Classification Metrics

Hands-on Machine Learning: Chapter 3

MNIST

Handwritten digits

Labels: 0 ... 9

70k: train 60k + test 10k

784 features: 28 x 28 pixels

Values: 0 ... 255, white ... black

Figure 3-1. A few digits from the MNIST dataset

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1, as_frame=False)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details', 'categories', 'url'])
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)

import matplotlib as mpl
import matplotlib.pyplot as plt
some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = mpl.cm.binary, interpolation="nearest")
plt.axis("off")
plt.show()

>>> y[0]
'5'
>>> y = y.astype(np.uint8)
>>> X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Training a Binary Classifier

```
y_train_5 = (y_train == 5) # True for all 5s, False for all other digits.
y_test_5 = (y_test == 5)

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])
```

Performance Measures

```python
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))
```

Accuracy Using Cross Validation

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])

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)

>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.91125, 0.90855, 0.90915])
```

Accuracy is generally not the preferred performance measure especially with *skewed datasets*

Confusion Matrix

Actual classes

Predicted classes

True negatives (TN)

False positives (FP)

False negatives (FN)

True positives (TP)

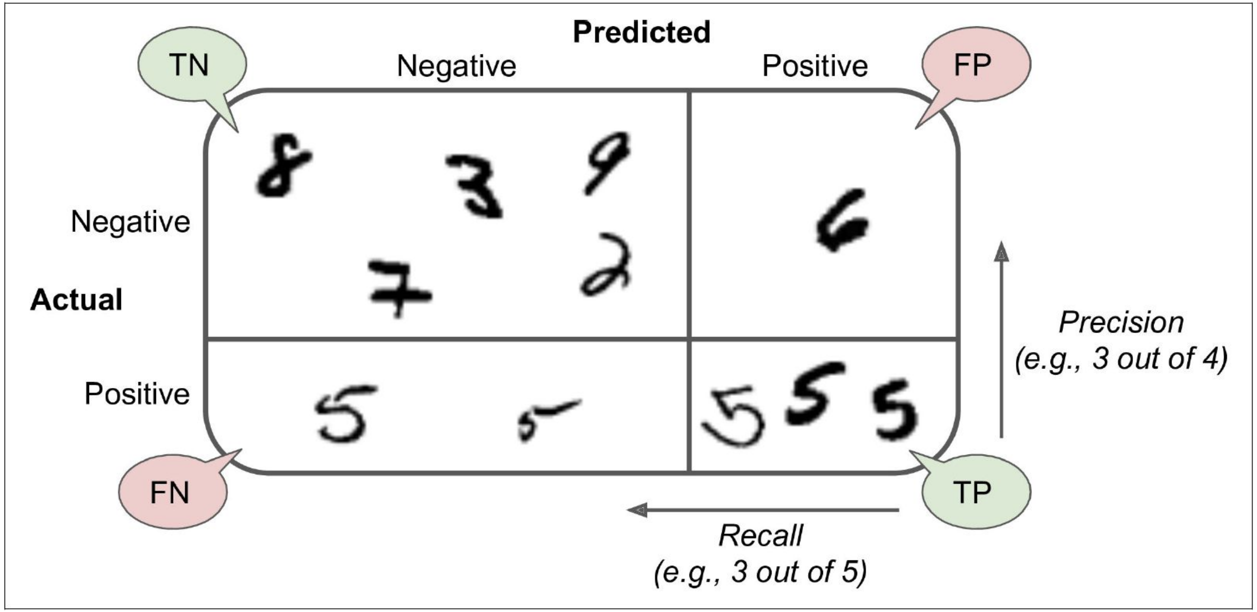

Figure 3-2. An illustrated confusion matrix

Confusion Matrix

```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)

>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057, 1522],
       [ 1325, 4096]])

>>> y_train_perfect_predictions = y_train_5 # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579, 0],
       [ 0, 5421]])
```

Perfect classifier only has TP and TN. Confusion matrix non-0s on main diagonal.

Precision

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

Trivial perfect precision make

1 positive correct prediction

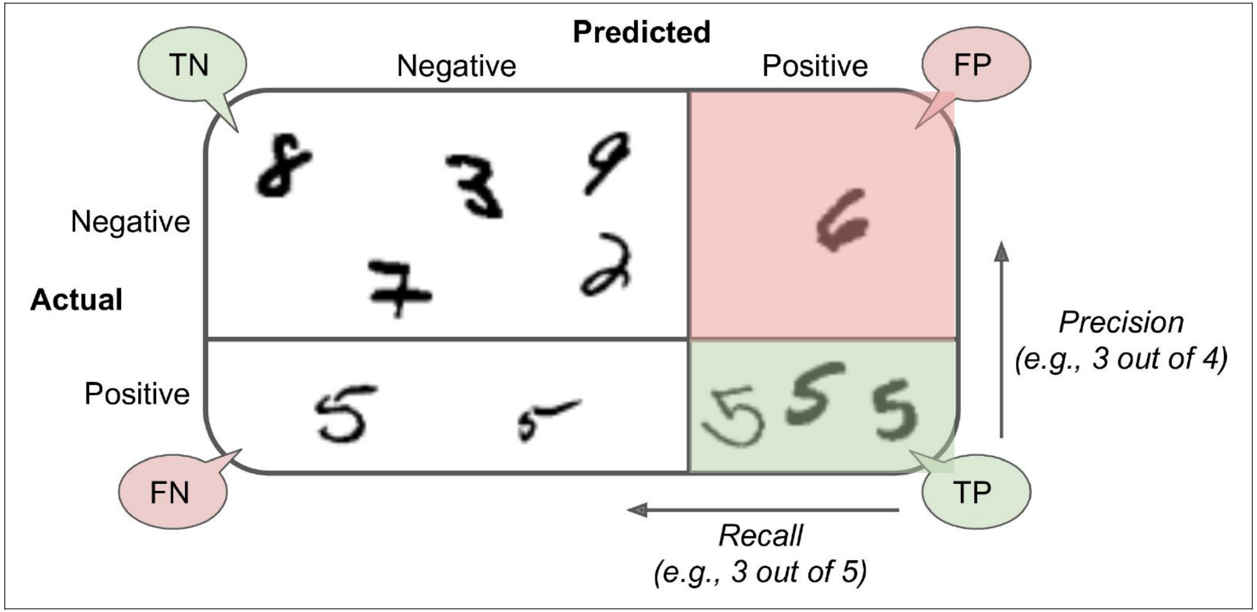

Figure 3-2. An illustrated confusion matrix

Recall

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

Sensitivity

True Positive Rate (TPR)

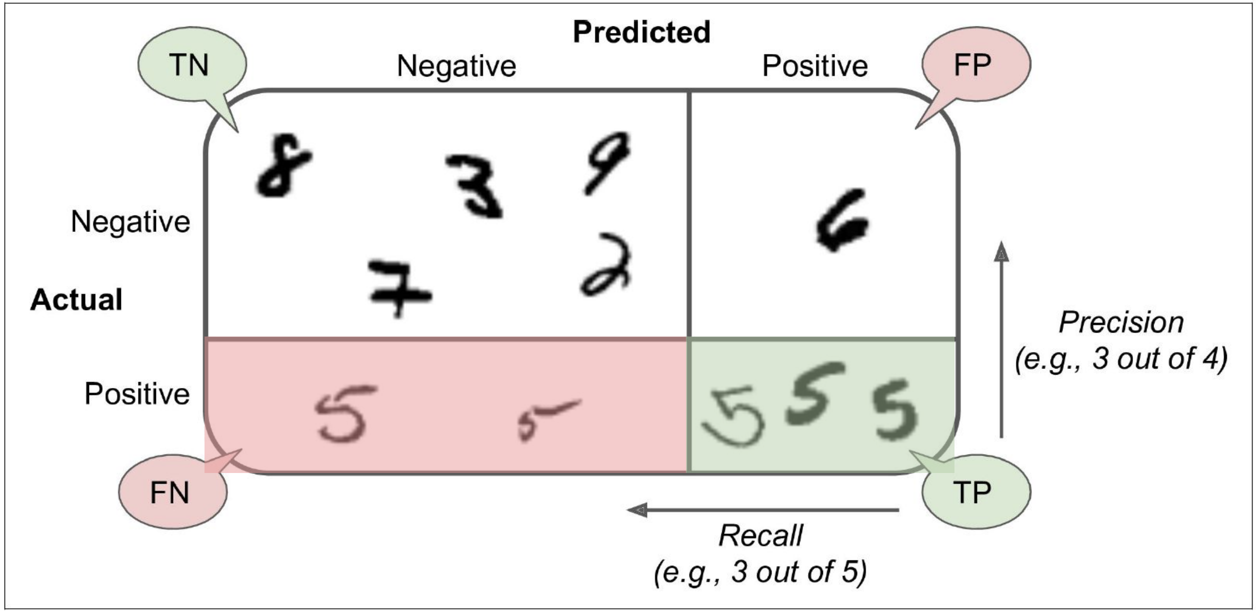

Figure 3-2. An illustrated confusion matrix

F_1 Score

Harmonic mean of precision and recall.

Regular mean treats all values equally. Harmonic mean gives more weight to low values.

Only get a high F_1 score if both recall and precision are high.

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

Precision and Recall

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654

>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044

>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

The F_1 score *favors* classifiers that have similar precision and recall.

Some contexts mostly care about precision and others recall.

Consider safe kid videos (precision), shoplifters surveillance (recall), *fraud, medical screening*.

Precision/Recall Tradeoff

Increasing precision reduces recall, and vice versa.
SGDClassifier *decision function* computes a score and makes decision based on *threshold*

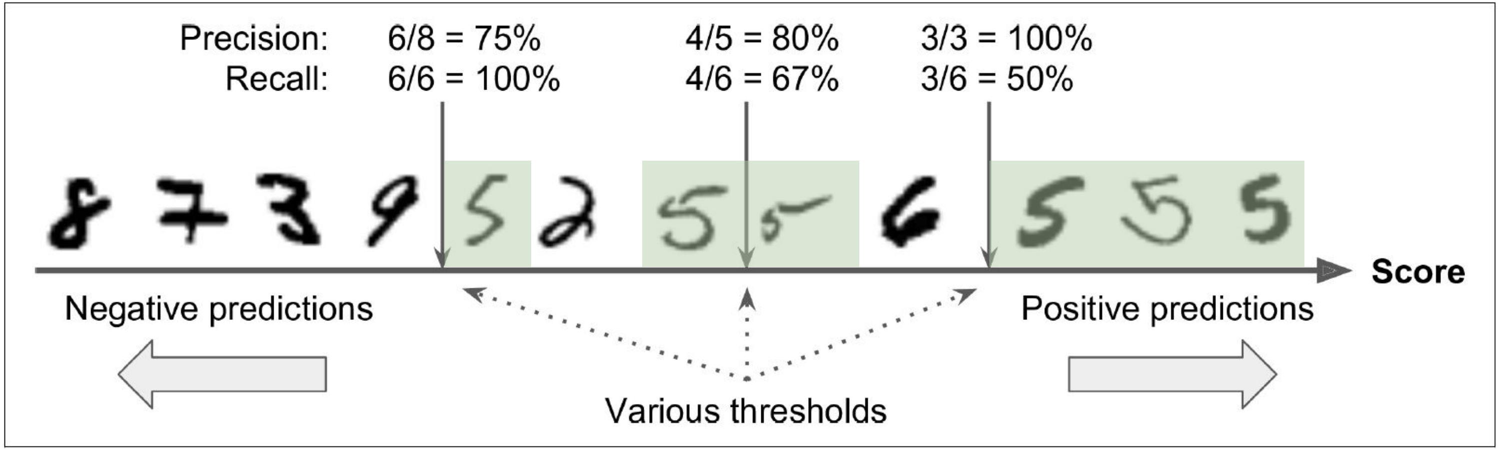

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

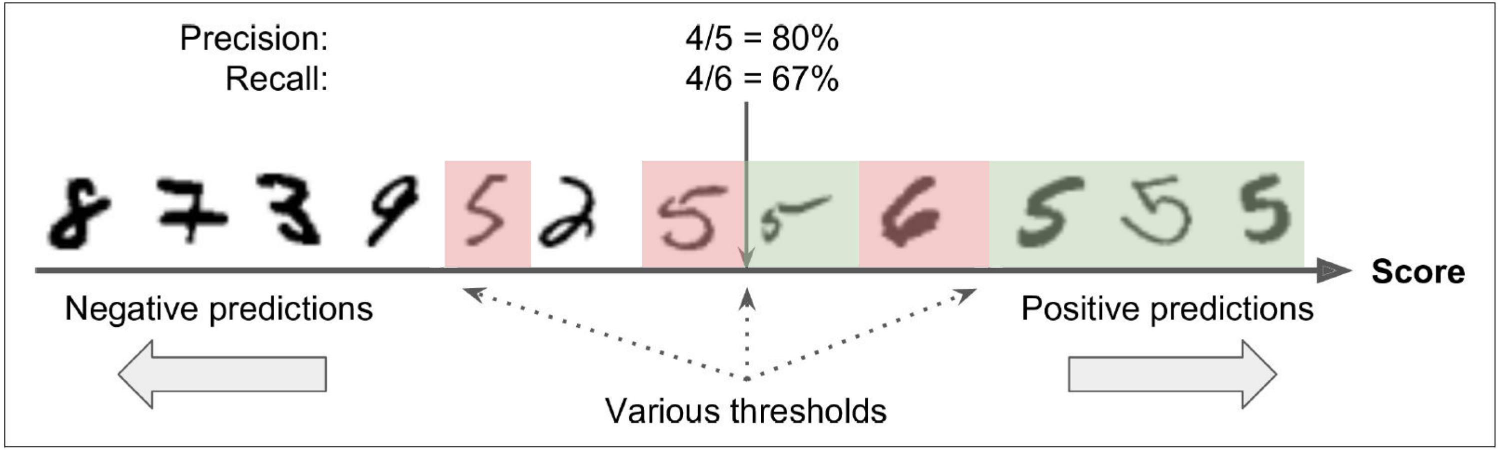

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

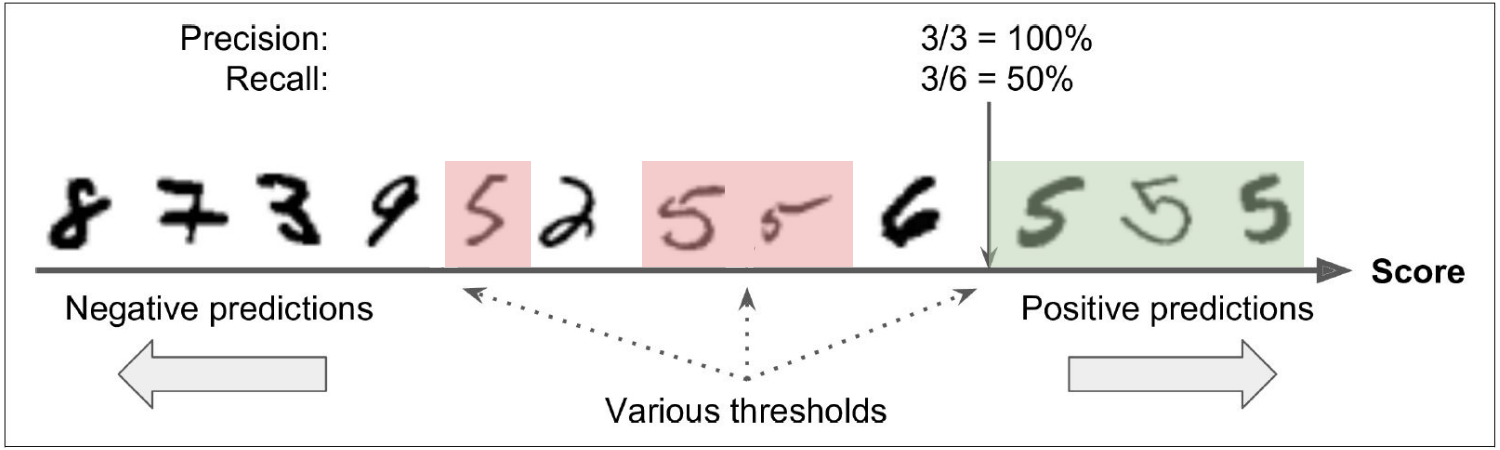

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

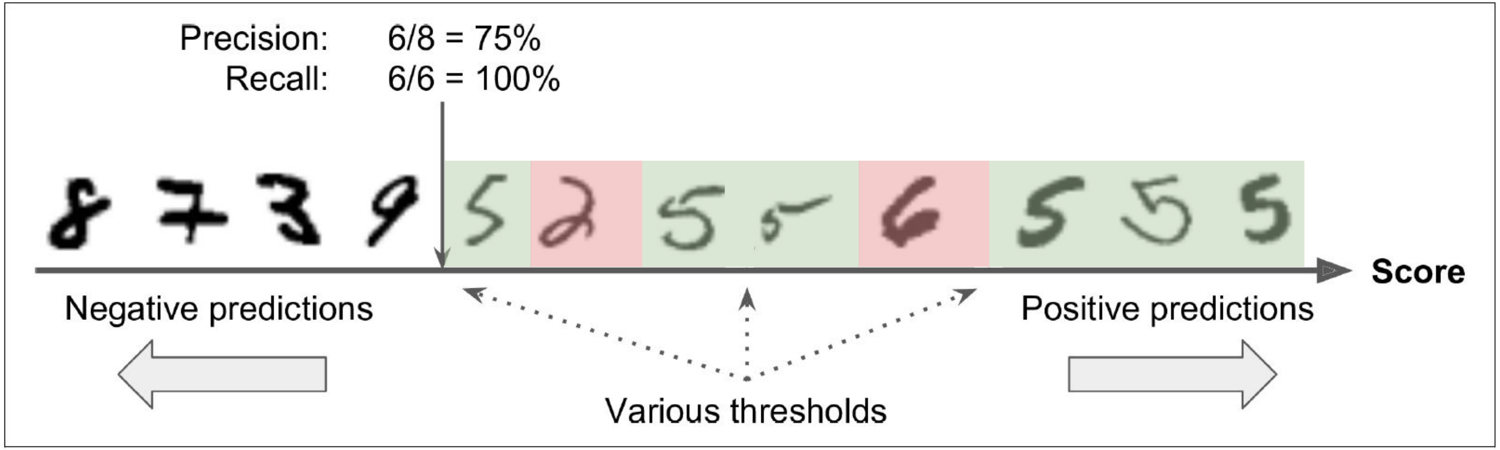

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

Increasing precision reduces recall, and vice versa.
SGDClassifier *decision function* computes a score and makes decision based on *threshold*

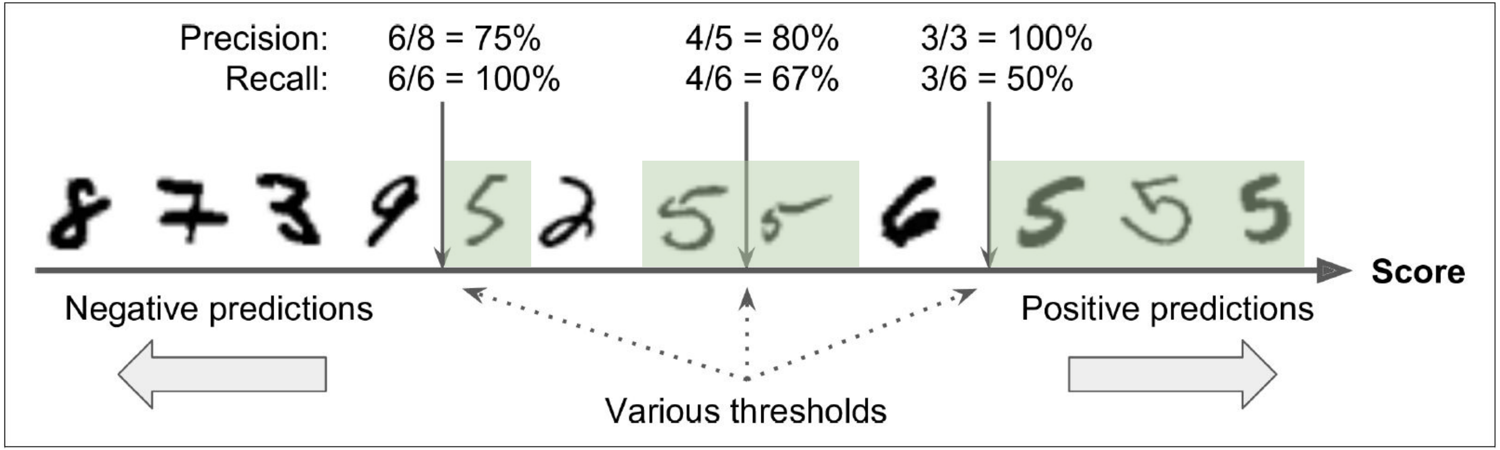

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

Cannot set threshold directly. Access decision function scores used to make predictions.

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2412.53175101])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])

>>> threshold = 8000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

Precision/Recall Tradeoff

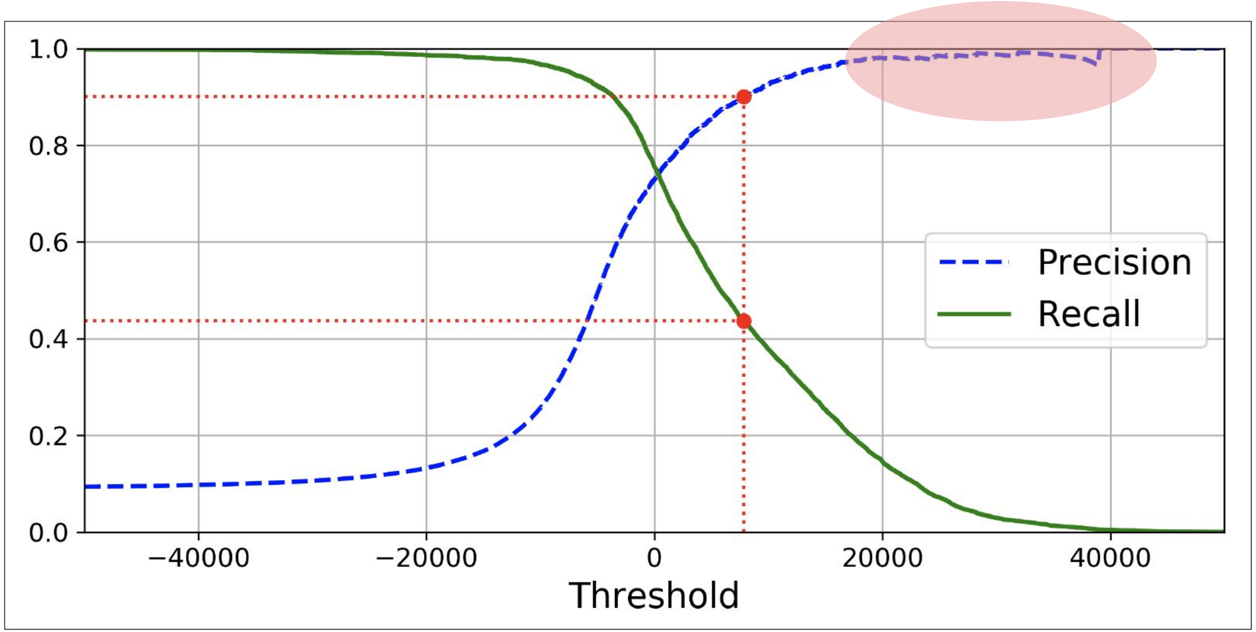

Figure 3-4. Precision and recall versus the decision threshold

Precision/Recall Tradeoff

Precision: 3/4 = 75%
Recall: 3/6 = 50%

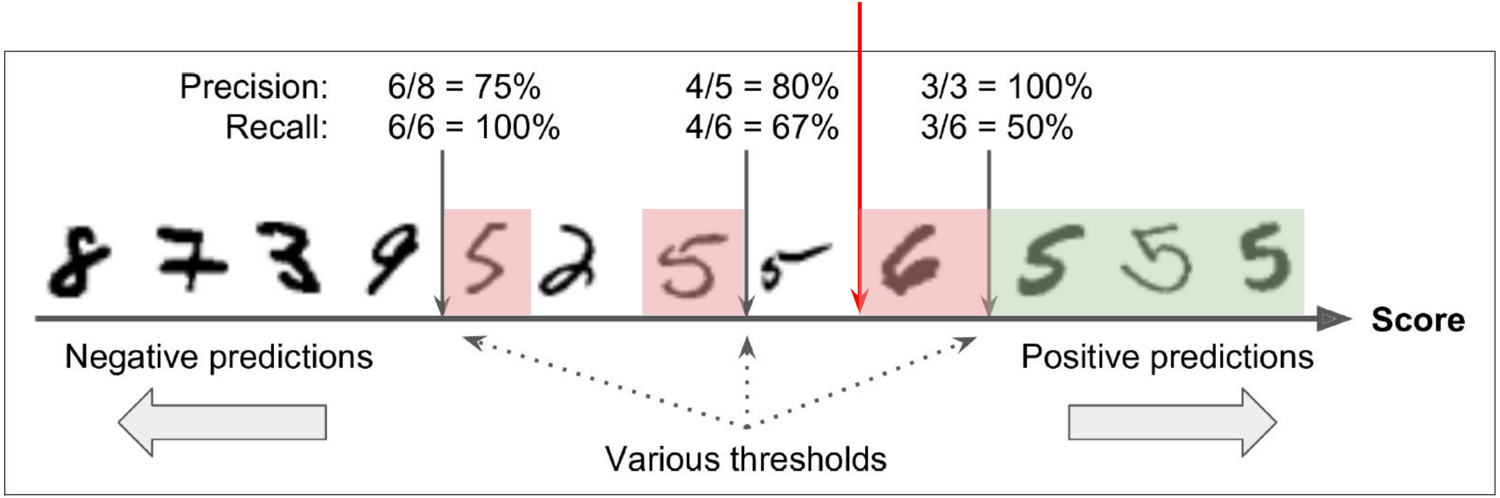

Figure 3-3. Decision threshold and precision/recall tradeoff

Precision/Recall Tradeoff

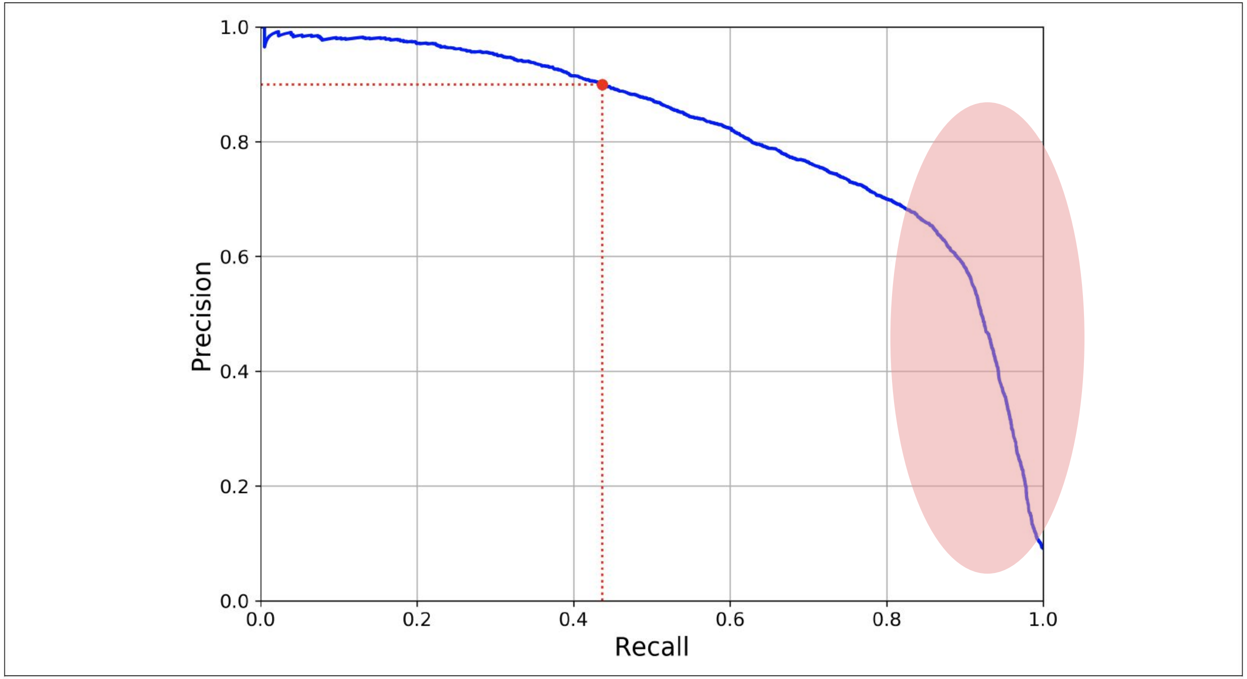

Figure 3-5. Precision versus recall

Precision/Recall Tradeoff

Use decision function scores instead of predictions

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision_function")
```

Plot precision, recall, threshold curve.

```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

Aim to find lowest threshold giving at least 90% precision

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
y_train_pred_90 = (y_scores >= threshold_90_precision)

>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

ROC Curve

Receiver operating characteristic

Another common metric for evaluating binary classifiers

Plot True Positive Rate (TPR) versus False Positive Rate (FPR)

ROC Curve

TPR vs FPR

$$\mathrm{TPR} = \frac{TP}{TP + FN}$$

True Positive Rate

Recall

Sensitivity

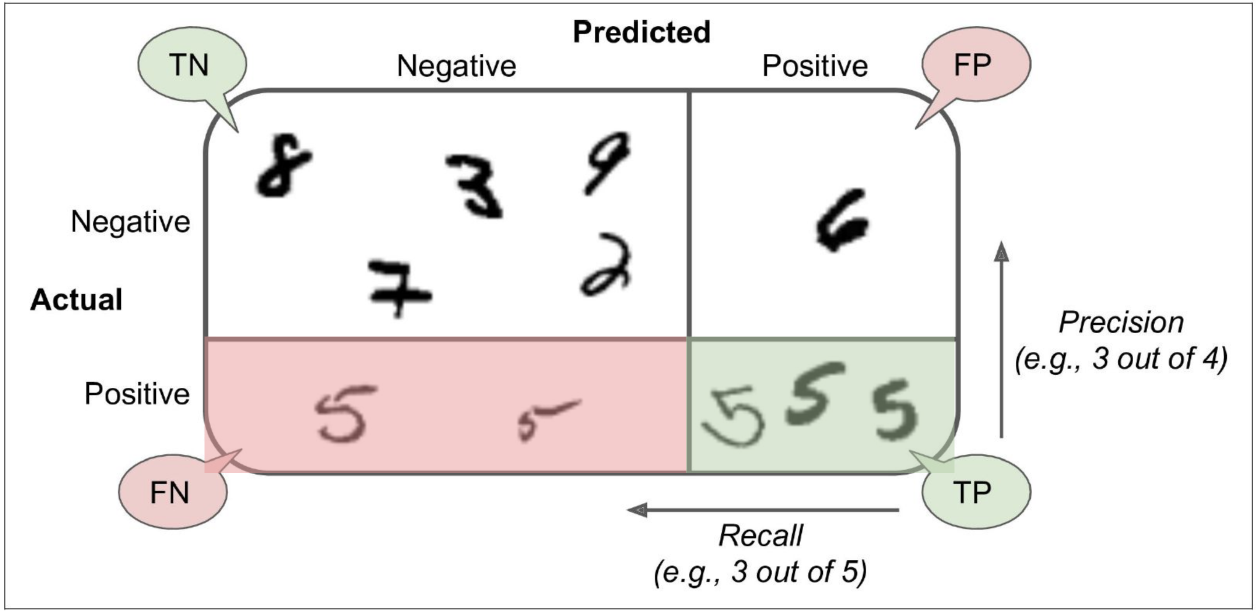

Figure 3-2. An illustrated confusion matrix

ROC Curve

TPR vs FPR

$$\text{TNR} = \frac{TN}{TN + FP}$$

True Negative Rate

Specificity

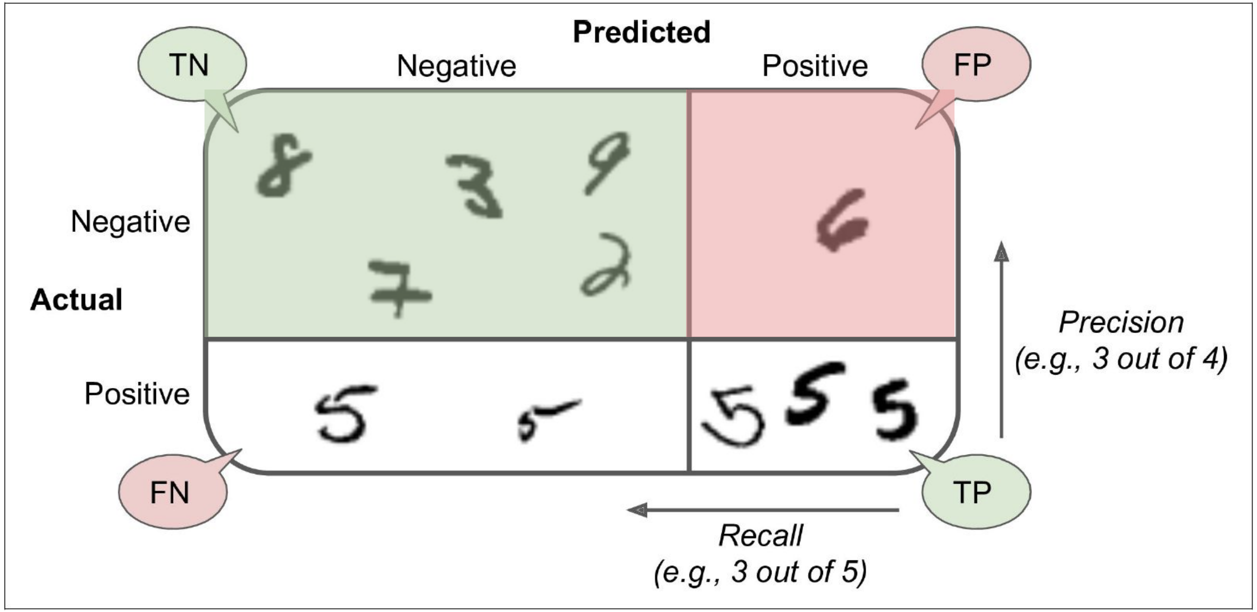

Figure 3-2. An illustrated confusion matrix

ROC Curve

TPR vs FPR

$$FPR = \frac{FP}{TN + FP}$$

False Positive Rate

1 - TNR

1 - Specificity

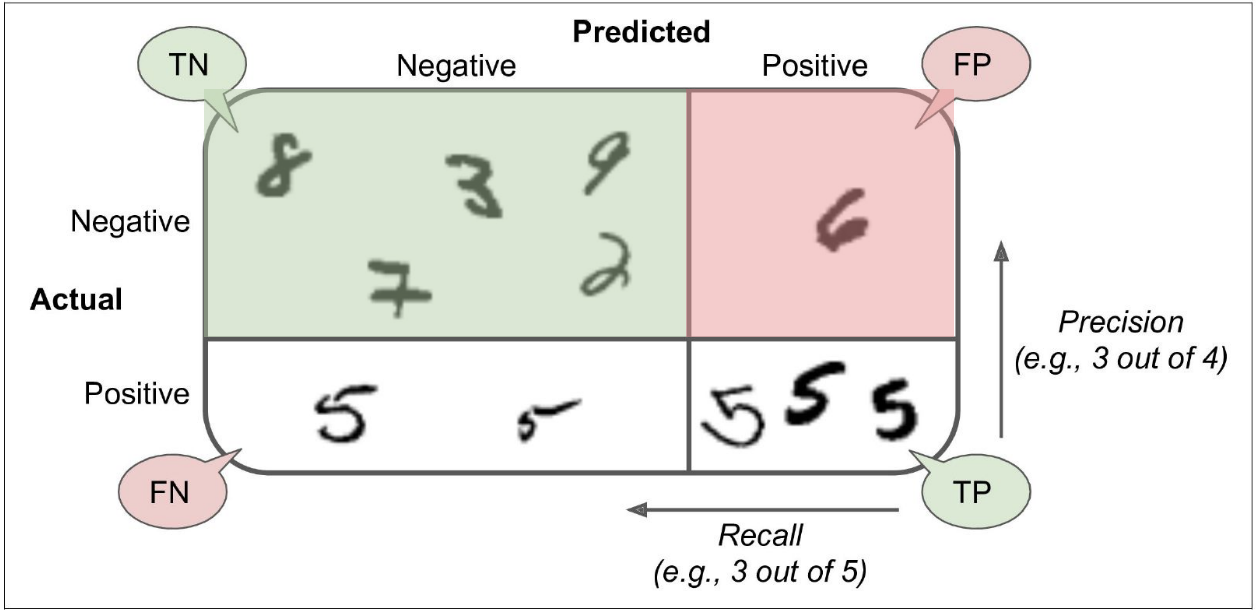

Figure 3-2. An illustrated confusion matrix

ROC Curve

TPR vs FPR

Higher the recall (TPR)
More false positives (FPR)

Random classifier dotted line
Good classifier stays top-left

Area under the curve (AUC)

Random classifier
ROC AUC = 0.5

Perfect classifier
ROC AUC = 1.0

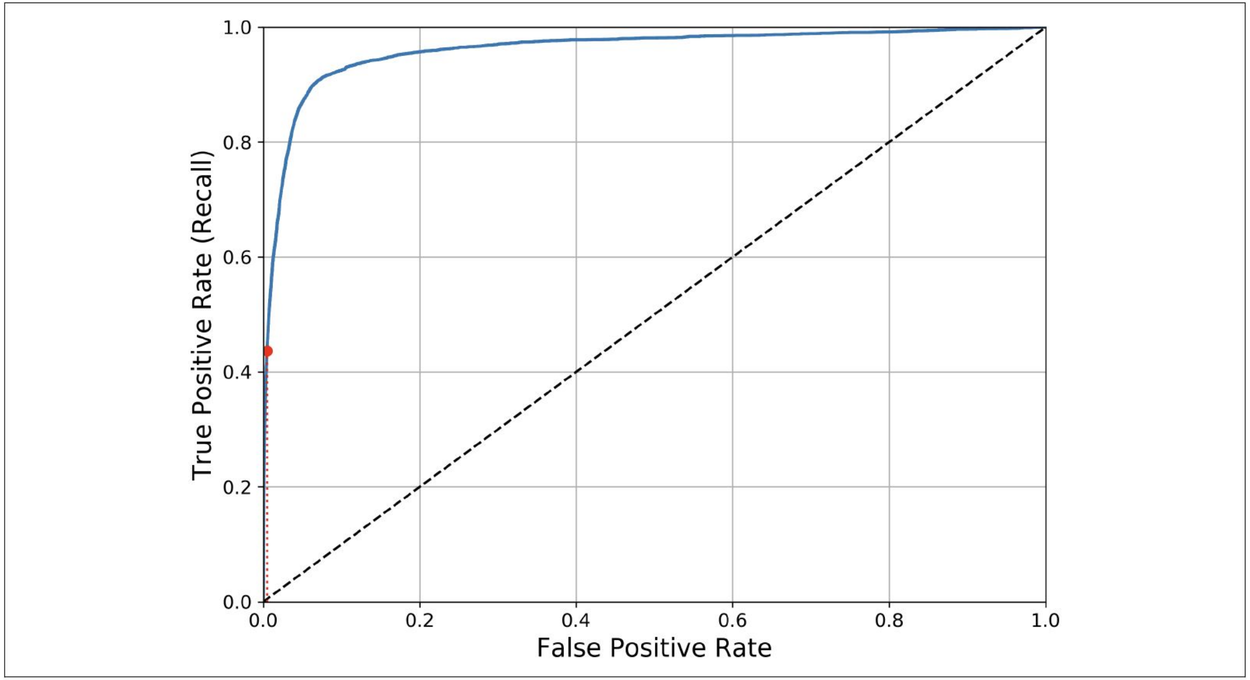

Figure 3-6. ROC curve

ROC Curve

Receiver operating characteristic. TPR (recall) vs FPR

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)

def plot_roc_curve(fpr, tpr, label=None):
      plt.plot(fpr, tpr, linewidth=2, label=label)
      plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal
      [...] # Add axis labels and grid
plot_roc_curve(fpr, tpr)
plt.show()
```

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9611778893101814
```

PR curve vs ROC curve?

PR curve: positive class is rare or FP important than FN. ROC curve otherwise

ROC Curve

Compare SGDClassifier with RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3, method="predict_proba")

y_scores_forest = y_probas_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)

plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()

>>> roc_auc_score(y_train_5, y_scores_forest)
0.9983436731328145
```

ROC Curve

Compare SGDClassifier with RandomForestClassifier

99.0% precision

86.6% recall

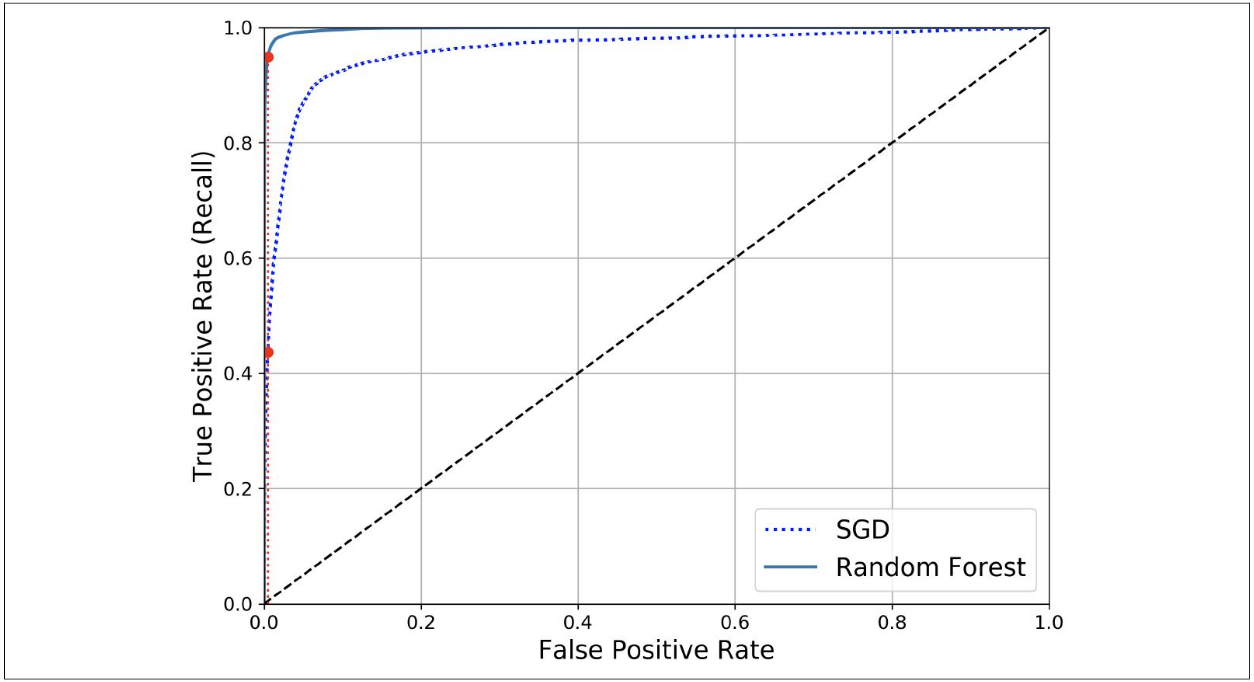

Figure 3-7. Comparing ROC curves