Week_7

June 9, 2020

1 Week 7

[1]:
```
import numpy as np
import random
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
#from sklearn.metrics import confusion_matrix
#from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression


#from sklearn.datasets import make_classification
#from sklearn.model_selection import KFold
```

2 1. Import Data and Preprocess

[3]:
```
df = pd.read_csv("GSE68086_TEP_data_matrix.txt",sep= "\t", index_col = 0)

x = df[(df.isin([0]).sum(axis=1)/285 <0.9 ) ]

X = x.T
```

[5]:
```
features = list(df.columns.values)

y = np.array([0 if ("HD-" in name or "Control" in name or "Type-Unknown-3" in␣
 ↪name) else 1 for name in features])

h_list= [name for name in features if ("HD-" in name or "Control" in name or␣
 ↪"Type-Unknown-3" in name) ]
```

3 2. Split data

[6]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=1/3,␣
↪stratify=y, random_state=0)
```

4 3. Model Training

[42]:
```
logisticRegr = LogisticRegression(max_iter =3000)
logisticRegr.fit(x_train, y_train)
```

[42]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=3000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

5 4. Model Test

[43]:
```
# Use score method to get accuracy of model
score = logisticRegr.score(x_test, y_test)
print(score)
```

```
0.9052631578947369
```

[79]:
```
y_test_pred = logisticRegr.predict(x_test)
metrics.roc_auc_score(y_test, y_test_pred)
```

[79]:
```
0.9415584415584415
```

[76]:
```
y_train_pred = logisticRegr.predict(x_train)
metrics.roc_auc_score(y_train, y_train_pred)
```

[76]:
```
1.0
```

[81]:
```
metrics.confusion_matrix(y_test, y_test_pred)
```

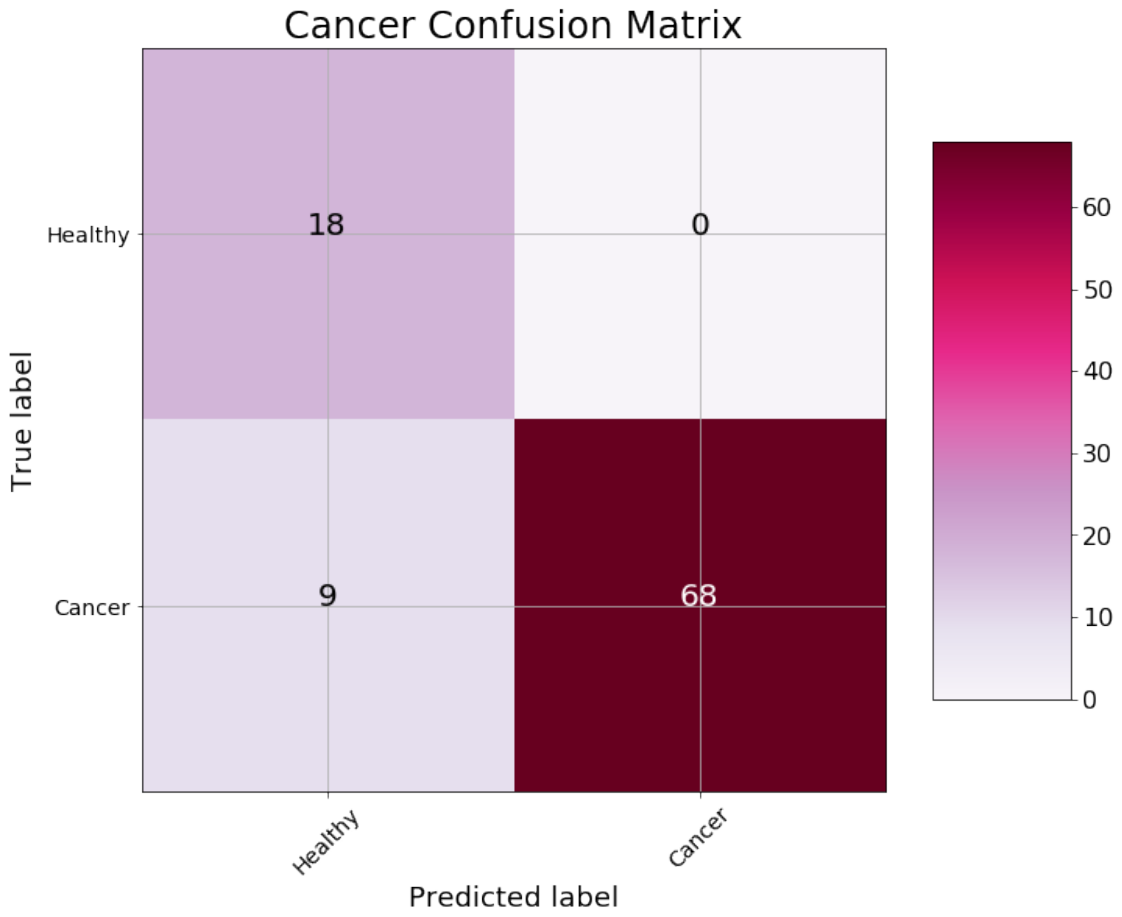
[81]:
```
array([[18,  0],
       [ 9, 68]])
```

[98]:
```
import itertools

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

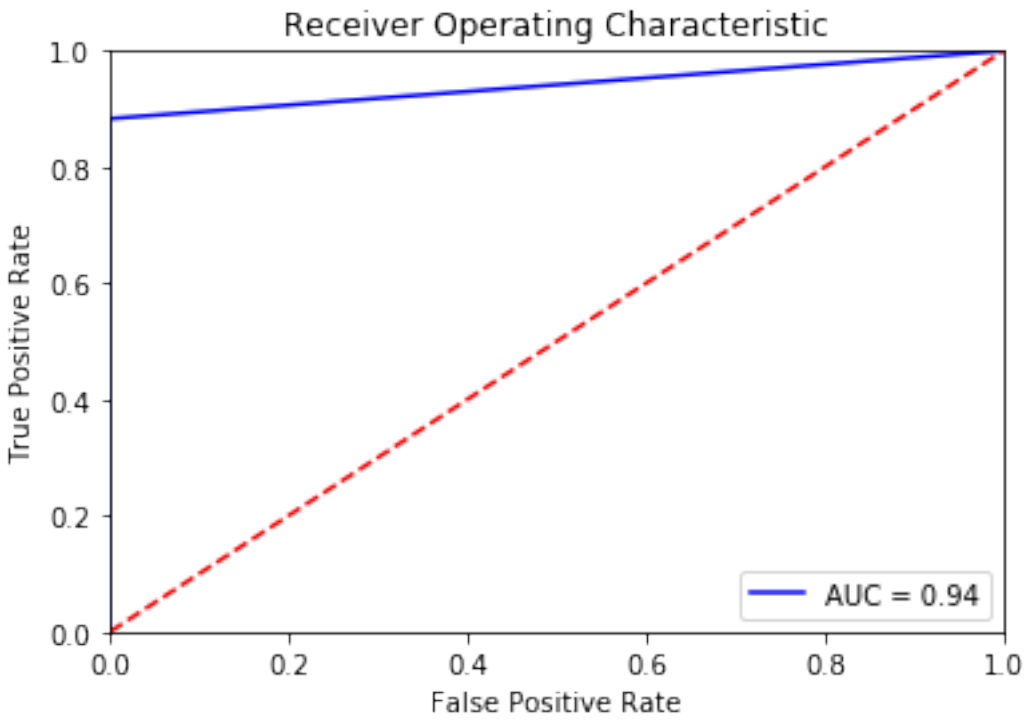
```
                          title='Confusion matrix',
                          cmap=plt.cm.PuRd):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    Source: http://scikit-learn.org/stable/auto_examples/model_selection/
↪plot_confusion_matrix.html
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    # Plot the confusion matrix
    plt.figure(figsize = (10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, size = 24)
    plt.colorbar(aspect=4)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, size = 14)
    plt.yticks(tick_marks, classes, size = 14)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    # Labeling the plot
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), fontsize = 20,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.grid(None)
    plt.tight_layout()
    plt.ylabel('True label', size = 18)
    plt.xlabel('Predicted label', size = 18)
```

[100]:
```
cm = metrics.confusion_matrix(y_test, y_test_pred)
plot_confusion_matrix(cm, classes = ['Healthy', 'Cancer'],
                      title = 'Cancer Confusion Matrix')
```

```
Confusion matrix, without normalization
```



[83]:
```
import scikitplot as skplt

fpr, tpr, threshold = metrics.roc_curve(y_test, predictions)
roc_auc = metrics.auc(fpr, tpr)
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])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



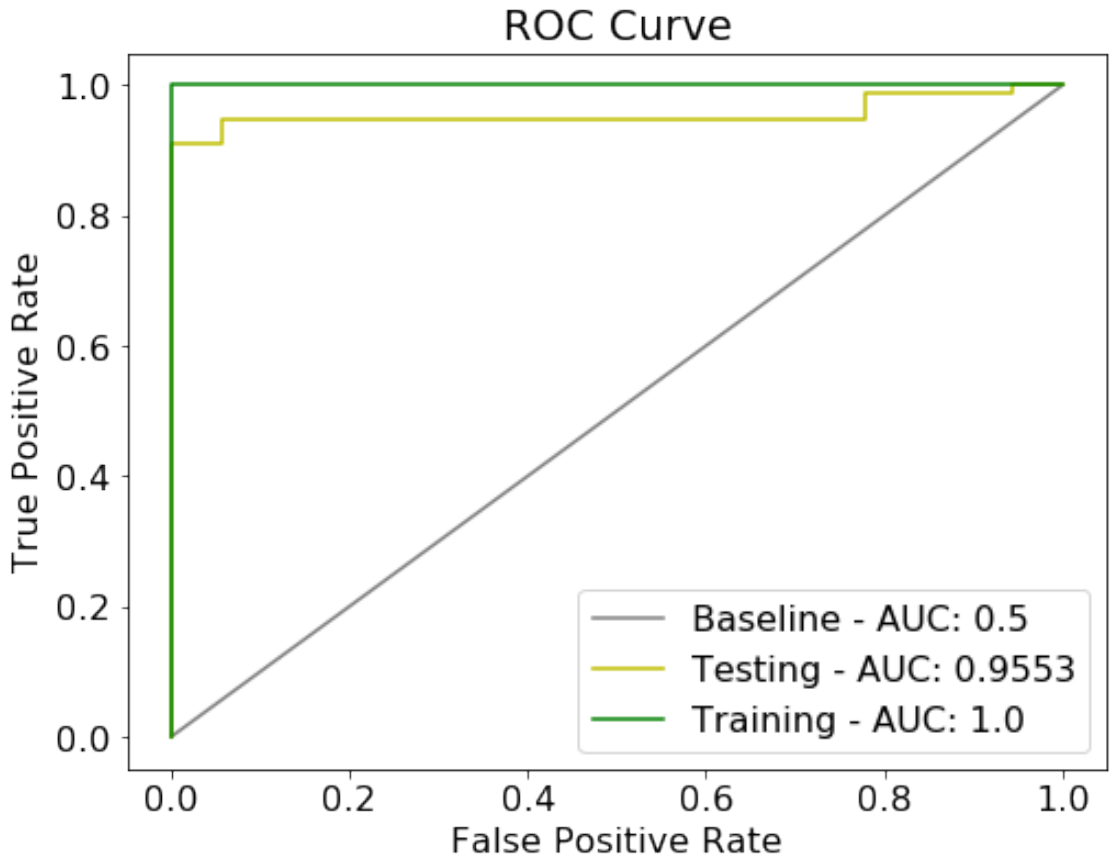
[94]:
```
model = logisticRegr
# Training predictions (to demonstrate overfitting)
train_rf_predictions = model.predict(x_train)
train_rf_probs = model.predict_proba(x_train)[:, 1]

# Testing predictions (to determine performance)
rf_predictions = model.predict(x_test)
rf_probs = model.predict_proba(x_test)[:, 1]
```

[95]:
```
from sklearn.metrics import precision_score, recall_score, roc_auc_score,␣
↪roc_curve, confusion_matrix
from tabulate import tabulate


def evaluate_model(predictions, probs, train_predictions, train_probs,␣
↪train_labels, test_labels):
    """Compare machine learning model to baseline performance.
    Computes statistics and shows ROC curve."""

    baseline = {}

    baseline['recall'] = recall_score(test_labels,
                                      [1 for _ in range(len(test_labels))])
```

```
    baseline['precision'] = precision_score(test_labels,
                                            [1 for _ in range(len(test_labels))])
    baseline['roc'] = 0.5

    results = {}

    results['recall'] = recall_score(test_labels, predictions)
    results['precision'] = precision_score(test_labels, predictions)
    results['roc'] = roc_auc_score(test_labels, probs)

    train_results = {}
    train_results['recall'] = recall_score(train_labels, train_predictions)
   train_results['precision'] = precision_score(train_labels, train_predictions)
    train_results['roc'] = roc_auc_score(train_labels, train_probs)


    metrics = []
    for metric in ['recall', 'precision', 'roc']:
        metrics.append((metric,round(baseline[metric], 4),␣
↪round(train_results[metric], 4), round(results[metric], 4)))
    print(tabulate(metrics, headers=['Metric', 'Baseline', 'Train', 'Test']))

    # Calculate false positive rates and true positive rates
    base_fpr, base_tpr, _ = roc_curve(test_labels, [1 for _ in␣
↪range(len(test_labels))])
    testing_fpr, testing_tpr, _ = roc_curve(test_labels, probs)
    training_fpr, training_tpr, _ = roc_curve(train_labels, train_probs)

    plt.figure(figsize = (8, 6))
    plt.rcParams['font.size'] = 16

    baseline_label = 'Baseline - AUC: %s' %(round(baseline['roc'], 4))
    testing_label = 'Testing - AUC: %s' %(round(results['roc'], 4))
    training_label = 'Training - AUC: %s' %(round(train_results['roc'], 4))
    # Plot both curves
    plt.plot(base_fpr, base_tpr, 'grey', label = baseline_label)
    plt.plot(testing_fpr, testing_tpr, 'y', label = testing_label)
    plt.plot(training_fpr, training_tpr, 'g', label = training_label)
    plt.legend();
    plt.xlabel('False Positive Rate');
    plt.ylabel('True Positive Rate');
    plt.title('ROC Curve');
    plt.show();
evaluate_model(rf_predictions, rf_probs, train_rf_predictions, train_rf_probs,␣
↪y_train, y_test)
```

```
Metric       Baseline    Train    Test
```

```
---------  ----------  -------  ------
recall          1             1  0.8831
precision       0.8105        1  1
roc             0.5           1  0.9553
```



6 5. Feature importance

[61]:
```
import matplotlib.pyplot as plt


coefs = np.abs(logisticRegr.coef_[0])
indices = np.argsort(coefs)[::-1]

features = list(X.columns.values)
imp_feat =  [features[i] for i in indices[:18]]

plt.figure()
plt.title("Feature importances (Logistic Regression)")
```

```
plt.bar(range(18), coefs[indices[:18]],
       color="r", align="center")
plt.xticks(range(18),imp_feat, rotation=45, ha='right')
plt.subplots_adjust(bottom=0.3)
```