CST 407 ML – Lecture 3

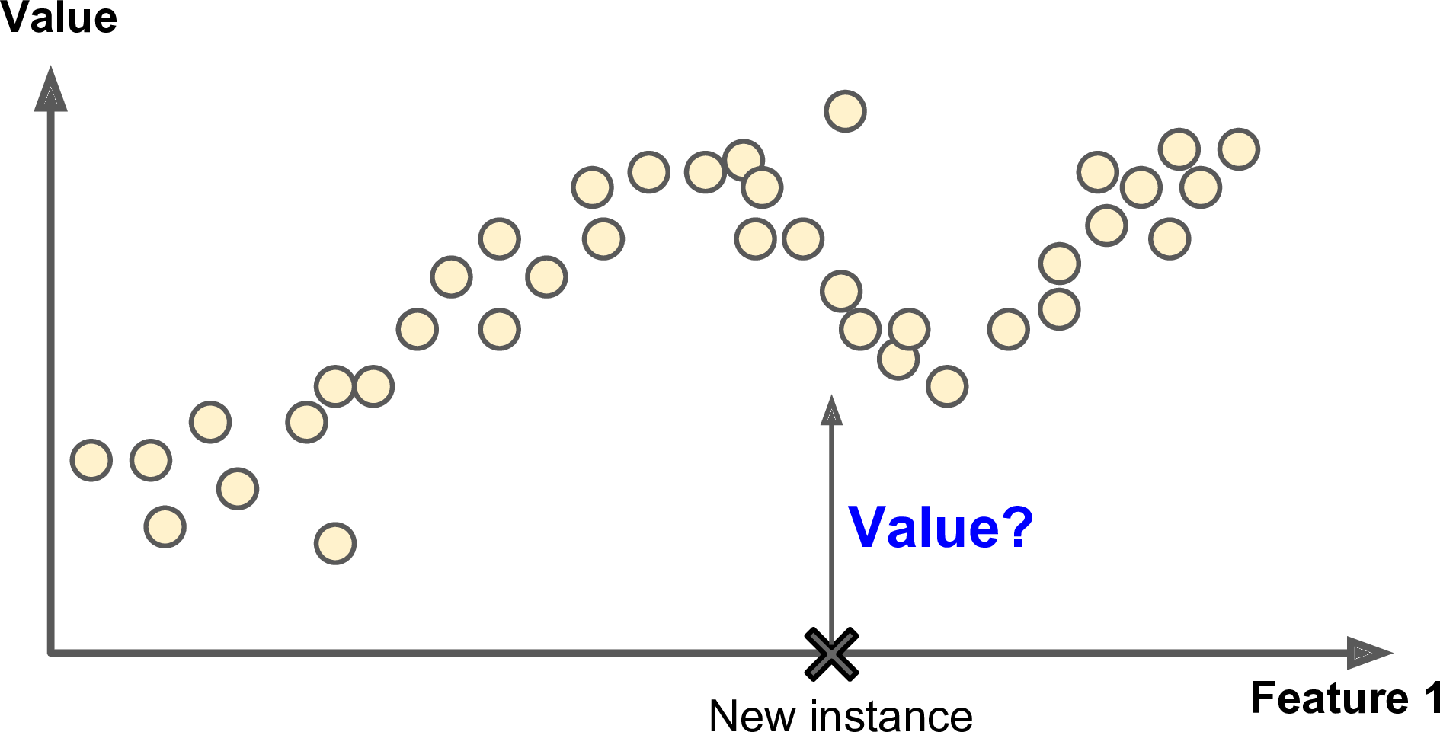
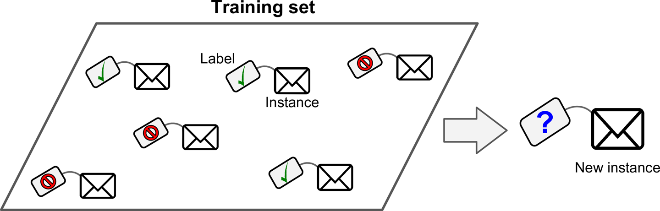
Ch 3, Classification techniques

# Classification

“classification is the problem of identifying which of a set of categories an observation belongs to. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed characteristics of the patient.” -- wikipedia

Classification model outputs are *abstract* or *categorical*

Which of the following is a classification problem and which is a regression problem?



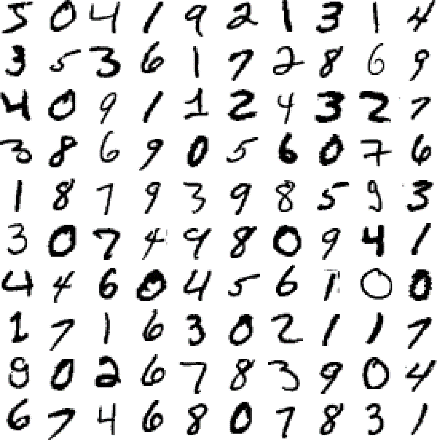
# Classification Example

From Chapter 3 of the text (Aurélien Géron)

Image Classification

“Image classification, at its very core, is the task of assigning a label to an image from a predefined set of categories. Practically, this means that our task is to analyze an input image and return a label that categorizes the image. The label is always from a predefined set of possible categories.”

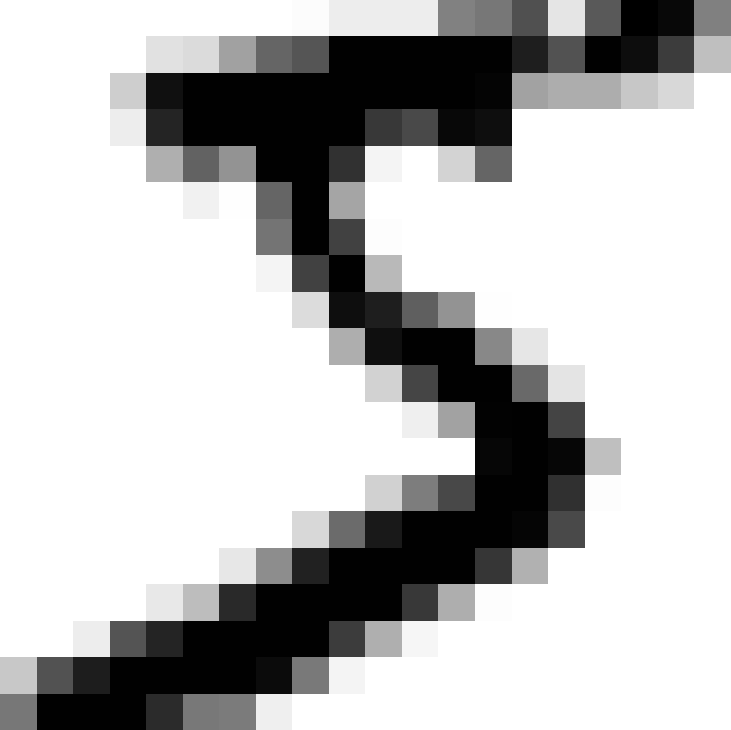
-- <https://www.pyimagesearch.com/2021/04/17/image-classification-basics/>

MNIST data set

“a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents.” -- Aurélien Géron

The challenge here? Recognize hand-written digits as one of 0 – 9

The classes are the digits and each training point will be **labeled** with one of these classes

What are the **features**?

Looking at the MNIST data set, we see 70,000 images

Each image is 28 × 28 pixels, and each pixel has intensity, from 0 (white) to 255 (black)

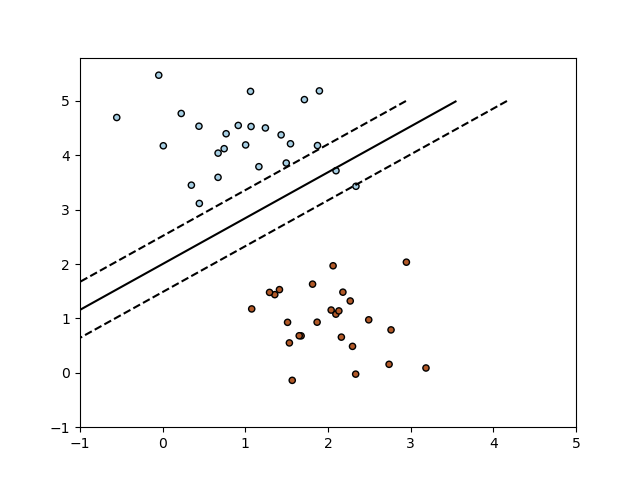
So, we have 28 x 28 = 784 features per image

The 70,000 images are pre-split into 60,000 Train and 10,000 Test instances

Why do we need **Test** data?  
We expect our trained model to predict well on future hand-written images  
We train the model with the Train set, and measure how well the model generalizes on the Test set

To start, let’s create a *binary classifier* and distinguish between just two classes: 5 and NOT-5  
We’ll use the **SGDClassifier** (Stochastic Gradient Descent classifier)

Remember “Gradient Descent”?

“Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.” -- <https://scikit-learn.org/stable/modules/sgd.html>

In SciKit-Learn, the default SGDClassifier, uses “hinge” loss function on a Support Vector Machine (SVM)

from sklearn.linear\_model import SGDClassifier

How do we know the classification worked well?

Measuring Accuracy Using Cross-Validation

**Accuracy** = # classified correctly / # should be classified, e.g.  
(15 classified as 5) / (17 should be classied as 5) = 88% accurate

“K-fold cross-validation means splitting the training set into K folds (in this case, three), 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

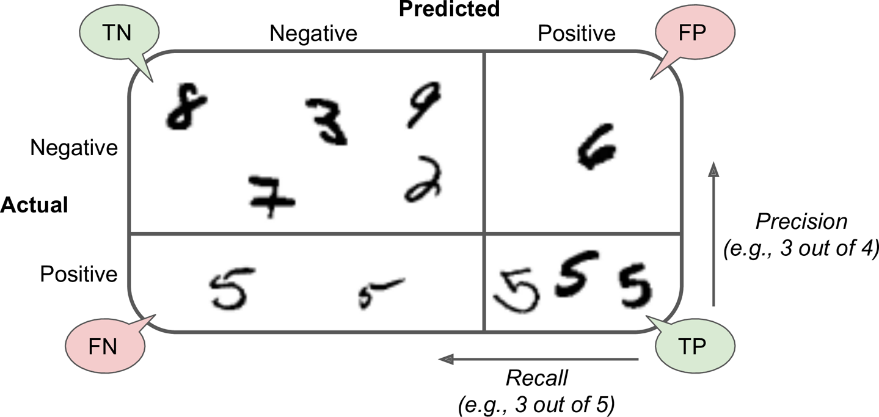
cross\_val\_score(sgd\_clf, X\_train, y\_train\_5, cv=3, scoring="accuracy")

array([0.96355, 0.93795, 0.95615])

93%-95% accuracy sounds great, right?

Until you think about the problem here… 5’s are 1 out of 10 possible classes  
Randomly guessing 5 vs NOT-5 will give us ~91% accuracy

“This demonstrates why accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others).”

Better? **Confusion Matrix**

TN = True Negative

FN = False Negative

FP = False Positive

TP = True Positive

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y\_train\_5, y\_train\_pred)

array([[53057, 1522],

[ 1325, 4096]])

precision = 4096 / (4096+1522) = 73% recall = 4096 / (4096+1325) = 76%

Okay, that’s actually not that good!

from sklearn.metrics import precision\_score, recall\_score

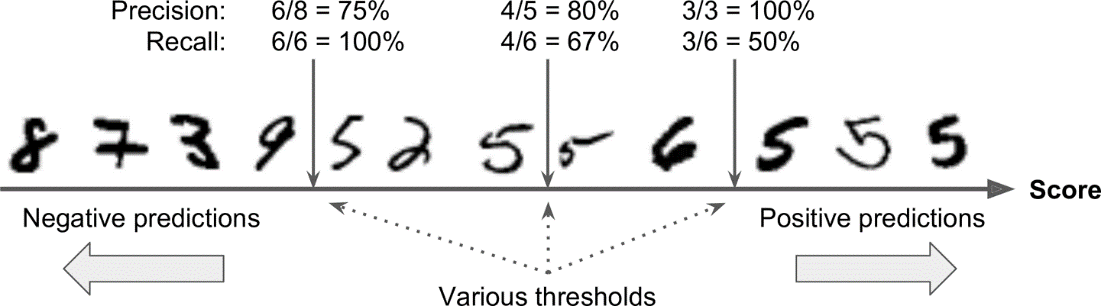
precision\_score(y\_train\_5, y\_train\_pred)

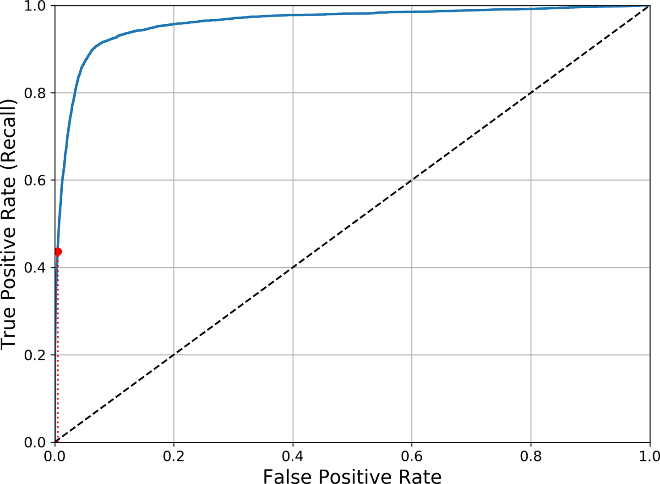
0.7290850836596654

recall\_score(y\_train\_5, y\_train\_pred)

0.7555801512636044

It’s possible to “tune” a classifier to trade off accuracy and recall, by setting the “threshold”…



Another way to look at this is the **ROC Curve**

This compares True Positive vs False Positive

For a particular trained model, we can achieve higher TP at the cost of more FP

The better the ROC curve, the more predictive the model

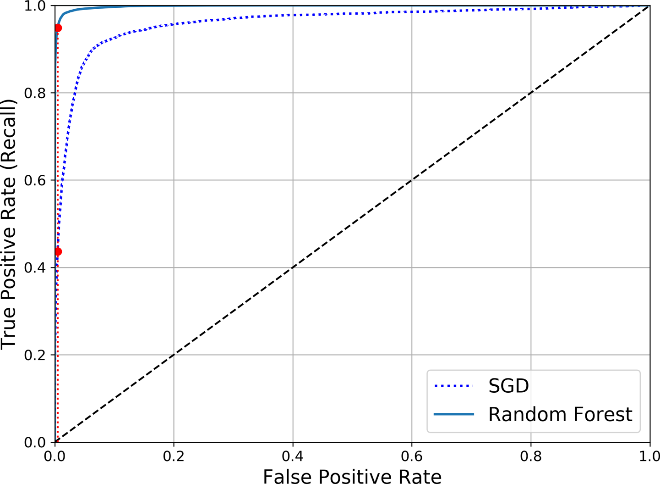
Compare classifiers by measuring the area under the curve (AUC)…  
Perfect classifier will have a ROC AUC equal to 1  
Purely random classifier will have a ROC AUC equal to 0.5

The SGDClassifier above scores 0.961 ROC AUC

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score(y\_train\_5, y\_scores)

0.9611778893101814

Choosing a different model, for example, a **RandomForestClassifier** and training it with the same data leads to a better curve

And a much better score: 0.998 ROC AUC

roc\_auc\_score(y\_train\_5, y\_scores\_forest)

0.9983436731328145



“A random forest is a meta estimator that fits a number of **decision tree classifiers** on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.”

-- <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

-- <https://en.wikipedia.org/wiki/Random_forest>