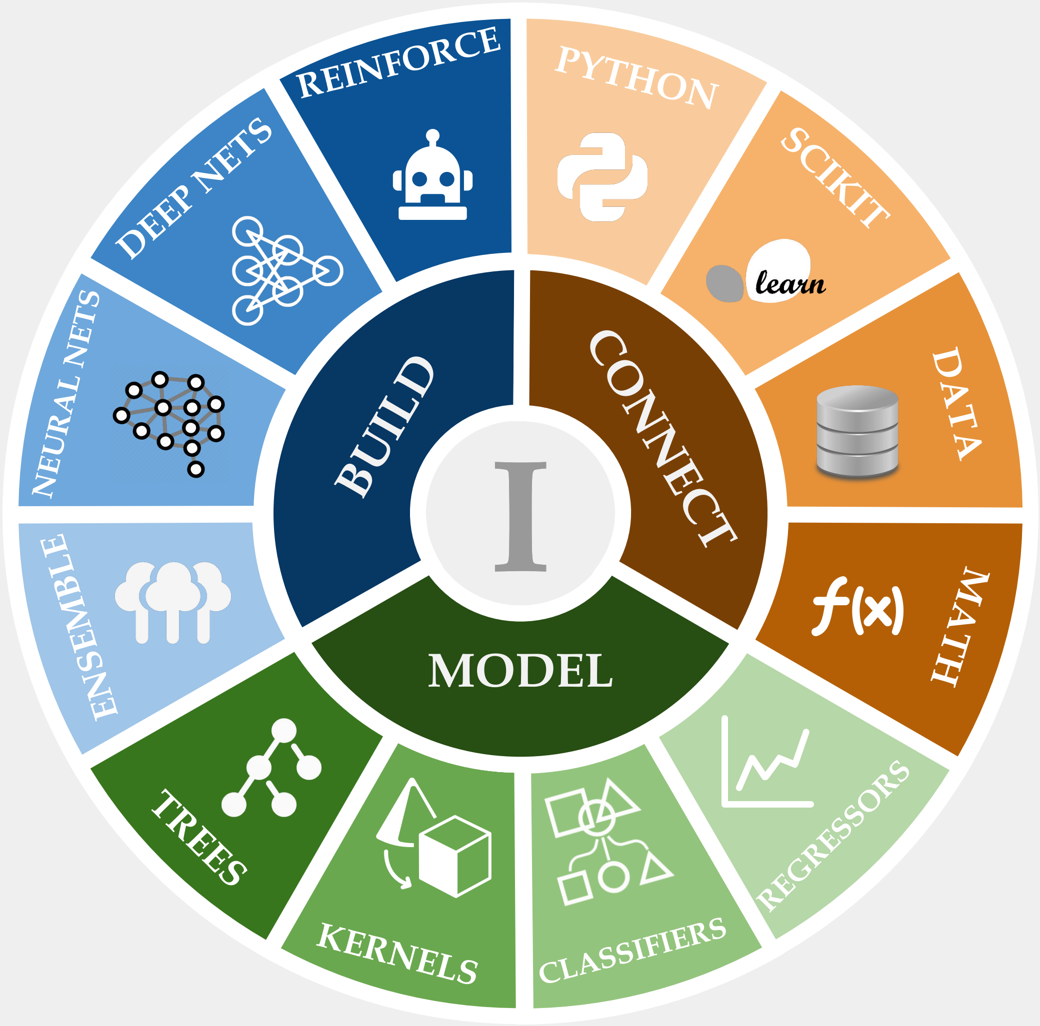
Classification

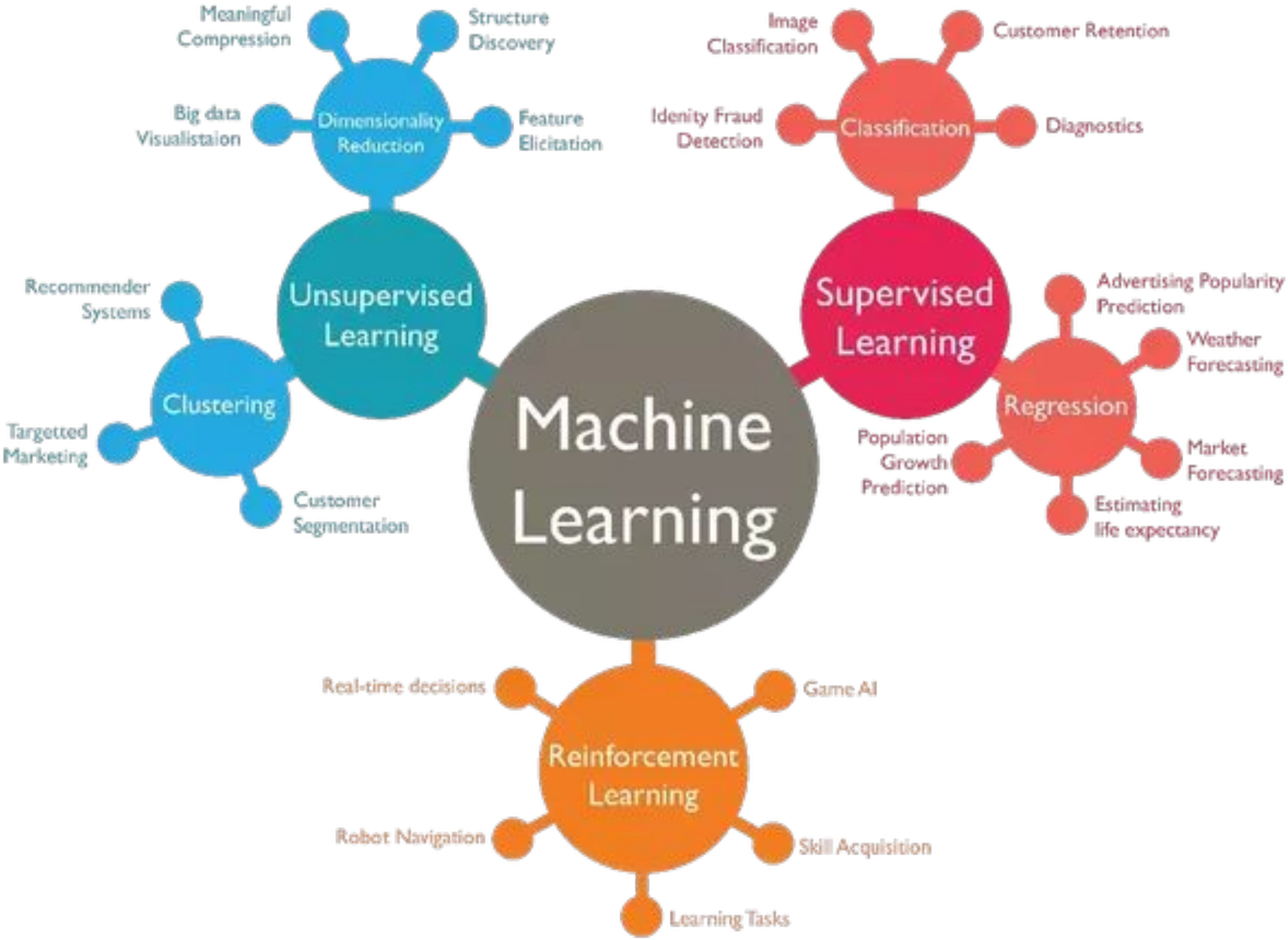
Lecture 5



I
BUILD
CONNECT
MODEL
REINFORCE
DEEP NETS
NEURAL NETS
ENSEMBLE
TREES
KERNELS
CLASSIFIERS
REGRESSORS
MATH
f(x)
DATA
SCIKIT
learn
PYTHON

Today: Classification

- ❏ Select good **performance measures** for classification tasks
- ❏ Know how to pick appropriate **precision/recall tradeoff**
- ❏ Extend to **Multiclass Classification** with One-versus-All or One-versus-One


Machine Learning
Unsupervised Learning
Dimensionality Reduction
Meaningful Compression
Structure Discovery
Big data Visualistaion
Feature Elicitation
Clustering
Recommender Systems
Targetted Marketing
Customer Segmentation
Supervised Learning
Classification
Image Classification
Customer Retention
Idenity Fraud Detection
Diagnostics
Regression
Advertising Popularity Prediction
Weather Forecasting
Population Growth Prediction
Market Forecasting
Estimating life expectancy
Reinforcement Learning
Real-time decisions
Game AI
Robot Navigation
Skill Acquisition
Learning Tasks

Terminology

- Instead of a real-value response, classification assign $\mathbf{x}$ to a category (**class**):
    - Regression: For pair $(\mathbf{x}, y)$, y is the response of $\mathbf{x}$
    - Classification: For pair $(\mathbf{x}, y)$, y is the class of $\mathbf{x}$
- Input: Measurement $x_1, ..., x_n$ in an input space
- Output: Discrete output is composed of K possible classes:
    - $\mathbf{Y} = \{-1, +1\}$ or $\{0, 1\}$ is called **binary** classification
    - $\mathbf{Y} = \{1, ..., K\}$ is called **multiclass** classification

Classification Problem Definition

Classification uses a function f (called a classifier) to map input x to class y.

$$y = f(x) : x \in \mathcal{X}, y \in \mathcal{Y}$$

Same as regression, the classification problem is twofold:

- Define the classifier f and its parameters
- Learn the classification rules using a training set of “labeled data”

MNIST dataset

- The “hello world” dataset of ML
- 70,000 small images of handwritten digits
- Written by high school students and employees of the US Census Bureau
- Each image is labeled with the digit it represents

How to load MNIST Dataset

```
[2]
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
mnist.keys()
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details', 'categories', 'url'])
```

```
[3]
X, y = mnist["data"], mnist["target"]
X.shape
```

```
(70000, 784)
```

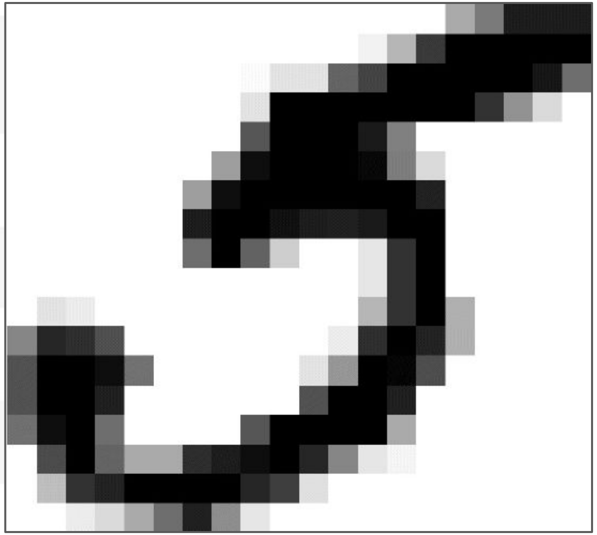
```
[4]
y.shape
```

```
(70000,)
```

```
[5]
28 * 28
```

```
784
```

- 70,000 images
- 28 x 28 pixels per image
- 784 features
- 256 intensity values



Splitting into train set and test set

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

```
import numpy as np

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

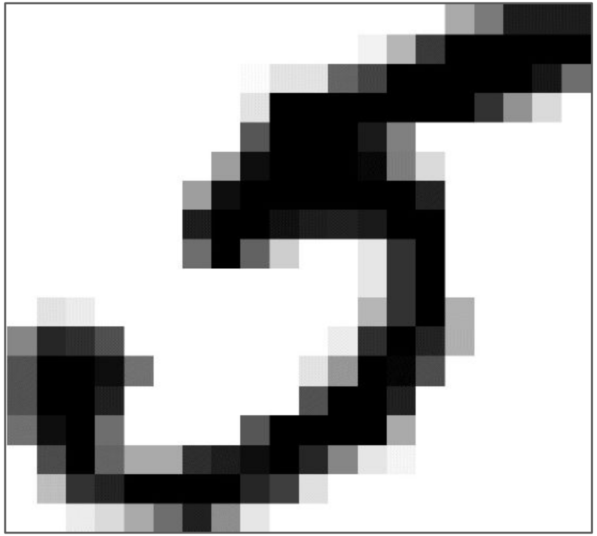
Why shuffle this training set?

Binary Classification (2-class)

- Simplify the digit identification problem into a "**detector of number 5**"

```
y_train_5 = (y_train == 5)
y_test_5 = (y_test == 5)
```

Note: Now y only contains 1s and 0s



Training a binary classifier

Pick Stochastic Gradient Descent (SGD) Classifier

- A linear classifier that use SGD to minimizes the training error
- Handling large dataset efficiently
- Dealing with training instances one at a time

```
from sklearn.linear_model import SGDClassifier

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

```
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])
```

← **Are these good enough?**

A “dumb” classifier and the imbalanced classes

```
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([ 0.909  ,  0.90715,  0.9128 ])
```

90% accuracy? Is this right?

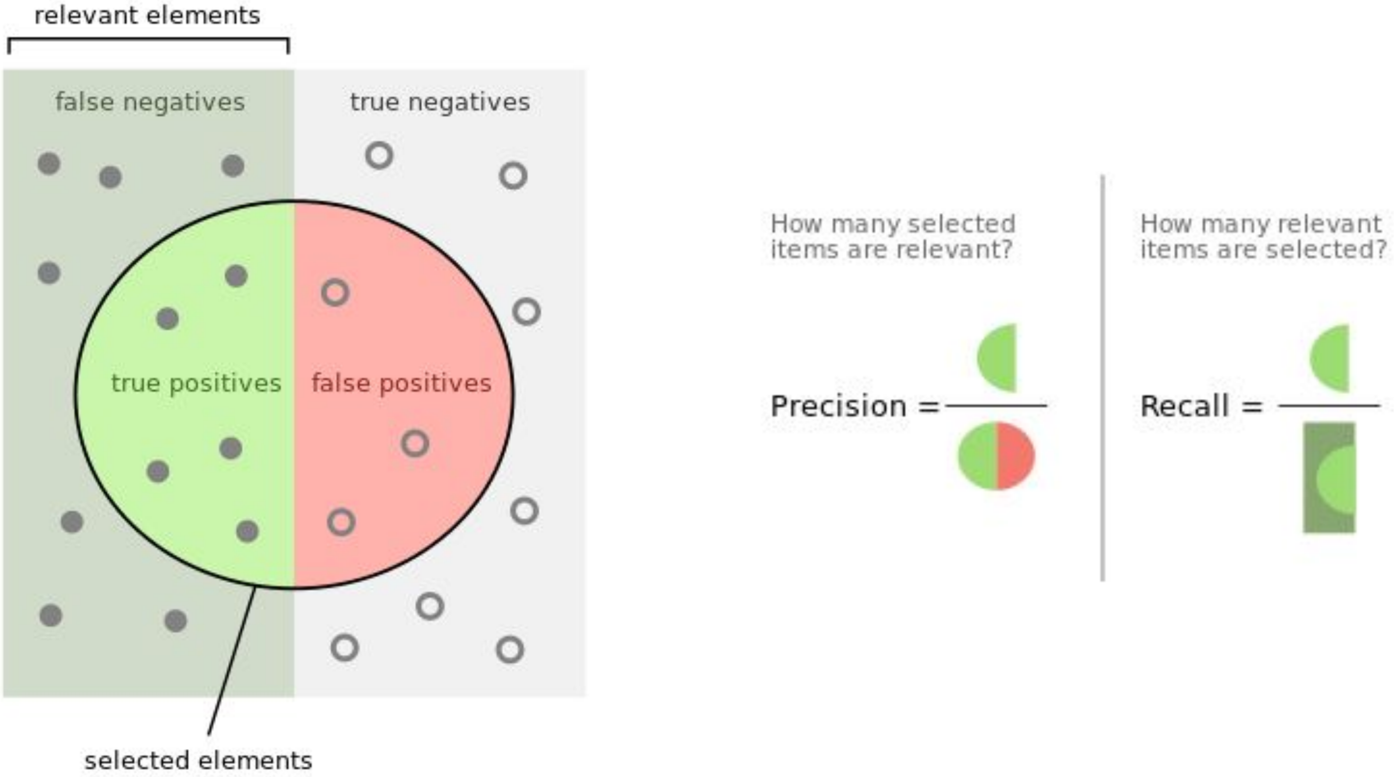
We need to find good performance measures

- Evaluating the classifier is a bit **trickier** than a regressor
- Many performance measures are available
- Let's discuss the following today:
    - Confusion Matrix
    - Precision and Recall
    - F-1 Score
    - ROC Curve
    - Area under the ROC

“Boy who cried wolf” analogy

- **True Positives:** Boy correctly called wolf. He saved the town.
- **False Negatives:**There is a wolf, but he didn’t see it. It ate all the sheeps.
- **False Positives:** Boy called wolf falsely. Everyone is mad at him.
- **True Negatives:** No wolf, no alarm. Everything is fine.

Terminology: a visual example

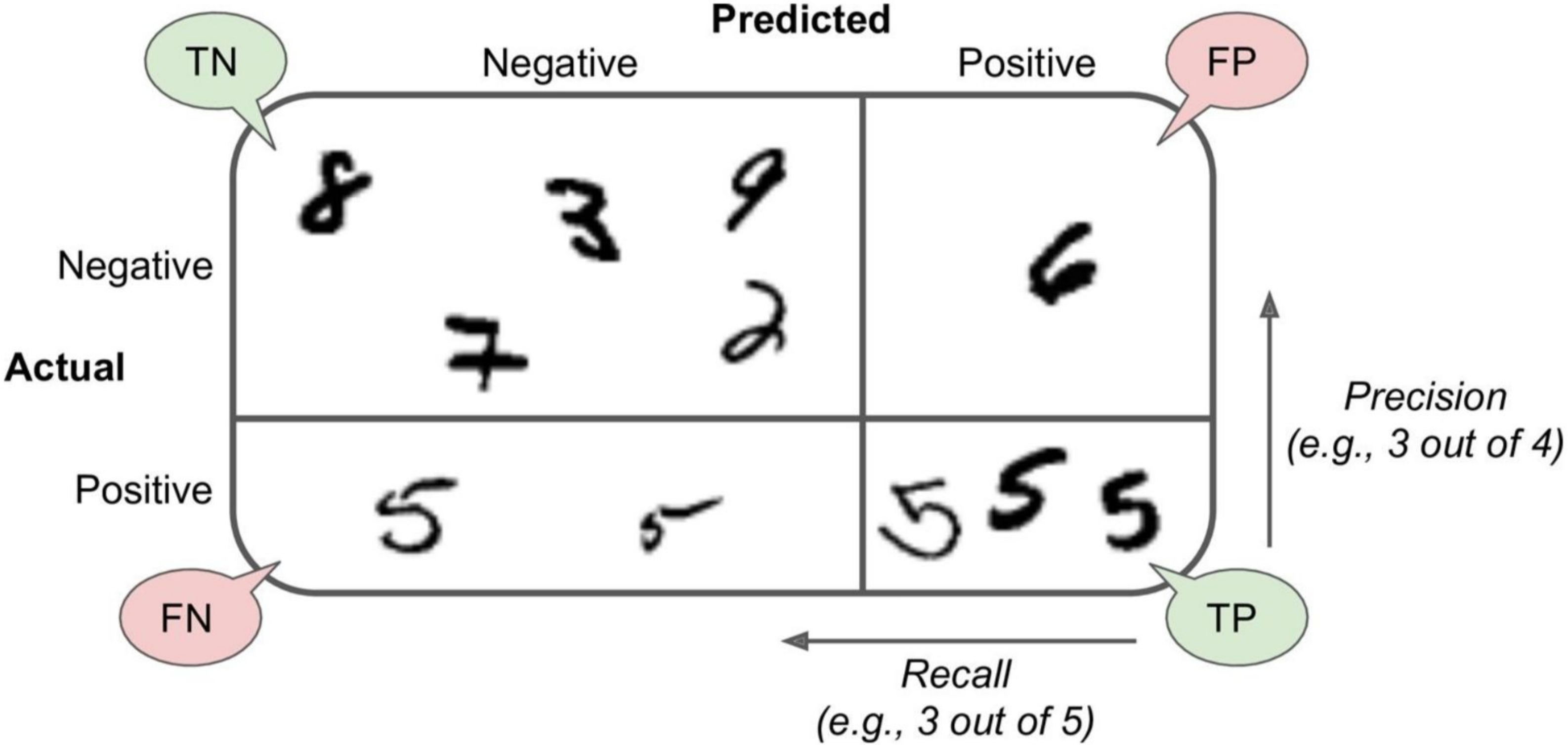


*Source: Wikipedia

“Boy who cried wolf” analogy

- **Precision**: (True Positives) / (All Positive Predictions)
    - Of all the times the boy cried wolf, how many time did he got it right?
    - Intuition: Did the model cry “wolf” too often?
- **Recall**: (True Positives) / (All Actual Positive)
    - Out of all the times the wolf comes, how many time did he got it right?
    - Intuition: Did the model miss any “wolves”?

Digit Visual Example



Confusion Matrix

- Helps gain insight about the performance of a classifier
- Counts the number of times instances are classified as a certain class

```
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([[53272,  1307],
       [ 1077,  4344]])
```

More concise metric: Precision and Recall

True Negatives (TN)

False Positives (FP)

```
array([[53272,   1307],
       [ 1077,   4344]])
```

False Negatives (FN)

True Positives (TP)

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN}$$

Code Example

```
from sklearn.metrics import precision_score, recall_score

precision_score(y_train_5, y_train_pred)
```

```
0.76871350203503818
```

```
4344 / (4344 + 1307)
```

```
0.7687135020350381
```

```
recall_score(y_train_5, y_train_pred)
```

```
0.80132816823464303
```

```
4344 / (4344 + 1077)
```

```
0.801328168234643
```

Getting more concise: F-1 Score

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

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

```
0.7846820809248547
```

```
4344 / (4344 + (1077 + 1307)/2)
```

```
0.784682080924855
```

A perfect prediction (we wish...)

```
y_train_perfect_predictions = y_train_5  # pretend we reached perfection
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

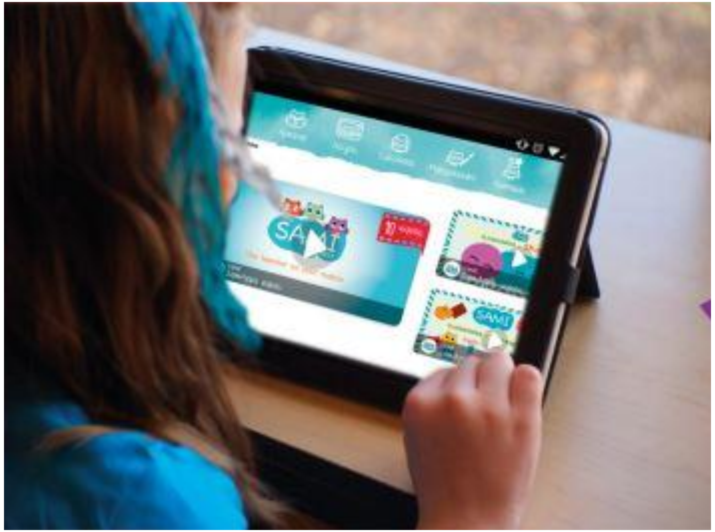
```
array([[54579,     0],
       [    0,  5421]])
```

Precision = 5421 / (5421+0) = 1

Recall = 5421 / (5421 + 0) = 1

F-score = 2 X (1 X 1) / (1 + 1) = 1

Scenario 1: Detect videos that are safe for children



Recall

vs.

Precision

Which measure should be kept **high?**

Scenario 2: Detecting shoplifter from the store videostream

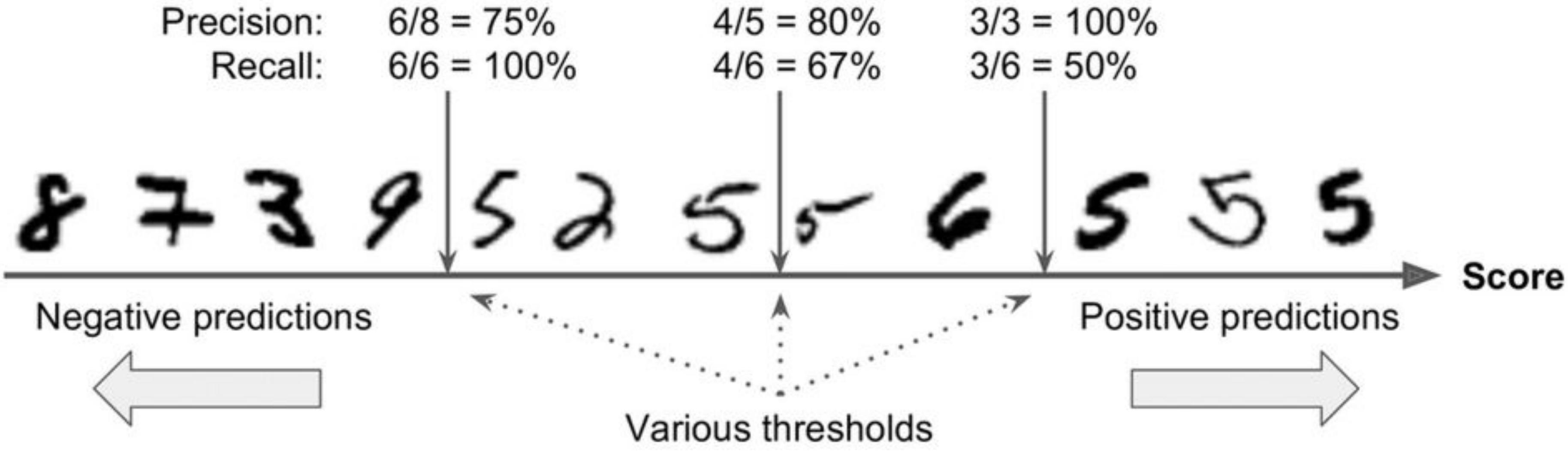
Recall

vs.

Precision

Which measure should be kept **high?**

Precision / Recall Tradeoff



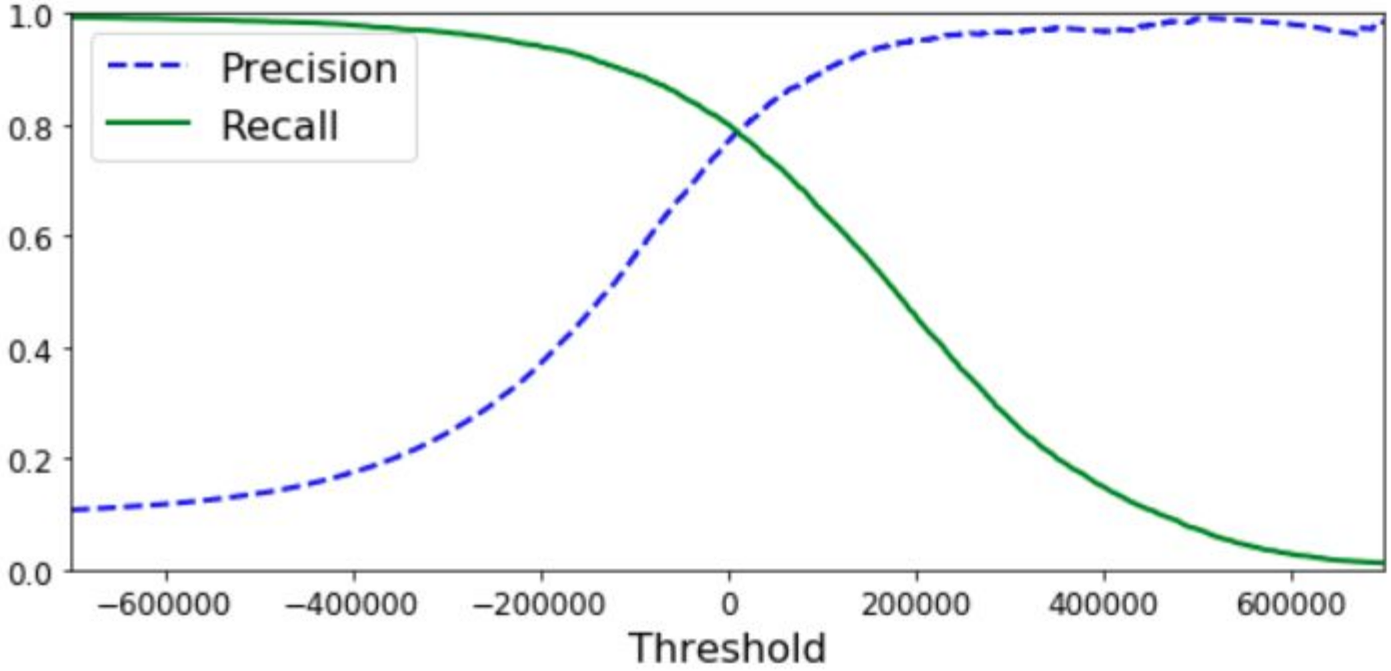
Deciding which Threshold to use?

Use Cross Validation and Prediction Function

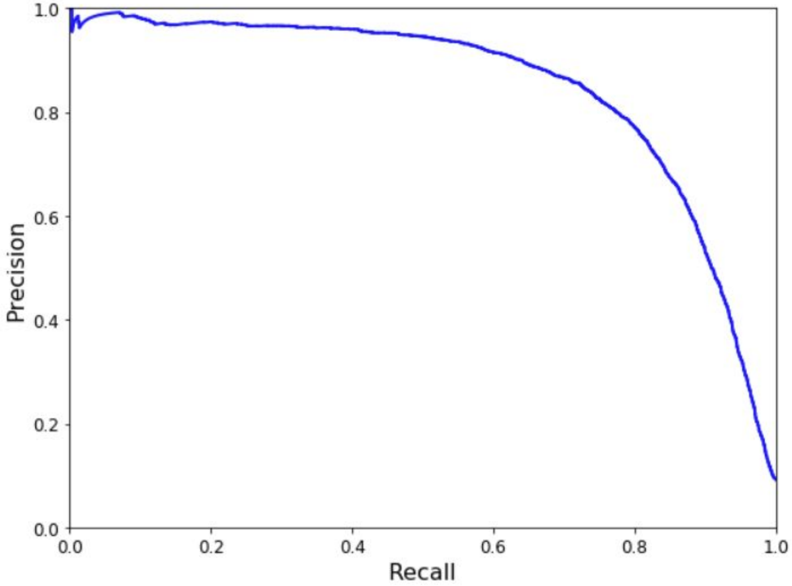
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

Precision / Recall Curve

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```



Precision versus Recall



If someone says **"I got 99% precision"**, you should ask **"Well, at what recall?"**

The ROC Curve

The Receiver Operating Characteristic Curve

Plot Recall vs. False Positive Rate FPR = $\frac{FP}{N} = \frac{FP}{FP + TN}$

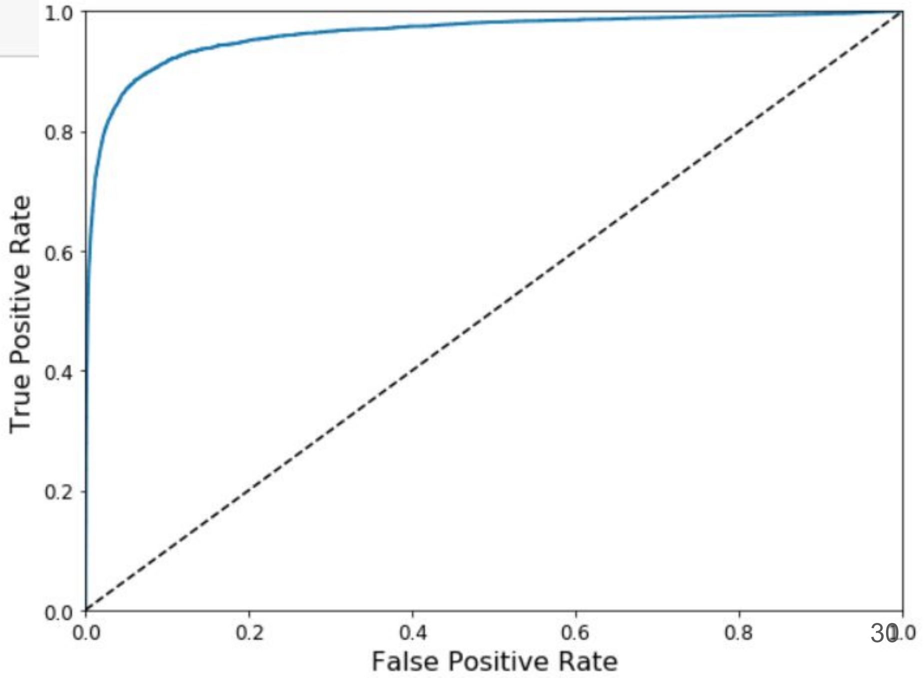
<u>**Intuition**</u>: gives a measure of performance aggregated across all possible classification thresholds

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

Area Under the Curve (AUC)

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

0.96244965559671547



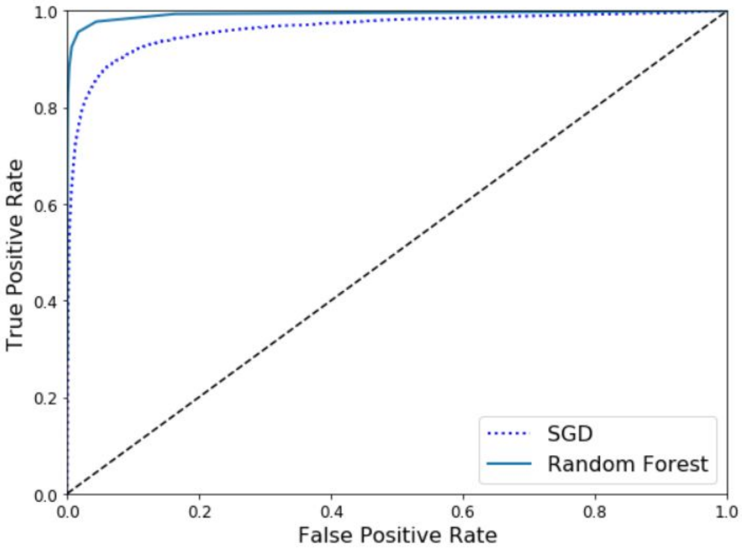
Let’s train another 5-classifier using RandomForest

```
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)
```

Performance Comparison

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right", fontsize=16)
save_fig("roc_curve_comparison_plot")
plt.show()
```



More specifically

```
roc_auc_score(y_train_5, y_scores_forest)
```

```
0.99312433660038291
```

```
y_train_pred_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3)
precision_score(y_train_5, y_train_pred_forest)
```

```
0.98529734474434938
```

```
recall_score(y_train_5, y_train_pred_forest)
```

```
0.82826046854823832
```

Now that's better! 98.5% precision and 82.8% recall

Can we classify more than just the 5s?

Multiclass Classification

While some learning algorithms (Random Forest Classifier or Naive Bayes Classifier) are capable of handling multiple classes directly, other algorithms (Support Vector Machine, Linear Classifiers) are **strictly binary.**

We need a strategy to perform multiclass classification using **multiple** binary classifier:

- One-versus-rest (OvR)
- One-versus-one (OvO)

One-versus-rest strategy

A system that can classify the digit images into 10 classes (0 to 9)

- Train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, ect)
- New image comes in, get the decision score from each classifier for the image
- Select the class whose classifier outputs the **highest score**

One-versus-one strategy

A system that can classify the digit images into 10 classes (0 to 9)

- Train a binary classifier for every pair of digits: one to classify 0s and 1s, another to classify 0s and 2s, another for 1s and 2s, ect.
- If there are **`n`** classes, we will need to train **`n(n-1)/2`** classifier. For MNIST, we will need to train 10*(10-1)=45 classifiers!
- New image comes in, run the image through all 45 classifiers and see which class wins the most duels
- **Advantage**: more discriminative training data (only 2 classes at a time)

Try it out: OneVsRestClassifier

```python
from sklearn.multiclass import OneVsRestClassifier
ovr_clf = OneVsRestClassifier(SVC(gamma="auto", random_state=42))
ovr_clf.fit(X_train[:1000], y_train[:1000])
ovr_clf.predict([some_digit])
```

```
array([5], dtype=uint8)
```

```python
len(ovr_clf.estimators_)
```

```
10
```

Try it out: OneVsOneClassifier

```
from sklearn.multiclass import OneVsOneClassifier
ovo_clf = OneVsOneClassifier(SGDClassifier(max_iter=5, random_state=42))
ovo_clf.fit(X_train, y_train)
ovo_clf.predict([some_digit])
```

```
array([5.])
```

```
len(ovo_clf.estimators_)
```

```
45
```

How’s our digit classifier doing?

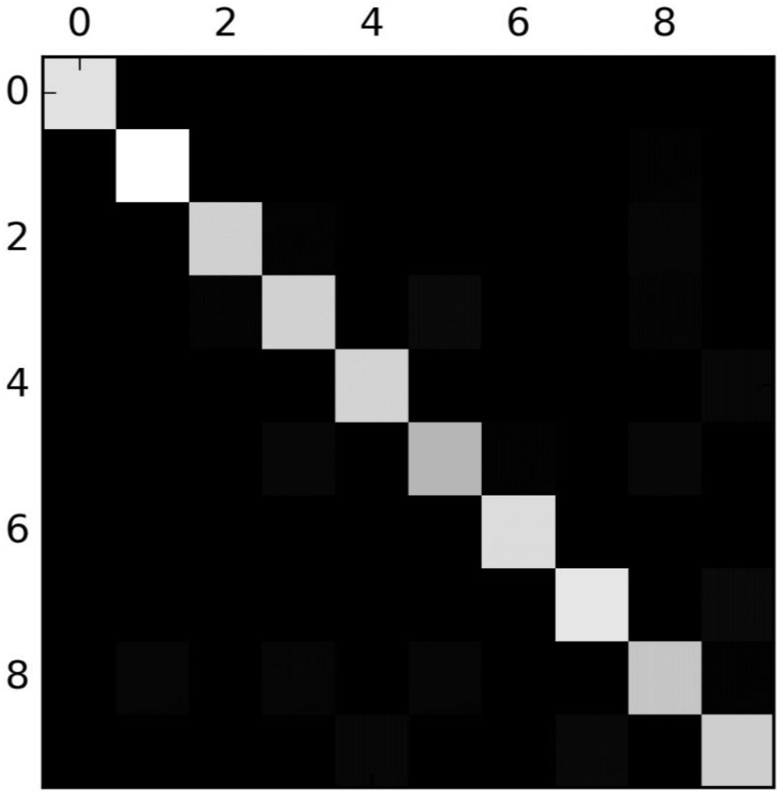
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

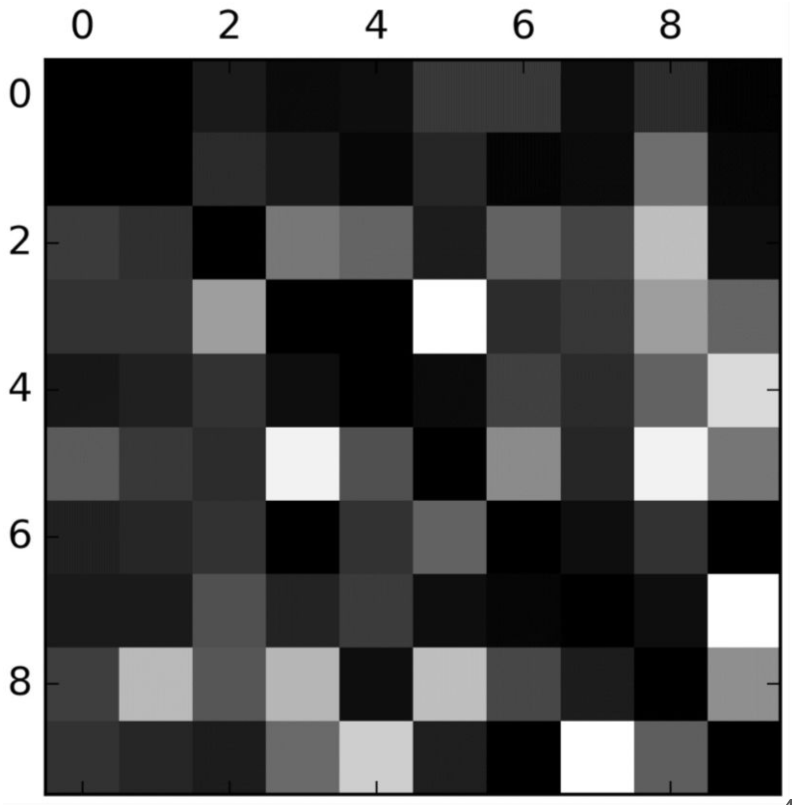
```
array([0.91011798, 0.90874544, 0.906636  ])
```

```
y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
conf_mx
```

```
array([[5725,    3,   24,    9,   10,   49,   50,   10,   39,    4],
       [   2, 6493,   43,   25,    7,   40,    5,   10,  109,    8],
       [  51,   41, 5321,  104,   89,   26,   87,   60,  166,   13],
       [  47,   46,  141, 5342,    1,  231,   40,   50,  141,   92],
       [  19,   29,   41,   10, 5366,    9,   56,   37,   86,  189],
       [  73,   45,   36,  193,   64, 4582,  111,   30,  193,   94],
       [  29,   34,   44,    2,   42,   85, 5627,   10,   45,    0],
       [  25,   24,   74,   32,   54,   12,    6, 5787,   15,  236],
       [  52,  161,   73,  156,   10,  163,   61,   25, 5027,  123],
       [  43,   35,   26,   92,  178,   28,    2,  223,   82, 5240]])
```

Confusion Matrix in image form

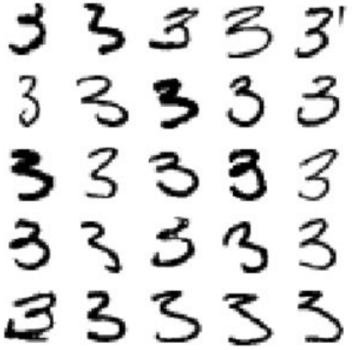




Error Analysis

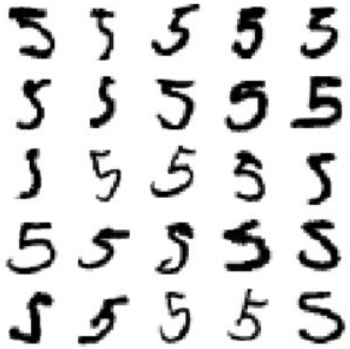
```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
save_fig("error_analysis_digits_plot")
plt.show()
```

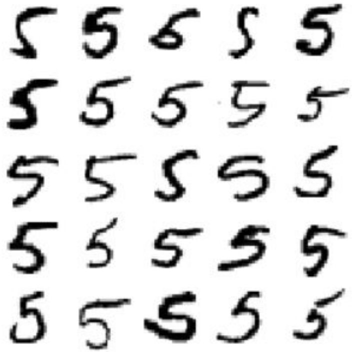
Analyze the misclassified images



examples which
were misclassified
as a 5

examples which
were misclassified
as a 3





Today: Learning Objectives

- ❏ Select good **performance measures** for classification tasks
- ❏ Know how to pick appropriate **precision/recall tradeoff**
- ❏ Extend to **Multiclass** with **One-versus-All** or **One-versus-One**

Coming next: LOGISTIC REGRESSION

Unused Slides

Confusion matrix (4 numbers for binary case)

Predicted Class

True Negatives (TN)

False Positives (FP)

Not 5 **5**

Actual Class

Not 5

5

```
array([[53272,  1307],
       [ 1077,  4344]])
```

False Negatives (FN)

True Positives (TP)