## Understanding Metrics of Classification

Importing libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import itertools

np.random.seed(42)  # for reproducibility
sns.set(rc={"figure.figsize": (8, 8)})
sns.set_style("ticks")
```

Loading BreastCancer Dataset from sklearn

```
data = load_breast_cancer()
print(data.DESCR[:760])  # print short description
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline appr
```

see the targets classes

```
print(f"Types of cancer (targets) are {data.target_names}")
```

```
Types of cancer (targets) are ['malignant' 'benign']
```

Print the target and the features

```
X = data.data  # features
y = data.target  # labels
print(f"Shape of features is {X.shape}, and shape of target is {y.shape}")
```

```
Shape of features is (569, 30), and shape of target is (569,)
```

Split the data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=200, random_state=42, stratify=y)
```

```
y_train[:10]
```

```
array([1, 1, 1, 1, 0, 1, 1, 1, 0, 1])
```

Training and predicting data

```
classifier = svm.SVC(kernel='linear', probability=True, verbose=True)
```

fit/train the model on our training dataset

```
classifier.fit(X_train, y_train)
```

```
[LibSVM]
SVC
SVC(kernel='linear', probability=True, verbose=True)
```

```
y_preds = classifier.predict(X_test)
y_proba = classifier.predict_proba(X_test)
```

reshaping y_proba to a 1D vector denoting the probability of having benign cancer.

```
y_proba = y_proba[:,1].reshape((y_proba.shape[0],))
```

```
y_proba[:5], y_preds[:5], y_test[:5]
```

```
(array([0.99697262, 0.08136644, 0.99999168, 0.96809392, 0.99999907]),
 array([1, 0, 1, 1, 1]),
 array([1, 0, 1, 1, 1]))
```

Confusion Matrix

```
conf = metrics.confusion_matrix(y_test, y_preds)
conf
```

```
array([[ 68,   7],
       [  2, 123]])
```

implementing on our own confusion matrix

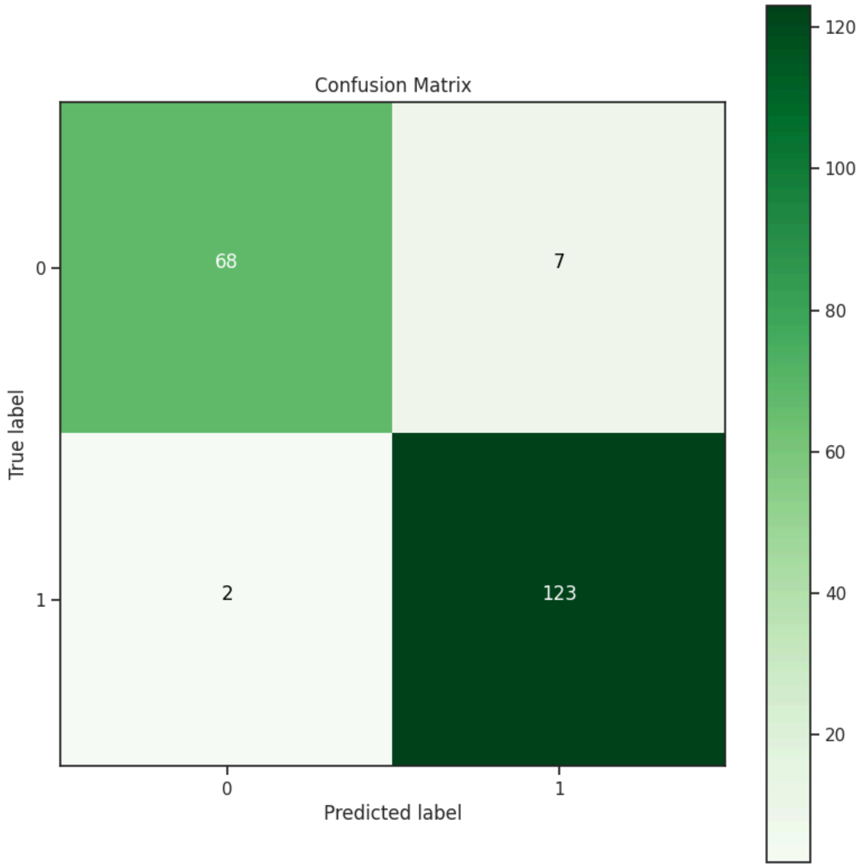
```
def get_confusion_matrix(y_true, y_pred):
    n_classes = len(np.unique(y_true))
    conf = np.zeros((n_classes, n_classes))
    for actual, pred in zip(y_true, y_pred):
        conf[int(actual)][int(pred)] += 1
    return conf.astype('int')
```

```
conf = get_confusion_matrix(y_test, y_preds)
conf
```

```
array([[ 68,   7],
       [  2, 123]])
```

```
classes = [0, 1]
# plot confusion matrix
plt.imshow(conf, interpolation='nearest', cmap=plt.cm.Greens)
plt.title("Confusion Matrix")
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)

fmt = 'd'
thresh = conf.max() / 2.
for i, j in itertools.product(range(conf.shape[0]), range(conf.shape[1])):
    plt.text(j, i, format(conf[i, j], fmt),
             horizontalalignment="center",
             color="white" if conf[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 114.24999999999993, 'Predicted label')
```




Results from Confusion Matrix

```
# from the confusion matrix
TP = true_pos = 123
TN = true_neg = 68
FP = false_pos = 7
FN = false_neg = 2
```

Some basic metrics

creating a dictionary results

```
results = {}
```

Accuracy

```
metric = "ACC"
results[metric] = (TP + TN) / (TP + TN + FP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
ACC is  0.955
```

True Positive Rate

```
# Sensitivity or Recall
metric = "TPR"
results[metric] = TP / (TP + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
TPR is  0.984
```

True Negative Rate

```
# Specificity
metric = "TNR"
results[metric] = TN / (TN + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
TNR is  0.907
```

Positive Predictive Value

```
# Precision
metric = "PPV"
results[metric] = TP / (TP + FP)
print(f"{metric} is {results[metric]: .3f}")
```

```
PPV is  0.946
```

Negative Predictive Value

```
metric = "NPV"
results[metric] = TN / (TN + FN)
print(f"{metric} is {results[metric]: .3f}")
```

```
NPV is  0.971
```

F1 score

```
metric = "F1"
results[metric] = 2 / (1 / results["PPV"] + 1 / results["TPR"])
print(f"{metric} is {results[metric]: .3f}")
```

```
F1 is  0.965
```

Matthew's correlation coefficient

```
metric = "MCC"
num = TP * TN - FP * FN
den = ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)) ** 0.5
results[metric] = num / den
print(f"{metric} is {results[metric]: .3f}")
```

```
MCC is  0.904
```

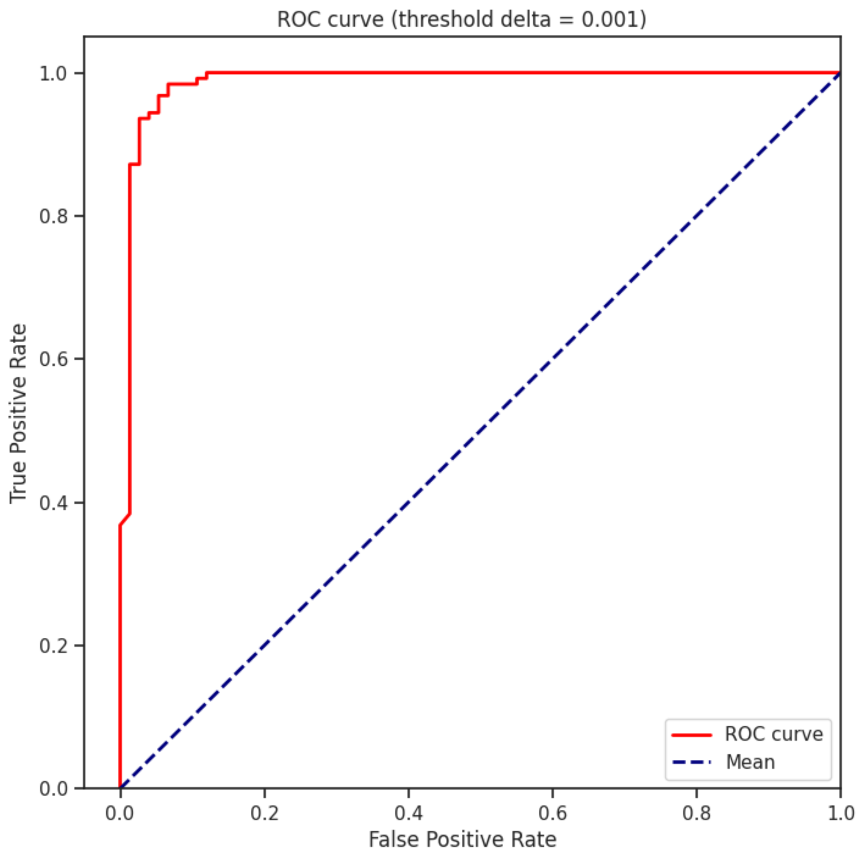
Comparing these calculated metrics

```
print(f"Calculated and Actual Accuracy:                          {results['ACC']: .3f}, {metrics.accuracy_score(y_test, y_preds): .3f}")
print(f"Calculated and Actual Precision score:                   {results['PPV']: .3f}, {metrics.precision_score(y_test, y_preds): .3f}"
print(f"Calculated and Actual Recall score:                      {results['TPR']: .3f}, {metrics.recall_score(y_test, y_preds): .3f}")
print(f"Calculated and Actual F1 score:                          {results['F1']: .3f}, {metrics.f1_score(y_test, y_preds): .3f}")
print(f"Calculated and Actual Matthew's correlation coefficient: {results['MCC']: .3f}, {metrics.matthews_corrcoef(y_test, y_preds): .3
```

```
Calculated and Actual Accuracy:                          0.955,  0.955
Calculated and Actual Precision score:                   0.946,  0.946
Calculated and Actual Recall score:                      0.984,  0.984
Calculated and Actual F1 score:                          0.965,  0.965
Calculated and Actual Matthew's correlation coefficient:  0.904,  0.904
```

ROC curve (Receiver Operating Characteristic curve)

```
def get_roc_curve(y_test, y_proba, delta=0.1):
    thresh = list(np.arange(0, 1, delta)) + [1]
    TPRs = []
    FPRs = []
    y_pred = np.empty(y_proba.shape)
    for th in thresh:
        y_pred[y_proba < th] = 0
        y_pred[y_proba >= th] = 1

        # confusion matrix from the function we defined
        (TN, FP), (FN, TP) = get_confusion_matrix(y_test, y_pred)

        TPR = TP / (TP + FN)  # sensitivity
        FPR = FP / (FP + TN)  # 1 - specificity
        TPRs.append(TPR)
        FPRs.append(FPR)
    return FPRs, TPRs, thresh


delta = 0.001
FPRs, TPRs, _ = get_roc_curve(y_test, y_proba, delta)


# Plot the ROC curve
plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC curve (threshold delta = {delta})')
plt.legend(loc="lower right")
plt.show()
```

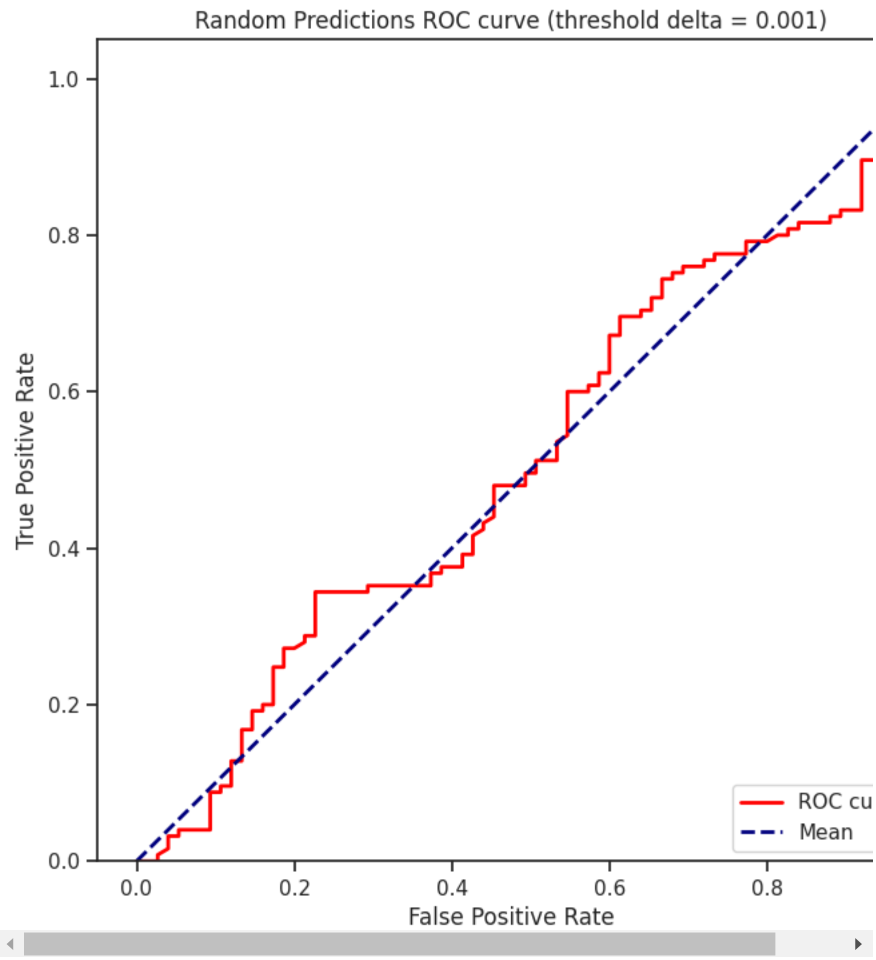



calculate an ROC curve with random predictions

```
# create random predictions
rand_proba = np.random.random(size=(y_proba.shape))
rand_proba[:5]  # 0.5 probability of being 0 or 1
```

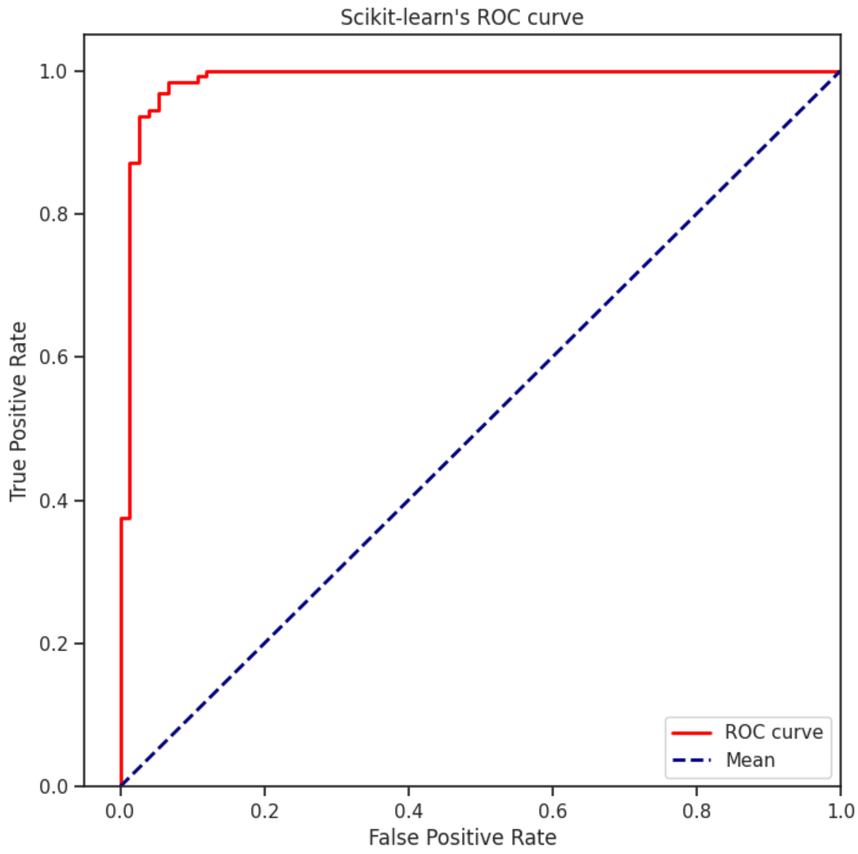
```
array([0.79654299, 0.18343479, 0.779691  , 0.59685016, 0.44583275])
```

```
FPRs, TPRs, _ = get_roc_curve(y_test, rand_proba, delta)  # passing random preds


# Plot the ROC curve
plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Random Predictions ROC curve (threshold delta = {delta})')
plt.legend(loc="lower right")
plt.show()
```

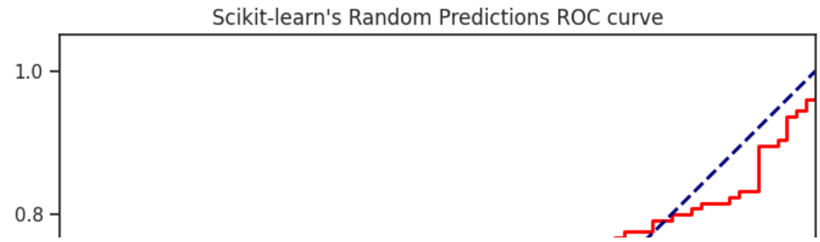



As we can see from the above plot, random predictions give the ROC curve nearly at the mean.

```
FPRs, TPRs, _ = metrics.roc_curve(y_test, y_proba)


# Plot the ROC curve
plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("Scikit-learn's ROC curve")
plt.legend(loc="lower right")
plt.show()
```




```
FPRs, TPRs, _ = metrics.roc_curve(y_test, rand_proba)  # passing random preds


# Plot the ROC curve
plt.plot(FPRs, TPRs, color='red',
         lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Mean")
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("Scikit-learn's Random Predictions ROC curve")
plt.legend(loc="lower right")
plt.show()
```




## ROC-AUC score

Lets check the AUC score of our model.

```
auc_score = metrics.roc_auc_score(y_test, y_proba)
print(f"Scikit's ROC-AUC score of SVC model is {auc_score: .4f}")
```

```
Scikit's ROC-AUC score of SVC model is  0.9872
```

We can also calculate the ROC-AUC score by summing up the areas under each observation of FPRs and TPRs.

```
def get_roc_auc_score(y_test, y_proba):

    # use the function get_roc_curve that we created.
    FPRs, TPRs, _ = get_roc_curve(y_test, y_proba)
    FPRs.reverse()
    TPRs.reverse()
    x1, y1 = FPRs[0], TPRs[0]
    auc = 0.0
    prev = 0.0
    # cumulative differences in x-axis
    diffs = [FPRs[i] - FPRs[i-1] for i in range(1, len(FPRs))]
    for x, y in zip(diffs, TPRs[1:]):
        auc += (x * y1)  # area of rectangle
        auc += (x * (y - y1)/2)  # area of triangle formed (if any)
        y1 = y
    return auc


auc_score = get_roc_auc_score(y_test, y_proba)
print(f"Our ROC-AUC score of SVC model is {auc_score: .4f}")
```

```
Our ROC-AUC score of SVC model is  0.9874
```

This is a good ROC-AUC score as we expected. (Also pretty close to Scikit's implementation). Lets try the ROC-AUC score of random predictions