# Digital Assignment III
# Classification Evaluation Metrics

Ishita Jaju
16BCE1059

### Review of model evaluation

Model chosen: Logistic Regression

## Procedure

### Train/test split

***Divide into train-test-split***

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 42)
```

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0, solver='lbfgs').fit(X_train, y_train)
```

### Model evaluation metric: Classification accuracy

The classification accuracy is the percentage of correct predictions. So for our test set, which we trained with the training set after the train-test split, the accuracy of the prediction can be found.

***Classification accuracy***

```
from sklearn.metrics import accuracy_score
y_pred = clf.predict(X_test)
accuracy_score(y_pred,y_test)
```

```
0.9791666666666666
```

Here, the accuracy is 97.91%.

### Examining the test set

Distribution of the test set: how many 0, how many 1:

```
y_test.value_counts()
```

```
0    35
1    13
Name: classification, dtype: int64
```

```
y_test.mean()
```

```
0.2708333333333333
```

```
# print the first 25 true and predicted responses
print('True:', y_test.values[0:25])
print('False:', y_pred[0:25])
```

```
True: [0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 1 1 1]
False: [0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 1 1]
```

### Confusion matrix :

***Confusion matrix***

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[35  0]
 [ 1 12]]
```

TP: 35 TN: 12 FP: 1 FN: 0

The confusion matrix gives us the number of true positives (correctly classified +ve values), true negatives (correctly classified -ve values), false positives (incorrectly classified +ve values), false negatives (incorrectly classified -ve values). This method is helpful in describing the performance of a classification model.

**Confusion matrix metrics** There are many metrics that can be computed with the four values obtained from the confusion matrix. Some of these are:

### Accuracy score

```
print((TP + TN) / float(TP + TN + FP + FN))
print(accuracy_score(y_test, y_pred))
```

```
0.9791666666666666
0.9791666666666666
```

**Classification error** "Frequency of the classifier being incorrect"

```
classification_error = (FP + FN) / float(TP + TN + FP + FN)

print(classification_error)
print(1 - accuracy_score(y_test, y_pred))
```

```
0.020833333333333332
0.02083333333333337
```

**Sensitivity** "Frequency of the prediction being correct when the actual value is positive"

Sensitivity:

```
from sklearn.metrics import recall_score
sensitivity = TP / float(FN + TP)
print(sensitivity)
print(recall_score(y_test, y_pred))
```

```
1.0
1.0
```

**Specificity** "Frequency of the prediction being correct when the actual value is negative"

Specificity:

```
specificity = TN / (TN + FP)

print(specificity)
```

```
0.9230769230769231
```

**Precision** "Frequency of the prediction being correct when the predicted value is positive"

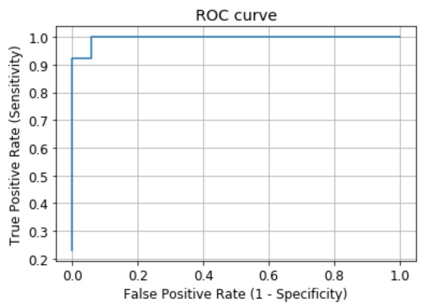
Precision:

```
from sklearn.metrics import precision_score
precision = TP / float(TP + FP)

print(precision)
print(precision_score(y_test, y_pred))
```

```
0.9722222222222222
0.9722222222222222
```

**ROC Curves** This plot shows us how sensitivity and specificity are affected by various thresholds, without actually changing the threshold

```python
from sklearn import metrics
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)

plt.plot(fpr, tpr)
plt.rcParams['font.size'] = 12
plt.title('ROC curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```




**AUC** Now let us see the percentage of the ROC plot that is underneath the curve

***AUC***

```python
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_pred_prob))
```

```
0.9956043956043956
```

Cross validated AUC:

```python
from sklearn.cross_validation import cross_val_score
cross_val_score(clf, X_test, y_test, cv=10, scoring='roc_auc').mean()
```

```
1.0
```

The main advantages of the ROC/AUC are that they do not require a set classification threshold, and it is still useful when there is high class imbalance.