Performance Measurement – Unit 11 – eportfolio

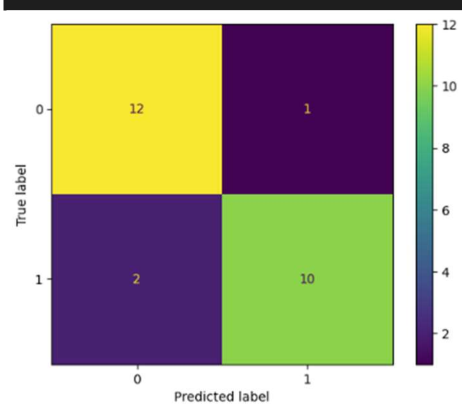
```
from sklearn.metrics import confusion_matrix
#imorting the confusion matrix library to evaluate qaccuracy of
classification
```

```
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
(tn, fp, fn, tp)
```

Giving labels to true negatives, false positives, false negatives and true positives

Here the data is being split into training and testing sets, where the SVC model is being trained, the trained model is then used to make predictions on the test set. The results are then put into a plot with binary classification. True positive was predicted correctly 10 times, true negative 12 times, false positive was incorrectly predicted 1 time and false negative 2 times.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
X, y = make_classification(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=0)
clf = SVC(random_state=0)
clf.fit(X_train, y_train)
SVC(random_state=0)
predictions = clf.predict(X_test)
cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
disp.plot()

plt.show()
```



F1, Accuracy, Rexall and Precision Scores

```
from sklearn.metrics import f1_score

y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]

print(f"Macro f1 score: {f1_score(y_true, y_pred, average='macro')}")

print(f"Micro F1: {f1_score(y_true, y_pred, average='micro')}")

print(f"Weighted Average F1: {f1_score(y_true, y_pred,
average='weighted')}")

print(f"F1 No Average: {f1_score(y_true, y_pred, average=None)}")

y_true = [0, 0, 0, 0, 0, 0]
y_pred = [0, 0, 0, 0, 0, 0]
f1_score(y_true, y_pred, zero_division=1)

# multilabel classification
y_true = [[0, 0, 0], [1, 1, 1], [0, 1, 1]]
y_pred = [[0, 0, 0], [1, 1, 1], [1, 1, 0]]
print(f"F1 No Average: {f1_score(y_true, y_pred, average=None)}")
```

```
Macro f1 score: 0.26666666666666666
Micro F1: 0.3333333333333333
Weighted Average F1: 0.26666666666666666
F1 No Average: [0.8 0.  0. ]
F1 No Average: [0.66666667 1.         0.66666667]
```

0 & 0 are correct, 2 & 1 are incorrect, 1 & 2 are incorrect, 3 & 3 are correct. This gives us an accuracy of 50%. The same methods are used for the future calculations of precision score and recall score.

```
 from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
```

```
0.5
```

```
from sklearn.metrics import precision_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
precision_score(y_true, y_pred, average='macro')
```

```
0.2222222222222222
```

```
from sklearn.metrics import recall_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
recall_score(y_true, y_pred, average='macro')
```

```
0.3333333333333333
```

```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

    accuracy                           0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

The ROC AUC score is 0.994 to 3 decimal places which shows this model can distinguish between positive and negative classes from the dataset.

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
X, y = load_breast_cancer(return_X_y=True)
clf = LogisticRegression(solver="liblinear", random_state=0).fit(X, y)
roc_auc_score(y, clf.predict_proba(X)[:, 1])
```

```
0.994767718408118
```

There is a high AUC score of 0.991 to 3 decimal places here which shows that the regression model can distinguish between different classes within the dataset.

```
#multiclass case
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
clf = LogisticRegression(solver="liblinear").fit(X, y)
roc_auc_score(y, clf.predict_proba(X), multi_class='ovr')
```

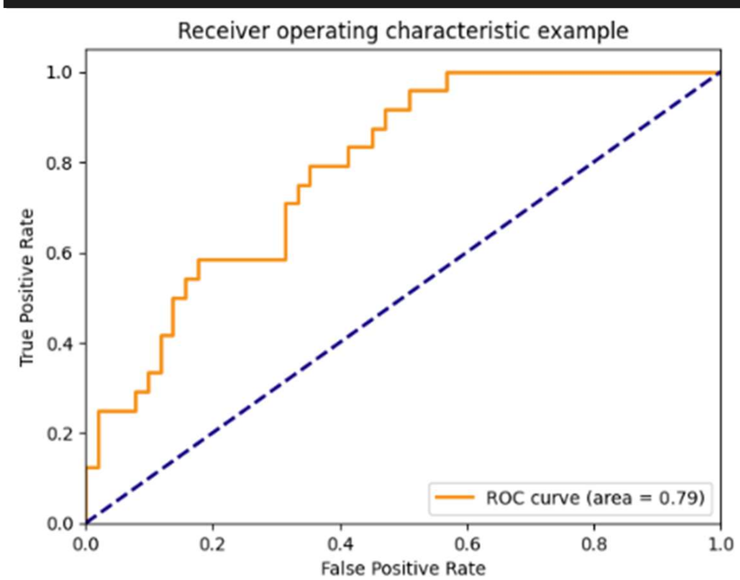
```
0.9913333333333334
```

```
import numpy as np
```

```python
import matplotlib.pyplot as plt
from itertools import cycle

from sklearn import svm, datasets
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_auc_score

# Import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Binarize the output
y = label_binarize(y, classes=[0, 1, 2])
n_classes = y.shape[1]

# Add noisy features to make the problem harder
random_state = np.random.RandomState(0)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

# shuffle and split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=0)

# Learn to predict each class against the other
classifier = OneVsRestClassifier(
    svm.SVC(kernel="linear", probability=True, random_state=random_state)
)
y_score = classifier.fit(X_train, y_train).decision_function(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

The ROC curve is displayed in orange and the blue line represents the random classifier.

```
plt.figure()
lw = 2
plt.plot(
    fpr[2],
    tpr[2],
    color="darkorange",
    lw=lw,
    label="ROC curve (area = %0.2f)" % roc_auc[2],
)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```



A log loss function with the result of 0.216 to 3 decimal places. The predicted probabilities match the true labels.

```
from sklearn.metrics import log_loss
```

```
log_loss(["spam", "ham", "ham", "spam"], [[.1, .9], [.9, .1], [.8, .2],
[.35, .65]])
```

```
0.21616187468057912
```

Regression Metrics

RMSE

A mean squared error pf 0.375

```
from sklearn.metrics import mean_squared_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

```
0.375
```

A mean absolute error of 0.5

```
from sklearn.metrics import mean_absolute_error
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
```

```
0.5
```

A score of 0.9486 shows a high level of predictive accuracy

```
from sklearn.metrics import r2_score

r2_score(y_true, y_pred)
```

```
0.9486081370449679
```

If we were to change the figures to have them be less of a match like the below we would receive a low level of predictive accuracy.

```
from sklearn.metrics import mean_squared_error
y_true = [9, -0.8, 5, 1]
y_pred = [2.5, 0.0, 2, 8]
mean_squared_error(y_true, y_pred)
```

```
25.2225
```

```
from sklearn.metrics import mean_absolute_error
y_true = [9, -0.8, 5, 1]
```

```
y_pred = [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
```

```
4.325
```

```
from sklearn.metrics import r2_score

r2_score(y_true, y_pred)
```

```
-0.7628865979381445
```