In [1]:
```
# imports

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import csv
import re
import pickle
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

In [2]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points {}%".format(round(((len(test_y)-np.trace(C))/len(test_y)*100),2)))
    fp = (int)(C[0][1])
    fn = (int)(C[1][0])
    print('Number of False Positives: ', fp)
    print('Number of False Negatives: ', fn)
    cost = (10 * fp) + (500 * fn)
    print('Total Cost (cost1+cost2): ', cost)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
```

```
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [5]:
```
def predict(X):
    """Give APS prediction for a input list/vector of features"""
    X = np.array(X)
    # Check input length
#     if len(X) != 170:
#         raise Exception('Please pass valid input. Some values are missing.')
    # Replace non-numeric values with NaN
    X.astype('<U32')
    #Refer: https://stackoverflow.com/questions/16223483/forced-conversion-of-non-numeric-numpy-arrays-with-nan-repla
    X = np.genfromtxt(X)
    # For a single input, shape should be (1, 170)
    X = X.reshape(1,-1)
    # Replace the missing values using saved imputers if present
    imputer = pickle.load(open("imputer.pkl", 'rb'))
    # As we dont know the class label, we can try with imputers (with class label 0 and 1)
    X = imputer.transform(X)
    #Standardize the data
    scaler = pickle.load(open("scaler.pkl", 'rb'))
    X = scaler.transform(X)
    # Remove the column with constant value ('cd_000') index= 89
    X = np.delete(X, 89,1)
    # Predict Y values with logistic regression as classifier
    clf = pickle.load(open("lr.sav", 'rb'))
    y_pred = clf.predict(X)
    return y_pred[0]
```

In [6]:
```
ip = '80168     0       2130706432      750     0       0       0       0       0       0       130496  3162754 13778
ip_list = ip.replace('\t', ',').split(',')
predict(ip_list)
```

Out[6]: 1

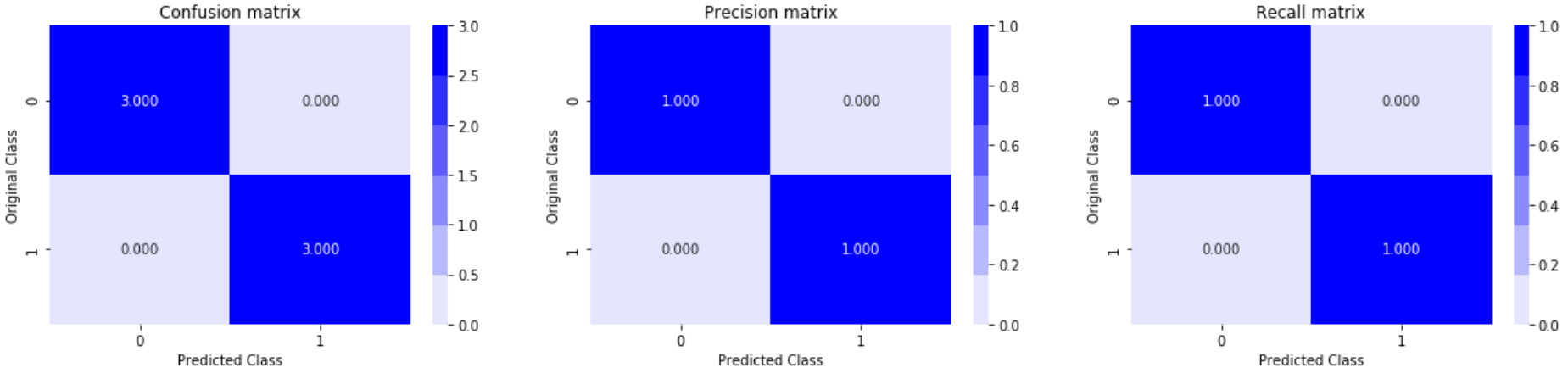
In [11]:
```
def checkPerformance(X, y):
    """For a given set of X and y values, predict and check model performance"""
    #Convert X (list of list) and y(list) to arrays
    X = np.array(X)
    y = np.array(y)
    #Convert Y values ('pos', 'neg' to 1 and 0)
    y = y=='pos'
    y.astype('int64')
    size = X.shape[0]
    # Predict for each data points using the selected classifier
    y_preds = []
    for i in range(size):
        y_pred = predict(X[i])
        y_preds.append(y_pred)

    # Plot confusion matrix with precision and recall
    plot_confusion_matrix(y, y_preds)
    # Label the predicted Y values back into 'pos' and 'neg'
    y_labels = ['pos' if y == 1 else 'neg' for y in y_preds]

    return y_labels
```

In [12]:
```
ip1 = '1055714  na      na      na      0       0       0       0       0       130     196186  10992134        4197
y1 = 'pos'
X1 = ip1.replace('\t', ',').split(',')
ip2 = '453634   na      na      na      0       0       0       0       0       5388    1141808 18432608        3814
y2 = 'pos'
X2 = ip2.replace('\t', ',').split(',')
ip3 = '351266   na      na      na      na      na      17706   192868  704464  3439760 7939556 8467710 1901626 2924
y3 = 'pos'
X3 = ip3.replace('\t', ',').split(',')
ip4 = '48746    na      80      74      0       0       na      na      na      na      na      na      na      na
y4 = 'neg'
X4 = ip4.replace('\t', ',').split(',')
ip5 = '268      4       118     96      0       0       0       0       0       0       1062    17980   18620   0
y5 = 'neg'
X5 = ip5.replace('\t', ',').split(',')
ip6 = '3300     na      2130706444      114     0       0       0       0       0       0       134     354     1722
```

```
y6 = 'neg'
X6 = ip6.replace('\t', ',').split(',')
```

In [13]:
```
X = [X1,X2,X3,X4,X5,X6]
y = [y1,y2,y3,y4,y5,y6]
checkPerformance(X, y)
```

```
Number of misclassified points 0.0%
Number of False Positives:  0
Number of False Negatives:  0
Total Cost (cost1+cost2):  0
```



Out[13]:
```
['pos', 'pos', 'pos', 'neg', 'neg', 'neg']
```

We are perfectly classified data points from test set. More data points can be used to test the performance.