Data Science Final (2)

October 26, 2020

[231]:
```
# import needed packages
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from numpy import *
from imblearn.over_sampling import SMOTE, BorderlineSMOTE, SMOTENC, SVMSMOTE
from sklearn.metrics import classification_report
from collections import Counter
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
import sklearn.metrics as metrics
import seaborn as sns
```

[232]:
```
# load the credit card dataset
dataset = pd.read_csv('UCI_Credit_Card.csv')
```

[233]:
```
# print part of dataset and the dataset information to ensure loading
print(dataset.info())
dataset.head()
```

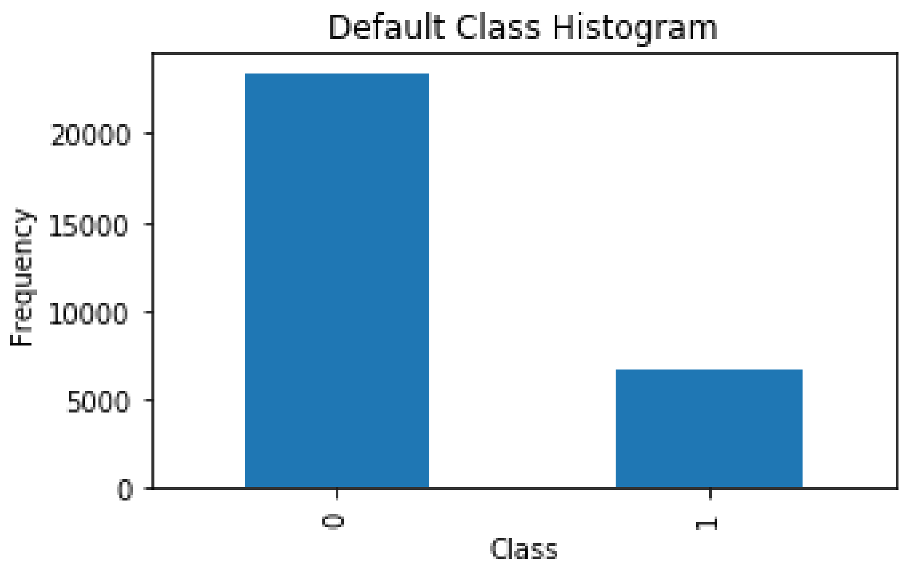
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
```

```
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
None
```

[233]:

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
3   4    50000.0    2          2         1   37      0      0      0      0
4   5    50000.0    1          2         1   57     -1      0     -1      0

   …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  …        0.0        0.0        0.0       0.0     689.0       0.0
1  …     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
2  …    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
3  …    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
4  …    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                           1
1    1000.0       0.0    2000.0                           1
2    1000.0    1000.0    5000.0                           0
3    1100.0    1069.0    1000.0                           0
4    9000.0     689.0     679.0                           0

[5 rows x 25 columns]
```

[234]:
```
# check the imbalance of the target default.payment.next.month.
dataset['default.payment.next.month'].value_counts().
    ↪plot(kind='bar',figsize=[5,3])
plt.title('Default Class Histogram')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
dataset['default.payment.next.month'].value_counts()
```



[234]:
```
0    23364
1     6636
Name: default.payment.next.month, dtype: int64
```

[235]:
```
# define dependent and independent variables
target = dataset['default.payment.next.month']
variables = dataset.drop(['default.payment.next.month','ID'], axis=1)
```

[236]:
```
# convert the categorical variables
categorical = ['SEX', 'MARRIAGE', 'EDUCATION', 'PAY_0', 'PAY_2', 'PAY_3',␣
    ↪'PAY_4', 'PAY_5', 'PAY_6']
variables = pd.get_dummies(data=variables, columns=categorical, drop_first=True)
variables.reset_index(drop=True, inplace=True)
```

[237]:
```
# re-scale numerical variables
numerical = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3',␣
    ↪'BILL_AMT4',
```

```
'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4',␣
↪'PAY_AMT5', 'PAY_AMT6']
sc = StandardScaler()
variables[numerical] = sc.fit_transform(variables[numerical])
```

[238]:
```
# train test split
x_train,x_test,y_train,y_test = train_test_split(variables, target, test_size =␣
↪0.2)
```

[239]:
```
# accuracy is equal to correct_prediction/total_number_in_test_set
# Model Performance Metrics

def generate_model_report(y_actual, y_predicted):
    Accuracy = accuracy_score(y_actual, y_predicted)
    Precision = precision_score(y_actual, y_predicted)
    Recall = recall_score(y_actual, y_predicted)
    F1_Score = f1_score(y_actual, y_predicted)
    return [Accuracy, Precision, Recall, F1_Score]
```

[240]:
```
# Plot ROC

def plot_roc(x_test, model, title):
    y_score = model.predict_proba(x_test)[:,1]
    # roc_auc_score and roc_auc does not always give the same result, due to␣
↪different thresholds
    # https://stackoverflow.com/questions/31159157/
↪different-result-with-roc-auc-score-and-auc
    # y_score = LogitReg.predict(x_test)  -- this will match with␣
↪roc_auc_score(y_test, y_pred)

    fpr, tpr, thresholds = roc_curve(y_test, y_score)
    roc_auc = roc_auc_score(y_test, y_pred)
    print(roc_auc)
    plt.title(title)
    plt.plot(fpr, tpr, 'b',label='AUC = %0.3f'% 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()
```

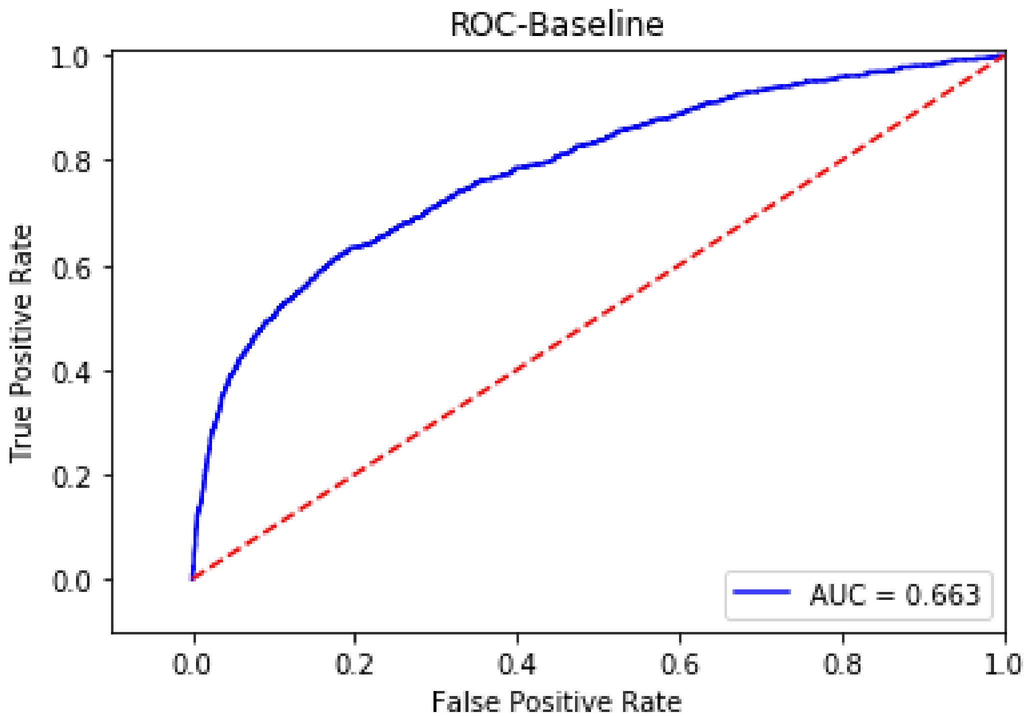
[241]:
```
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
```

```
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    # normalize confusion matrix by set normalize to 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(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')
```

[242]:
```
# model fitting using logistic regression classification
# fit the model and predict the target value for test set
LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train, y_train)
y_pred = LogitReg.predict(x_test)
```

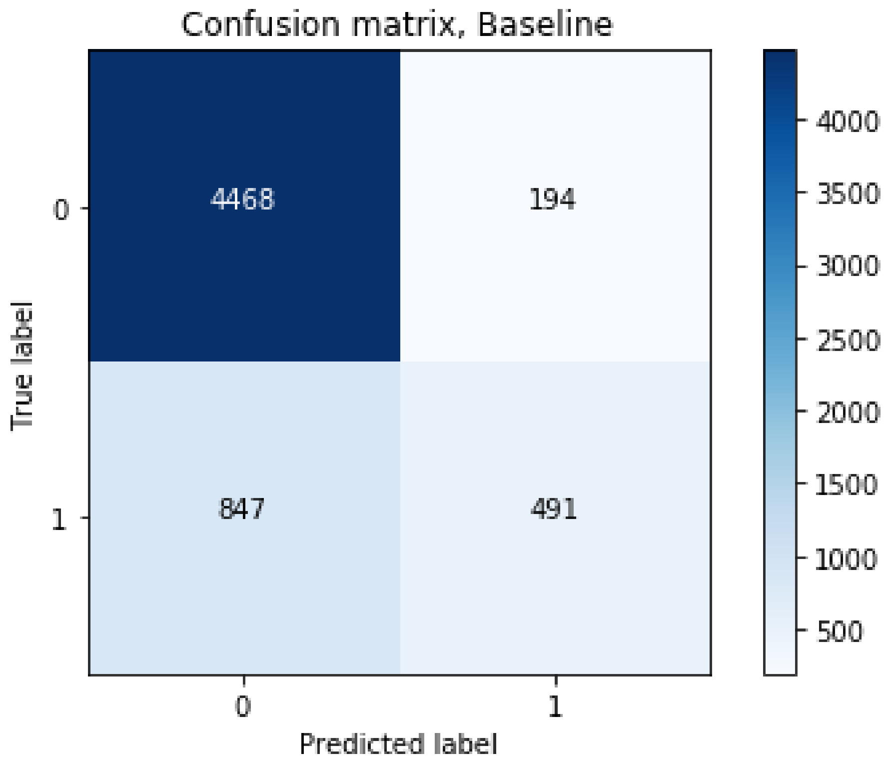
[243]:
```
plot_roc(x_test, LogitReg.fit(x_train, y_train),"ROC-Baseline")
```

```
0.6626762893579038
```



[244]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, Baseline')
```

```
[[4468  194]
 [ 847  491]]
```



[245]:
```
metrics_without = generate_model_report(y_test, y_pred)
```

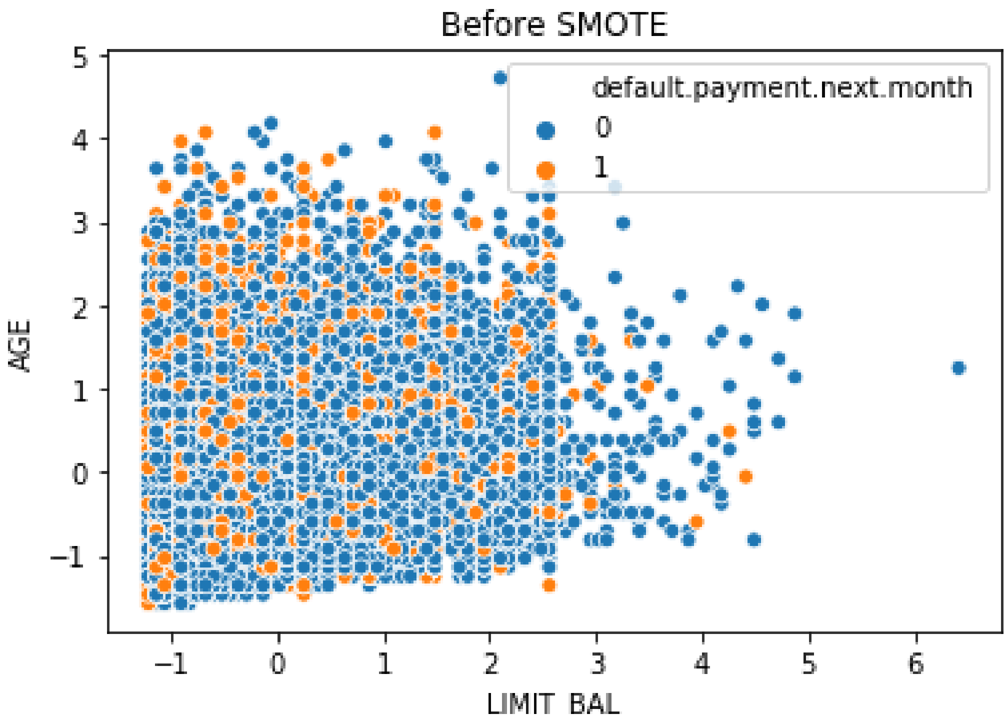
[246]:
```
# Treatment 1 - Smote
```

[247]:
```
# unique, count = np.unique(y_train, return_counts=True)
counter = Counter(y_train)
print(Counter(y_train))
```

```
Counter({0: 18702, 1: 5298})
```

[248]:
```
# scatter plot by class label
# Graph is normalized
tempset = x_train.join(y_train)
sns.scatterplot(data=tempset, x="LIMIT_BAL", y="AGE",hue="default.payment.next.
 ↪month").set(title="Before SMOTE")
```

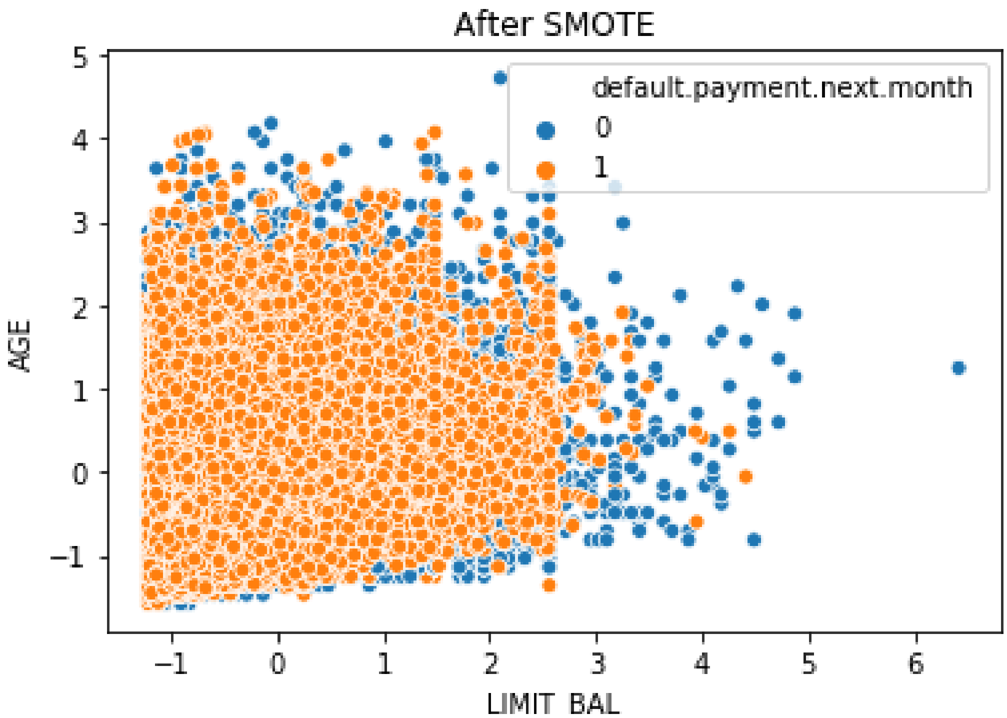
[248]:
```
[Text(0.5, 1.0, 'Before SMOTE')]
```



[249]:
```
sm = SMOTE(random_state=12, k_neighbors = 10)
x_train_res, y_train_res = sm.fit_sample(x_train, y_train)
```

[250]:
```
# scatter plot after SMOTE
# Graph is normalized
tempset = x_train_res.join(y_train_res)
sns.scatterplot(data=tempset, x="LIMIT_BAL", y="AGE",hue="default.payment.next.
 ↪month").set(title="After SMOTE")
```

[250]:
```
[Text(0.5, 1.0, 'After SMOTE')]
```



[251]:
```
print(Counter(y_train_res))
```

Counter({1: 18702, 0: 18702})

[252]:
```
# model fitting using logistic regression classification
# try again on test data after applying SMOTE

LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train_res, y_train_res)
y_pred = LogitReg.predict(x_test)
```

[253]:
```
generate_model_report(y_test, y_pred)
```

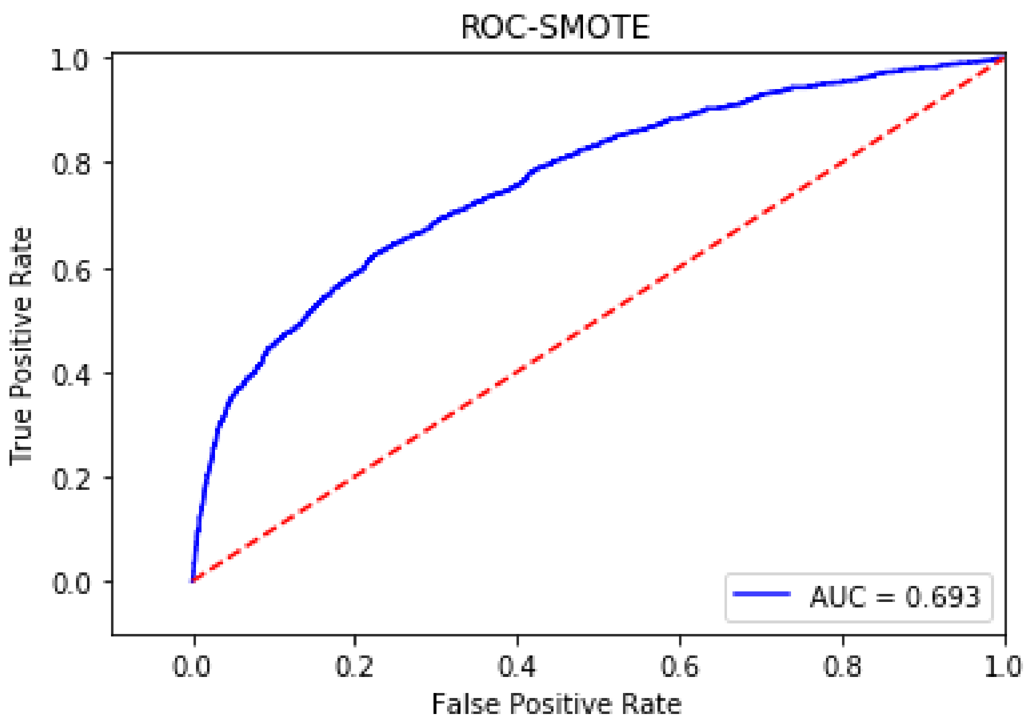
[253]: [0.756, 0.46236559139784944, 0.57847533632287, 0.5139442231075698]

[254]:
```
# So recall score uped 60%, but both accuracy and precision score dropped after␣
 ↪SMOTE. This is expected as SMOTE is a trade-off between
# precision vs. recall.That's because this technique puts more weight to the␣
 ↪small class, makes the model bias to it.
# The model will now predict the minority class with higher accuracy but the␣
 ↪overall accuracy will decrease.
# However, we can change the parameters within SMOTE (eg. K), or use␣
 ↪SMOTE-variant to maximize improvement in recall
```

```
# with respect to drop in precision
```

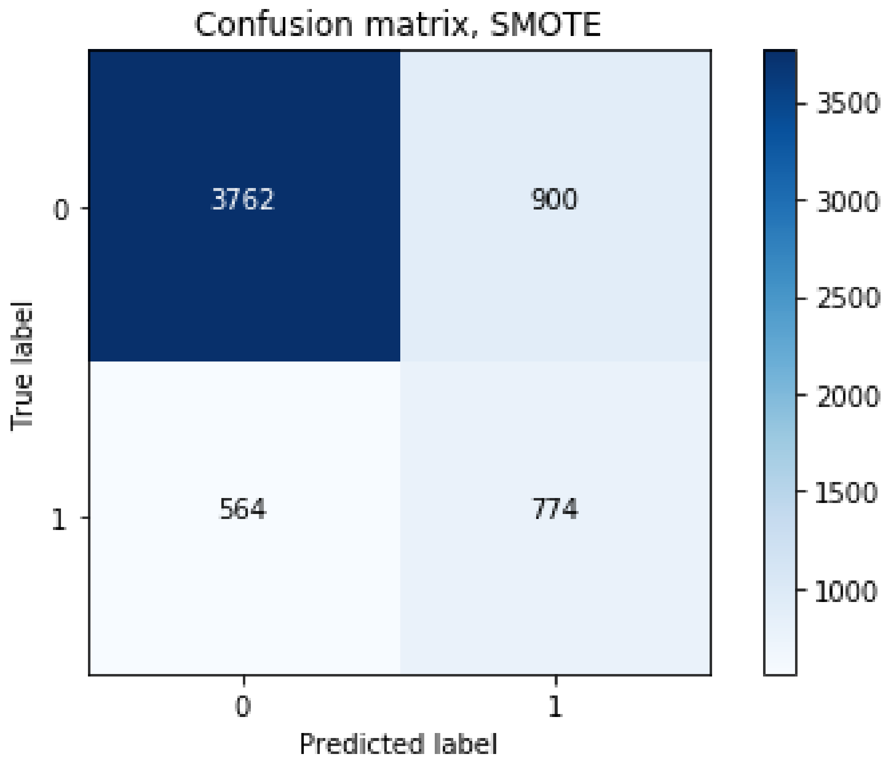
[255]:
```
plot_roc(x_test, LogitReg.fit(x_train_res, y_train_res),"ROC-SMOTE")
```

```
0.6927125716363385
```



[256]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, SMOTE')
```

```
[[3762  900]
 [ 564  774]]
```



[257]:
```
metrics_smote = generate_model_report(y_test, y_pred)
```

[258]:
```
# Try  Borderline SMOTE, in which only the minority examples near the␣
↪borderline are over-sampled
smb = BorderlineSMOTE(random_state=12, k_neighbors = 10)
x_train_res, y_train_res = smb.fit_sample(x_train, y_train)
```

[259]:
```
print(Counter(y_train_res))
```

```
Counter({1: 18702, 0: 18702})
```

[260]:
```
LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train_res, y_train_res)
y_pred = LogitReg.predict(x_test)
```

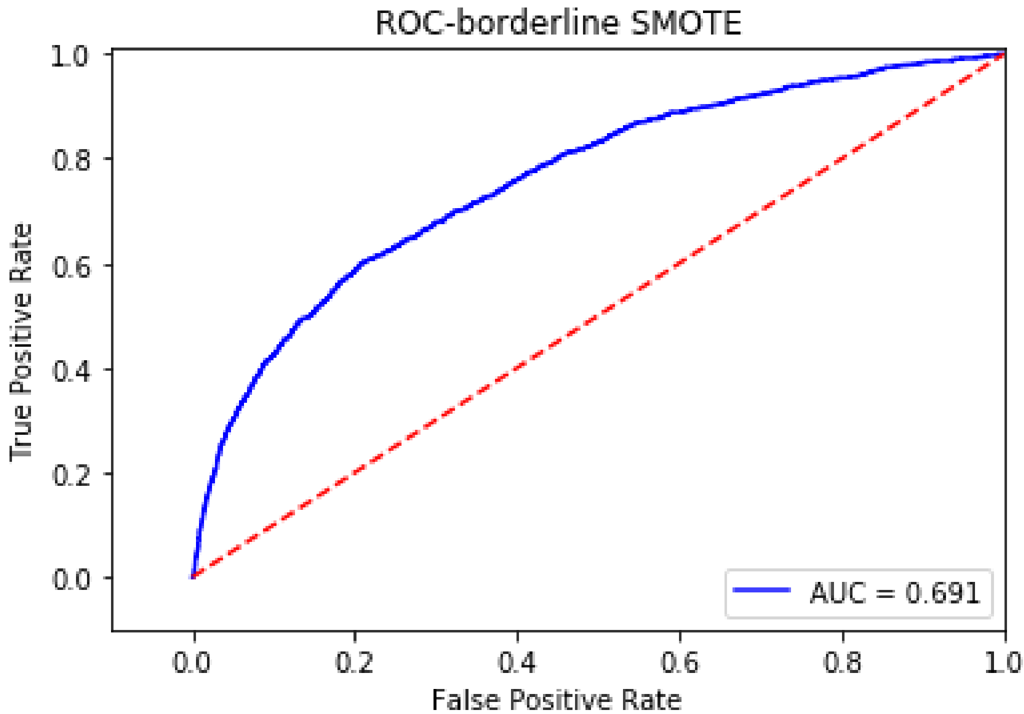
[261]:
```
generate_model_report(y_test, y_pred)
```

[261]:
```
[0.7318333333333333, 0.4295371814872595, 0.617339312406577, 0.5065930696105488]
```

So with broderline SMOTE, recall up bit further, but accuracy and precision are even worse

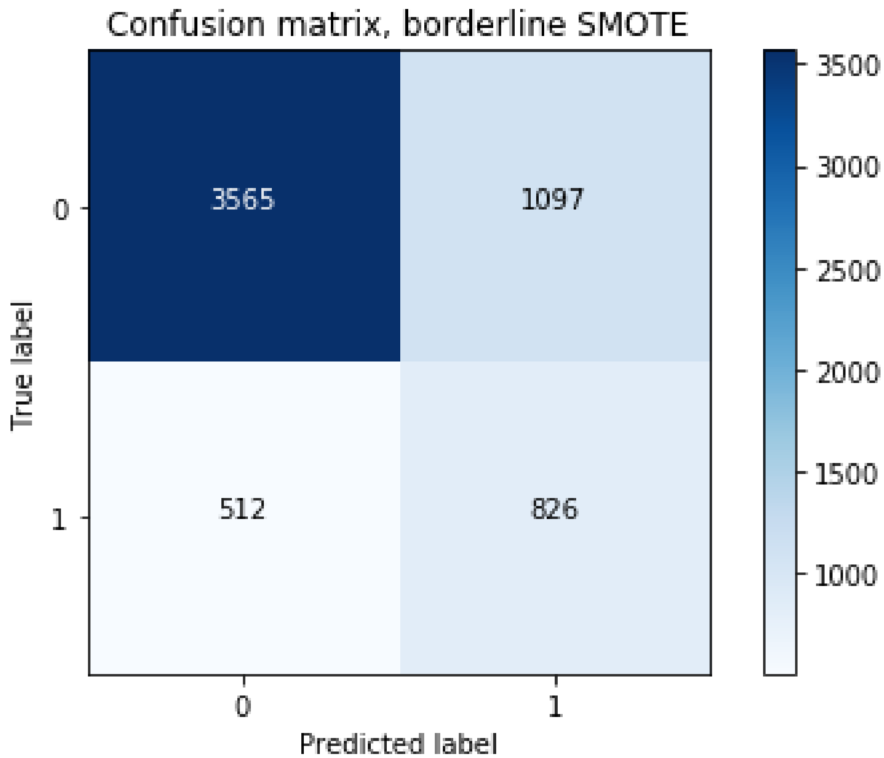
[262]:
```
plot_roc(x_test, LogitReg.fit(x_train_res, y_train_res),"ROC-borderline SMOTE")
```

```
0.6910162885499208
```



[263]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, borderline␣
 ↪SMOTE')
```

```
[[3565 1097]
 [ 512  826]]
```



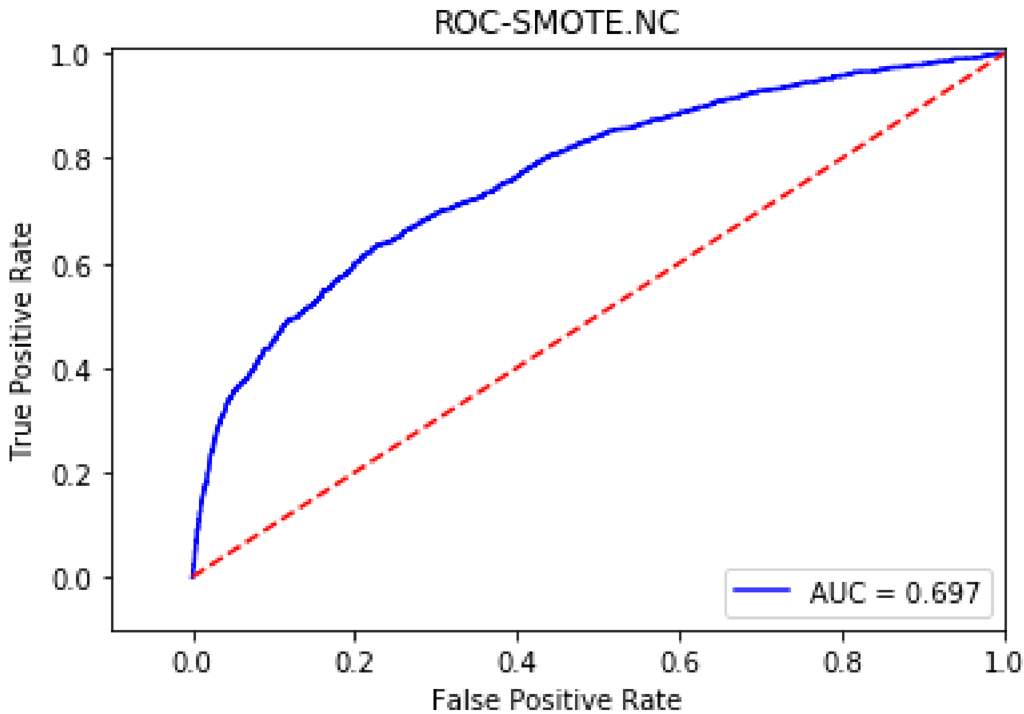
[264]:
```
metrics_border = generate_model_report(y_test, y_pred)
```

[265]:
```
# Try SMOTENC, for dataset with continuous and categorical features
smote_nc = SMOTENC(categorical_features=[0, 1], random_state=12, k_neighbors =␣
 ↪10)
x_train_res, y_train_res = smote_nc.fit_sample(x_train, y_train)
LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train_res, y_train_res)
y_pred = LogitReg.predict(x_test)
generate_model_report(y_test, y_pred)
```

[265]:
```
[0.7551666666666667,
 0.46185206755969715,
 0.5926756352765321,
 0.5191489361702128]
```

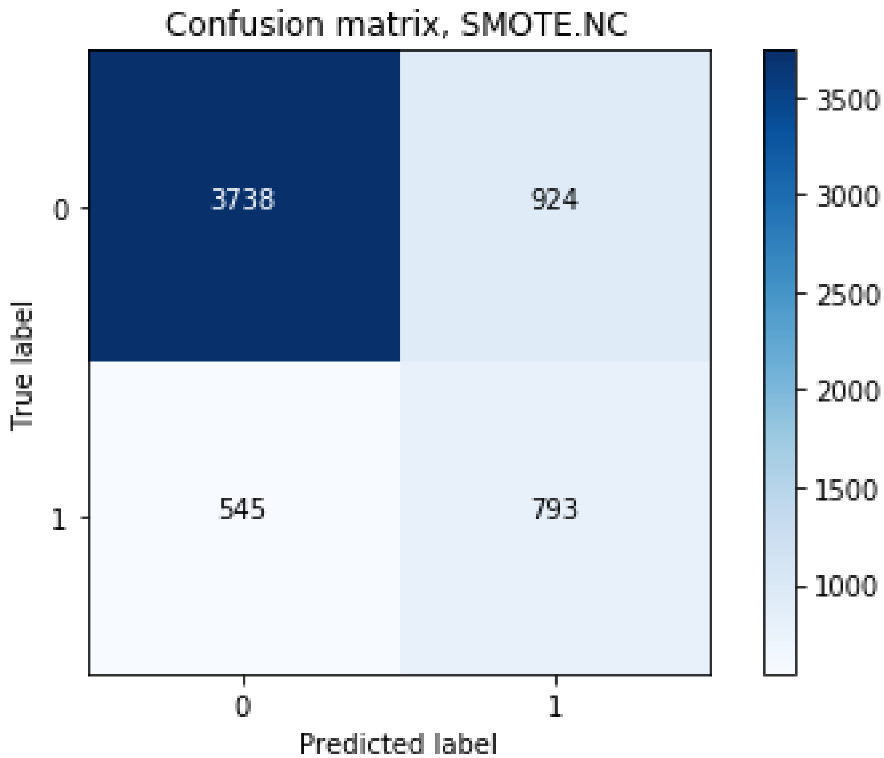
[266]:
```
plot_roc(x_test, LogitReg.fit(x_train_res, y_train_res),"ROC-SMOTE.NC")
```

```
0.697238718539167
```



[267]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, SMOTE.NC')
```

```
[[3738  924]
 [ 545  793]]
```



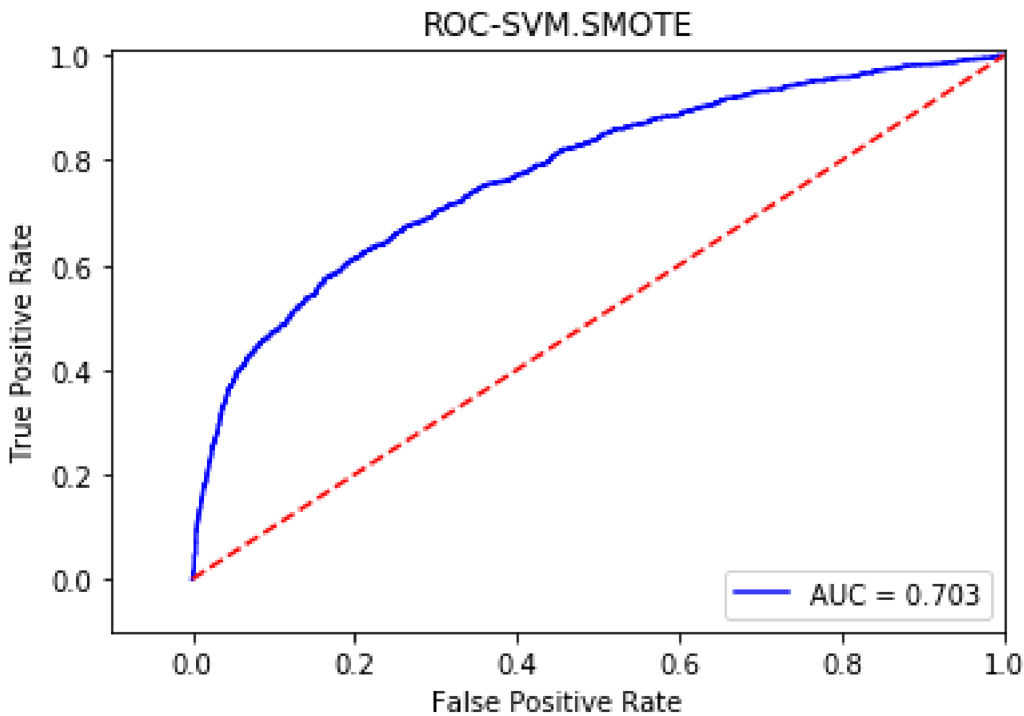
[268]:
```
metrics_nc = generate_model_report(y_test, y_pred)
```

[269]:
```
# Try SVMSMOTE, which use SVM algorithm to detect sample
smote_svm = SVMSMOTE(random_state=12,k_neighbors = 10)
x_train_res, y_train_res = smote_svm.fit_sample(x_train, y_train)
LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train_res, y_train_res)
y_pred = LogitReg.predict(x_test)
generate_model_report(y_test, y_pred)
```

[269]: [0.767, 0.48164014687882495, 0.5881913303437967, 0.5296096904441453]

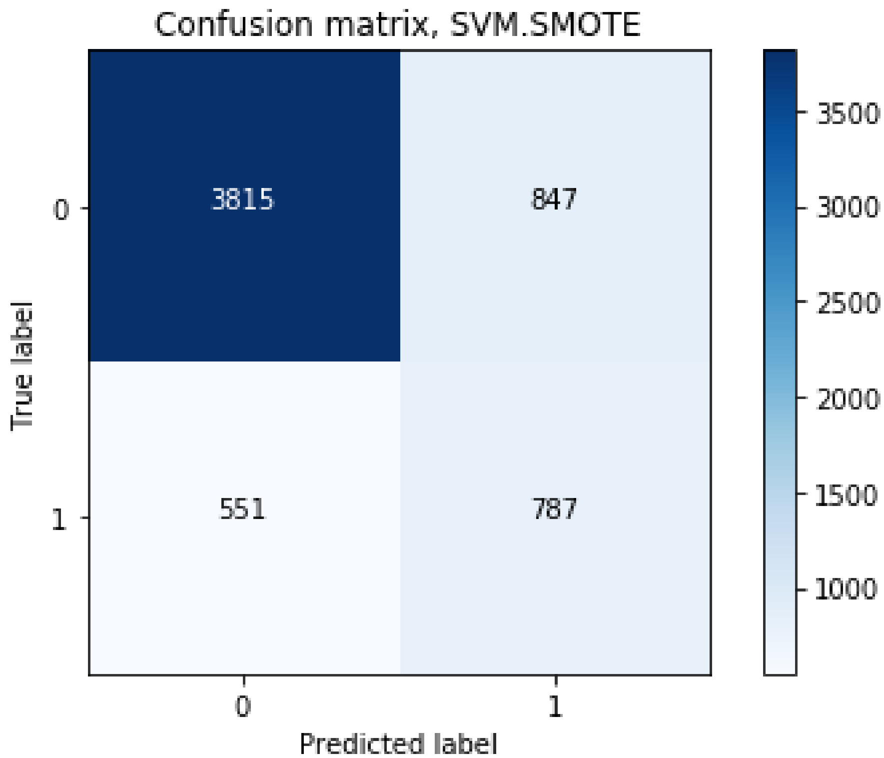
[270]:
```
plot_roc(x_test, LogitReg.fit(x_train_res, y_train_res),"ROC-SVM.SMOTE")
```

0.7032548243310575



[271]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, SVM.SMOTE')
```

```
[[3815  847]
 [ 551  787]]
```



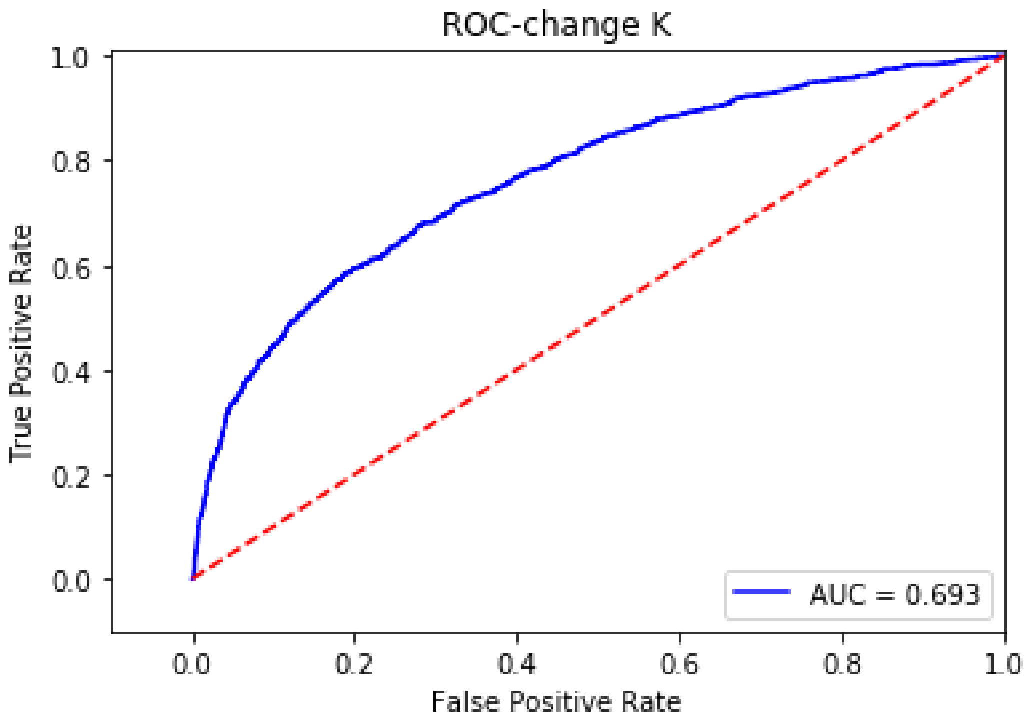
[272]:
```
metrics_svm = generate_model_report(y_test, y_pred)
```

[273]:
```
# Change k_neighbors
smb = BorderlineSMOTE(random_state=12, k_neighbors = 3)
x_train_res, y_train_res = smb.fit_sample(x_train, y_train)
LogitReg = LogisticRegression(max_iter=500)
LogitReg.fit(x_train_res, y_train_res)
y_pred = LogitReg.predict(x_test)
generate_model_report(y_test, y_pred)
```

[273]:
```
[0.7243333333333334,
 0.4217046580773043,
 0.6360239162929746,
 0.5071513706793803]
```

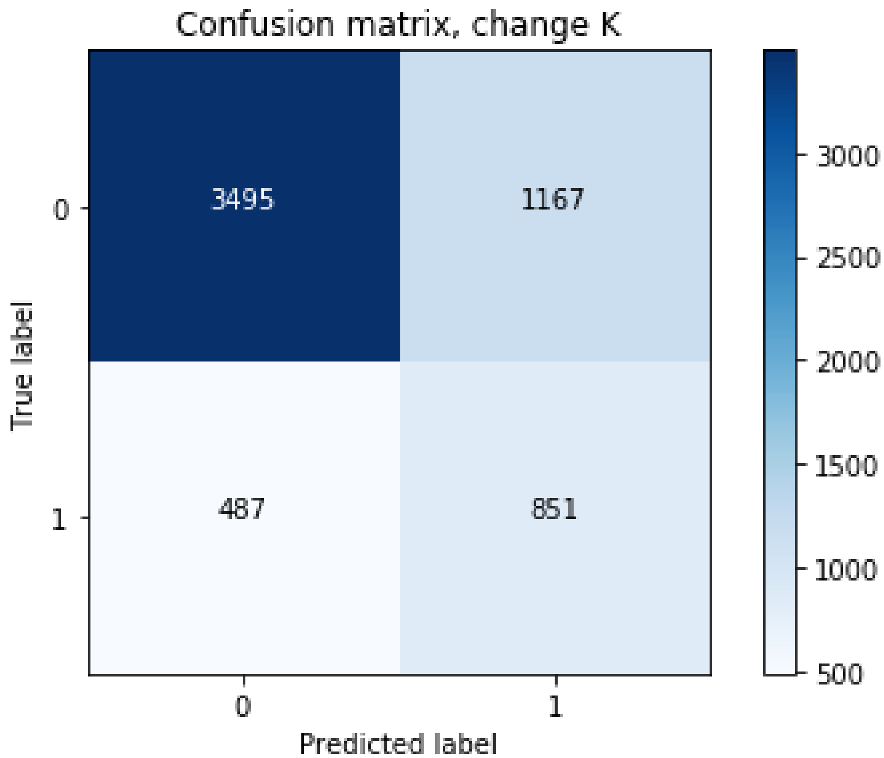
[274]:
```
plot_roc(x_test, LogitReg.fit(x_train_res, y_train_res),"ROC-change K")
```

```
0.6928510829856122
```



[275]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plot_confusion_matrix(cm, classes=[0,1], title='Confusion matrix, change K')
```

```
[[3495 1167]
 [ 487  851]]
```



[276]:
```
metrics_changek = generate_model_report(y_test, y_pred)
```

[277]:
```
# k_neightbors has little effect on accuracy and precision, but shows a␣
↪moderate
# negative relationship with recall score.
# Therefore, use smaller K
```

[278]:
```
# SVMSMOTE takes a long to time to run, but it produce the most desired outcome
```

[279]:
```
result_tbl = None
metrics_list = [metrics_without,metrics_smote,metrics_border,metrics_nc,␣
↪metrics_svm, metrics_changek]
for metrics in metrics_list:
    result_tbl = pd.concat([result_tbl,pd.Series(metrics)],axis = 1)

result_tbl.columns = ["Baseline", "SMOTE","Borderline-SMOTE", \
                                          "SMOTE-NC", "SVM-SMOTE","Change K"]
result_tbl.index = ["Accuracy", "Precision", "Recall", "F1 Score"]
```

[280]:
```
round(result_tbl,3)
```

[280]:

	Baseline	SMOTE	Borderline-SMOTE	SMOTE-NC	SVM-SMOTE	Change K
Accuracy	0.826	0.756	0.732	0.755	0.767	0.724
Precision	0.717	0.462	0.430	0.462	0.482	0.422
Recall	0.367	0.578	0.617	0.593	0.588	0.636
F1 Score	0.485	0.514	0.507	0.519	0.530	0.507

[281]:

```
sub_table = result_tbl.drop(index = ["Precision"])
round(sub_table,3)
```

[281]:

	Baseline	SMOTE	Borderline-SMOTE	SMOTE-NC	SVM-SMOTE	Change K
Accuracy	0.826	0.756	0.732	0.755	0.767	0.724
Recall	0.367	0.578	0.617	0.593	0.588	0.636
F1 Score	0.485	0.514	0.507	0.519	0.530	0.507

[ ]:

```
```