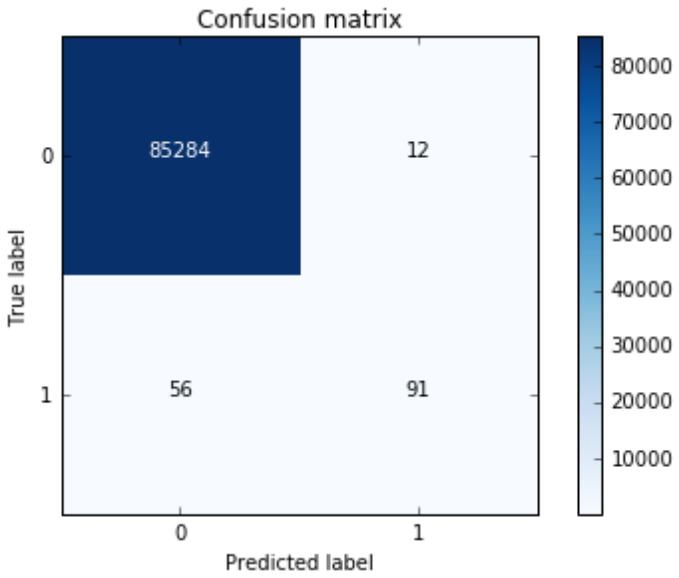
('Iteration ', 1, ': recall score = ', 0.56716417910447758)
('Iteration ', 2, ': recall score = ', 0.61643835616438358)
('Iteration ', 3, ': recall score = ', 0.6833333333333335)
('Iteration ', 4, ': recall score = ', 0.58461538461538465)
('Iteration ', 5, ': recall score = ', 0.52500000000000002)

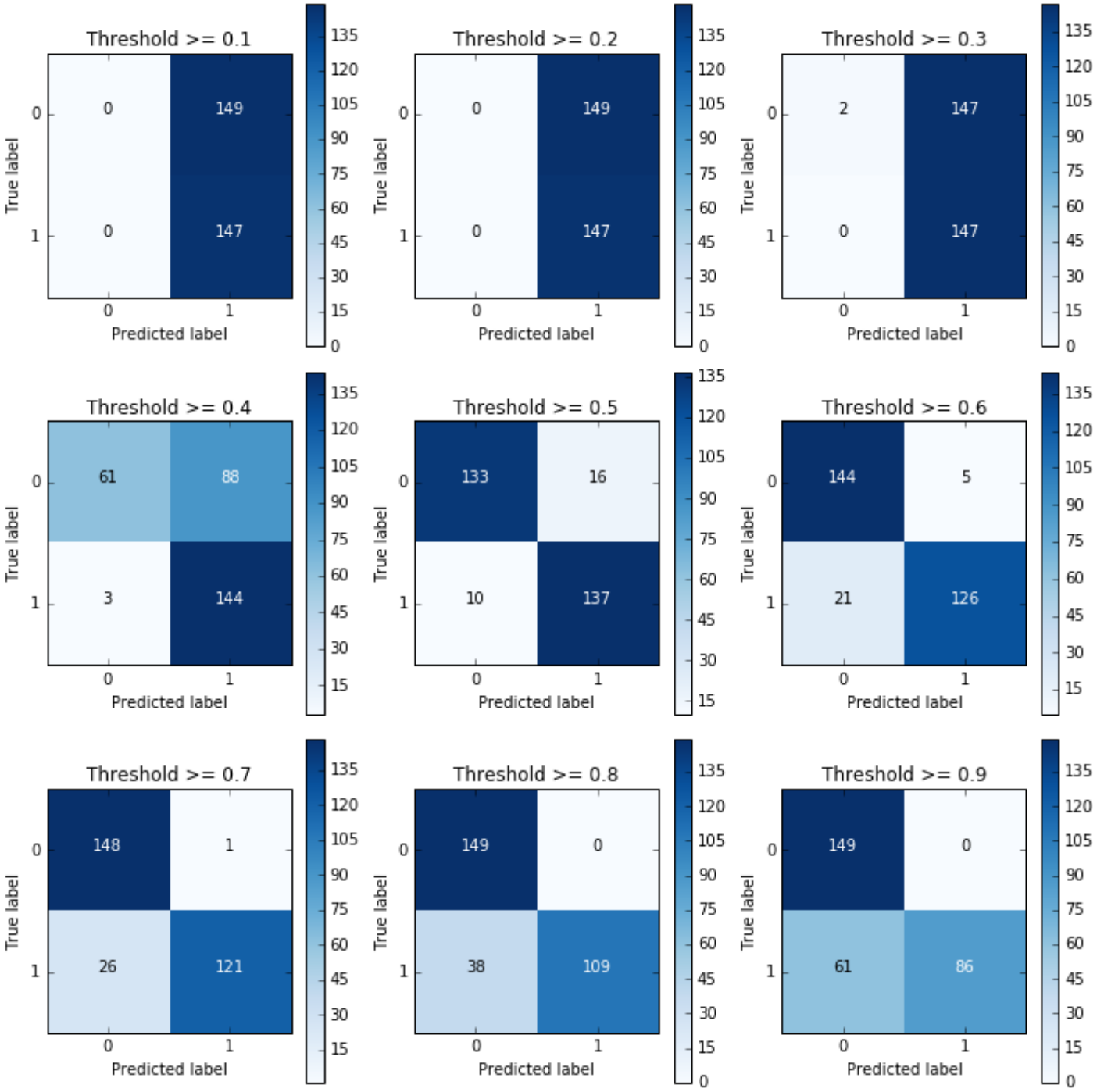
('Mean recall score ', 0.59531025064351584)

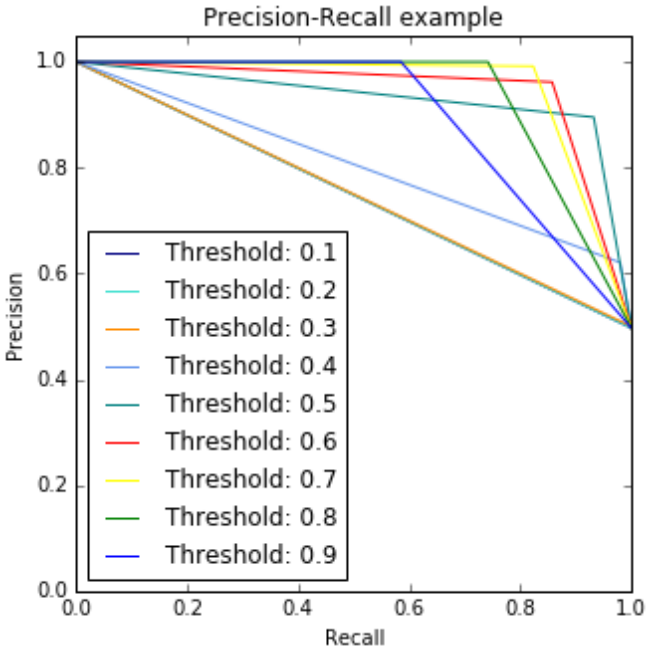
('C parameter: ', 1)

```
('Iteration ', 1, ': recall score = ', 0.55223880597014929)
('Iteration ', 2, ': recall score = ', 0.61643835616438358)
('Iteration ', 3, ': recall score = ', 0.71666666666666667)
('Iteration ', 4, ': recall score = ', 0.61538461538461542)
('Iteration ', 5, ': recall score = ', 0.5625)

('Mean recall score ', 0.61264568883716297)

-------------------------------------------
('C parameter: ', 10)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.55223880597014929)
('Iteration ', 2, ': recall score = ', 0.61643835616438358)
('Iteration ', 3, ': recall score = ', 0.73333333333333328)
('Iteration ', 4, ': recall score = ', 0.61538461538461542)
('Iteration ', 5, ': recall score = ', 0.57499999999999996)

('Mean recall score ', 0.61847902217049633)

-------------------------------------------
('C parameter: ', 100)
-------------------------------------------

('Iteration ', 1, ': recall score = ', 0.55223880597014929)
('Iteration ', 2, ': recall score = ', 0.61643835616438358)
('Iteration ', 3, ': recall score = ', 0.73333333333333328)
('Iteration ', 4, ': recall score = ', 0.61538461538461542)
('Iteration ', 5, ': recall score = ', 0.57499999999999996)

('Mean recall score ', 0.61847902217049633)

*********************************************************************************
('Best model to choose from cross validation is with C parameter = ', 10.0)
*********************************************************************************

In [10]: # Use this C_parameter to build the final model with the whole training dataset and predict the classes in the test
   ...: # dataset
   ...: lr = LogisticRegression(C = best_c, penalty = 'l1')
   ...: lr.fit(X_train,y_train.values.ravel())
   ...: y_pred_undersample = lr.predict(X_test.values)
   ...:
   ...: # Compute confusion matrix
   ...: cnf_matrix = confusion_matrix(y_test,y_pred_undersample)
   ...: np.set_printoptions(precision=2)
   ...:
   ...: print("Recall metric in the testing dataset: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))
   ...:
   ...: # Plot non-normalized confusion matrix
   ...: class_names = [0,1]
   ...: plt.figure()
   ...: plot_confusion_matrix(cnf_matrix
   ...:                       , classes=class_names
   ...:                       , title='Confusion matrix')
   ...: plt.show()
   ...:
   ...: lr = LogisticRegression(C = 0.01, penalty = 'l1')
   ...: lr.fit(X_train_undersample,y_train_undersample.values.ravel())
   ...: y_pred_undersample_proba = lr.predict_proba(X_test_undersample.values)
   ...:
   ...: thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
   ...:
   ...: plt.figure(figsize=(10,10))
   ...:
   ...: j = 1
   ...: for i in thresholds:
   ...:     y_test_predictions_high_recall = y_pred_undersample_proba[:,1] > i
   ...:
   ...:     plt.subplot(3,3,j)
```

```
...:     j += 1
...:
...:     # Compute confusion matrix
...:     cnf_matrix = confusion_matrix(y_test_undersample,y_test_predictions_high_recall)
...:     np.set_printoptions(precision=2)
...:
...:     # Plot non-normalized confusion matrix
...:     class_names = [0,1]
...:     plot_confusion_matrix(cnf_matrix
...:                           , classes=class_names
...:                           , title='Threshold >= %s'%i)
...:
...:     from itertools import cycle
...:
...: lr = LogisticRegression(C = 0.01, penalty = 'l1')
...: lr.fit(X_train_undersample,y_train_undersample.values.ravel())
...: y_pred_undersample_proba = lr.predict_proba(X_test_undersample.values)
...:
...: thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
...: colors = cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal', 'red', 'yellow', 'green', 'blue','black'])
...:
...: plt.figure(figsize=(5,5))
...:
...: j = 1
...: for i,color in zip(thresholds,colors):
...:     y_test_predictions_prob = y_pred_undersample_proba[:,1] > i
...:
...:     precision, recall, thresholds = precision_recall_curve(y_test_undersample,y_test_predictions_prob)
...:
...:     # Plot Precision-Recall curve
...:     plt.plot(recall, precision, color=color,
...:                   label='Threshold: %s'%i)
...:     plt.xlabel('Recall')
...:     plt.ylabel('Precision')
...:     plt.ylim([0.0, 1.05])
...:     plt.xlim([0.0, 1.0])
...:     plt.title('Precision-Recall example')
...:     plt.legend(loc="lower left")
...:
('Recall metric in the testing dataset: ', 0)
```




Threshold >= 0.1
0
149
0
147
True label
Predicted label
Threshold >= 0.2
0
149
0
147
True label
Predicted label
Threshold >= 0.3
2
147
0
147
True label
Predicted label
Threshold >= 0.4
61
88
3
144
True label
Predicted label
Threshold >= 0.5
133
16
10
137
True label
Predicted label
Threshold >= 0.6
144
5
21
126
True label
Predicted label
Threshold >= 0.7
148
1
26
121
True label
Predicted label
Threshold >= 0.8
149
0
38
109
True label
Predicted label
Threshold >= 0.9
149
0
61
86
True label
Predicted label


Precision-Recall example
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Threshold: 0.1
Threshold: 0.2
Threshold: 0.3
Threshold: 0.4
Threshold: 0.5
Threshold: 0.6
Threshold: 0.7
Threshold: 0.8
Threshold: 0.9

In [**11**]: