3-randomforest-rf

May 8, 2024

[1]:

```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
c:\users\user\anaconda3\lib\site-packages (0.7.0)
Requirement already satisfied: scipy>=0.19.1 in
c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn) (1.5.0)
Requirement already satisfied: numpy>=1.13.3 in
c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn) (1.19.4)
Requirement already satisfied: joblib>=0.11 in c:\users\user\anaconda3\lib\site-
packages (from imbalanced-learn) (0.16.0)
Requirement already satisfied: scikit-learn>=0.23 in
c:\users\user\anaconda3\lib\site-packages (from imbalanced-learn) (0.23.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\user\anaconda3\lib\site-packages (from scikit-learn>=0.23->imbalanced-
learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:

```
# Check version number
import imblearn
from imblearn.over_sampling import RandomOverSampler
print(imblearn.__version__)
```

```
0.7.0
```

[3]:

```
# Importing packages
import numpy as np
from numpy import where
from numpy import mean
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

%matplotlib inline
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from collections import Counter
from imblearn.over_sampling import SMOTE
```

[4]:

```
# Importing and cleaning the data
train_data = pd.read_csv('DBS.csv', sep=';')
test_data = pd.read_csv('DBS_2020.csv', sep=';')
train_data.head()
```

[4]:

```
   access  tests tests_grade  exam  project project_grade  assignments  \
0    1256  57.00           A    19    91.54             A         40.0
1     985  42.87           B    19    75.96             A         13.7
2    1455  54.50           A    16    96.79             A         40.0
3     998  54.50           A    16    93.36             A         40.0
4    1347  55.00           A    16    92.86             A         39.0

   result_points result_grade  graduate  year  acad_year
0          189.92            A         1  2019  2019/2020
1          189.43            A         1  2017  2017/2018
2          188.91            A         1  2019  2019/2020
3          186.85            A         1  2019  2019/2020
4          186.38            A         1  2019  2019/2020
```

[5]:

```
train_data.head()
```

[5]:

```
   access  tests tests_grade  exam  project project_grade  assignments  \
0    1256  57.00           A    19    91.54             A         40.0
1     985  42.87           B    19    75.96             A         13.7
2    1455  54.50           A    16    96.79             A         40.0
3     998  54.50           A    16    93.36             A         40.0
4    1347  55.00           A    16    92.86             A         39.0

   result_points result_grade  graduate  year  acad_year
0          189.92            A         1  2019  2019/2020
1          189.43            A         1  2017  2017/2018
2          188.91            A         1  2019  2019/2020
3          186.85            A         1  2019  2019/2020
4          186.38            A         1  2019  2019/2020
```

[6]:

```
X_train = np.asarray(train_data[['access', 'tests', 'assignments']])
y_train = np.asarray(train_data['graduate'])
```

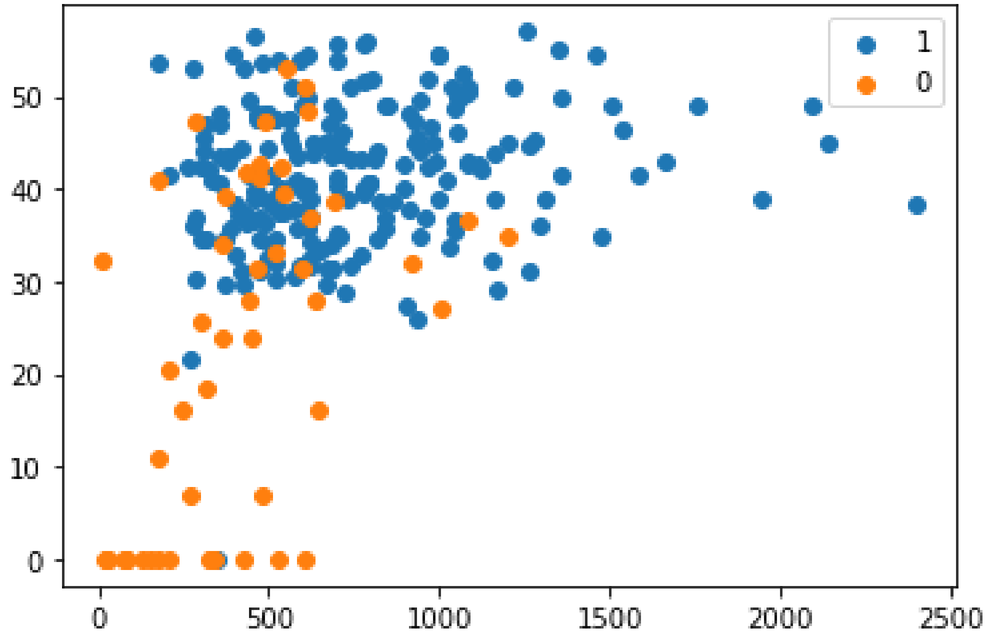
[7]:

```
X_test = np.asarray(test_data[['access', 'tests', 'assignments']])
y_test = np.asarray(test_data['graduate'])
```

[8]:

```
counter = Counter(y_train)
print(counter)

# scatter plot of examples by class label
for label, _ in counter.items():
        row_ix = where(y_train == label)[0]
        plt.scatter(X_train[row_ix, 0], X_train[row_ix, 1], label=str(label))
plt.legend()
plt.show()
```

```
Counter({1: 210, 0: 51})
```



[9]:

```
# Data normalization with sklearn
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range = (0,1))

scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[10]:

```
# Transform the dataset
oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

[11]:

```
# Modelling
from sklearn.ensemble import RandomForestRegressor

forest = RandomForestClassifier(random_state = 1,
                                  n_estimators = 1000,
                                  max_features = 'auto',
                                  max_depth = 50,
                                  bootstrap = False,
                                  min_samples_split = 2,  min_samples_leaf = 1)
model = forest.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

[12]:

```
from sklearn.metrics import mean_absolute_error
def evaluate(forest, X_test, y_test):
    predictions = model.predict(X_test)
    errors = abs(predictions - y_test)
    mape = mean_absolute_error(predictions, y_test)*100
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

[13]:

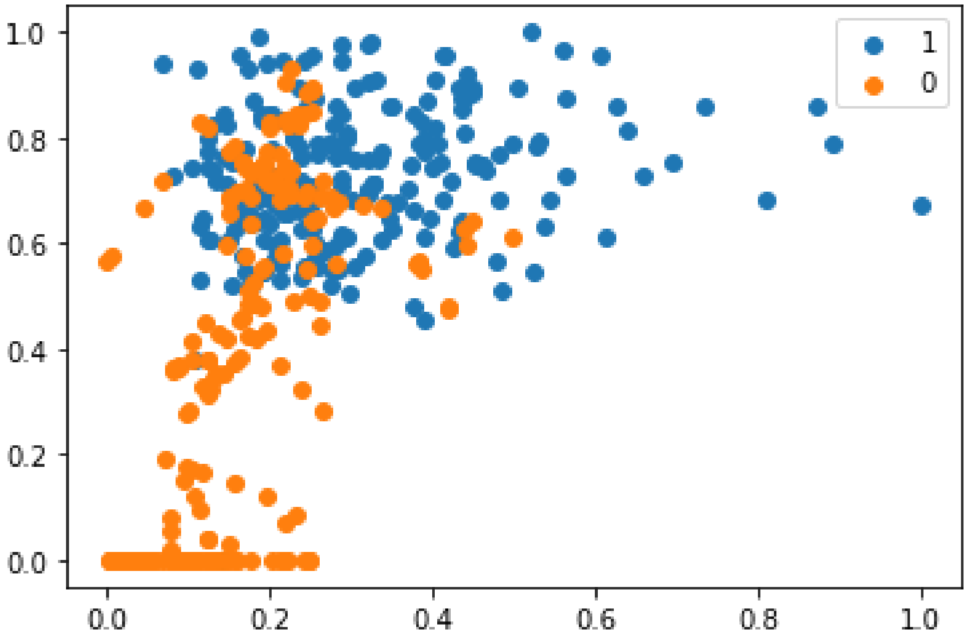
```
base_accuracy = evaluate(forest, X_test, y_test)
```

```
Model Performance
Average Error: 0.0833 degrees.
Accuracy = 91.67%.
```

[14]:

```
counter = Counter(y_train)
print(counter)
# scatter plot of examples by class label
for label, _ in counter.items():
        row_ix = where(y_train == label)[0]
        plt.scatter(X_train[row_ix, 0], X_train[row_ix, 1], label=str(label))
plt.legend()
plt.show()
```

```
Counter({1: 210, 0: 210})
```



[15]:

```python
# Make predictions for the test set
y_pred_test = forest.predict(X_test)
```

[16]:

```python
# View accuracy score
accuracy_score(y_test, y_pred_test)
```

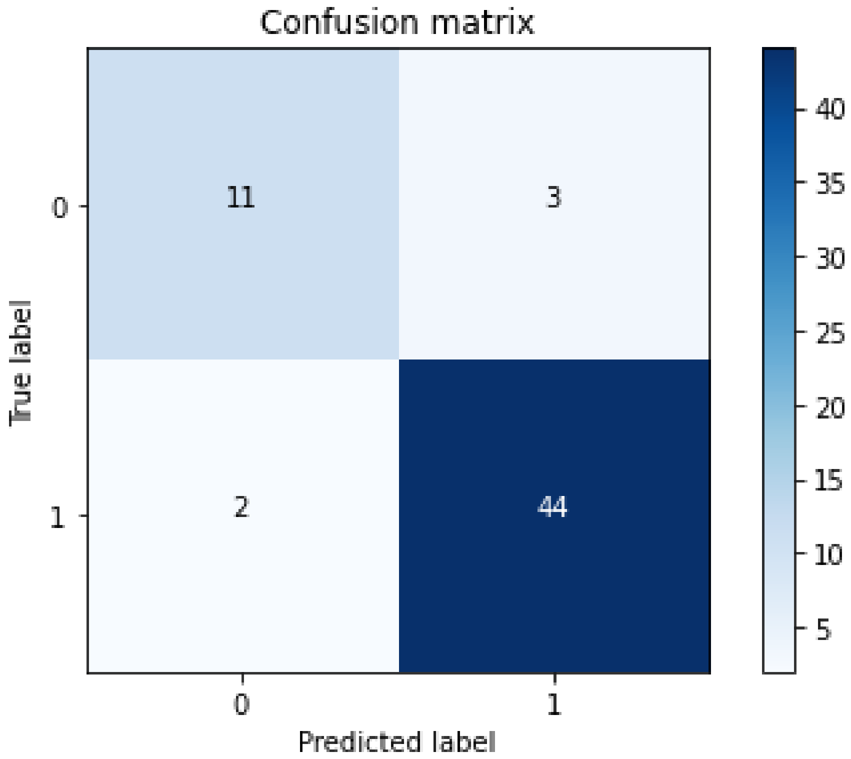
[16]:

```
0.9166666666666666
```

[17]:

```python
# Classificaton report
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')
y_pre = forest.predict(X_test)

cnf_matrix = confusion_matrix(y_test,  y_pre)

print("Recall metric in the testing dataset: {}%".format(100*cnf_matrix[1,1]/
 ↪(cnf_matrix[1,0]+cnf_matrix[1,1])))
#print("Precision metric in the testing dataset: {}%".
 ↪format(100*cnf_matrix[0,0]/(cnf_matrix[0,0]+cnf_matrix[1,0])))
# 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: 95.65217391304348%
```



```
[18]: # View confusion matrix for test data and predictions
      confusion_matrix(y_test,  y_pred_test)
```

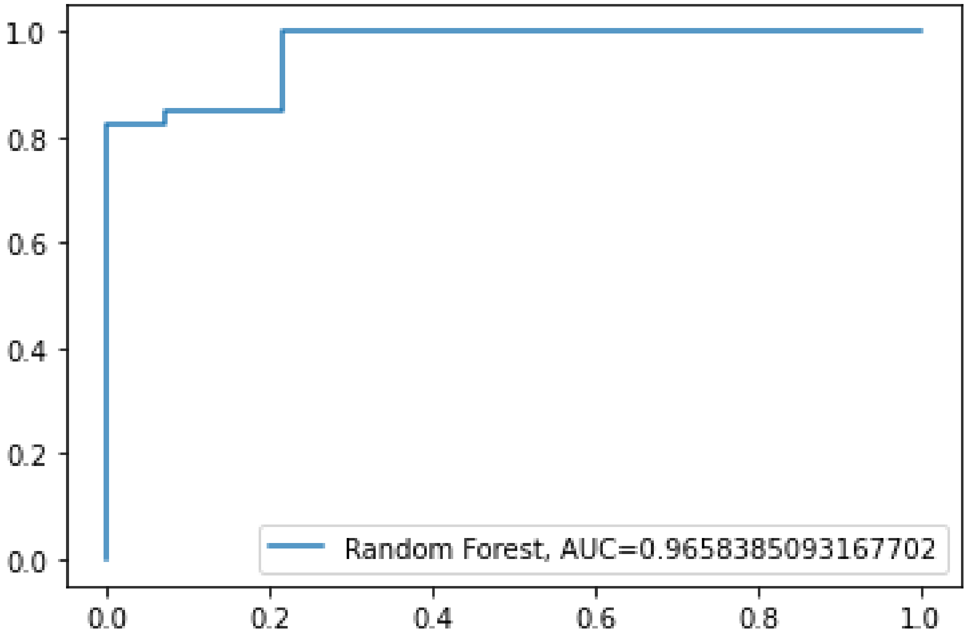
```
[18]: array([[11,  3],
             [ 2, 44]], dtype=int64)
```

```
[19]: # View the classification report for test data and predictions
      print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.81        14
           1       0.94      0.96      0.95        46

    accuracy                           0.92        60
   macro avg       0.89      0.87      0.88        60
weighted avg       0.92      0.92      0.92        60
```

```
[20]: # Import the metrics class
      from sklearn import metrics
```

```
y_pred_proba = forest.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))
plt.legend(loc=4)
plt.show()
```


Random Forest, AUC=0.9658385093167702