CITS5508 Machine Learning
Overview of a Machine Learning project

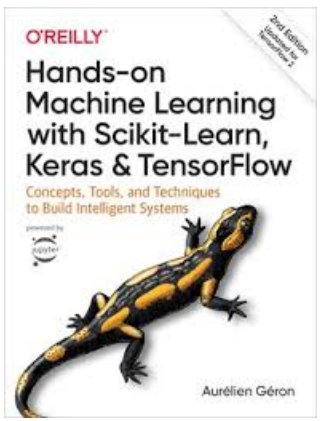
Débora Corrêa (Unit Coordinator and Lecturer)

2023

Today

Chapter 2 and Chapter 3.

Hands-on Machine Learning with Scikit-Learn & TensorFlow



End-to-End Machine Learning Project

In this chapter, you will go through an example project end to end, pretending to be a recently hired data scientist in a real estate company. Here are the main steps you will go through:

- Look at the big picture.
- Get the data.
- Explore and visualize the data to gain insights.
- Prepare the data for Machine Learning algorithms.
- Select a model and train it (can include fine-tuning).
- Present your solution.
- Launch, monitor, and maintain your system.

Working with real data

See, for example, text book for a list of places to get large public data sets.

We use a version of some old California housing prices.

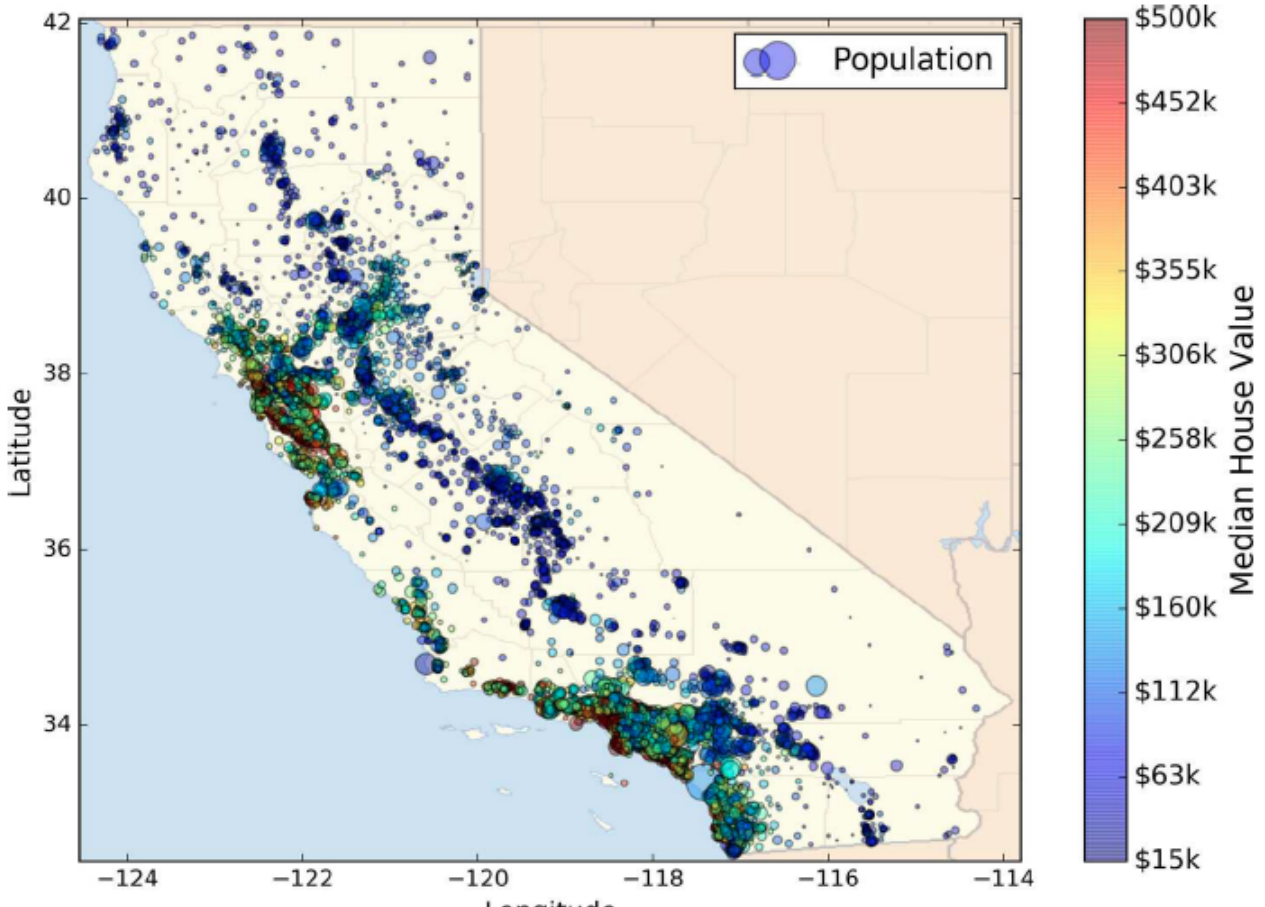


Figure 2-1. California housing prices

Frame the problem

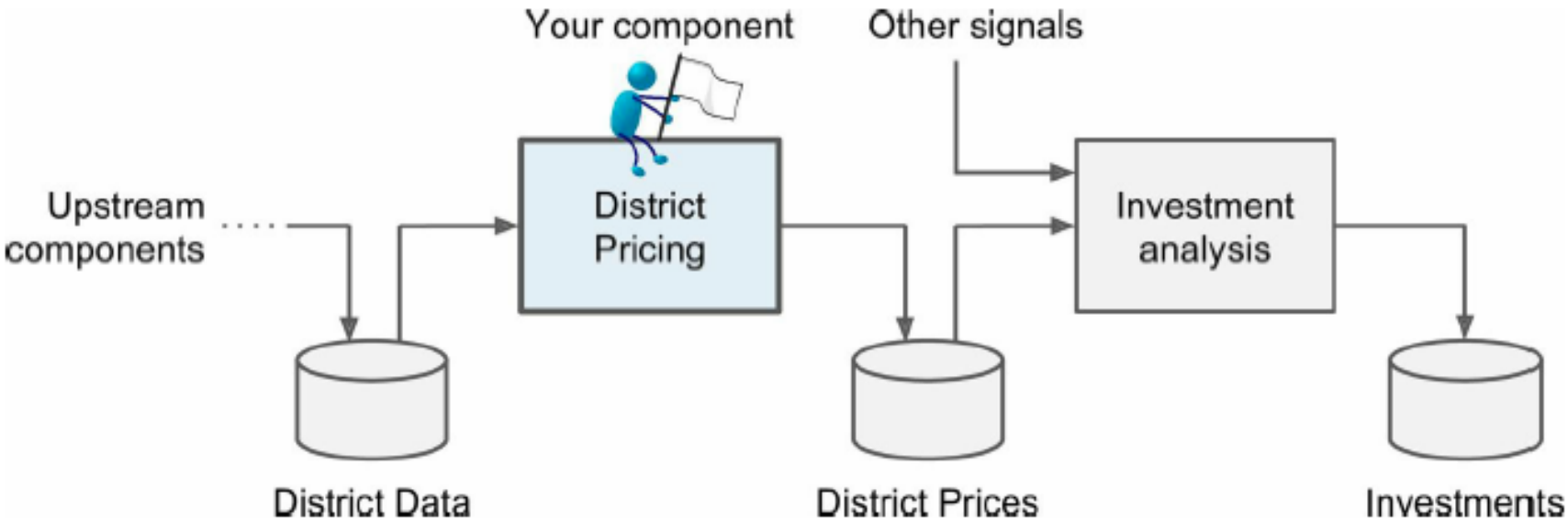


Figure 2-2. A Machine Learning pipeline for real estate investments

Example of a data pipeline. (Asynchronous).
Task is a supervised, regression problem.

Select a performance measure

Common options:
Root Mean Square Error:

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

where $\mathbf{X} = [(\mathbf{x}^{(i)})^T]_{i=1}^{m}$ is the matrix of feature values, $y^{(i)}$ is the label of the i-th training instance, and h is your prediction function.

Mean Absolute Error:

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h(\mathbf{x}^{(i)}) - y^{(i)}\right|$$

Also, **check assumptions** before proceeding.

Get the data

Go through the text book chapter 2 yourselves to set up a Jupyter notebook to analyse the downloaded the data.

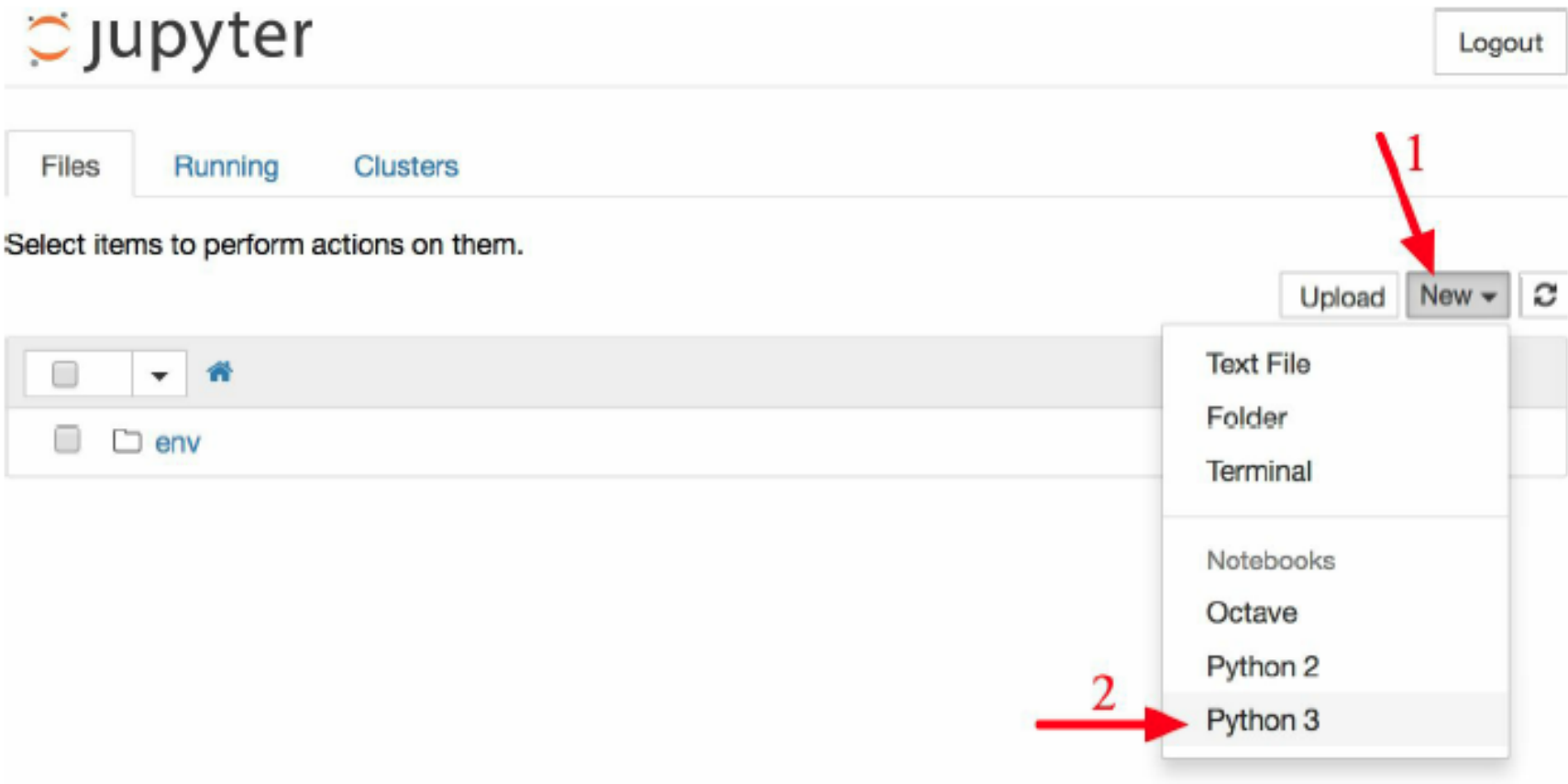


Figure 2-3. Your workspace in Jupyter

Try a small example notebook

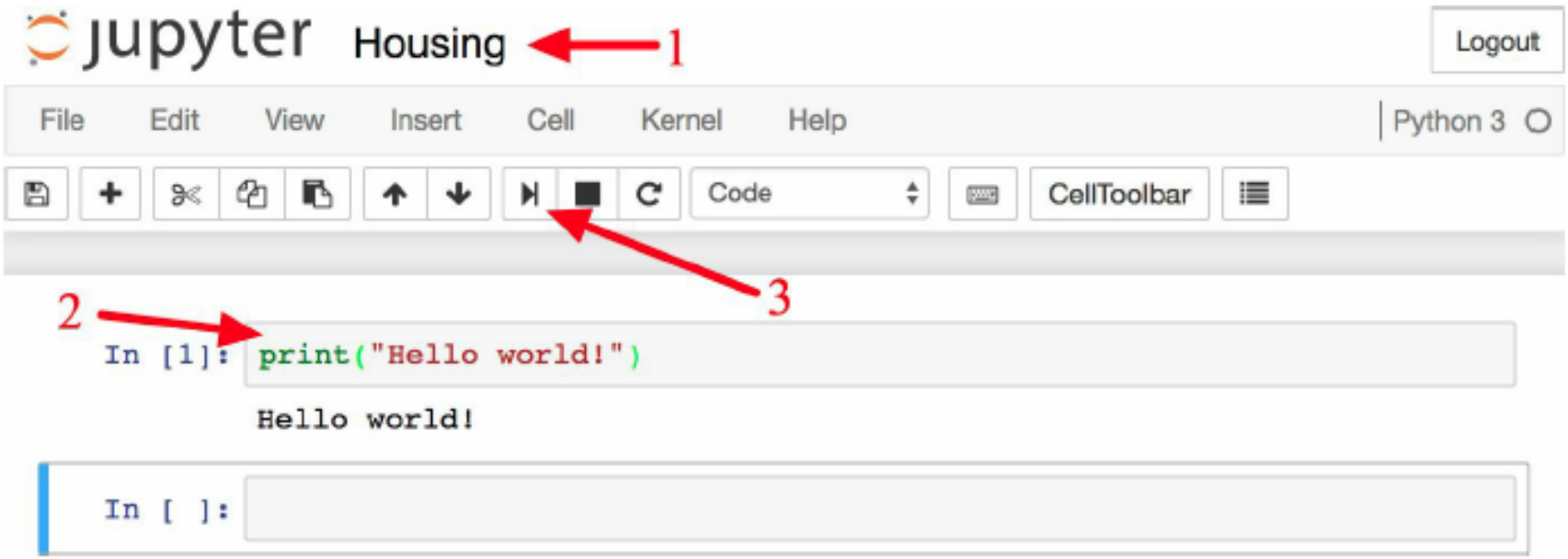


Figure 2-4. Hello world Python notebook

How to build the chapter notebook

The final, and very long, notebook can be found from the book's github site
https://github.com/ageron/handson-ml2/blob/master/02_end_to_end_machine_learning_project.ipynb

Go through it yourselves, implementing your own version step by step, while reading the chapter.

We only have time in lectures to look at a few highlights...

Get the Data

In [2]:
```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

In [3]:
```
fetch_housing_data()
```

In [4]:
```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

In [5]:
```
housing = load_housing_data()
housing.head()
```

Take a quick look at the data structure

In [5]:
```
housing = load_housing_data()
housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Histograms for each attribute

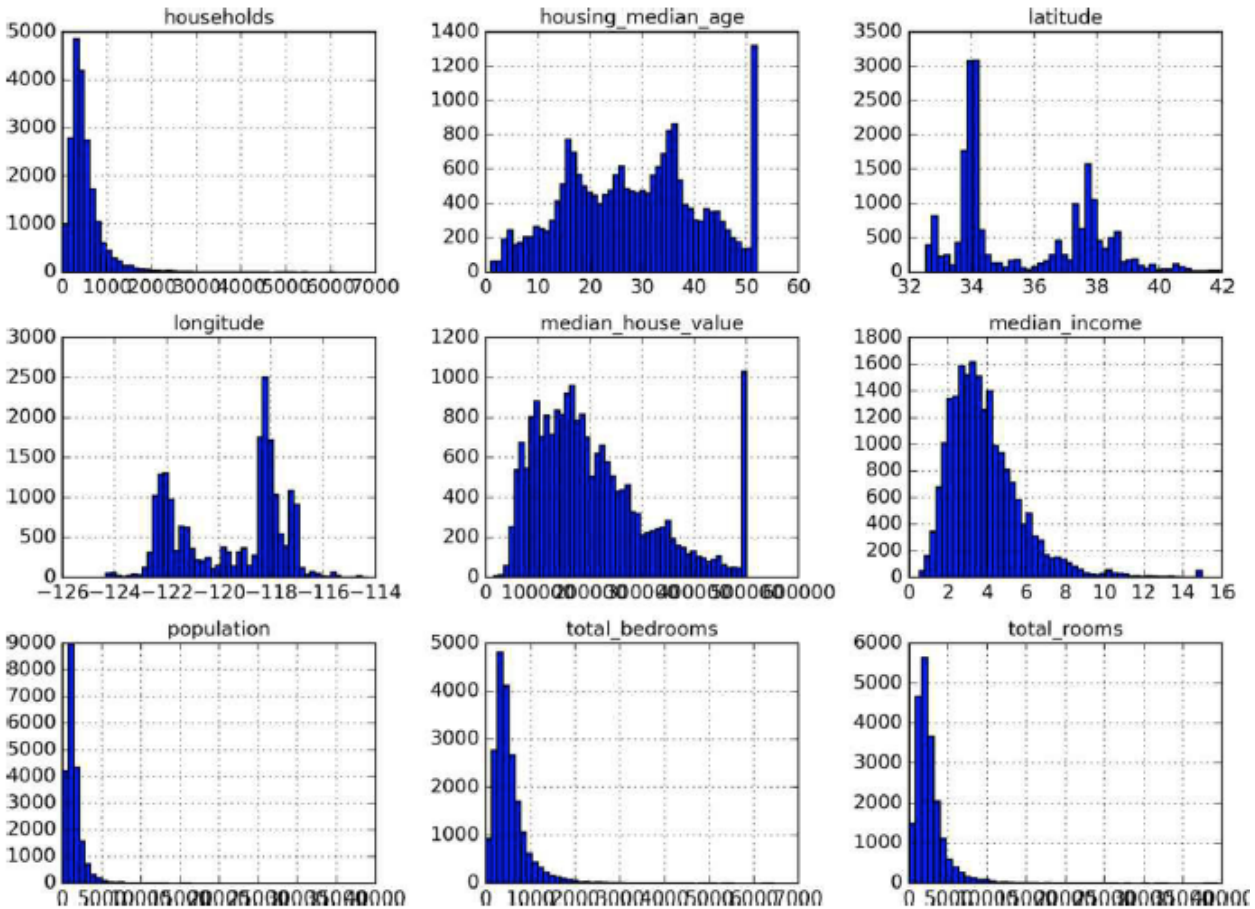


Figure 2-8. A histogram for each numerical attribute

Create a test set

Avoid *data snooping bias*: put the test set away now!
Typically 20% at random.
But be careful that you don't keep resampling.
Also consider stratified sampling.

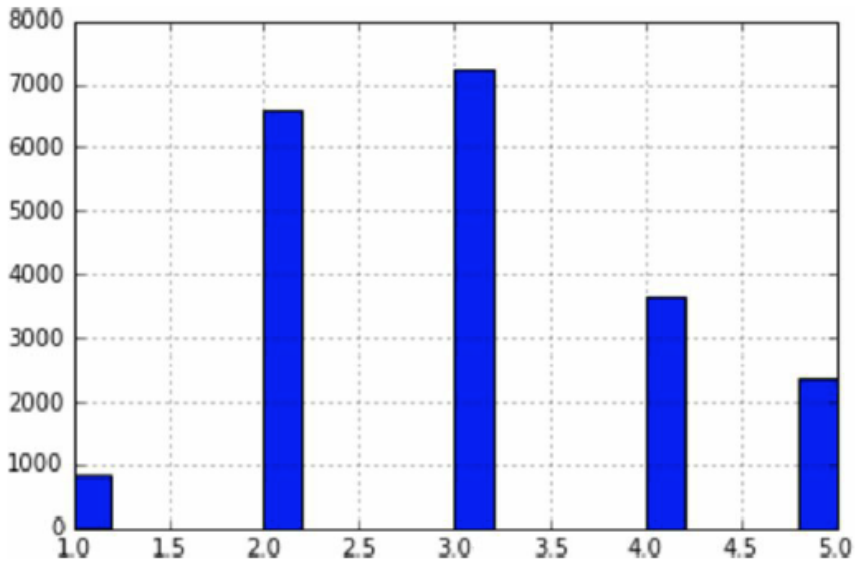


Figure 2-9. Histogram of income categories

Explore and Visualize the Data

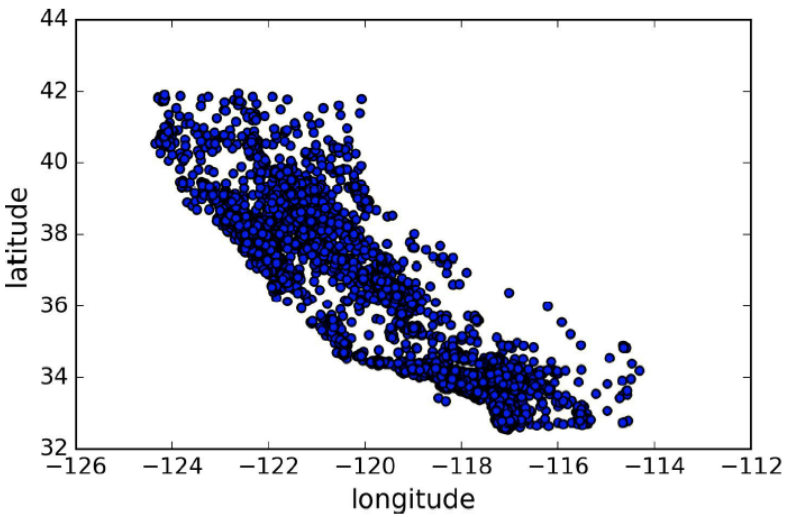


Figure 2-11. A geographical scatterplot of the data

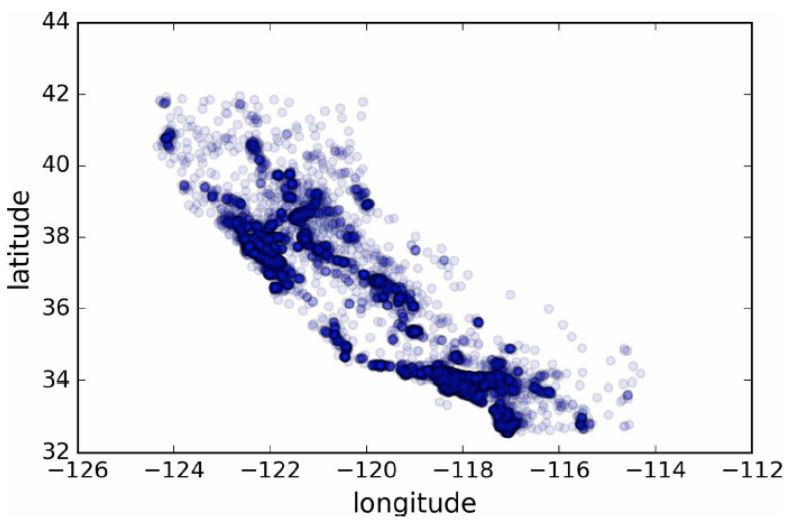


Figure 2-12. A better visualization highlighting high-density areas

Experimenting with Attribute Combinations. E.g.,

```
housing["rooms_per_household"] =
    housing["total_rooms"] / housing["households"]
```

Prepare the data for machine learning algorithms

(Use functions so it can be redone easily!)

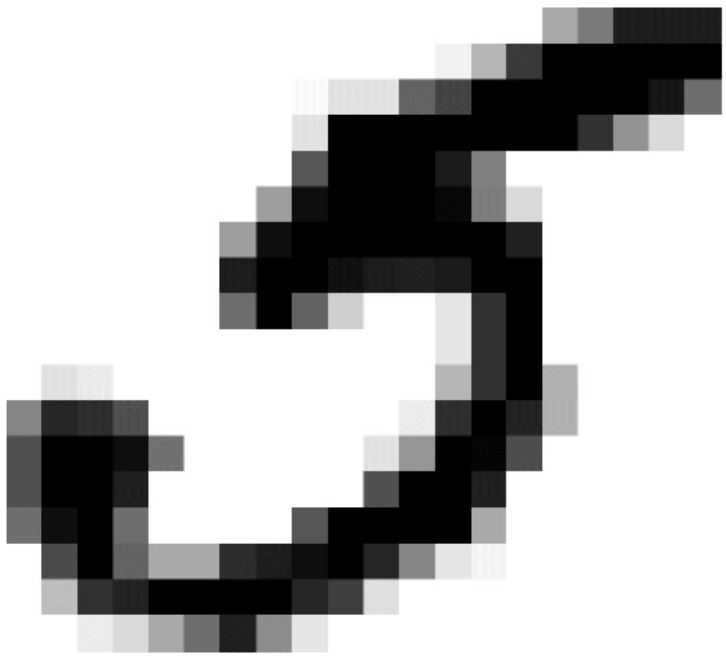
- Data Cleaning (e.g., the `Imputer` class from Scikit-Learn)
- Scikit-Learn Design (read the short description of the main types of functions)
- Handling Text and Categorical Attributes (convert to numbers)
- Custom Transformers (read how to implement your own)
- Feature Scaling (*min-max scaling* vs *standardisation*)
- Transformation Pipelines (specify the order of these transformations)

Let's have a look at a classification task example

- MNIST: a very well studied dataset
- $70,000$ small images of hand-written digits
- easy to download via Scikit-Learn
- each image has 784 features as it is 28×28 grey-scale pixels
- each target, or label, is $0, 1, ..., 9$
- the text book has a Jupyter notebook for this chapter on its github site.

MNIST example

Easy to view via Matplotlib's `imshow()` function, e.g.,
`plt.imshow(X[36000],...)`



y[36000] is 5.0

More MNIST examples

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

Training a binary classifier

E.g., let's make a "5-detector" distinguishing between 5s and not-5s.
Set up the target and task.

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

Performance strategies: Cross-Validation

- Recall that we can use cross-validation to evaluate a technique.
- We break the training data into *K folds* (= distinct sets). Then for each fold, train a model on all the other folds, and measure the accuracy of prediction on that fold.
- Can use Scikit-Learn's `cross_val_score` method.
- In this example we measure *accuracy* which is ratio of correct predictions.
- With 3 folds we get

```
>>> 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])
```

Wow! Over 93% accuracy on all folds.

Performance measures: ... but there is a problem with using accuracy

Here is a different classifier on the same data.

```
from sklearn.base import BaseEstimator

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

This one just says that every input is a not-5.
But it is over 90% accurate!!

Why?

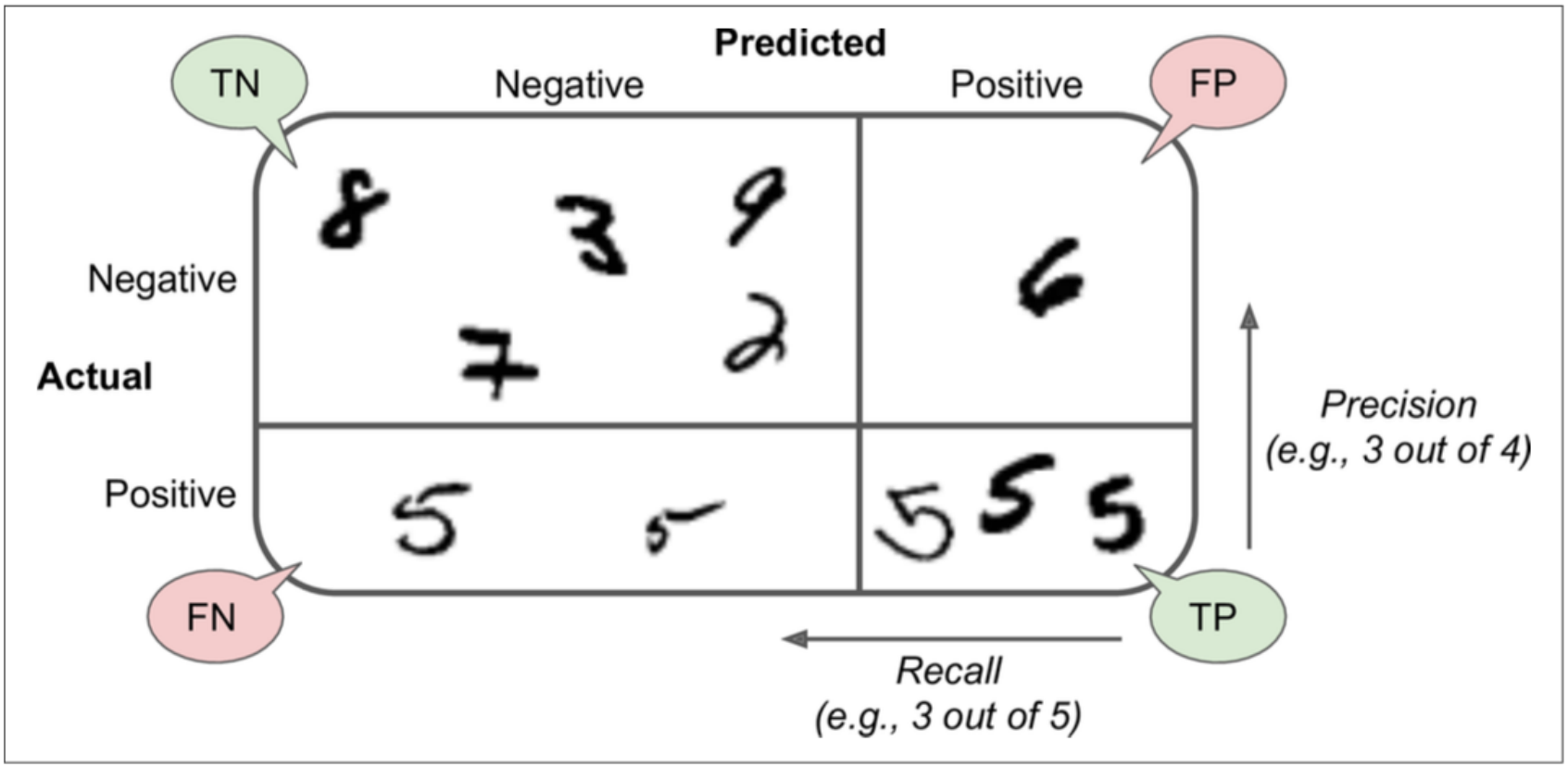
Accuracy has issues with *skewed datasets*.

Performance measures: confusion matrix

The general idea is to count the number of times instances of class A are classified as class B.

This gives us the confusion matrix which contains the true negatives, false positives, false negatives and true positives.

Performance measures: confusion matrix



Performance measures: Precision and Recall

We can extract some more concise metrics from the confusion matrix.

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{accuracy of positive predictions } (=0.75)$$

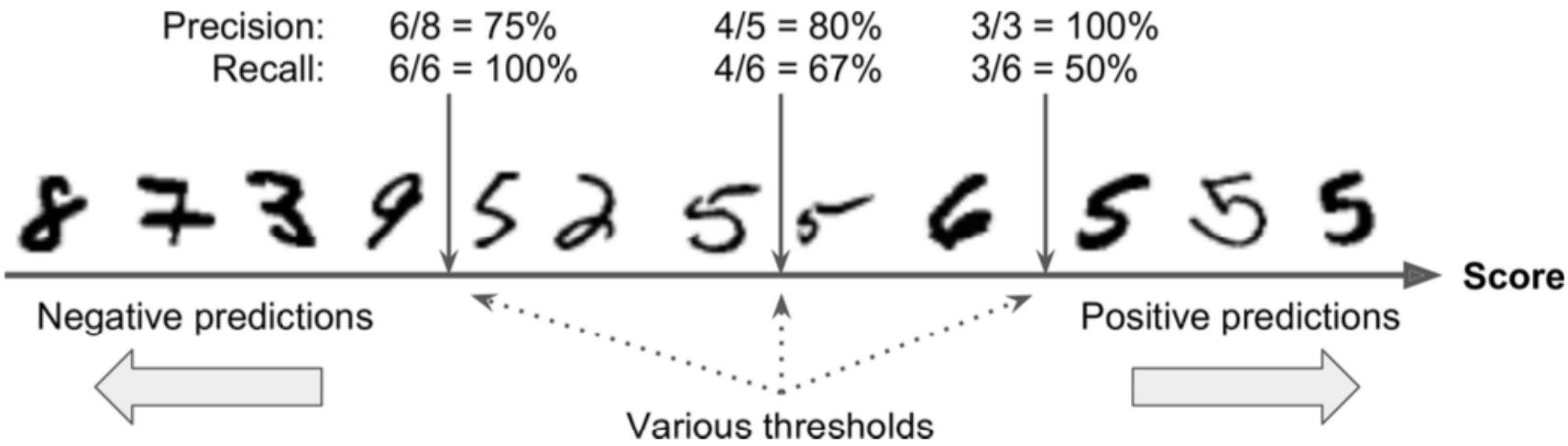
$$\text{recall} = \frac{TP}{TP + FN} \quad \text{true positive rate } (=0.6)$$

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

which is the *harmonic mean* of precision and recall (approx 67% in our example).

Performance measures: Precision/Recall tradeoff

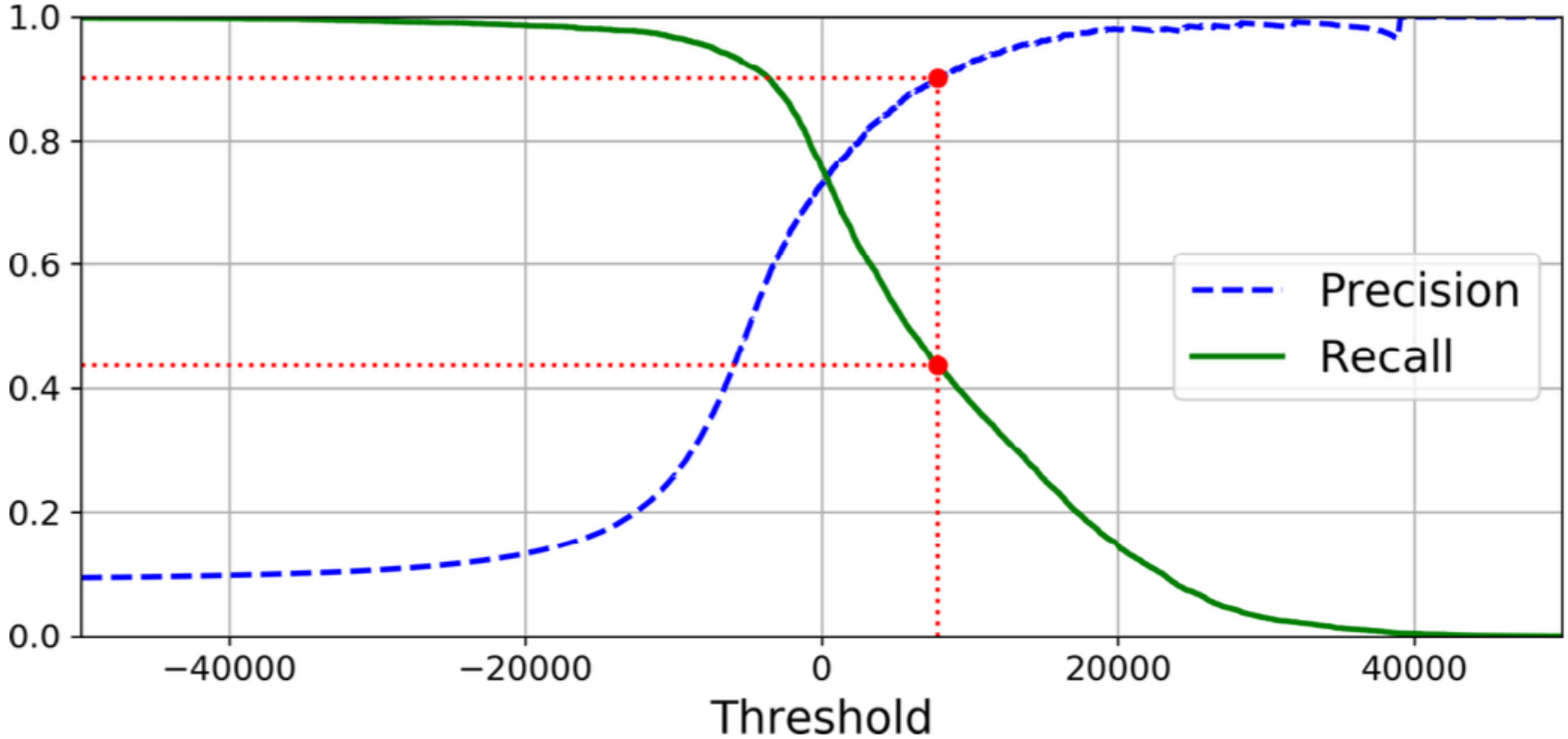
For some applications you may want higher precision or higher recall. But *you can not have both* ...



This is because classifiers typically use score calculated by a *decision function* and a *threshold* applied to the score.

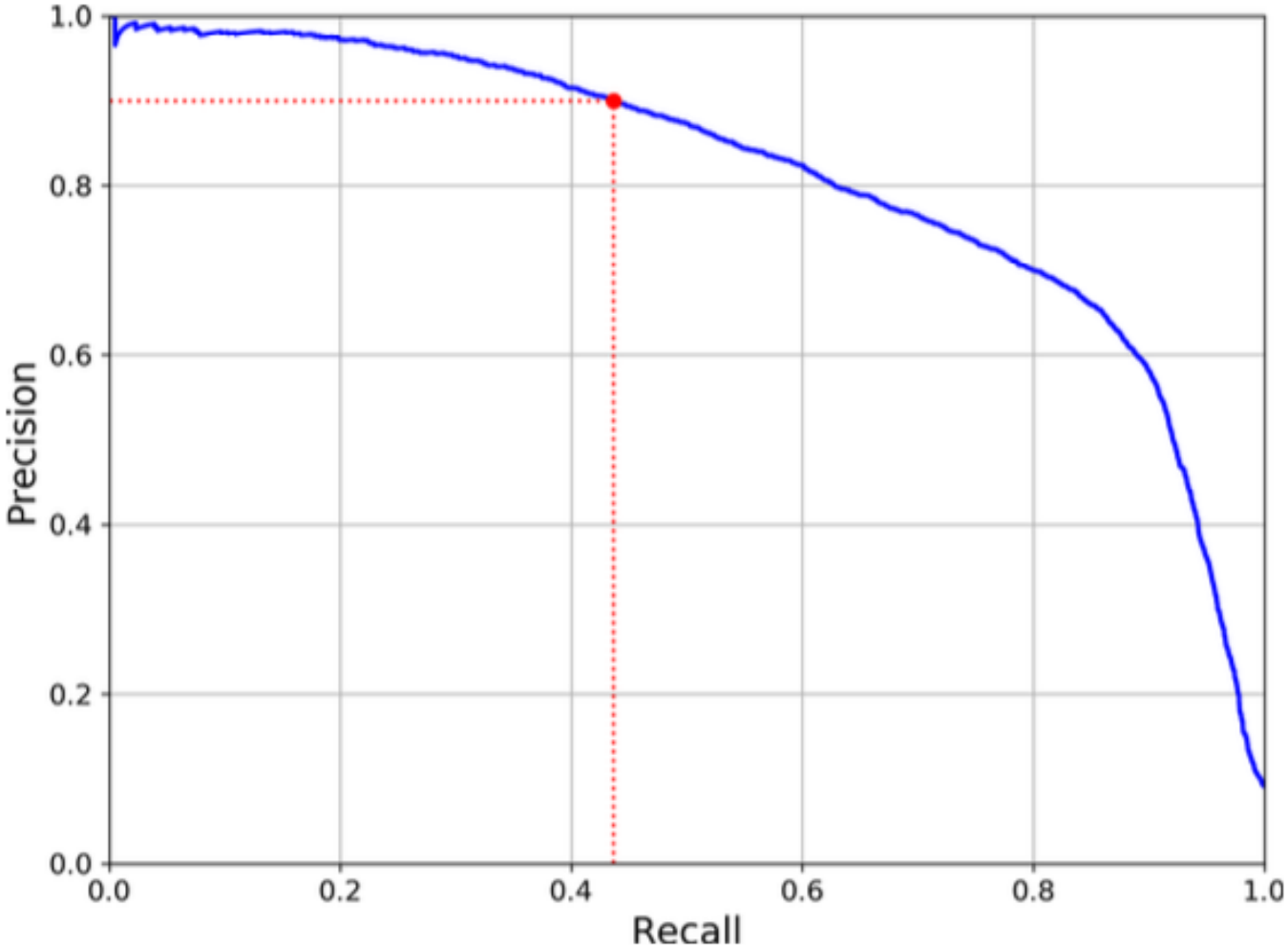
Performance measures: Precision/Recall tradeoff

You can get the score via the `decision_function()` method and use your own threshold.



Performance measures: Precision/Recall tradeoff

Or plot precision against recall (across choices of thresholds).



Performance measures: the ROC curve

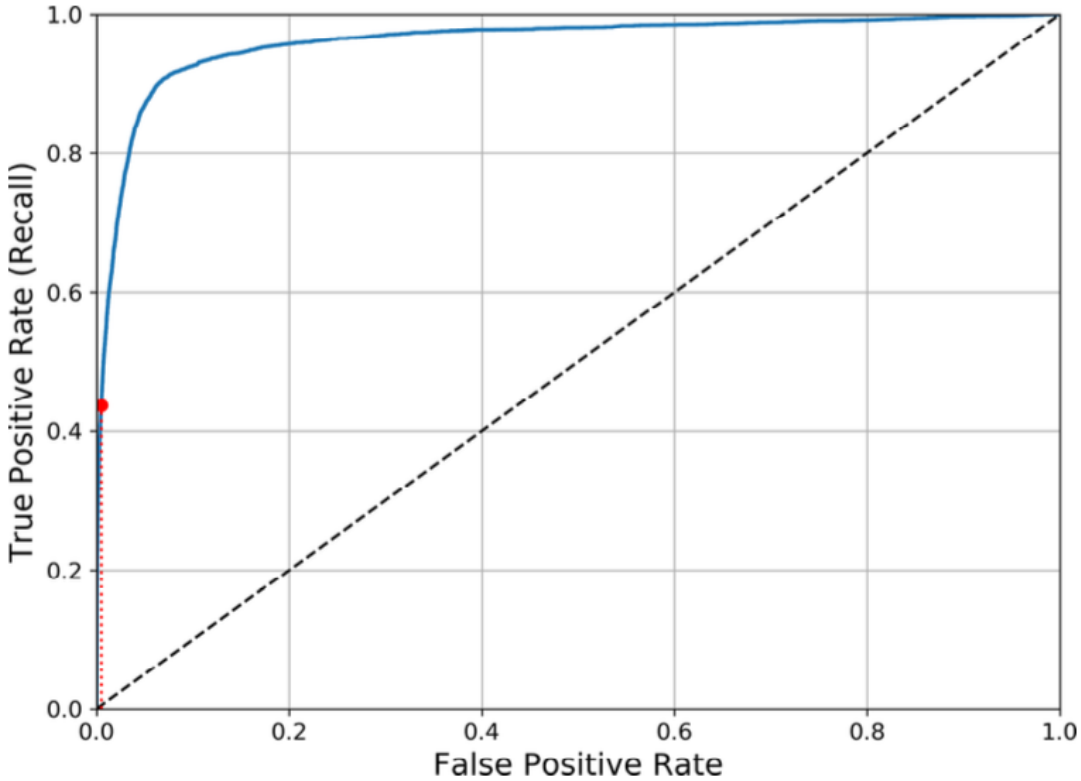
Similarly is the *receiver operating characteristic (ROC) curve*, which plots the *true positive rate* (TPR) against the false positive rate (FPR) for varying threshold settings.

TPR (also known as **sensitivity** and **recall**) = proportion of positive instances that are correctly classified as positives
$= \frac{TP}{TP+FN}$

TNR (also known as **specificity**) = proportion of negative instances that are correctly classified as negatives
$= \frac{TN}{FP+TN}$

FPR = proportion of negative instances that are incorrectly classified as positives
$= \frac{FP}{FP+TN} = 1 - \textbf{specificity}$

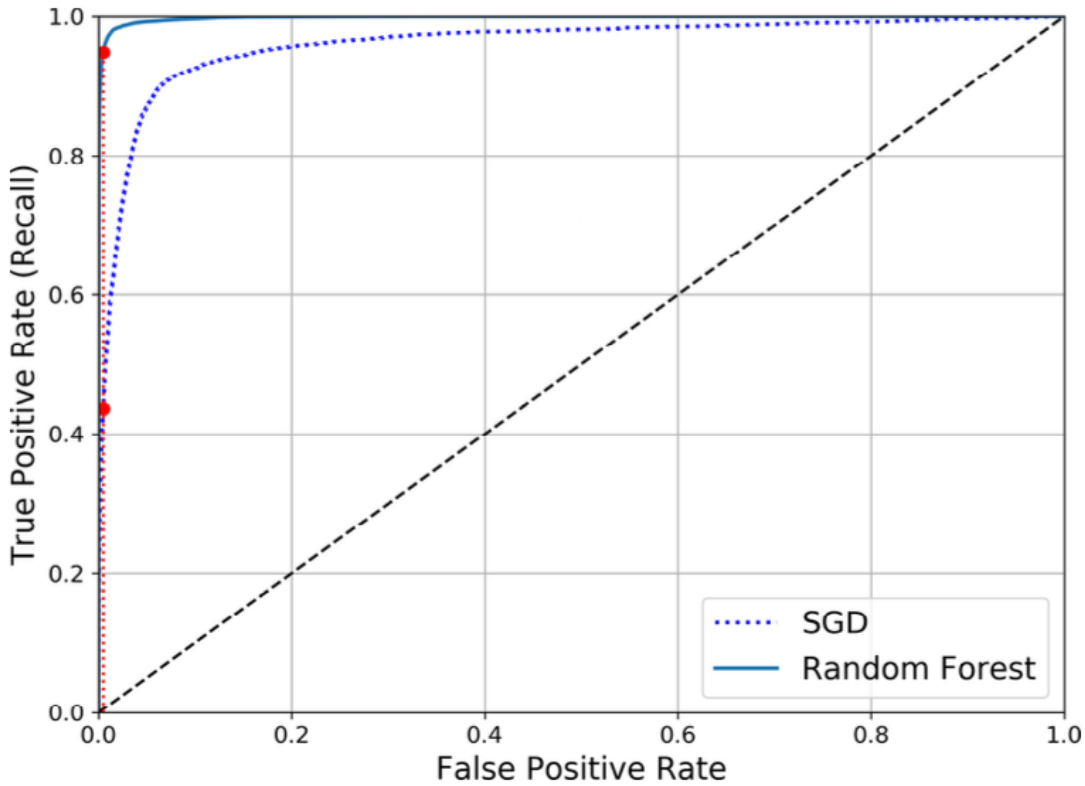
Performance measures: the ROC curve



We plot recall (vertical axis) against $1 -$ specificity (horizontal axis) for the ROC curve.

Performance measures: the ROC curve

A good classifier will have a large *area under the curve* (AUC).



Which of these two algorithms is better?

Sampling bias: example

US presidential election in 1936: Landon (Republican) against Roosevelt (Democratic).

The Literary Digest conducted a very large poll, sending mail to about 10 million people. It got 2.4 million answers, and predicted with high confidence that Landon would get 57% of the votes. Instead, Roosevelt won with 62% of the votes.

Problems with Literary Digest's sampling method:

- Lists of addresses were tended to favour wealthier people who were more likely to vote Republican.
- Less than 25% of the people who were polled answered.

Any other examples?

Once you have prepared your data

- Select and train a model
- Test the performance on the test set
- Fine-tune the model
- Launch, monitor and maintain your system
- Look for things you can automate

Present your solution

Now comes the project pre-launch phase: you need to present your solution (highlighting what you have learned, what worked and what did not, what assumptions were made, and what your system's limitations are), document everything, and create nice presentations with clear visualizations and easy-to-remember statements (e.g., "the median income is the number one predictor of housing prices").

Launch, monitor and maintain your system

- Plugging the production input data sources into your system and writing tests.
- Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops.
- Sampling the system's predictions and evaluating them
- Evaluate the system's input data quality
- Train your models on a regular basis using fresh data.

For this week

Attend the supervised lab. The Unit Coordinator and/or a casual Teaching Assistant will be there to help.

Make sure your software is correctly installed and you can run the notebooks of the textbook (chapters 1, 2 and 3). Get familiarized with Jupyter Notebooks.

Read up to Chapter 4 (Training Models).

And that's all for the second lecture.

Have a good week.