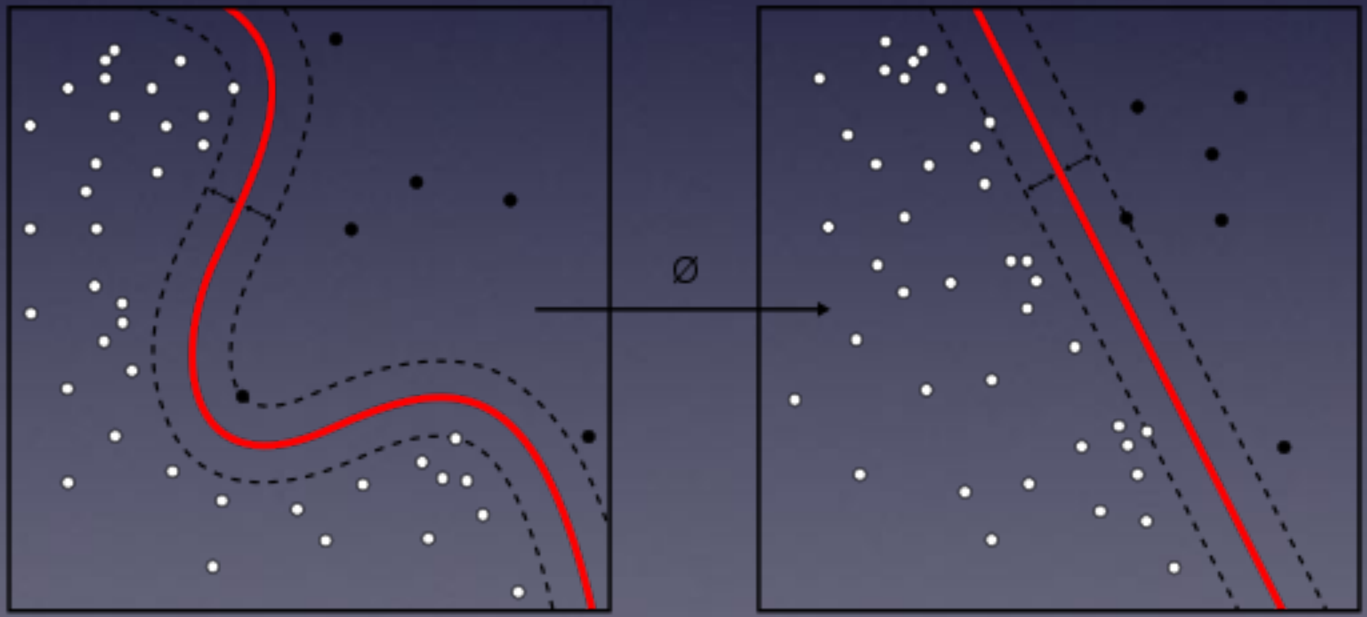
Data Science Workflow

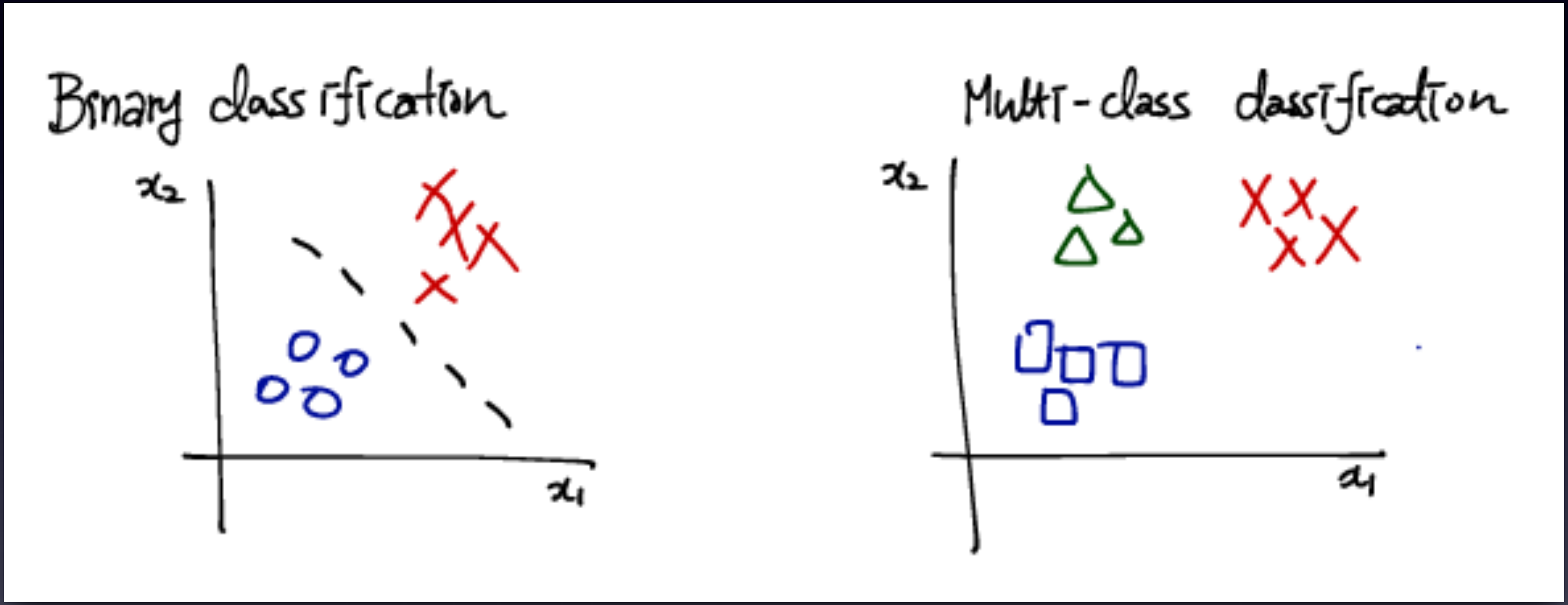
Binary Classification

Parvin Shakibaei

What is a classifier

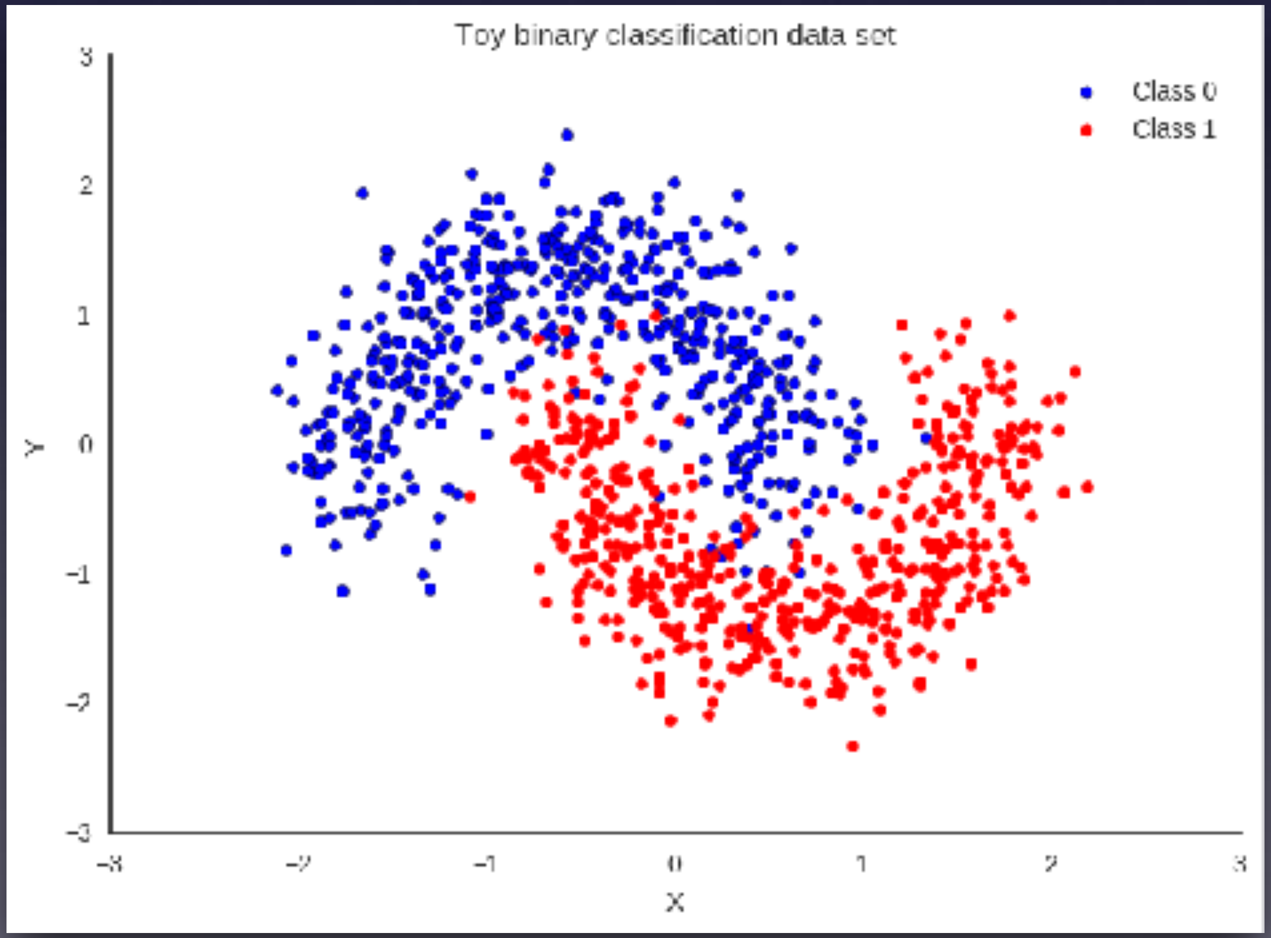
- In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-population) a new observation belongs

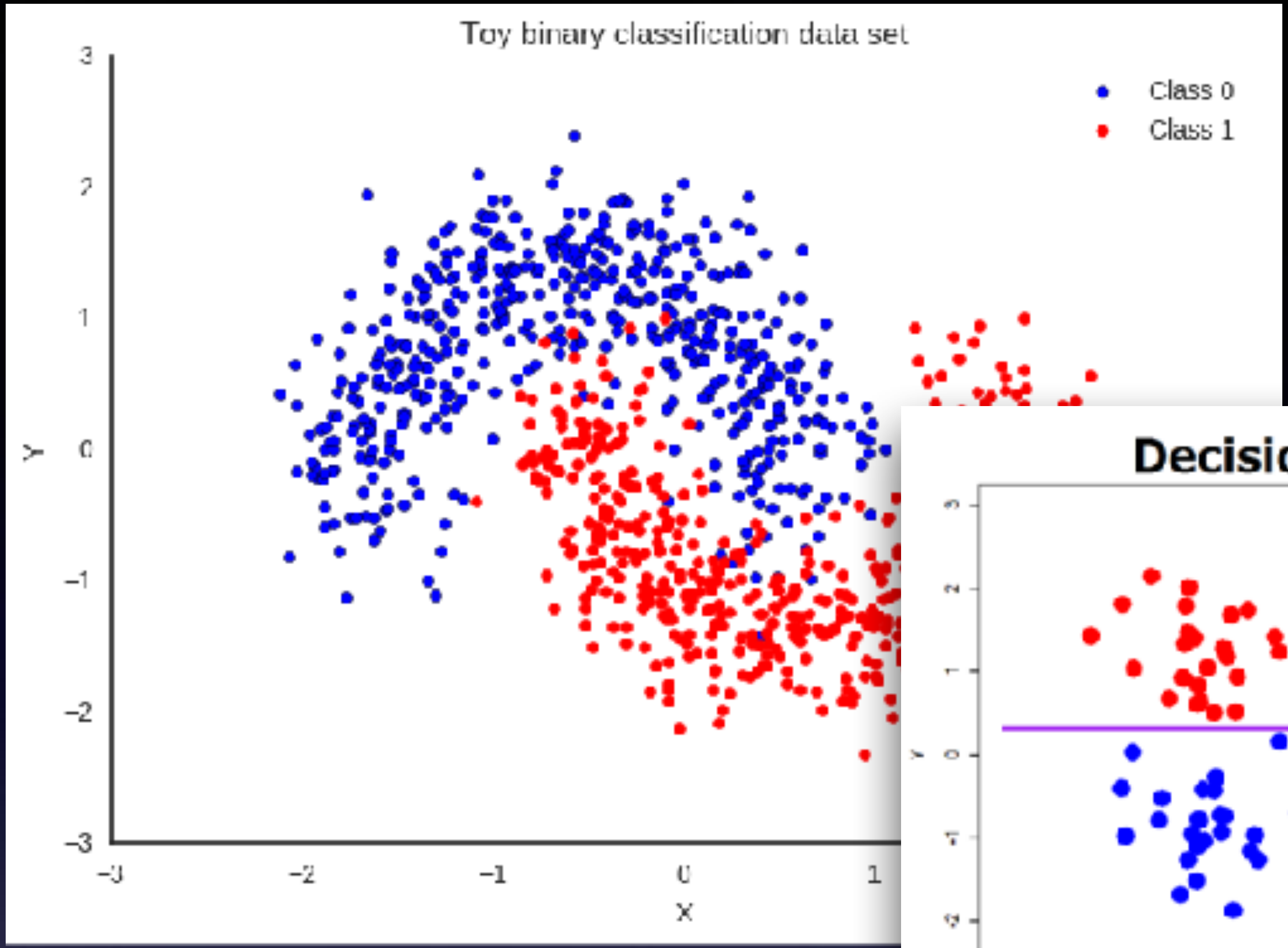


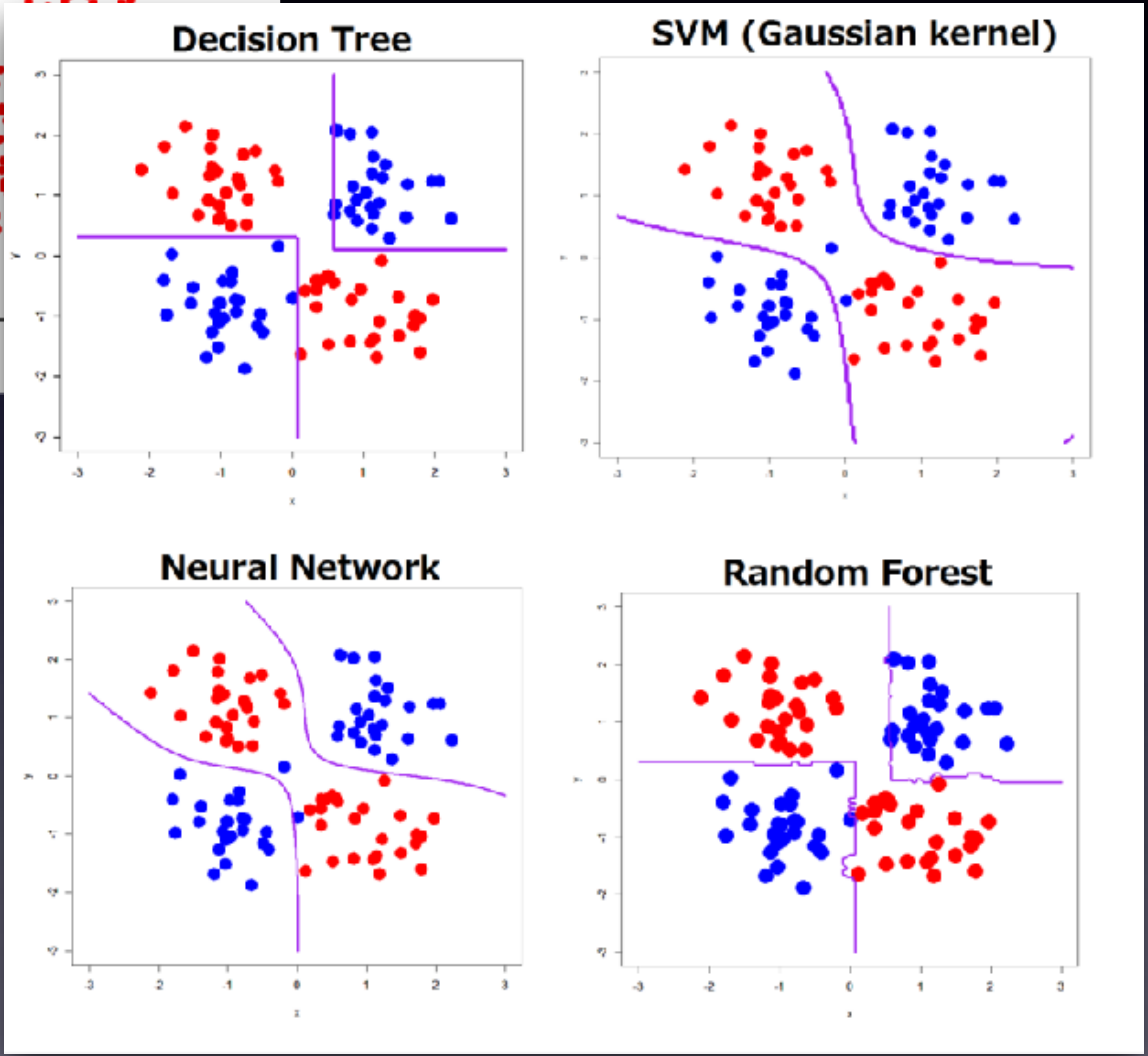

Binary classification
x_2
x_1
Multi-class classification
x_2
x_1

Binary Classifier

- Binary or binomial classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule.

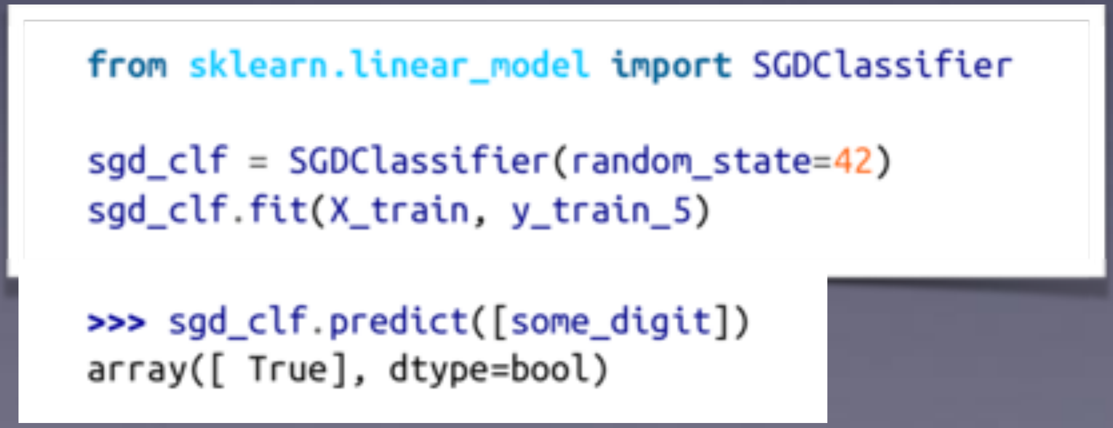



Toy binary classification data set
Class 0
Class 1
Y
X


Decision Tree
SVM (Gaussian kernel)
Neural Network
Random Forest

Training a Binary Classifier

- SGD or Stochastic gradient descent:
  This classifier has the advantage of being capable of handling very large datasets efficiently.

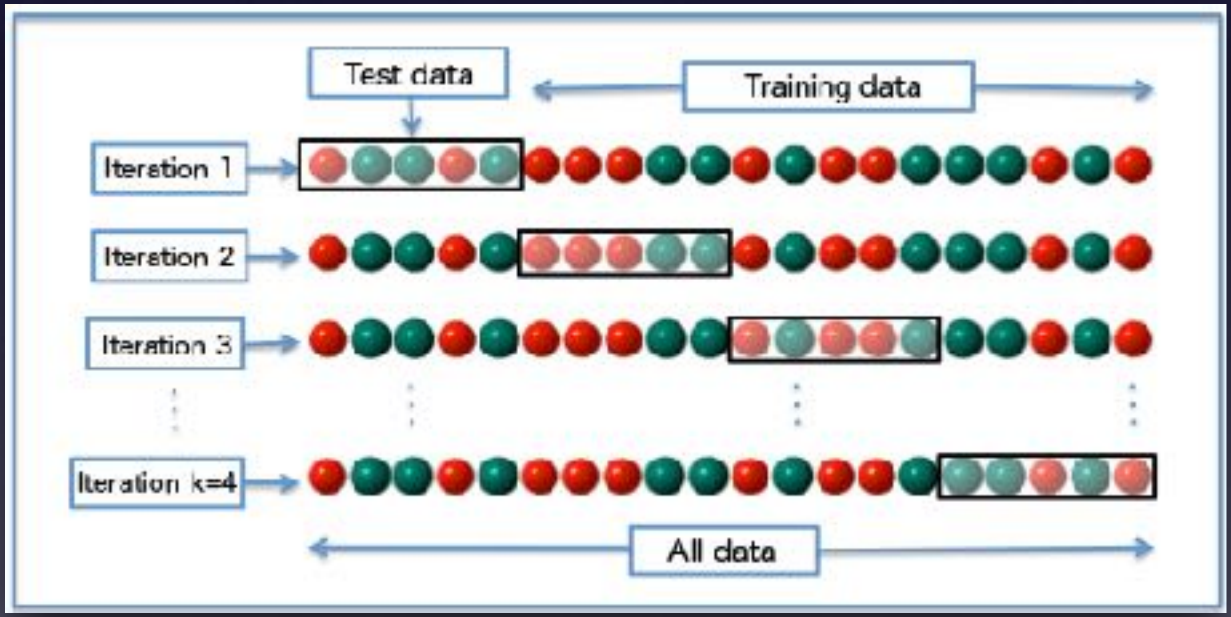


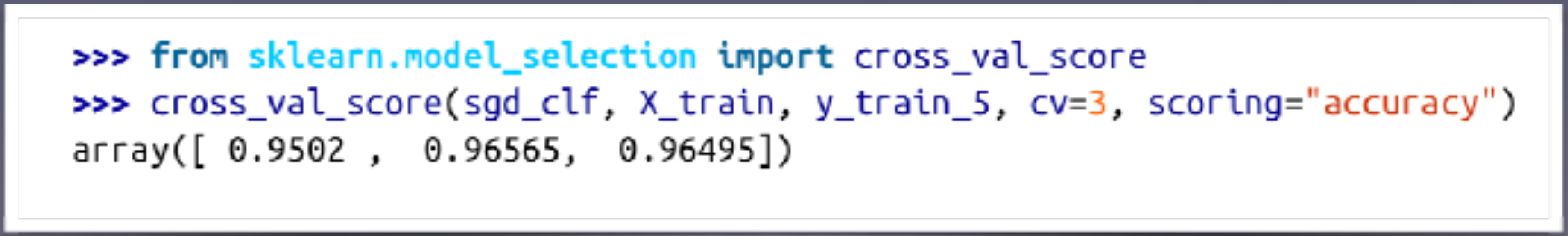
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

```
>>> sgd_clf.predict([some_digit])
array([ True], dtype=bool)
```

Performance Measures

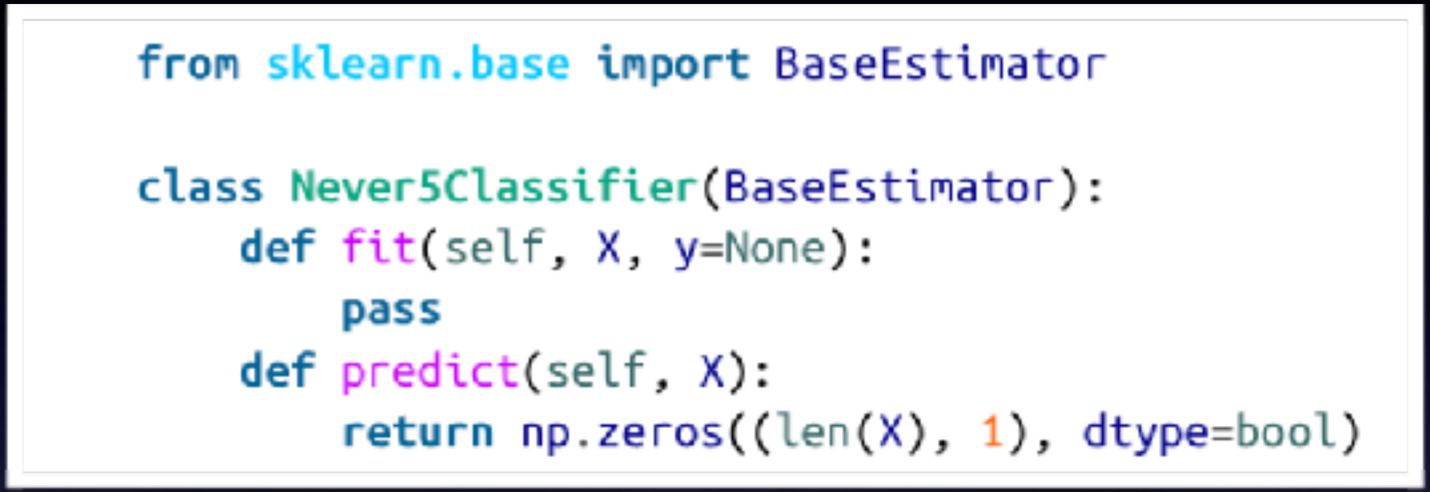
- Measuring Accuracy Using Cross validation
  K-fold cross validation means splitting the training set into k-folds, then making predictions and evaluating them on each fold using a model trained on the remaining folds





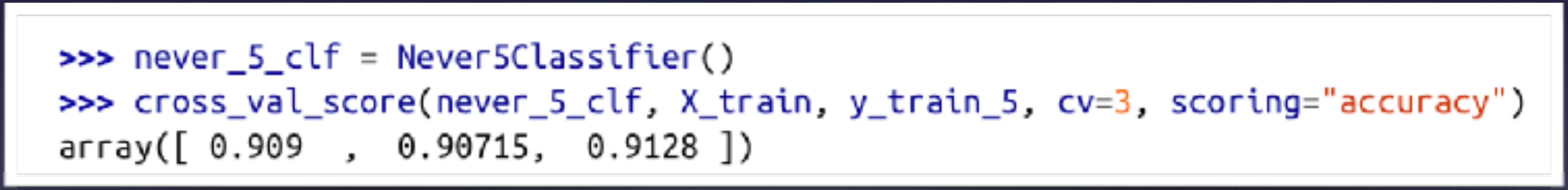
```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.9502 ,  0.96565,  0.96495])
```

- A dumb classifier that just classifies every single image in the "not-5" class:

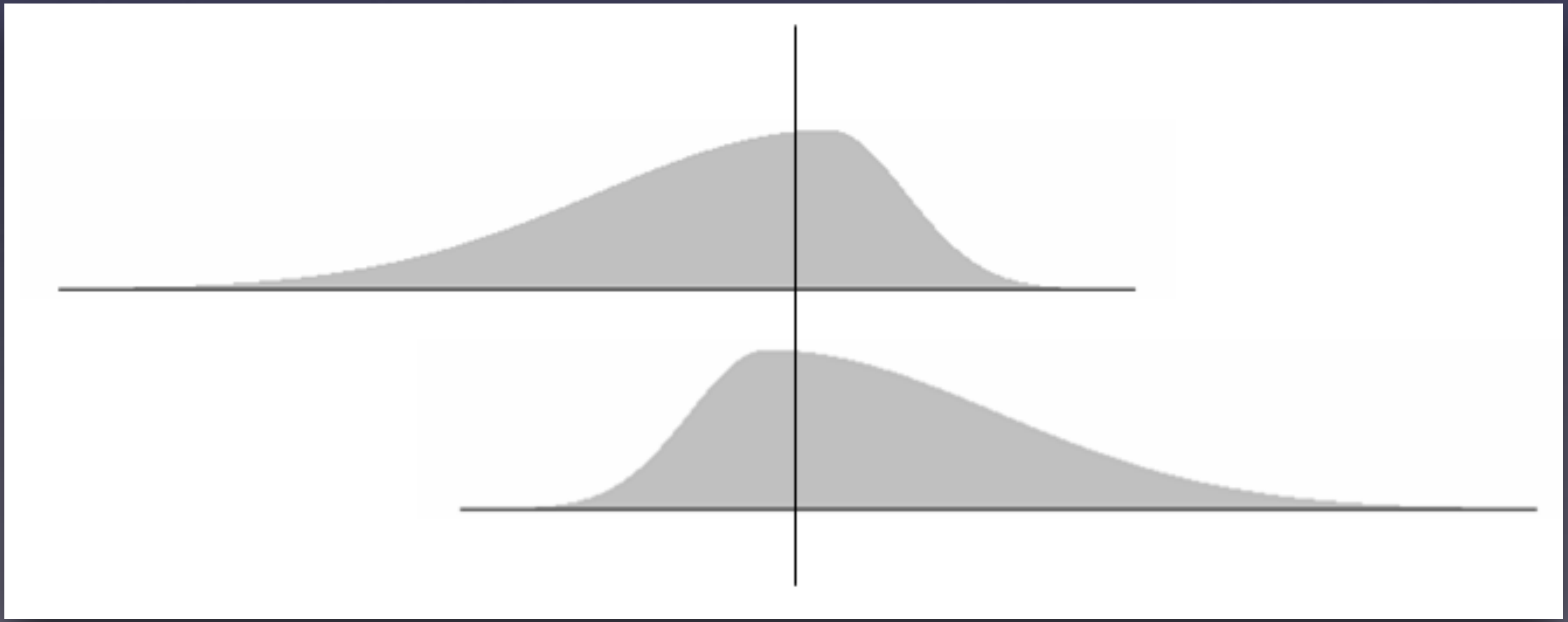


```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

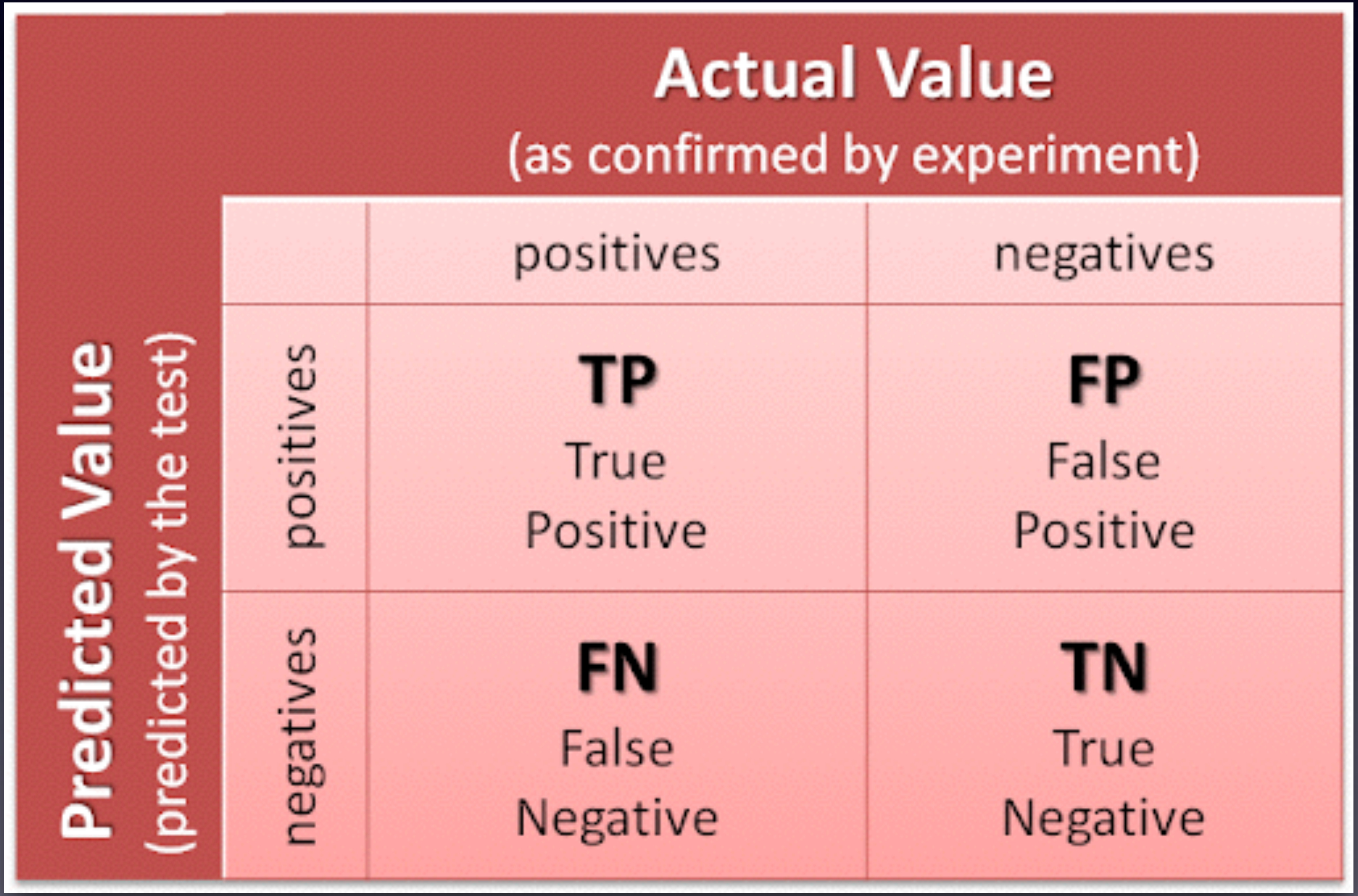
- Measuring accuracy:



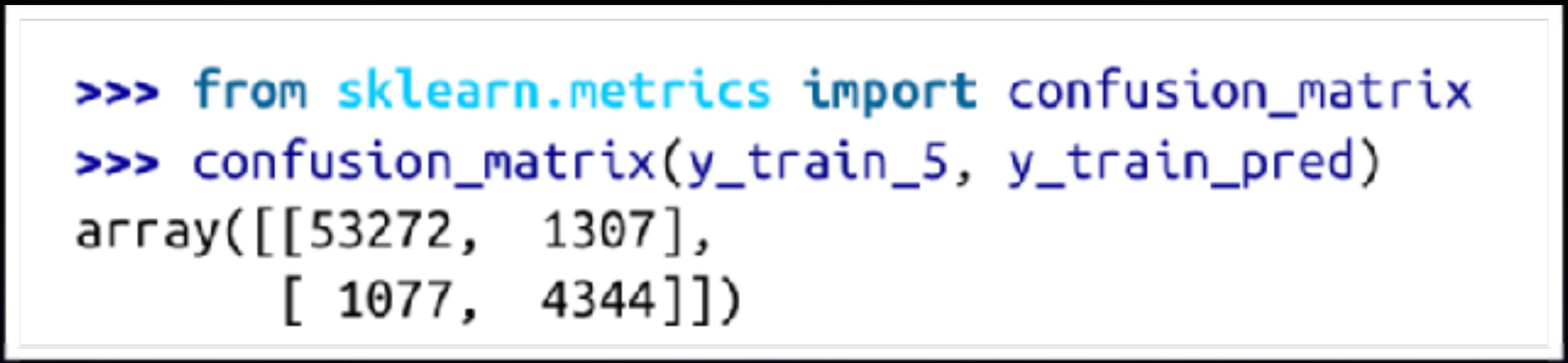
```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.909   ,  0.90715,  0.9128 ])
```



Confusion Matrix



		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative



```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53272,  1307],
       [ 1077,  4344]])
```

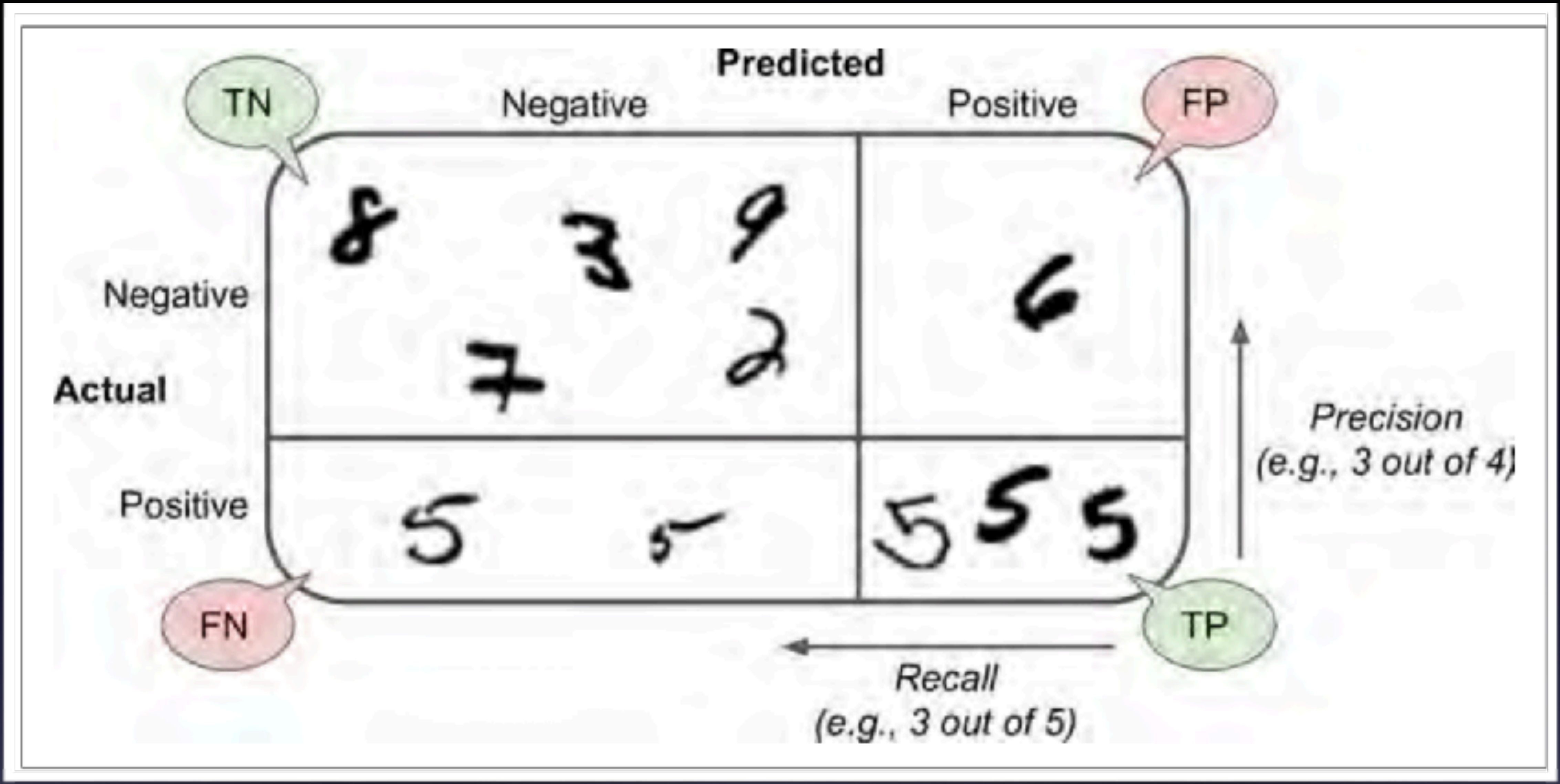
- Each row in a confusion matrix represents an actual class, while each column represents a predicted class. The first row of this matrix considers non-5 images (the negative class): 53,272 of them were correctly classified as non-5s (they are called true negatives), while the remaining 1,307 were wrongly classified as 5s (false positives). The second row considers the images of 5s (the positive class): 1,077 were wrongly classified as non-5s (false negatives), while the remaining 4,344 were correctly classified as 5s (true positives).

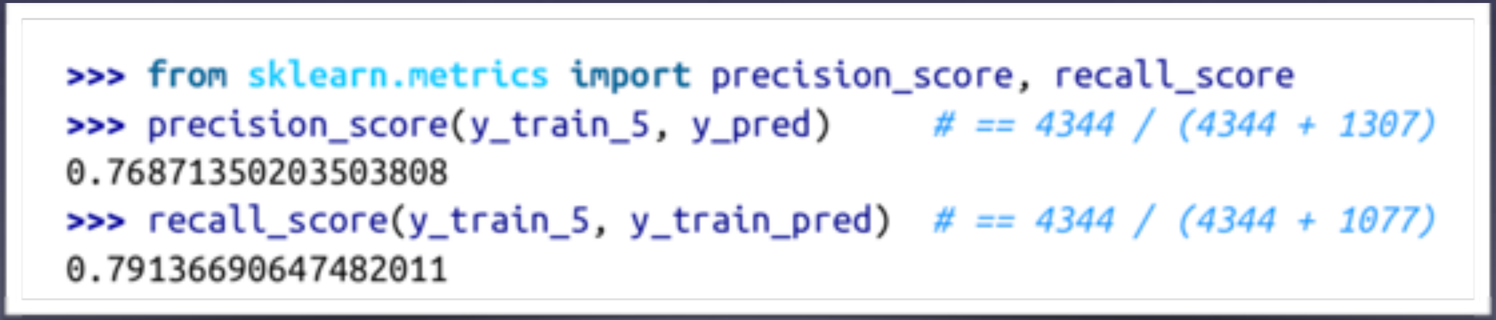
- accuracy of the positive predictions

$$\text{precision} = \frac{TP}{TP + FP}$$

- *A trivial way to have perfect precision is to make one single positive prediction and ensure it is correct (precision = 1/1 = 100%). This would not be very useful since the classifier would ignore all but one positive instance. So precision is typically used along with another metric named recall, also called sensitivity or true positive rate (TPR): this is the ratio of positive instances that are correctly detected by the classifier*

$$\text{recall} = \frac{TP}{TP + FN}$$





```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_pred)      # == 4344 / (4344 + 1307)
0.76871350203503808
>>> recall_score(y_train_5, y_train_pred)  # == 4344 / (4344 + 1077)
0.79136690647482011
```

Now your 5-detector does not look as shiny as it did when you looked at its accuracy. When it claims an image represents a 5, it is correct only 77% of the time. Moreover, it only detects 79% of the 5s.

- The F1 score is the harmonic mean of precision and recall. Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values. As a result, the classifier will only get a high F1 score if both recall and precision are high.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

if you trained a classifier to detect videos that are safe for kids, you would probably prefer a classifier that rejects many good videos (low recall) but keeps only safe ones (high precision), rather than a classifier that has a much higher recall but lets a few really bad videos show up in your product (in such cases, you may even want to add a human pipeline to check the classifier's video selection). On the other hand, suppose you train a classifier to detect shoplifters on surveillance images: it is probably fine if your classifier has only 30% precision as long as it has 99% recall (sure, the security guards will get a few false alerts, but almost all shoplifters will get caught).

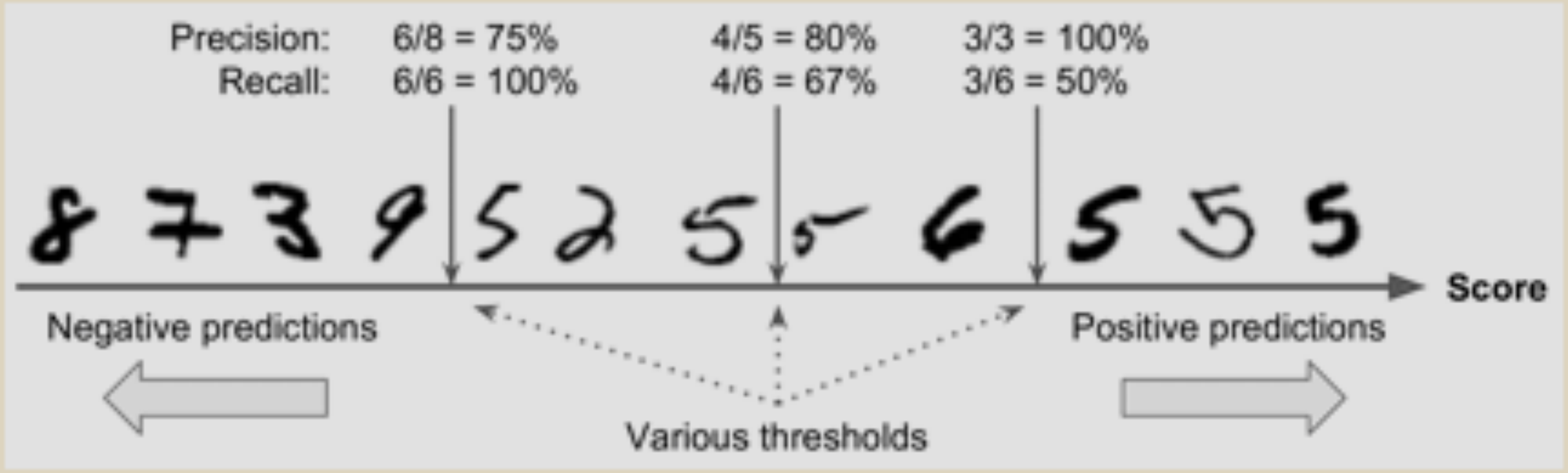
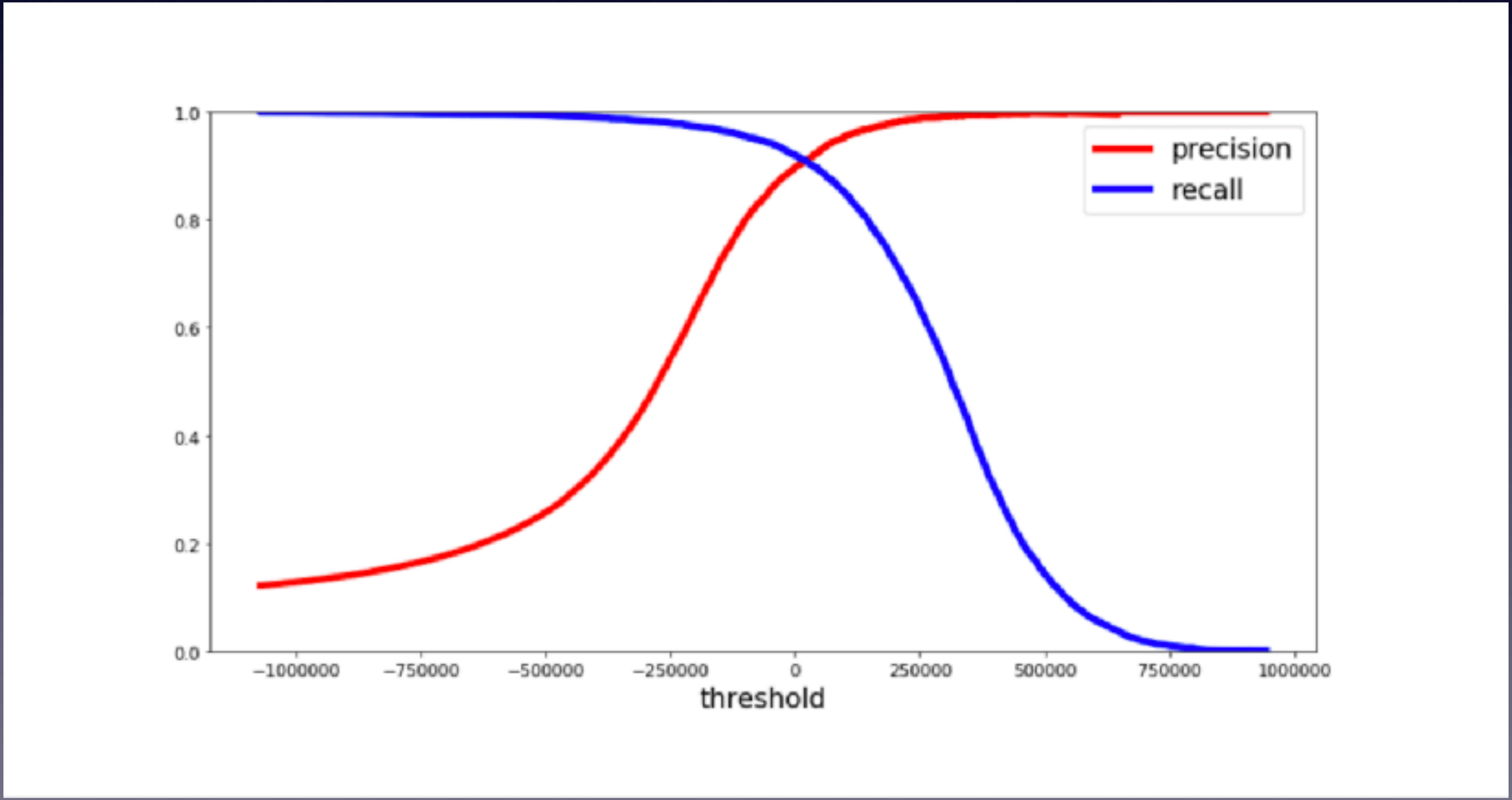


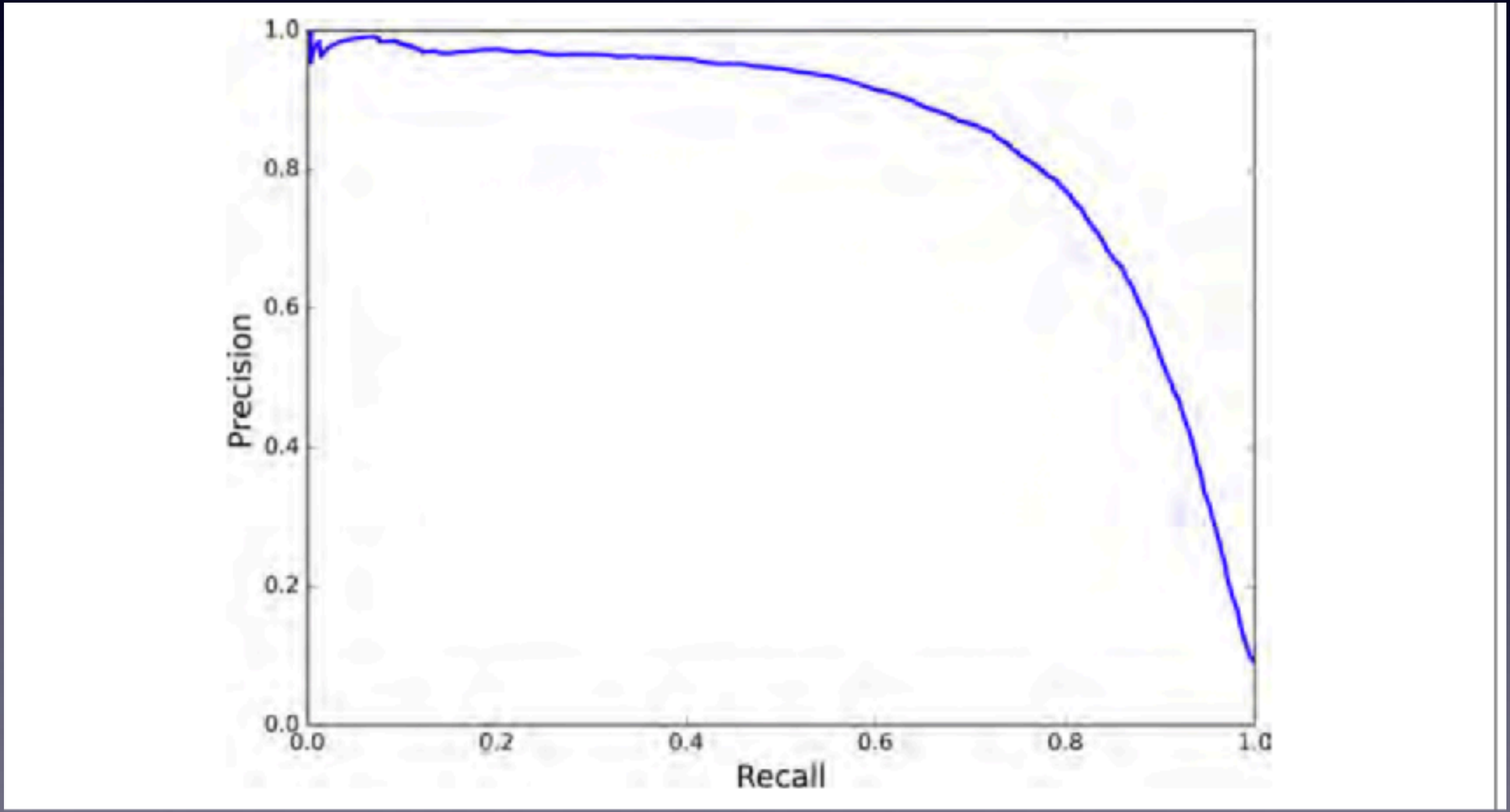
Figure 3-3. Decision threshold and precision/recall tradeoff

Suppose the decision threshold is positioned at the central arrow : you will find 4 true positives (actual 5s) on the right of that threshold, and one false positive (actually a 6). Therefore, with that threshold, the precision is 80% (4 out of 5). But out of 6 actual 5s, the classifier only detects 4, so the recall is 67% (4 out of 6). Now if you raise the threshold (move it to the arrow on the right), the false positive (the 6) becomes a true negative, thereby increasing precision (up to 100% in this case), but one true positive becomes a false negative, decreasing recall down to 50%. Conversely, lowering the threshold increases recall and reduces precision.

Precision and recall versus the decision threshold

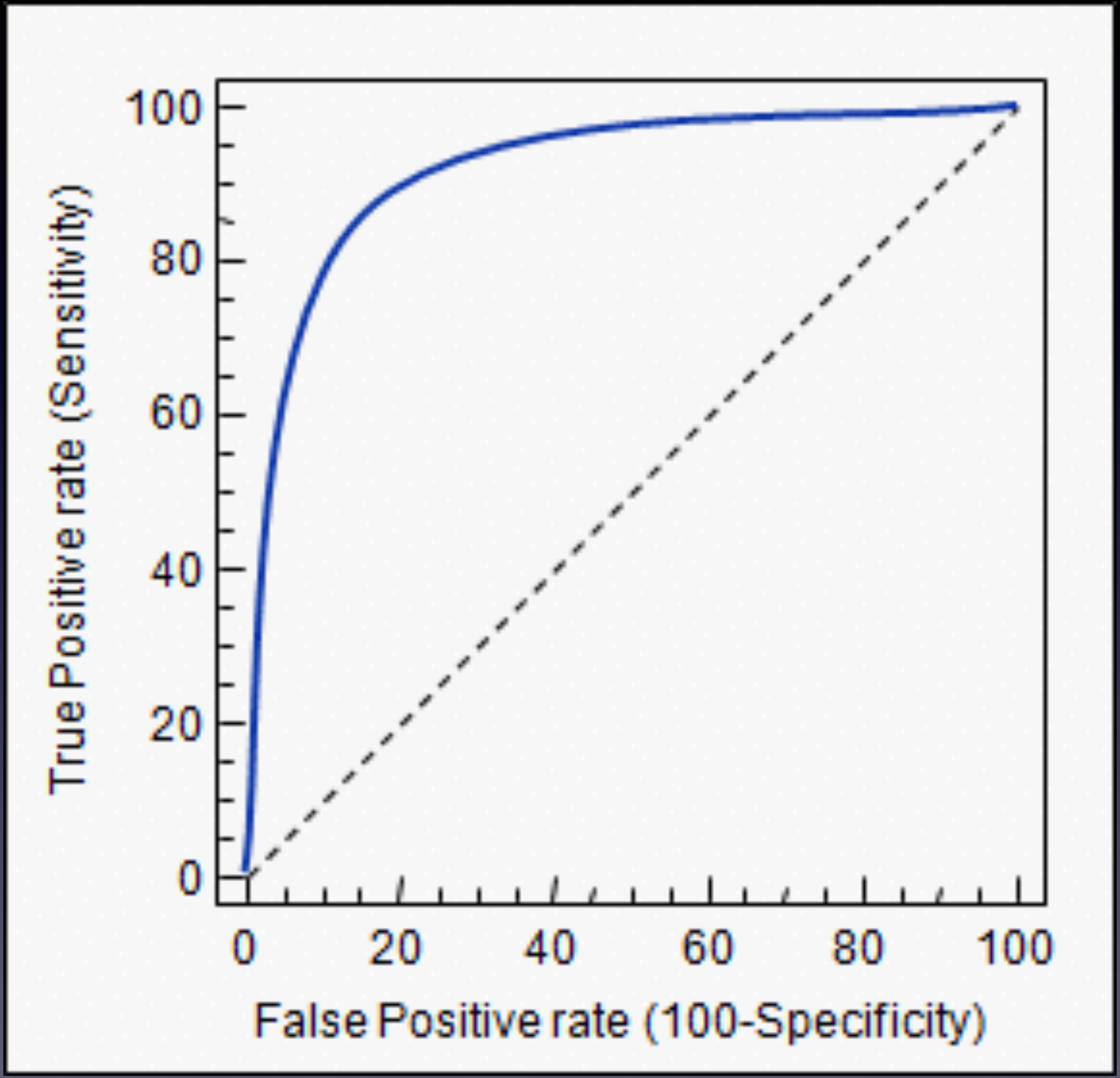


Precision versus recall

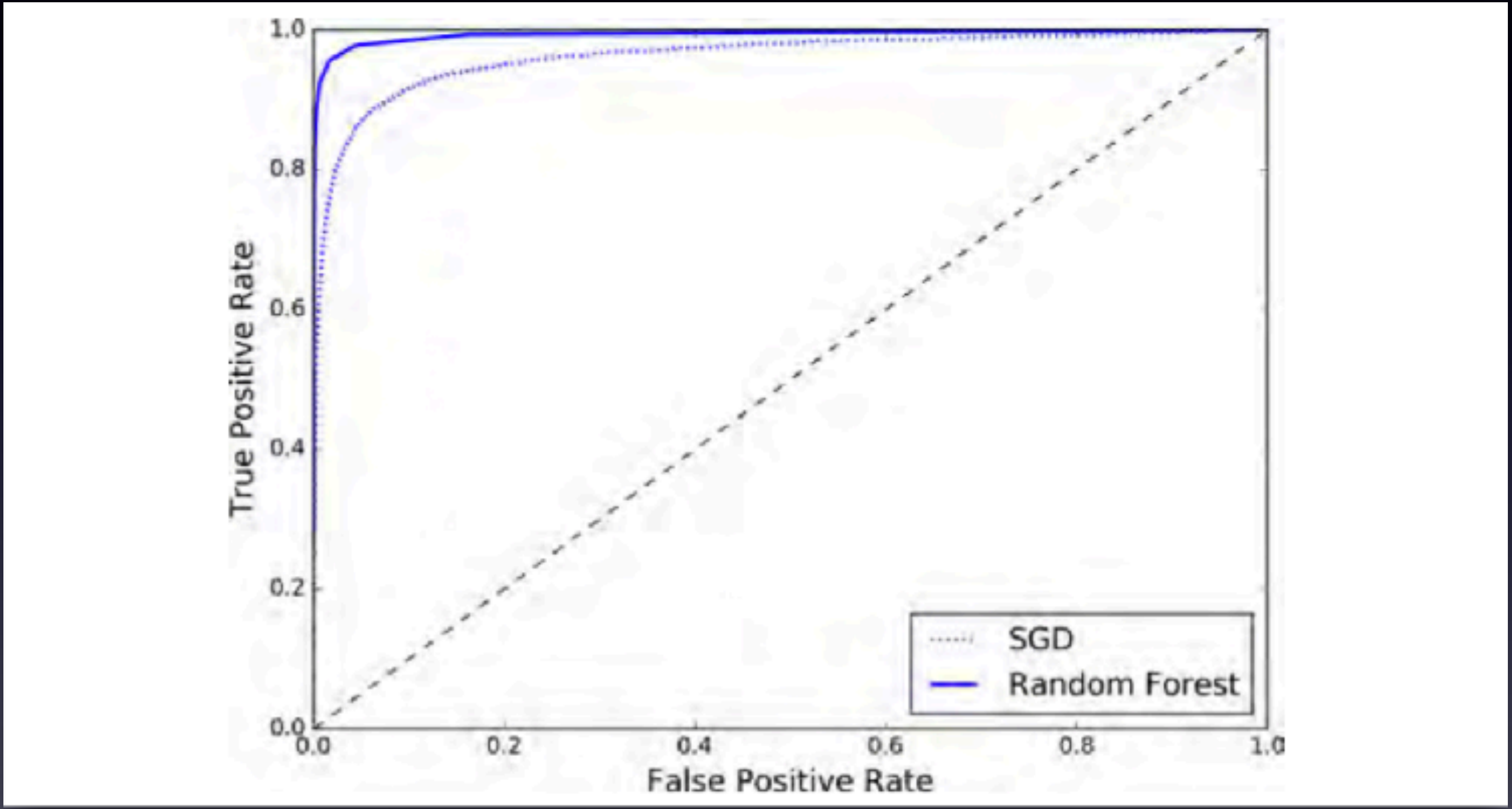


ROC curve

In Statistics, a receiver operating characteristic curve, i.e. ROC curve is a plot that illustrates the diagnostic ability of binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection.



Comparing ROC curves

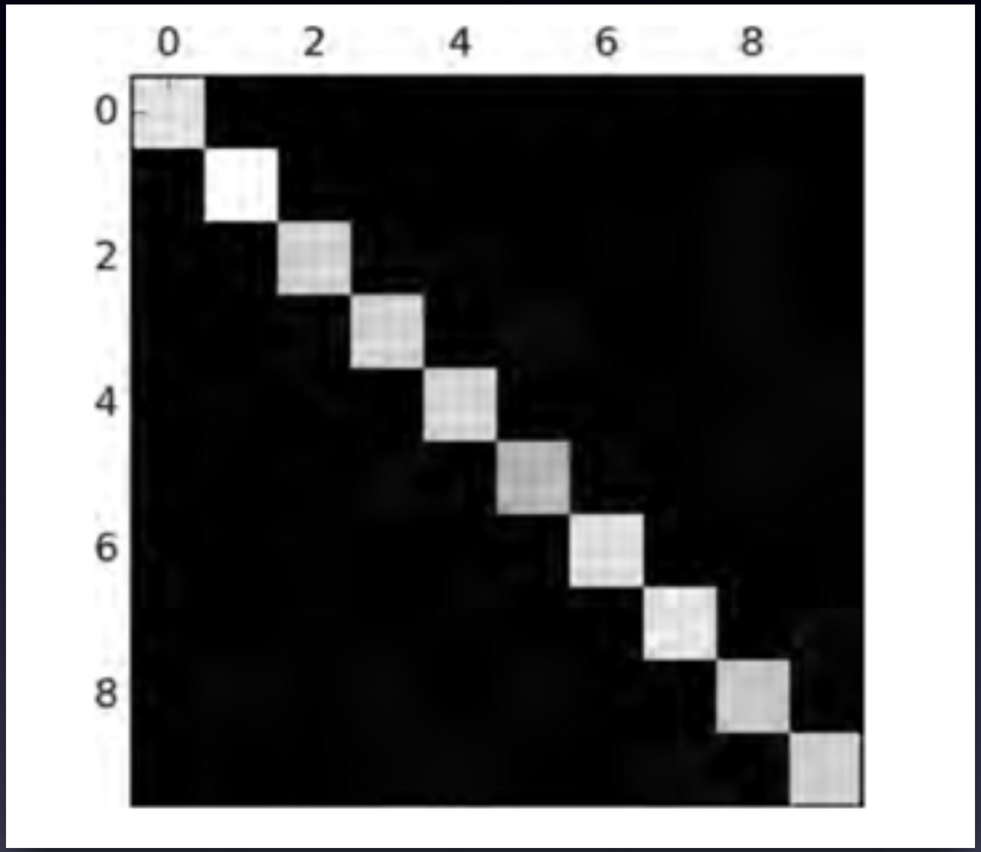


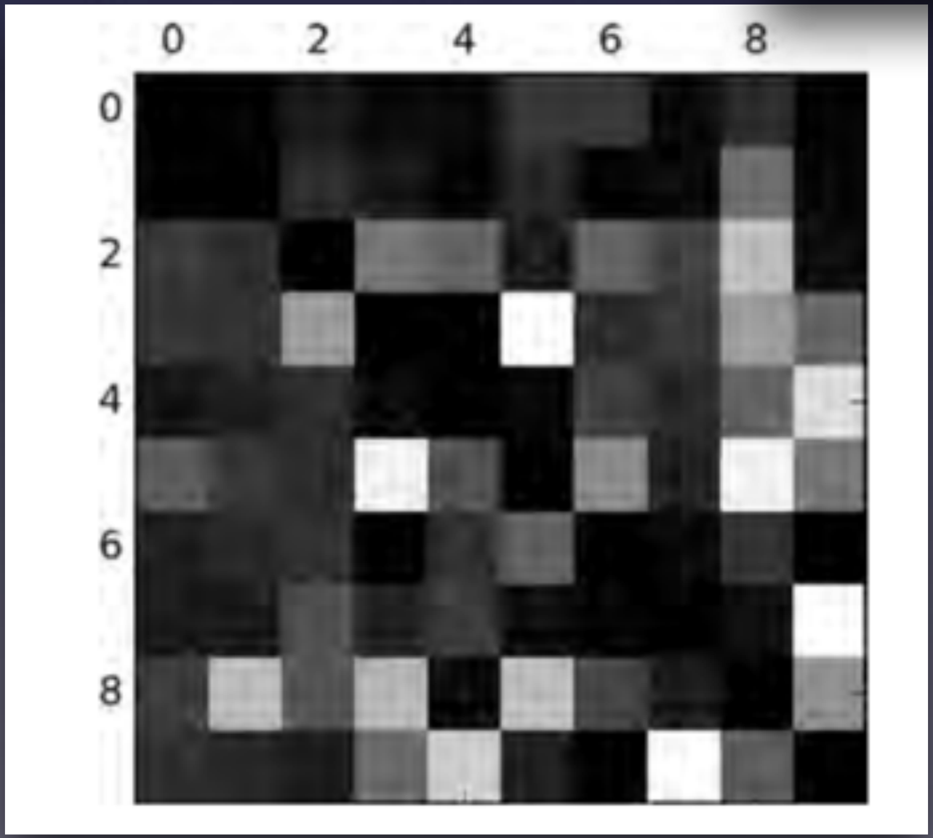
the area under the curve (AUC). A perfect classifier will have a ROC AUC equal to 1, whereas a purely random classifier will have a ROC AUC equal to 0.5
RandomForestClassifier's ROC curve looks much better than the SGDClassifier's: it comes much closer to the top-left corner. As a result, its ROC AUC score is also significantly better

Multiclass Classification

Whereas binary classifiers distinguish between two classes, multi class classifiers can distinguish between more than two classes.
Support Vector Machine classifiers or Linear classifiers
one-versus-all (OvA):
classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).
(OvO) strategy:
train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on. This is called the one-versus-one

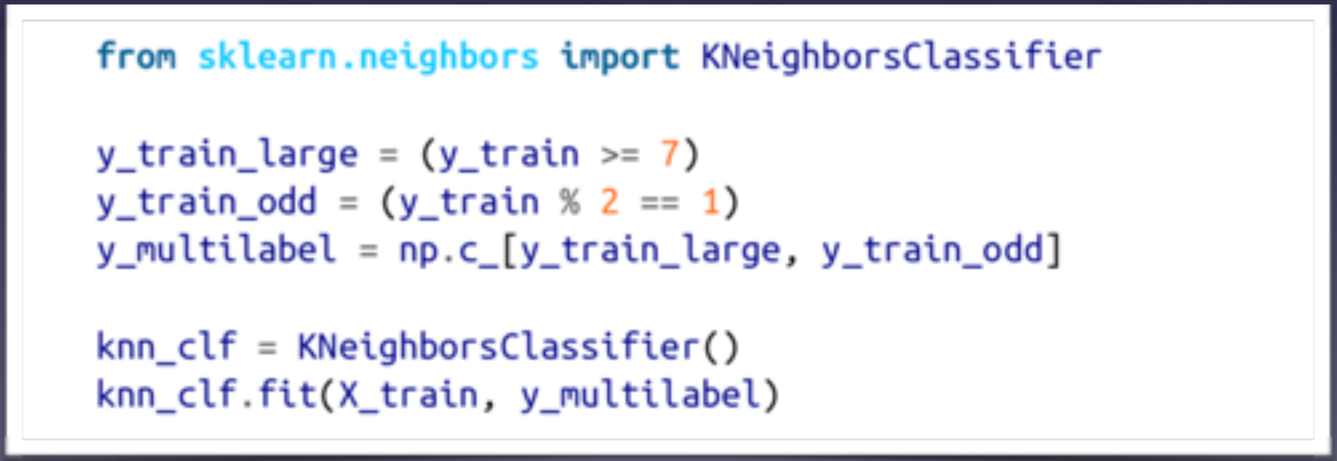
Error Analysis



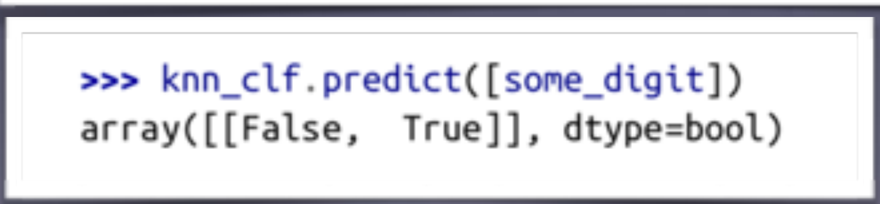


Multilabel classification

This code creates a y-multilabel array containing two target labels for each digit image: the first indicates whether or not the digit is large (7, 8, or 9) and the second indicates whether or not it is odd. The next lines create a KNeighborsClassifier instance (which supports multilabel classification, but not all classifiers do) and we train it using the multiple targets array. Now you can make a prediction, and notice that it outputs two labels



```python
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```



```
>>> knn_clf.predict([some_digit])
array([[False,  True]], dtype=bool)
```

Mulitoutput classification