MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Machine Learning

Lecture 7: Evaluation of results

Classification outcomes

- A classifier is a function mapping the d-dimensional feature space to the label space comprising k labels

$$f: \mathbb{R}^d \rightarrow \{1, \ldots, k\}$$

- Given n samples of the function's input and output

$$(X, t) \in \mathbb{R}^{n \times d} \times \{1, \ldots, k\}^n$$

- A sample is correctly classified, if

$$f[x_i] = t_i$$

- A sample is incorrectly classified, if

$$f[x_i] \neq t_i$$

Loss functions

- To quantify the impact of a misclassification a (domain specific) loss function is introduced
- For example the 0/1 loss function could be simply

$$\mathrm{L}[s,t] = \begin{cases} 0 & if\ s = t \\ 1 & if\ s \neq t \end{cases}$$

- Other loss functions are necessary, in particular if mis-classification has asymmetric impact
- Example:
  - Being wrongly diagnosed with cancer leads to more follow-up examinations
  - Being wrongly diagnosed as cancer-free leads to not treating the condition adequately

Loss functions

- In the Boolean case we name the four possible outcomes of a classification as follows

  - **True positives** ($f[x_i] = true \wedge t_i = true$)
    In case the function correctly predicts a positive result

  - **True negatives** ($f[x_i] = false \wedge t_i = false$)
    In case the function correctly predicts a negative result

  - **False positives** ($f[x_i] = true \wedge t_i = false$)
    In case the function incorrectly predicts a positive result despite the actual value is negative

  - **False negatives** ($f[x_i] = false \wedge t_i = true$)
    In case the function incorrectly predicts a negative result despite the actual value is positive

Classification error

- The classification error is defined via the loss function L as

$$\Omega = \frac{1}{n}\sum_{i} L[f[x_i], t_i]$$

- In case of the 0/1-loss function this simply counts the number of mis-classifications

- Every Machine Learning algorithms is a minimisation problem of the classification error Ω

Confusion matrix

- We can count the different types of mis-classifications separately

- This can be summarise in a $k \times k$ **confusion matrix** defined as

$$\mathrm{C} = \left[\sum_i 1_{f[x_i]=q \wedge t_i=r} \right]_{q=1..k, r=1..k}$$

- Using the indicator function

$$1_b = \begin{cases} 1 & if\ b\ is\ true \\ 0 & otherwise \end{cases}$$

- Intuition: We are counting the number of times a classifier predicts a label q while the actual value is supposed to be r for all samples in a given dataset
- Diagonal elements are correct classifications, while off-diagonal elements are the number of incorrect classifications

Example: confusion matrix

- Iris dataset (3 labels: 0=Iris-Setosa, 1=Iris-Versicolour, 2=Iris-Virginica)

```
clf.fit(train_features, train_labels)
predicted_labels = clf.predict(test_features)

print(test_labels)
print(predicted_labels)
print(metrics.confusion_matrix(test_labels, predicted_labels))
```

sklearn.metrics has a function for computing the confusion matrix

```
[0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2 0]
[0 2 0 0 2 0 2 1 1 1 2 2 1 2 0 1 2 2 0 1 1 2 1 0 0 0 2 1 2 0]
[[10  0  0]
 [ 0  9  2]
 [ 0  0  9]]
```

10x the class 0 was predicted as 0

9x the class 1 was predicted as 1

9x the class 2 was predicted as 2

2x the class 1 was predicted as 2

Example: confusion matrix

- Breast cancer dataset (0=benign (negative), 1=malignant (positive))

```
clf.fit(train_features, train_labels)
predicted_labels = clf.predict(test_features)

print(test_labels)
print(predicted_labels)
print(metrics.confusion_matrix(test_labels, predicted_labels))
```

Same code on the breast cancer dataset (with different classifier)

```
[0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0
 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1
 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0
 0 0]
[0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0
 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1
 0 1 0 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0
 0 0]
[[68  2]
 [ 1 42]]
```

1 false negatives, where a negative test result occurs for a patient with cancer

2 false positives, where a positive test result occurs for a healthy patient

Receiver-Operating-Characteristic (ROC)

- False positives and false negatives are inherently linked

- A trivial classifier that always predicts a positive output, i.e.
$$f[x] = true$$
regardless of the feature vector x, has by definition a false negative rate of 0 (note that this means it has a true positive rate of 100%)
- Similar, a trivial classifier always predicting a negative output, i.e.
$$f[x] = false$$
regardless of the feature vector x, has by definition a false positive rate of 0 (note that this means it has a true negative rate of 100%)

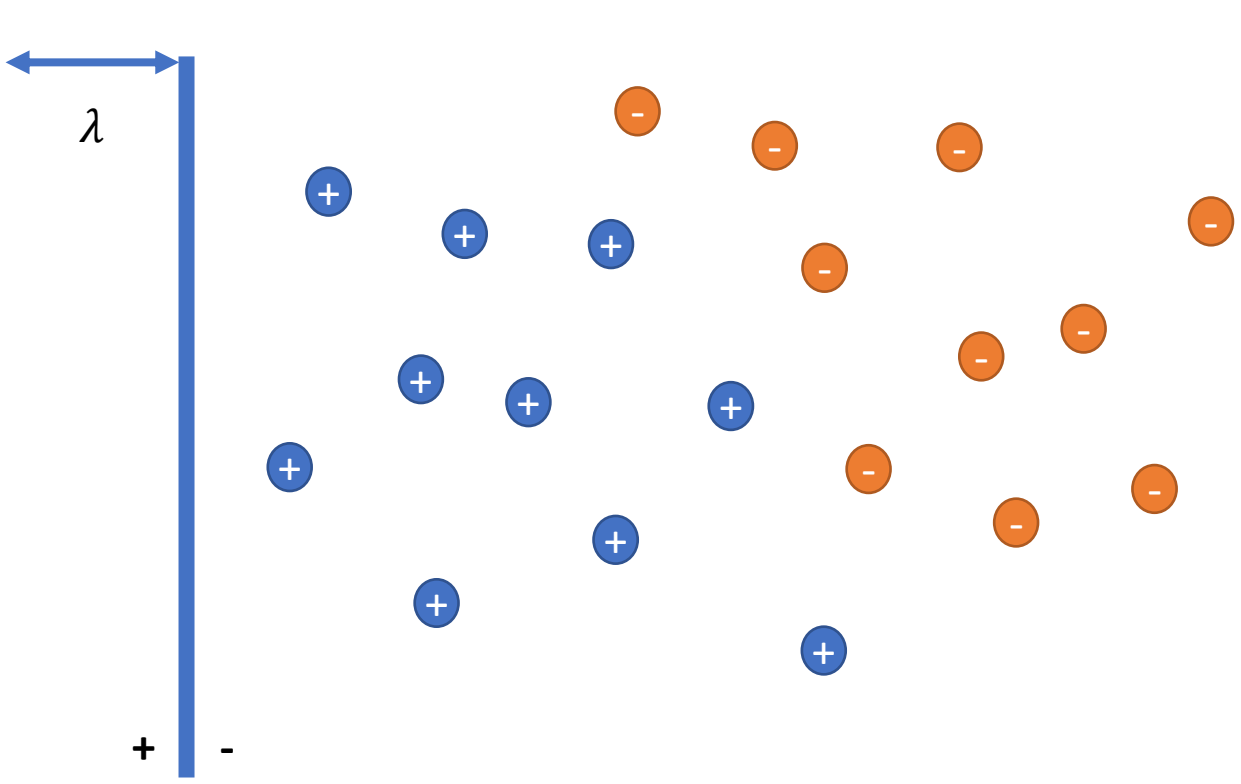
- Therefore it is easy to construct a classifier minimising either one quantity

- Typically we are interested in a solution that performs well on both indicators

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”

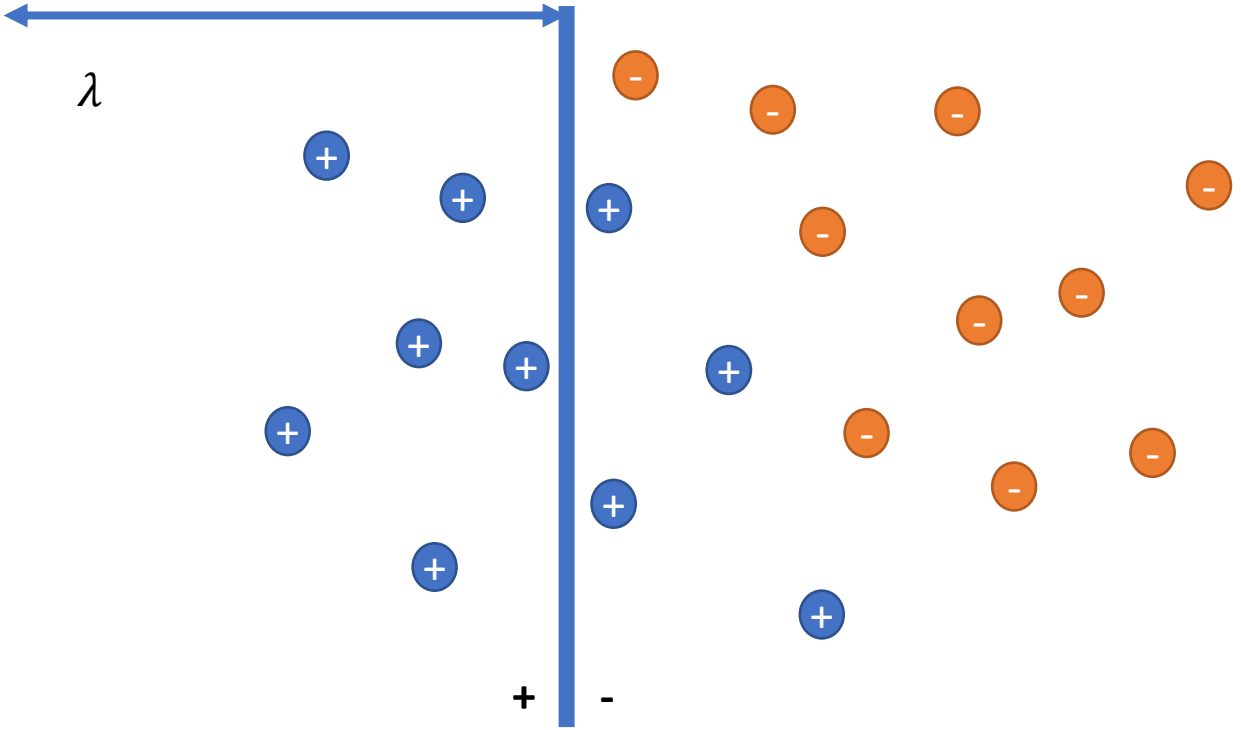


False positives: 0
True positives: 0

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”



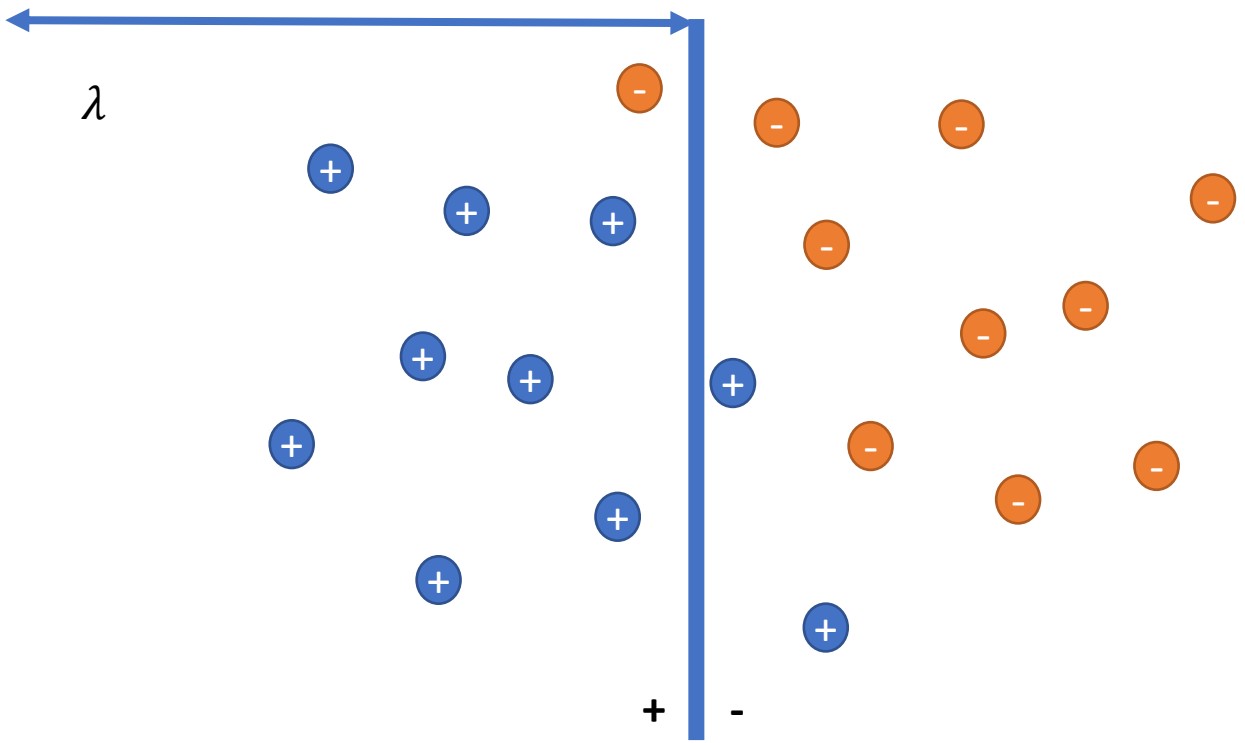
False positives: 0
True positives: 0

False positives: 0
True positives: 6

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”



False positives: 0
True positives: 0

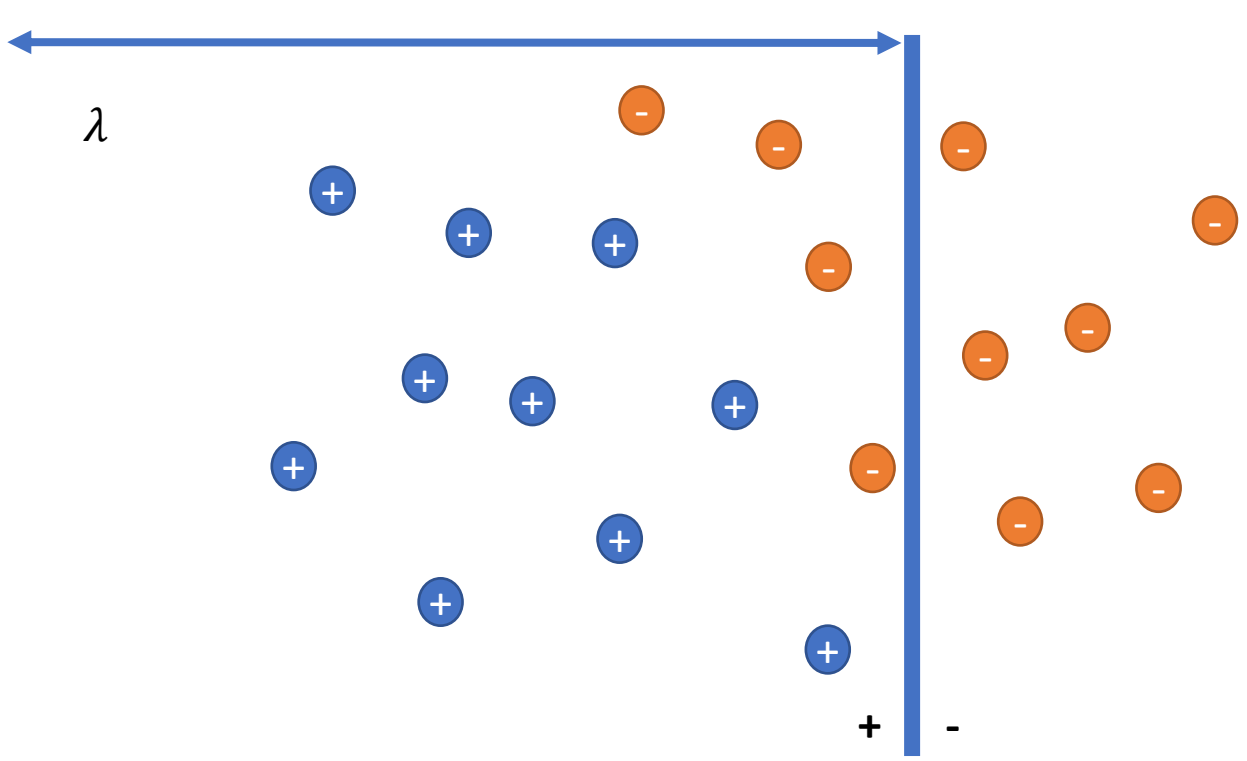
False positives: 0
True positives: 6

False positives: 1
True positives: 8

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”



False positives: 0
True positives: 0

False positives: 0
True positives: 6

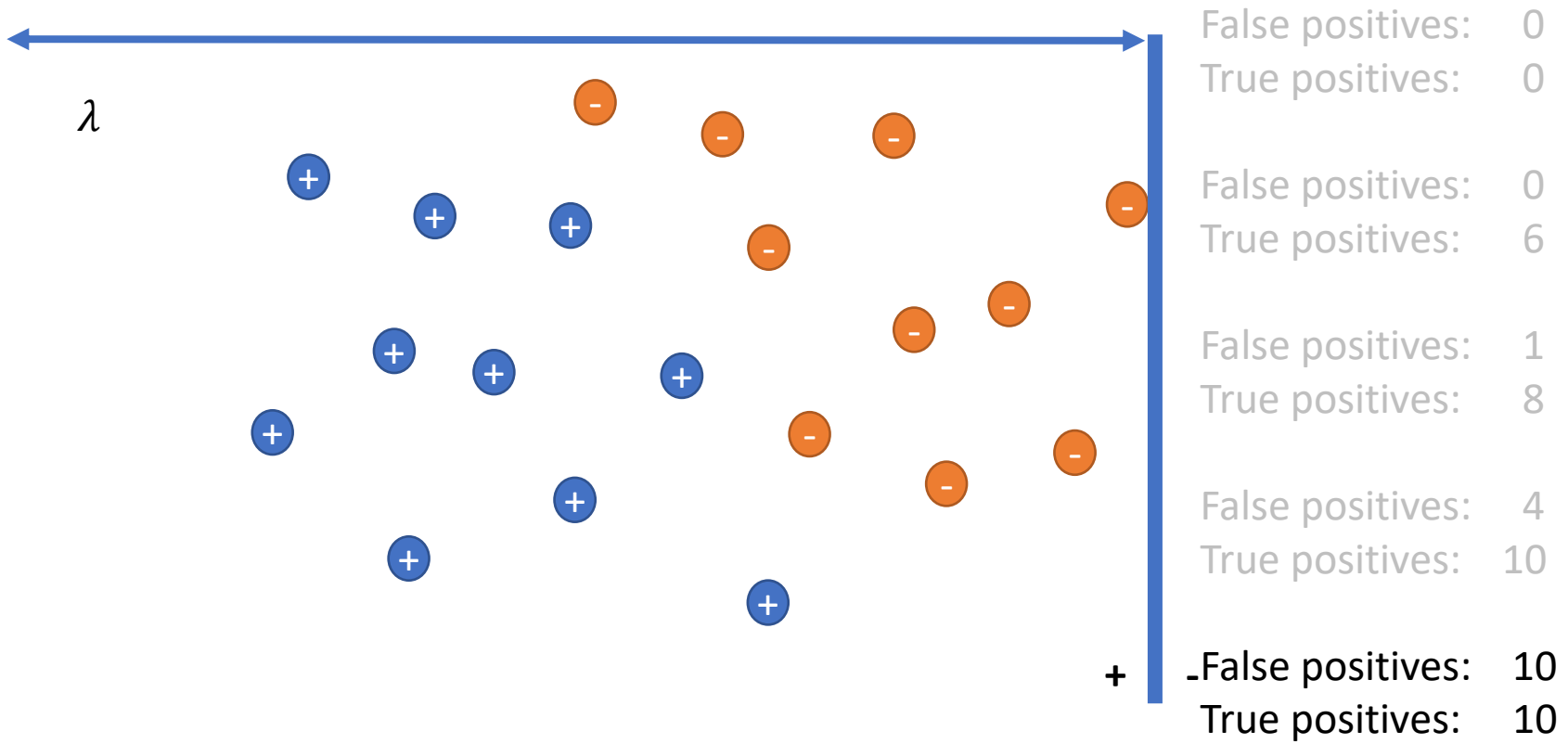
False positives: 1
True positives: 8

False positives: 4
True positives: 10

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

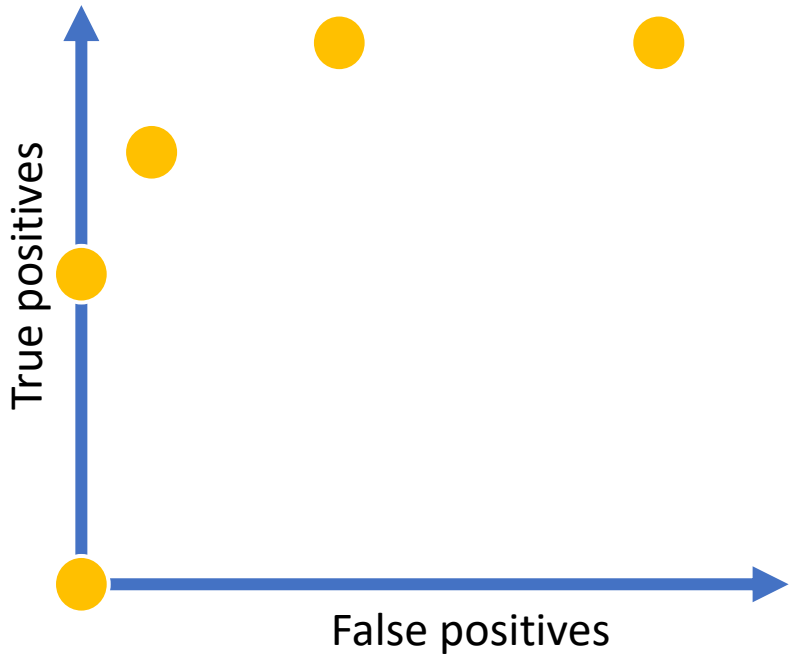
- Example: “Vertical line classifier”



Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”
- We can plot this into a graph



False positives: 0
True positives: 0

False positives: 0
True positives: 6

False positives: 1
True positives: 8

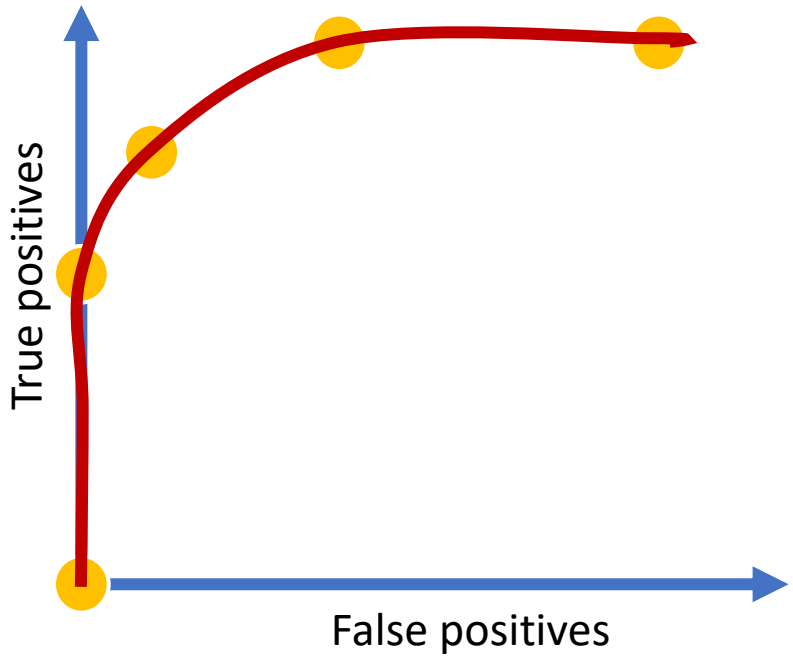
False positives: 4
True positives: 10

False positives: 10
True positives: 10

Receiver-Operating-Characteristic (ROC)

- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- Example: “Vertical line classifier”
- We can plot this into a graph and connect the dots



False positives: 0
True positives: 0

False positives: 0
True positives: 6

False positives: 1
True positives: 8

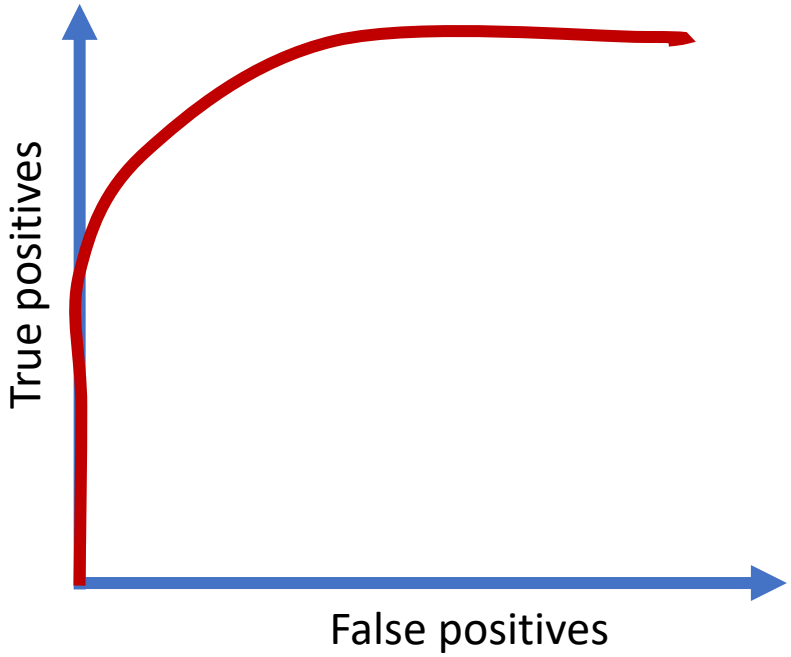
False positives: 4
True positives: 10

False positives: 10
True positives: 10

Receiver-Operating-Characteristic (ROC)

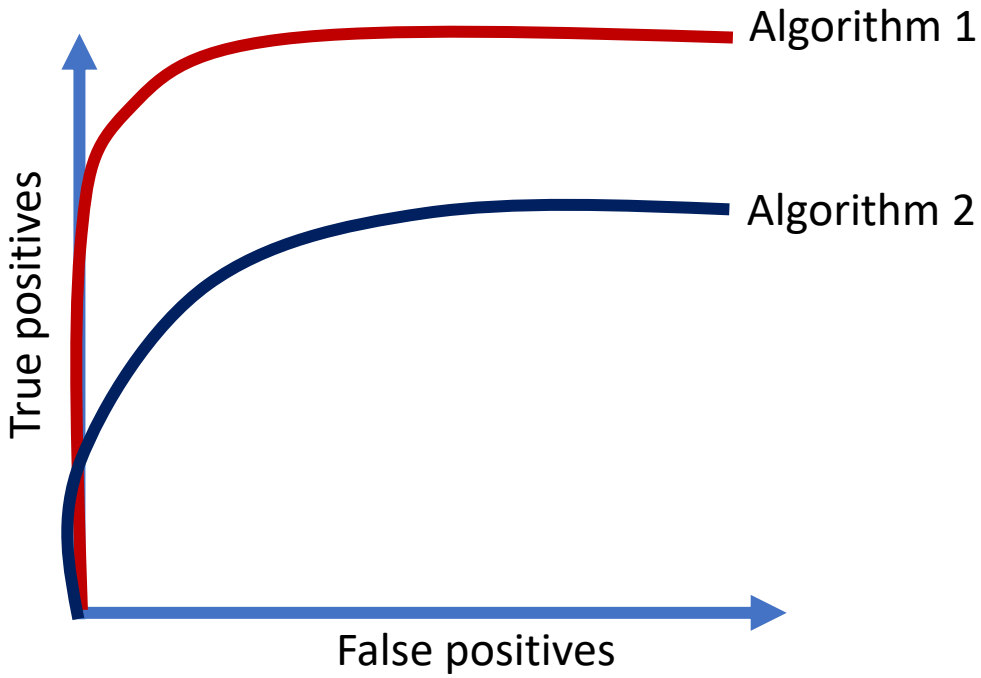
- Sometimes it is possible to identify a parameter λ that controls the relationship between false positives and false negatives

- This graph is called the **Receiver-Operating-Characteristic** or **ROC-curve**



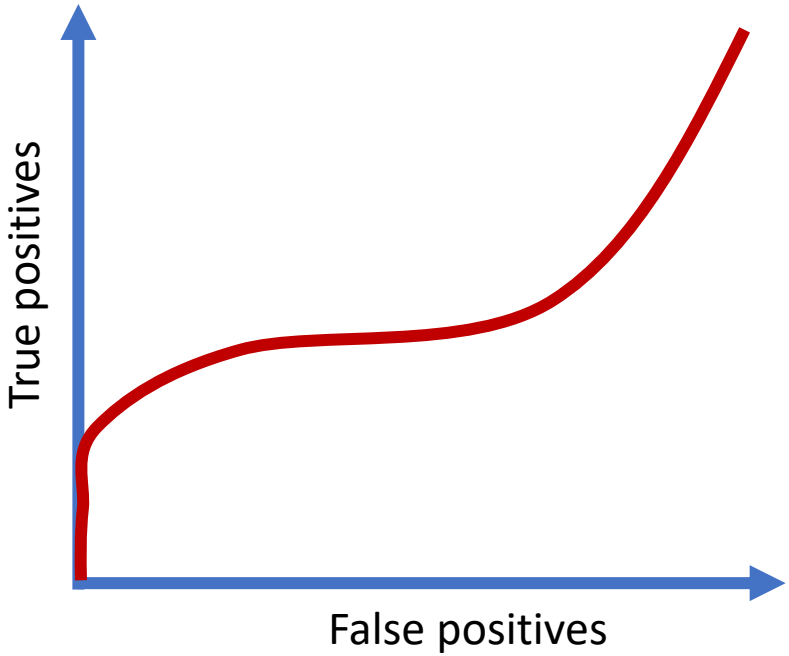
Receiver-Operating-Characteristic (ROC)

- A ROC-curve is frequently used to benchmark different classifiers
- Better classifiers are towards the top-left of the graph, worse towards the bottom-right



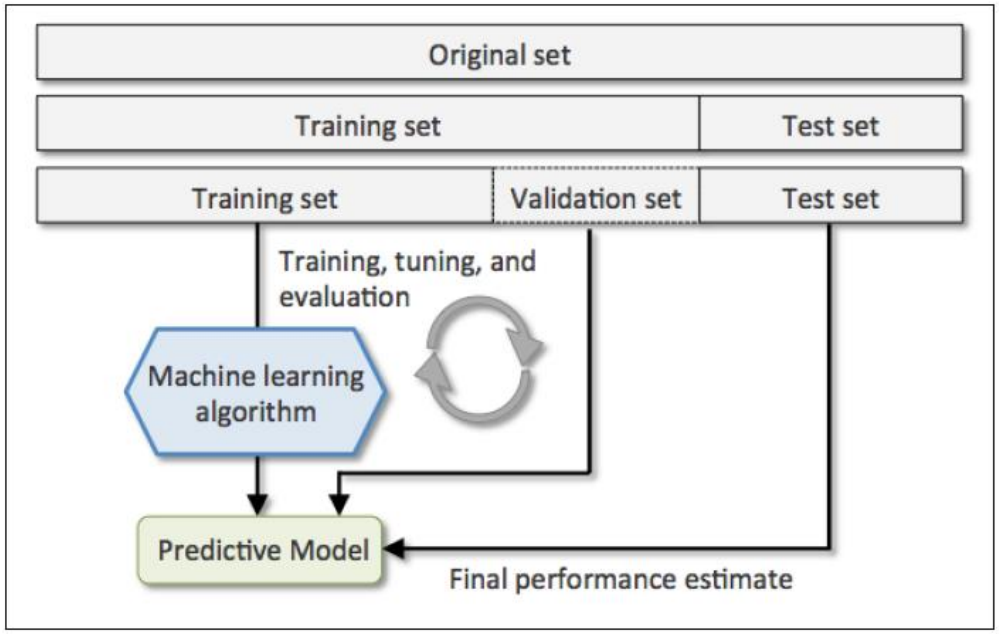
Receiver-Operating-Characteristic (ROC)

- The ROC is independent of the loss function
- It can be used to optimise the control parameter λ for different loss functions once computed
- Note, that the ROC curve does not need to be symmetric
- Also note, it does not even need to be concave in all circumstances



Holdout cross-validation

- In holdout cross-validation we evaluate the performance of the classifier based on three sets of samples
  - The training set used for fitting the classification function
  - The validation set used for optimising classifier parameters
  - The test set for evaluating the classifier



- Drawback: the outcome depends on the way we partition the data

K Fold Cross Validation

- In k-fold cross-validation, we randomly split the training dataset into **k** folds without replacement, where **k −1** folds are used for the model training and one fold is used for testing. This procedure is repeated k times so that we obtain k models and performance estimates.

- We then calculate the average performance of the models based on the different, independent folds to obtain a performance estimate that is less sensitive to the subpartitioning of the training data compared to the holdout method.

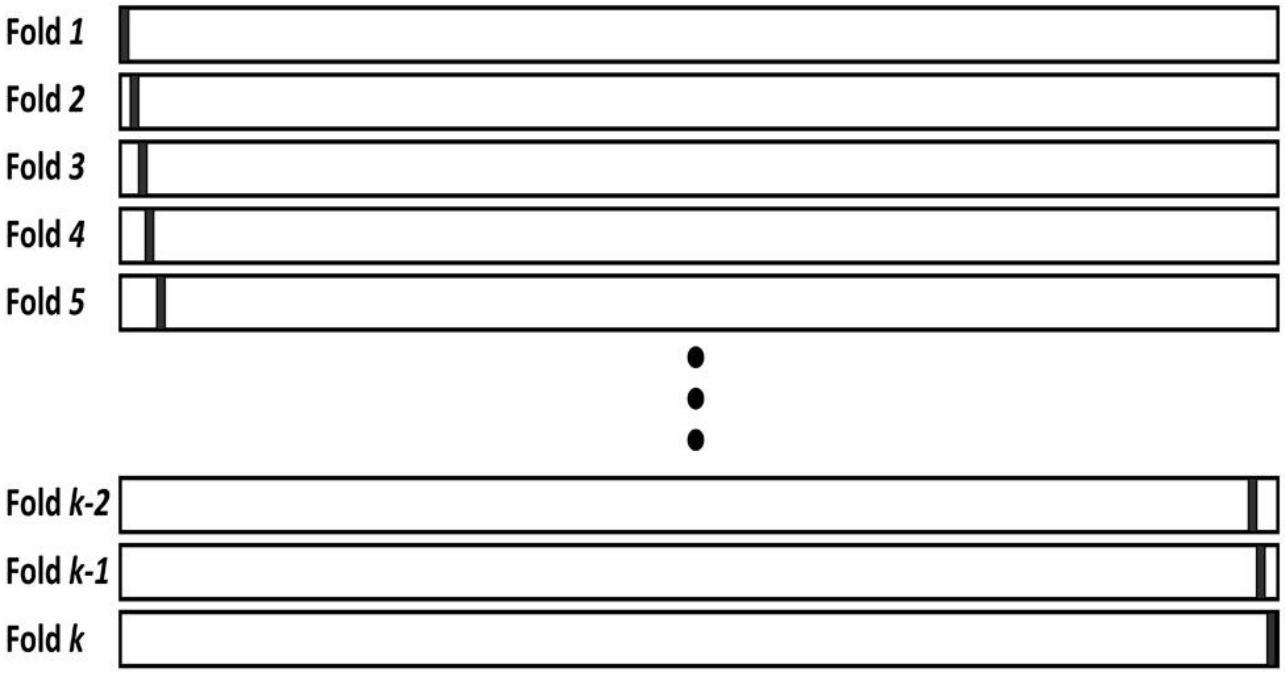
- Typically, we use k-fold cross-validation for model tuning, that is, finding the optimal hyperparameter values that yield a satisfying generalization performance.

- Once we have found satisfactory hyperparameter values, we can retrain the model on the complete training set and obtain a final performance estimate using the independent test set.

K Fold Cross Validation

Leave-one-out Cross Fold Validation

- LOO cross validation is an extreme form of cross validation where k = number of training instances.
- Therefore, for each fold the test set just contains a single instance.
- LOO is useful when the amount of data available is too small to allow big enough training sets in a k fold cross validation. However, it can also be very time-consuming.



Stratified Cross Fold Validation

- A slight improvement over the standard k-fold cross-validation approach is stratified k-fold cross-validation, which can yield better bias and variance estimates, especially in cases of unequal class proportions.
- In stratified cross-validation, the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset

Assessing Accuracy (Cross Fold Validation)

- The simplest way to use cross-validation is to call the *cross_val_score* function on the classifer and the dataset. Please note we use stratified cross fold validation by default

```
from sklearn import model_selection
from sklearn import datasets
from sklearn import svm


iris = datasets.load_iris()

clf = svm.SVC()

scores = model_selection.cross_val_score(clf, iris.data, iris.target, cv=10)

print (scores)

print (scores.mean(), scores.std())
```

In the example, we apply stratified cross fold validation.

Notice the scores variable holds the result after each fold. To get the final result we obtain the mean.

Also notice we don't have to direct call the fit function

```
[1.   0.93   1.   1.   1.   0.93   0.93   1.   1.   1. ]
0.98   0.03
```

k Fold Cross Validation

- While the approach presented in the previous slides is simple, often we want to get access to not just the accuracy for each fold but what classes were correctly or incorrectly predicted during each fold. In the example that follow we use normal k-fold cross validation.

- Therefore, it can often be very useful to perform cross fold validation manually.

- We can achieve this by using **sklearn.model_selection.Kfold**.
  - Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).
  - Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Please note there is also a **sklearn.model_selection.StratifiedKFold**

Scikit Learn - k Fold Cross Validation

```
from sklearn import datasets
from sklearn import svm
from sklearn import model_selection
from sklearn import metrics

iris = datasets.load_iris()
allResults = []

kf = model_selection.KFold(n_splits=6, shuffle=True)

for train_index, test_index in kf.split(iris.data):

    clf = svm.SVC()
    clf.fit( iris.data[train_index], iris.target[train_index] )

    results= clf.predict(iris.data[test_index])

    print (results[ results != iris.target[test_index]])

    allResults.append(metrics.accuracy_score(results, iris.target[test_index]))
print ("Accuracy is ", np.mean(allResults))
```

When we create the Kfold object we specify the number of folds as 6. Each time we iterate we call the **split** function, which will divide the indices into training and test (5/6 for training and 1/6 for test). Each time the loop iterates it generates a different fold.

Scikit Learn - k Fold Cross Validation

```
from sklearn import datasets
from sklearn import svm
from sklearn import model_selection
from sklearn import metrics

iris = datasets.load_iris()
allResults = []

kf = model_selection.KFold(n_splits=6, shuffle=True)

for train_index, test_index in kf.split(iris.data):

    clf = svm.SVC()
    clf.fit( iris.data[train_index], iris.target[train_index] )

    results= clf.predict(iris.data[test_index])

    print (results[ results != iris.target[test_index]])

    allResults.append(metrics.accuracy_score(results, iris.target[test_index]))
print ("Accuracy is ", np.mean(allResults))
```

Notice that NumPy array results contains all the predicts made by our model. If I want to determine the classes the model got wrong for each I compare them to the actual results using array-based indexing.

```
[]
[2]
[]
[2]
[]
[2]
Accuracy is  0.98
```

Thank you for your attention