

Machine Learning

Lecture 1: Introduction

Instructor: Dr. Farhad Pourkamali Anaraki

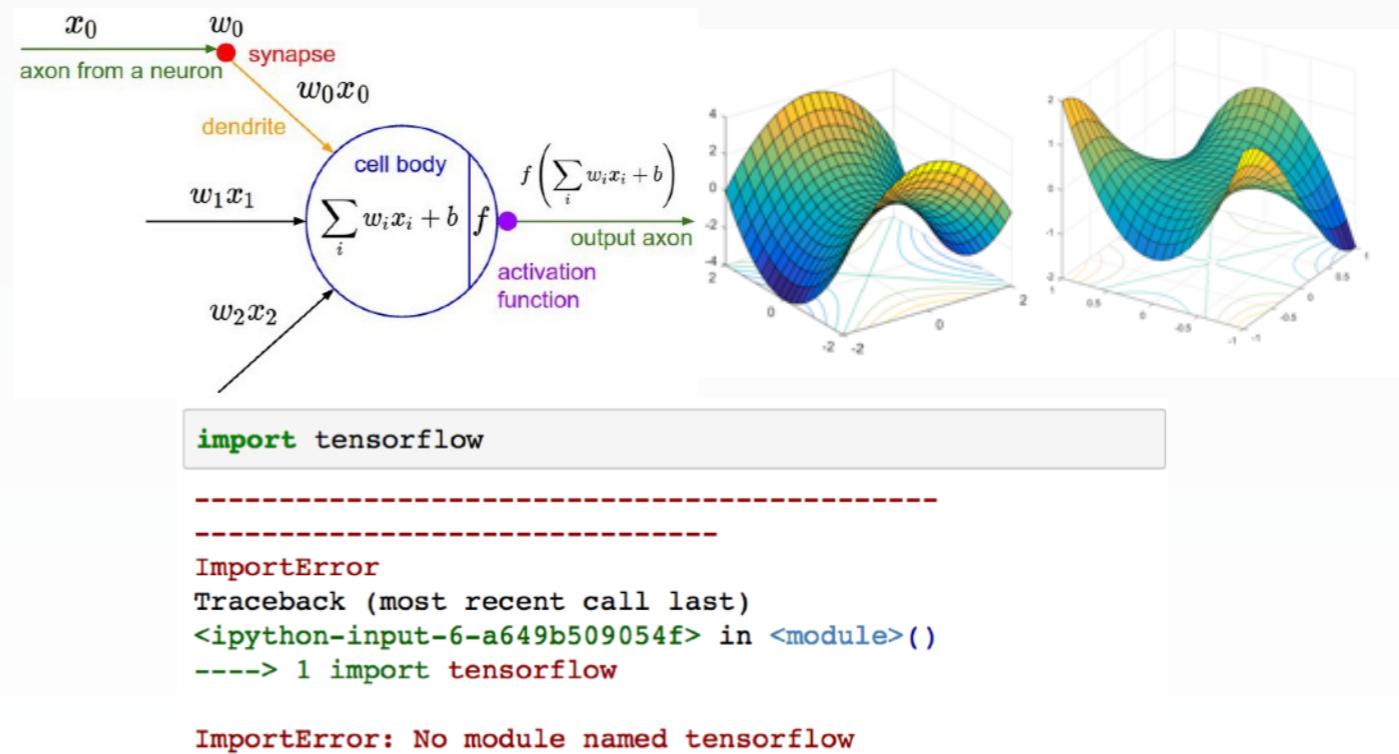
What do you learn from this course?

- Concepts, tools, and intuition you need to implement programs capable of **learning from data**
- From simple techniques such as linear regression to some of deep learning techniques
- A **hands-on approach** with just a little bit of theory to use production-ready Python frameworks

Public perception of ML



Reality



Course prerequisites

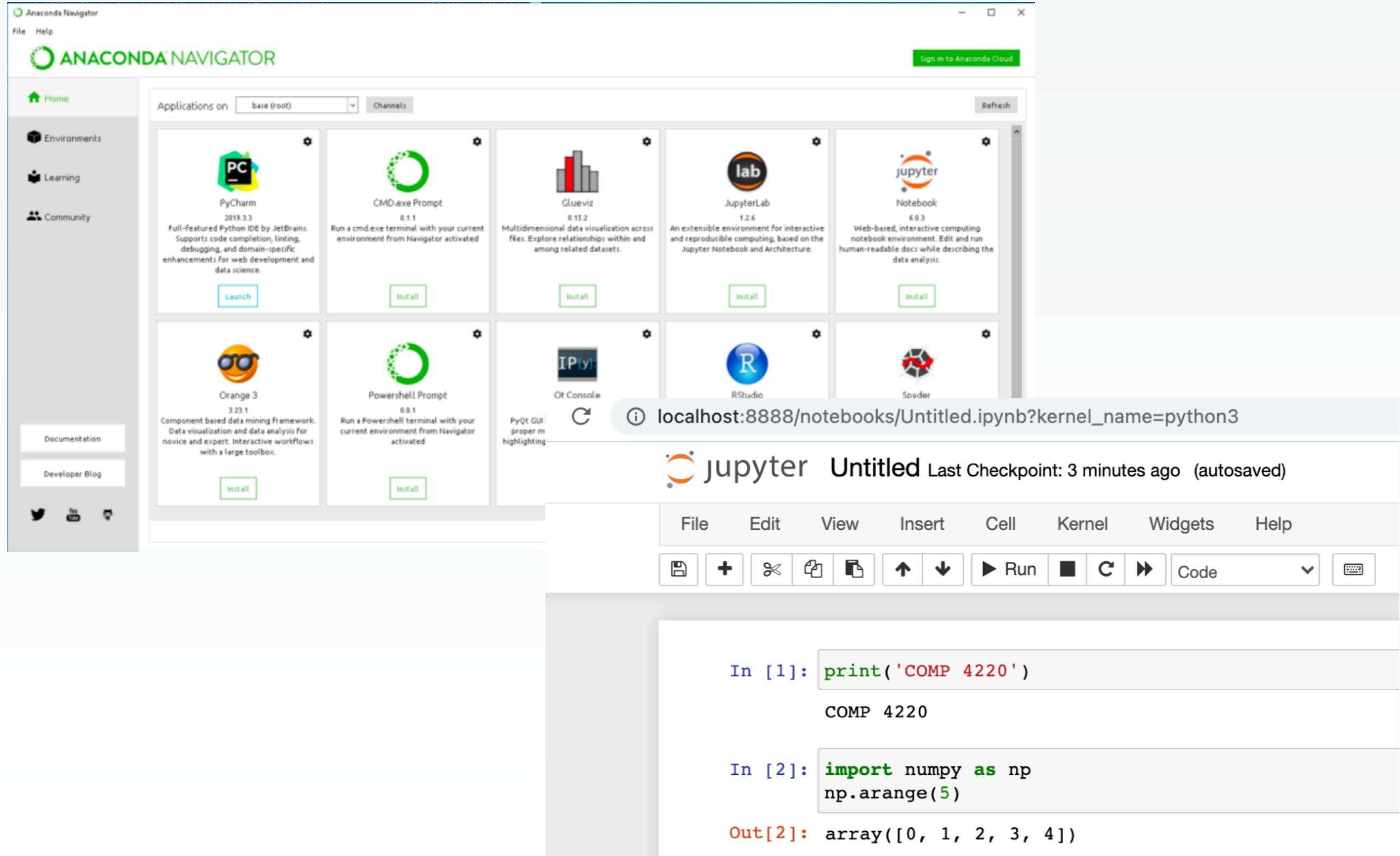
- We need some level of calculus and linear algebra
 - Vector, matrix, inner product, matrix-matrix multiplication
 - Derivatives of polynomials and exponential functions
 - Chain rule for finding the derivative of a composite function

$$\begin{bmatrix} y = f(t) \\ t = g(x) \end{bmatrix} \rightarrow \frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dt}{dx}}$$

- Basics of probability and statistics
 - Probability distributions (normal distribution, Bernoulli distribution, etc.)
 - Mean, standard deviation, ...

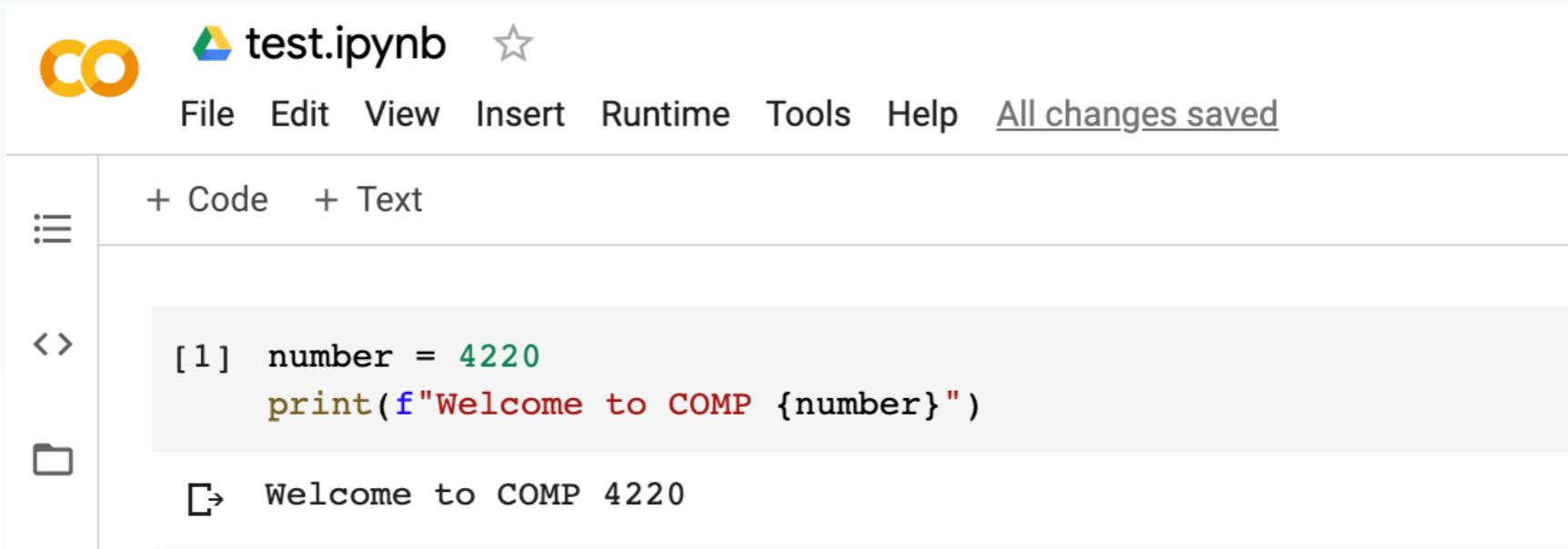
Anaconda

- <https://www.anaconda.com/>



Google Colab

- Google Colaboratory, or “Colab” for short, allows you to write and execute Python in your browser
 - Zero configuration required
 - Free access to GPUs and other computing resources
 - Use terminal commands on Google Colab



The screenshot shows the Google Colab interface. At the top, there's a toolbar with a 'CO' logo, the file name 'test.ipynb', and a star icon. Below the toolbar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and a status message 'All changes saved'. The main area has a sidebar on the left with icons for code (+ Code), text (+ Text), and a folder. A code cell is open, displaying the following Python code:

```
[1] number = 4220
    print(f"Welcome to COMP {number}")
```

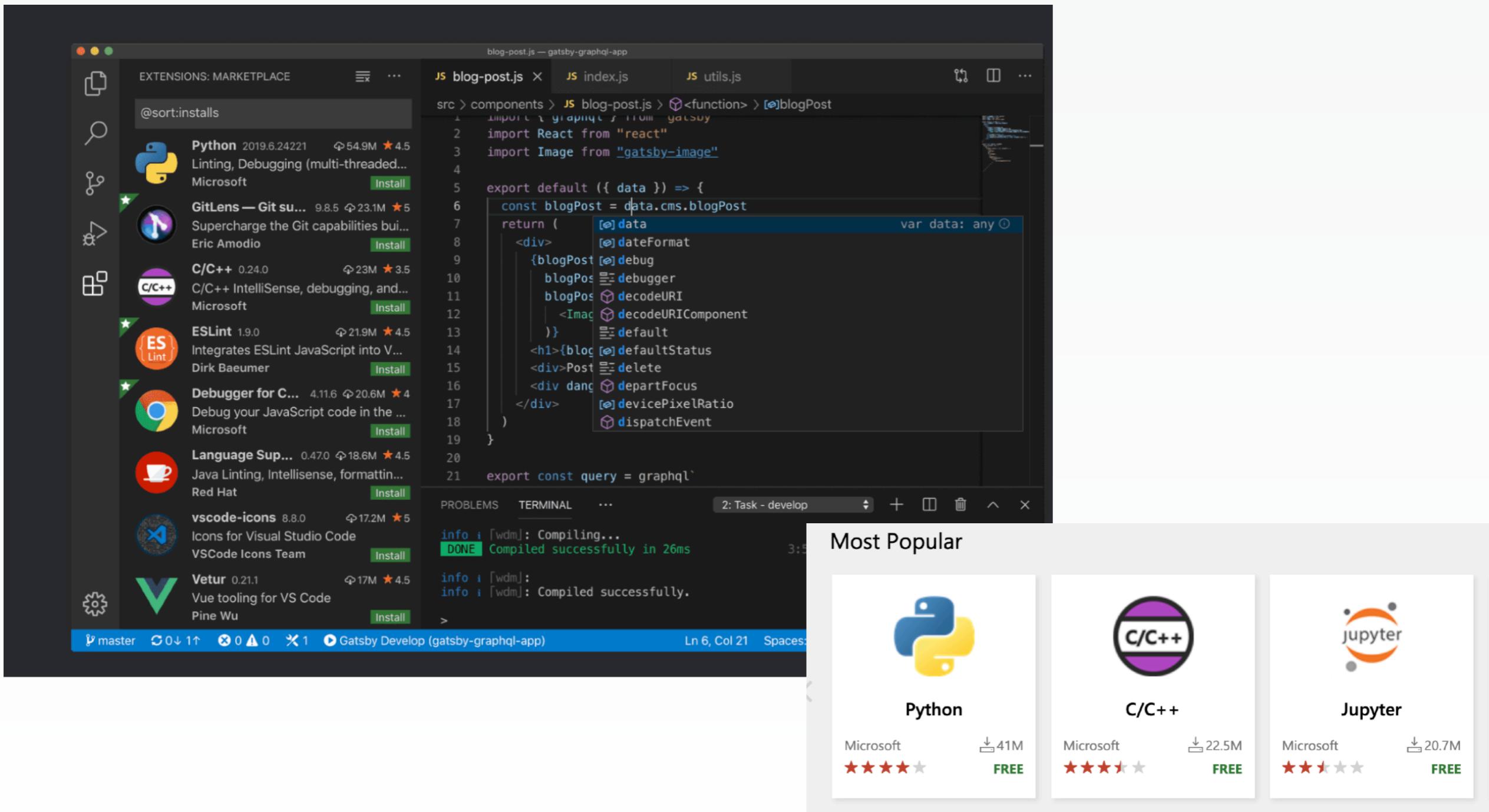
The cell's output is shown below it, displaying the result:

```
Welcome to COMP 4220
```

- <https://drive.google.com> (for the first time, select “Connect more apps”)

Visual Studio Code

- Better alternative for “real” machine learning projects
- <https://code.visualstudio.com/>

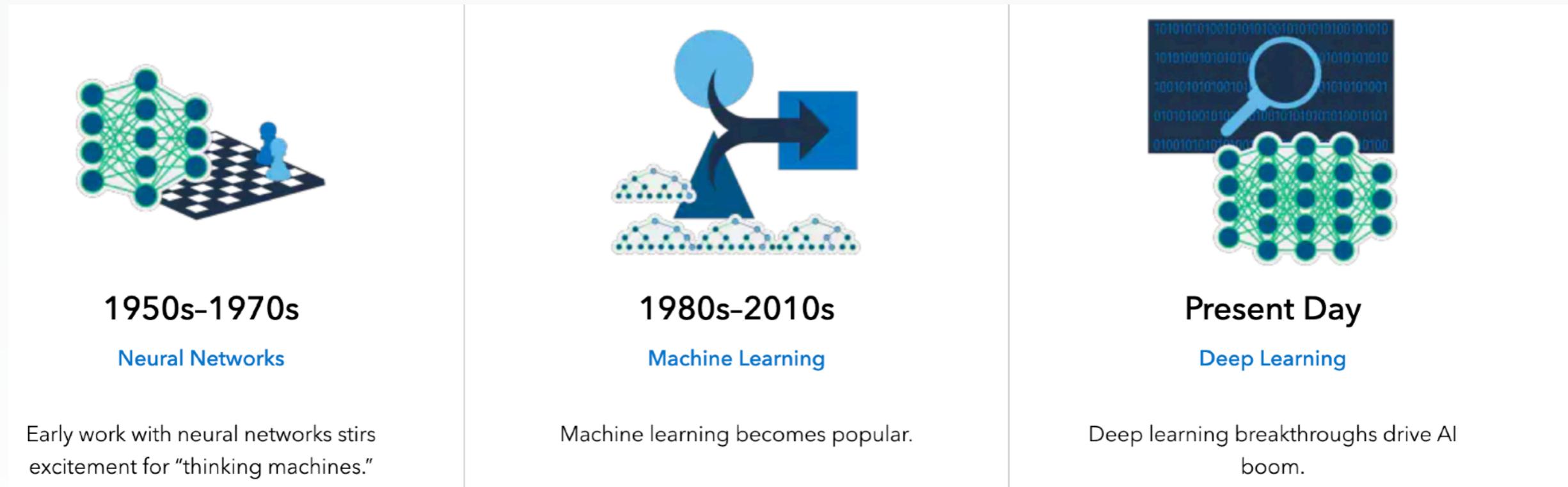


Course structure

- Part 1: Fundamentals of Machine Learning ([Scikit-Learn](#))
 - Review of Python's main scientific libraries: NumPy, pandas, and Matplotlib
 - Steps in a typical machine learning project
 - Learning by fitting a model to data and optimizing a cost function
 - Challenges of using machine learning systems
- Part 2: Neural Networks and Deep Learning ([TensorFlow 2.0](#), [Keras](#), and [PyTorch](#))
 - What neural nets are and what they are good for
 - Building and training feedforward neural nets (basics and implementation)
 - Convolutional nets for deep computer vision and recurrent neural nets (mainly fundamental concepts)

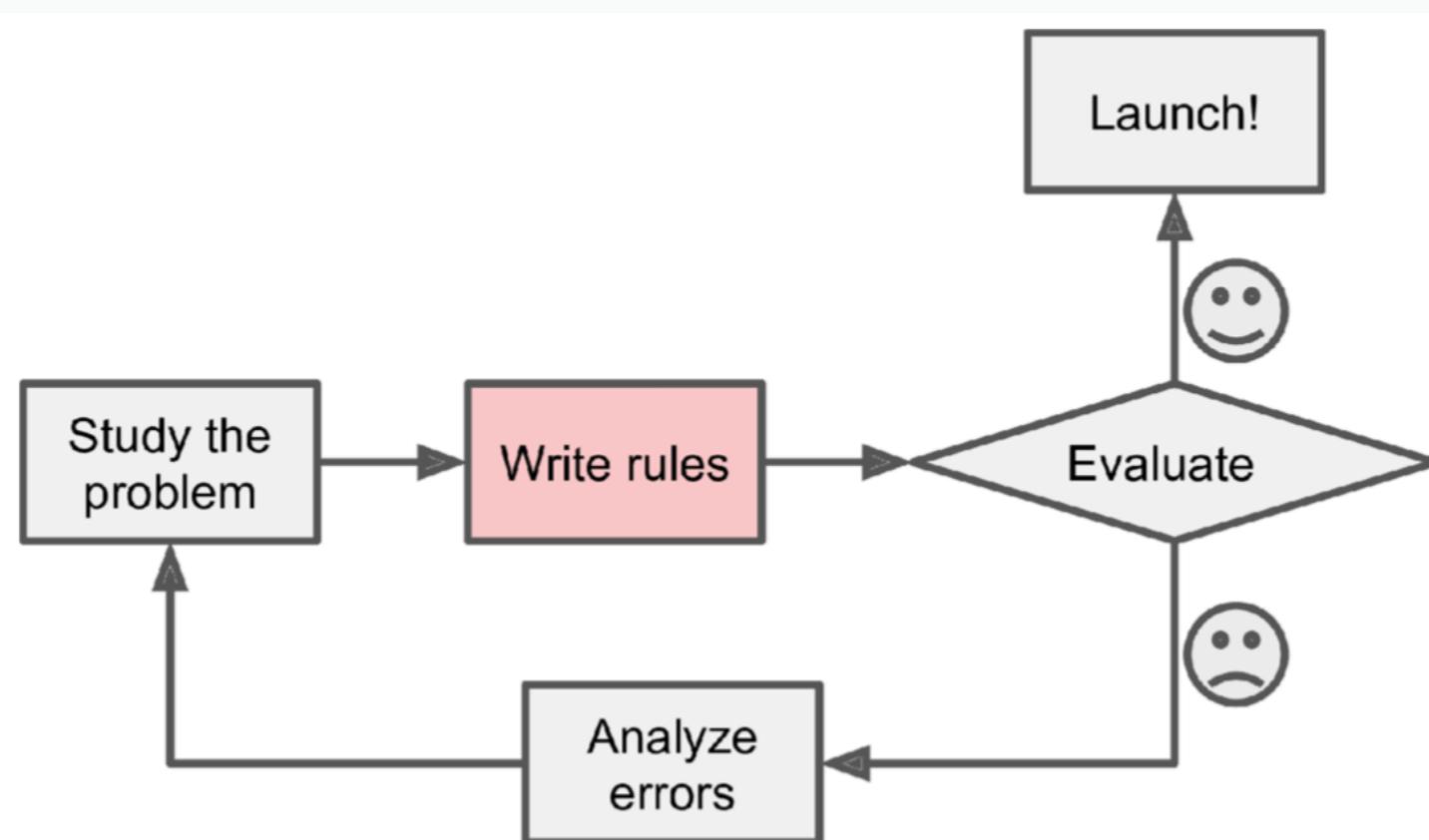
A little bit history

- Alan Turing gave quite possibly the earliest public lecture on this topic in 1947:
“What we want is a machine that can learn from experience”



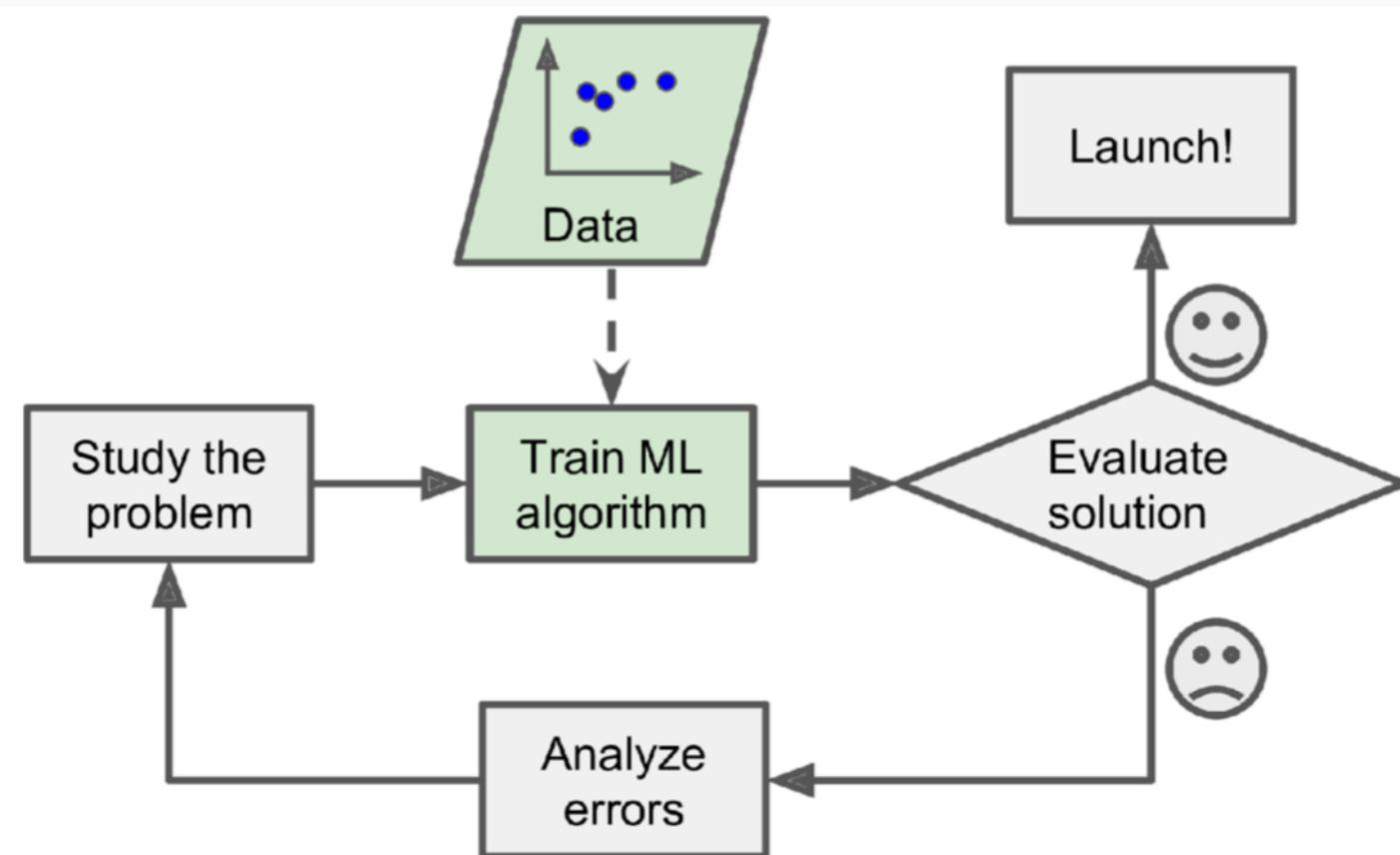
Why should we use machine learning?

- Let's say we want to design a spam filtering system
- One approach is to use traditional programming techniques
 - Look for words or phrases such as: credit card, free, bank information, "C U"
 - Write a detection algorithm for each of the patterns
 - Test your program and improve it



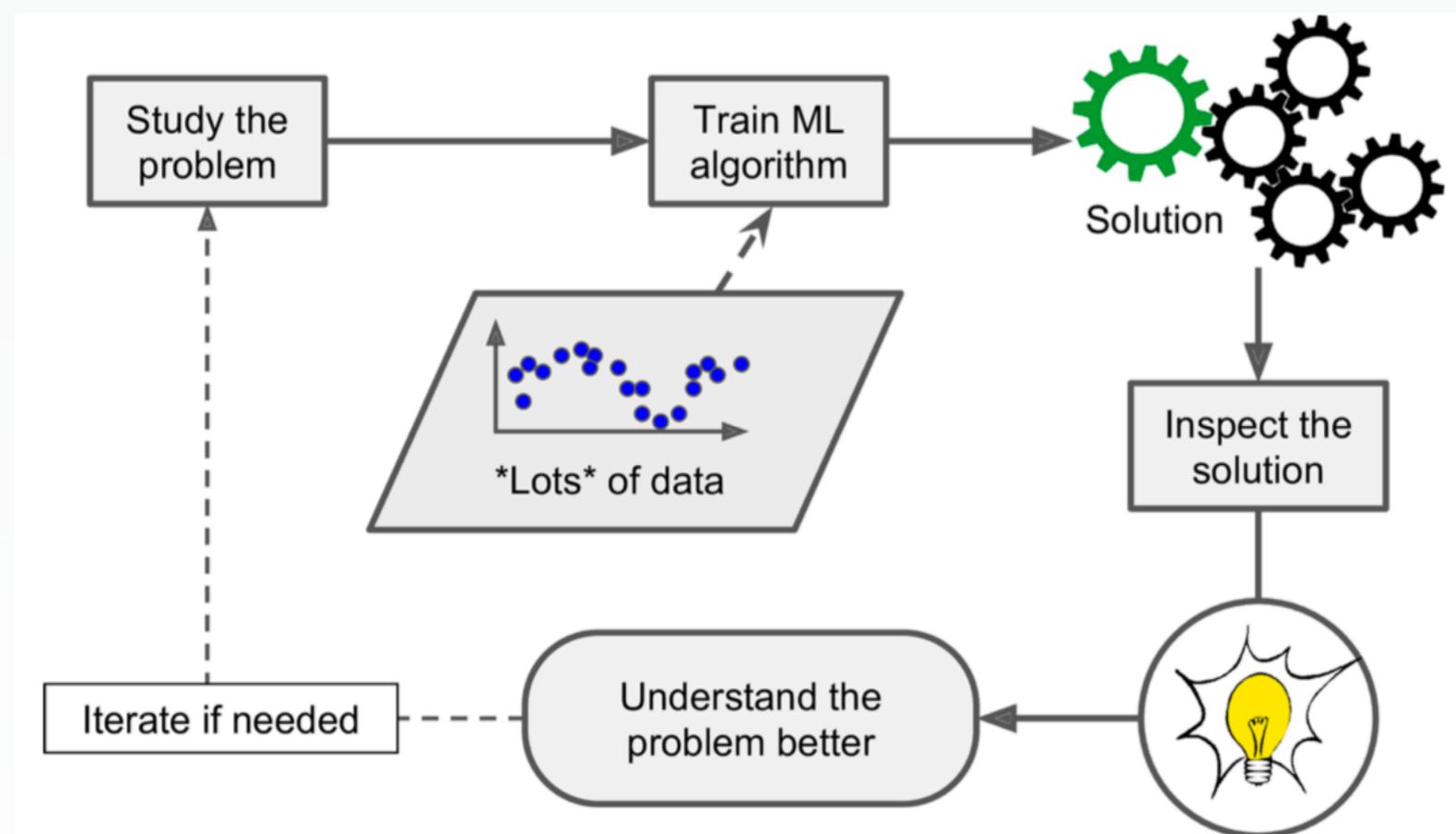
Why use machine learning?

- A spam filter based on machine learning automatically learns which words and phrases are good predictors
- Possible benefits: simpler, easier to maintain, and more accurate
- The examples that the system uses are called the training set
- We should also think of a performance measure



Beyond prediction

- Machine learning can help humans learn and discover patterns from large amounts of data that are not immediately apparent
- This is usually called data mining

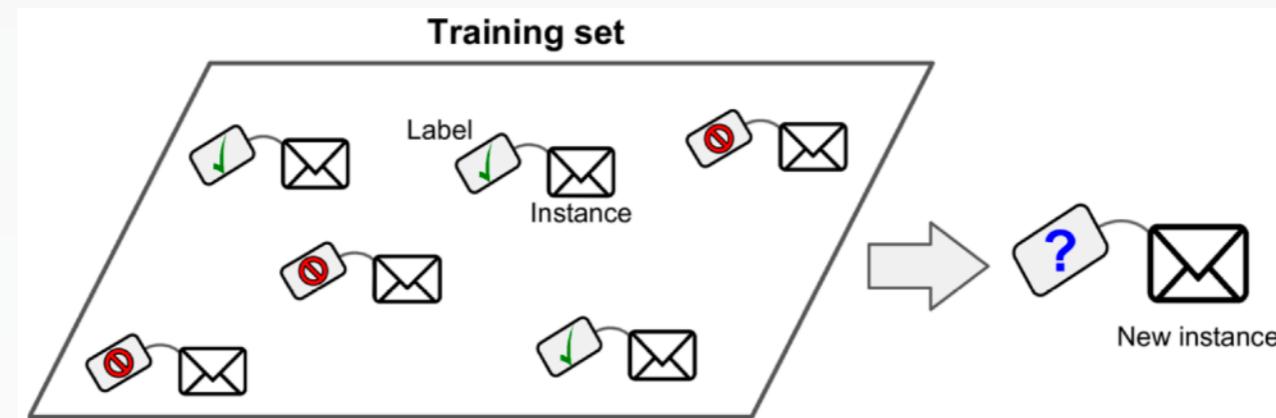


A few applications of machine learning

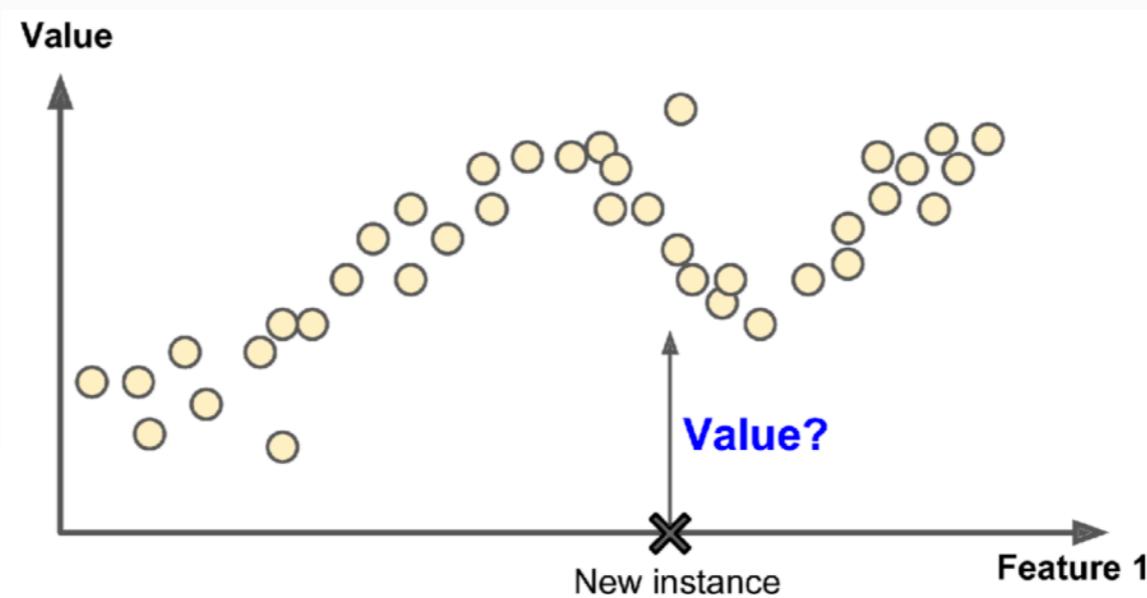
- Autonomous navigation systems
- Analyzing images of items on a product line
- Analyzing medical images
- Automatically flagging offensive comments on social media platforms
- Forecasting revenue
- Creating a recommendation engine
- Speech recognition (like Siri and Alexa)
- And many more!

Types of machine learning

- ML systems can be classified according to the amount and type of supervision
 - 1. **Supervised learning:** training set includes the desired solutions or labels
 - Classification: labels are classes (e.g., spam or ham)
 - Regression: labels are numeric values (e.g., price of car)

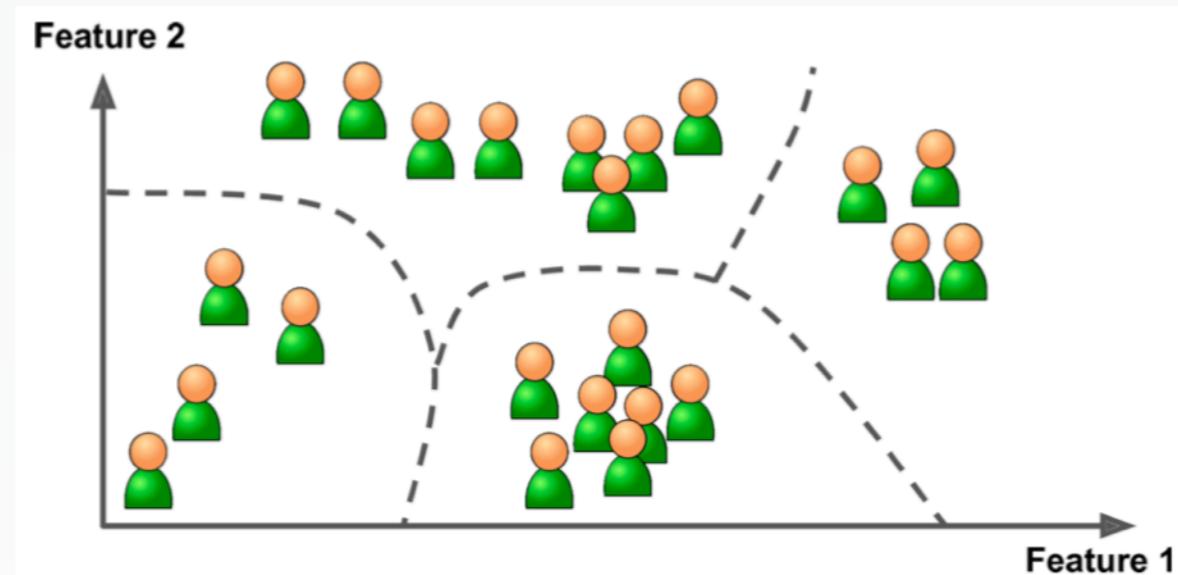


- Regression: labels are numeric values (e.g., price of car)

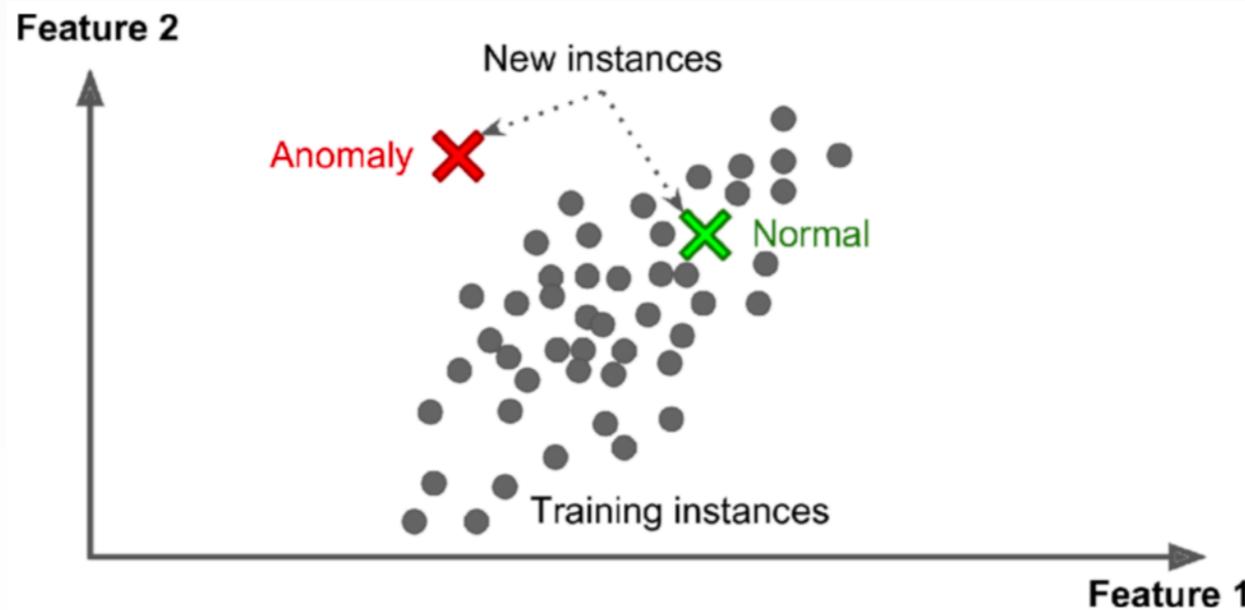


Types of machine learning

- 2. Unsupervised learning: ML system tries to learn without a teacher
 - For example, based on the shopping history, detect groups of similar visitors



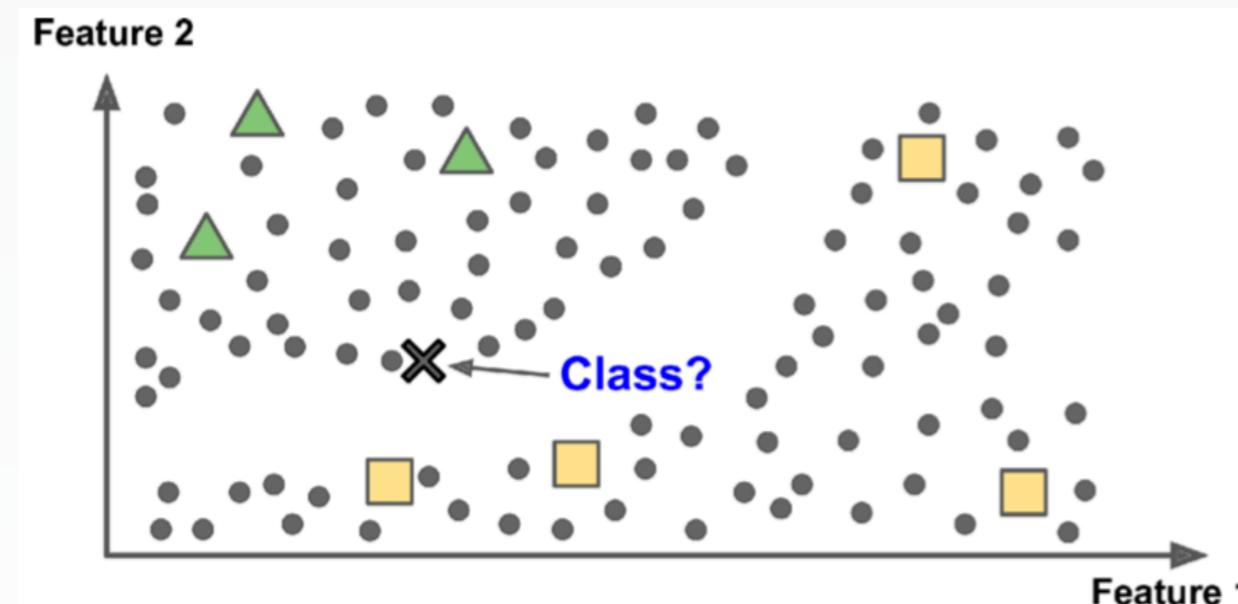
- Another example is novelty detection: find new instances that look different



Types of machine learning

3. **Semisupervised learning:** plenty of unlabeled instances and few labeled

- Why? Labeling data is costly and time-consuming
- Application: Google Photo
- Tools: Amazon SageMaker Ground Truth



4. **Reinforcement learning:** select and perform actions, and get rewards or penalties in return

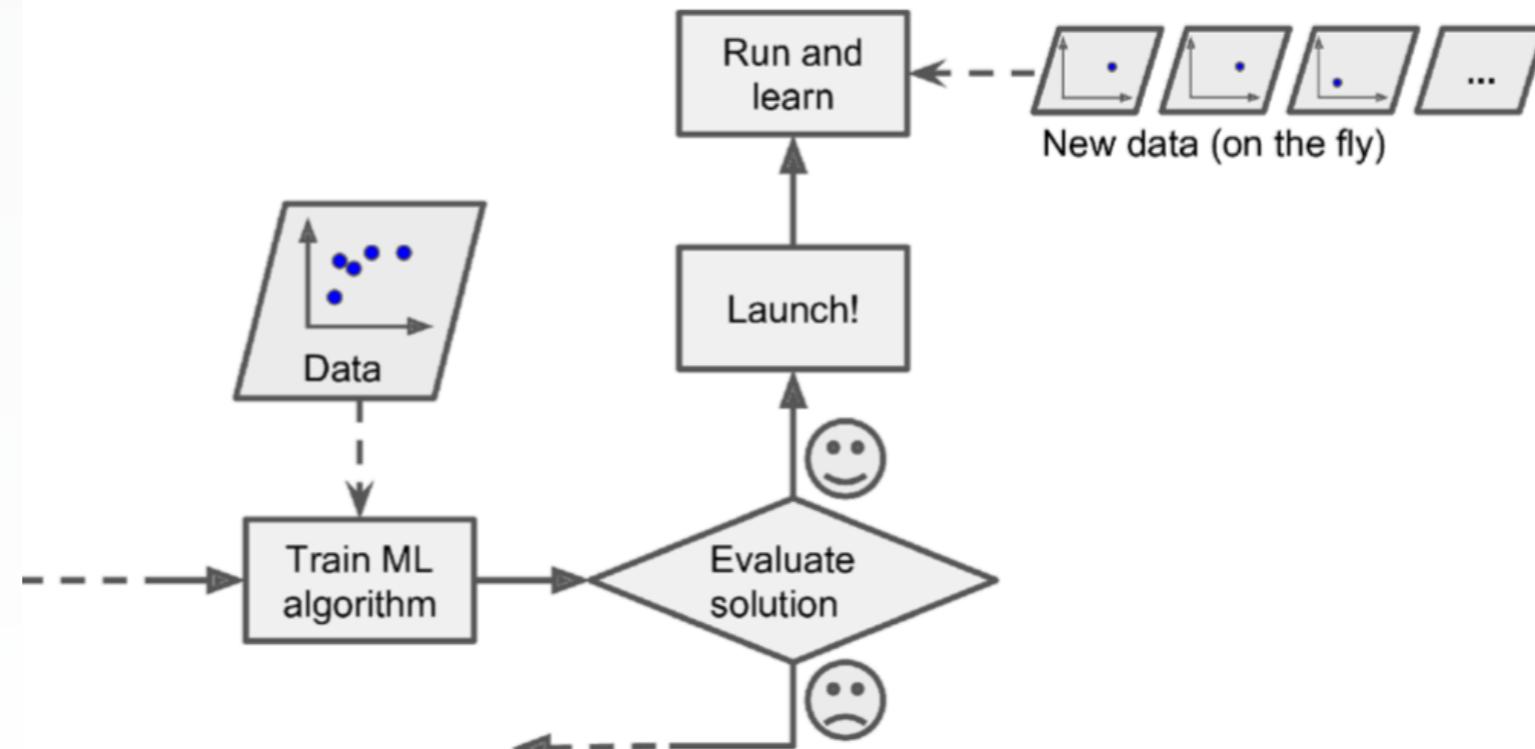
- Find the best strategy known as policy to get the most reward

Another categorization

1. **Batch learning or offline learning**: the system must be trained using all the available data

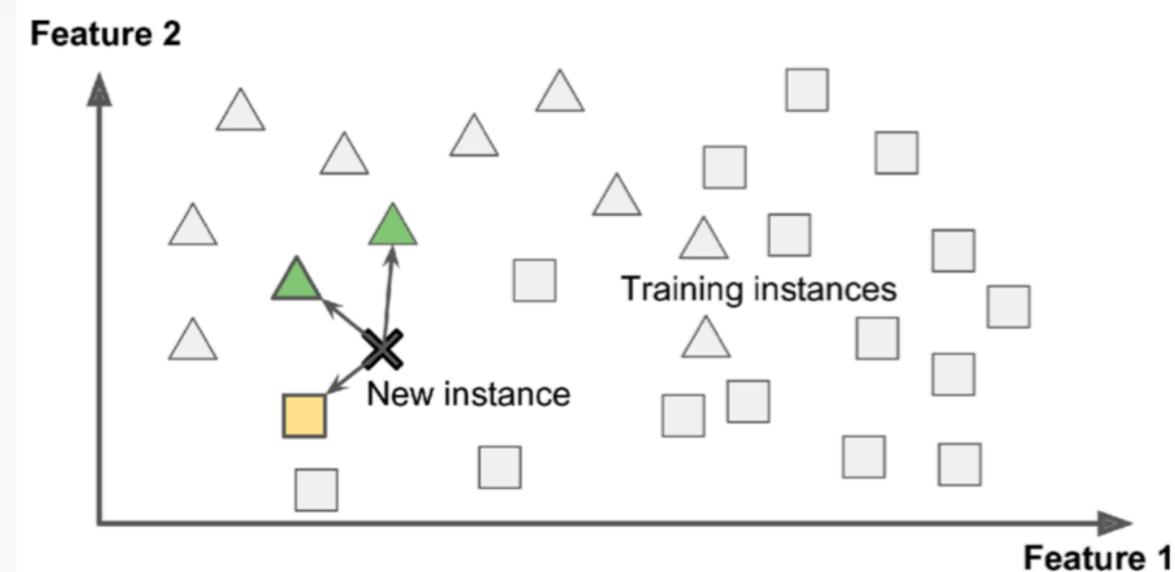
- Drawbacks:
 - Take a lot of time, computing resources, and storage space
 - Not suitable for rapidly changing data

2. **Incremental learning**: train the system incrementally by feeding data instances sequentially

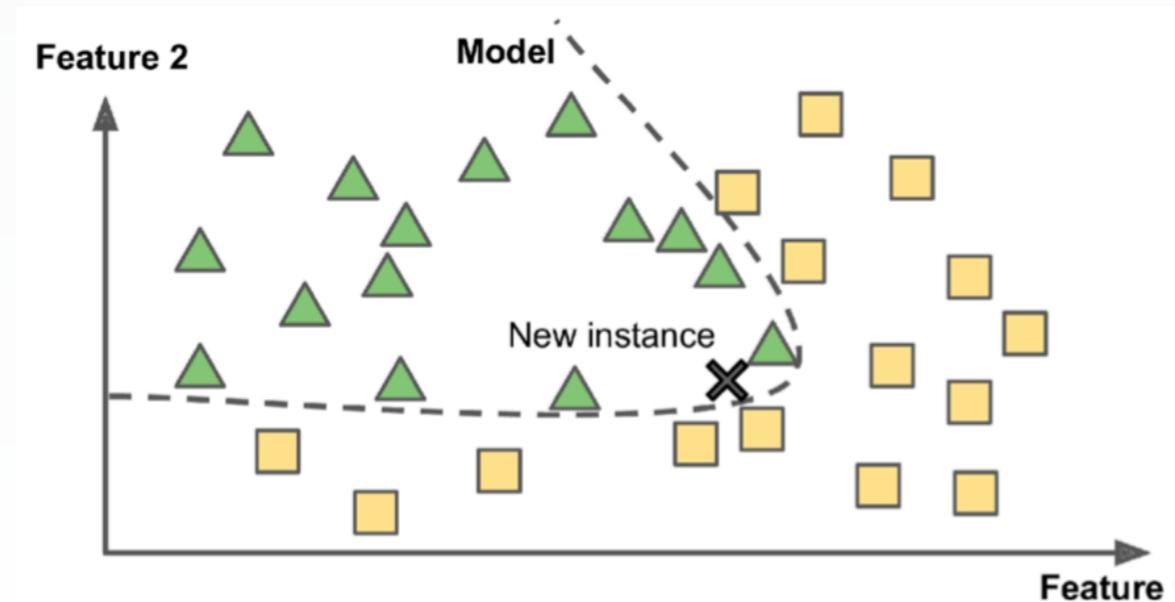


Another categorization

1. **Instance-based learning:** the system uses a similarity measure to compare new cases to the learned examples



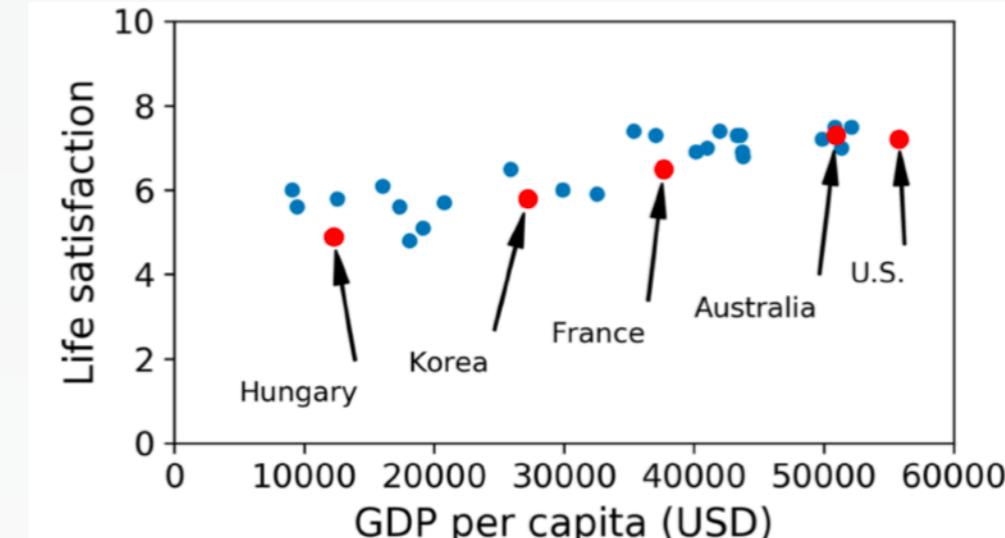
6. **Model-based learning:** build a model of the learned examples and then use that model to make predictions



Model-based learning

- Does money make people happier?

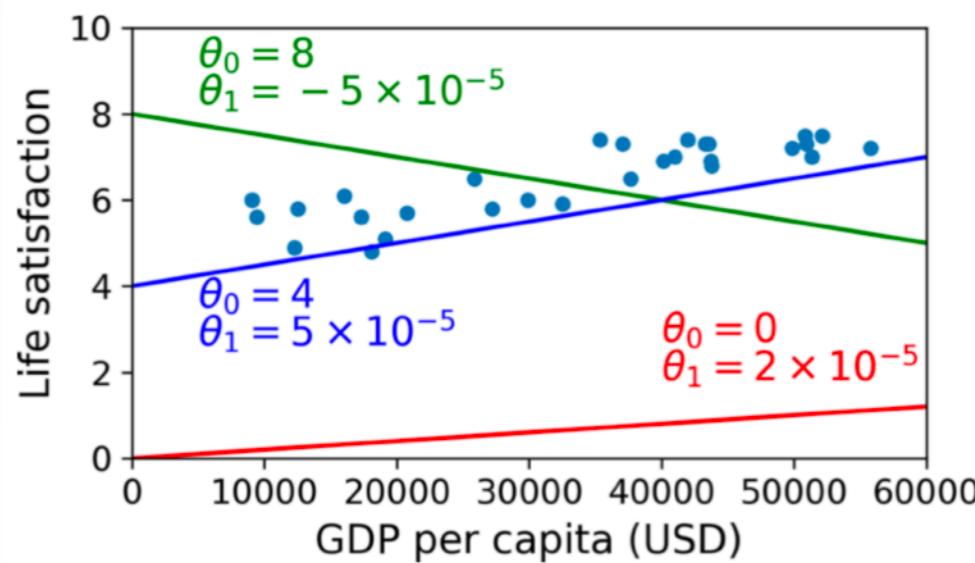
Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2



- Let us model life satisfaction as a linear function of GDP
 - A simple linear model

$$\text{life_satisfaction} = \theta_0 + \theta_1 \times \text{GDP_per_capita}$$

- Examples



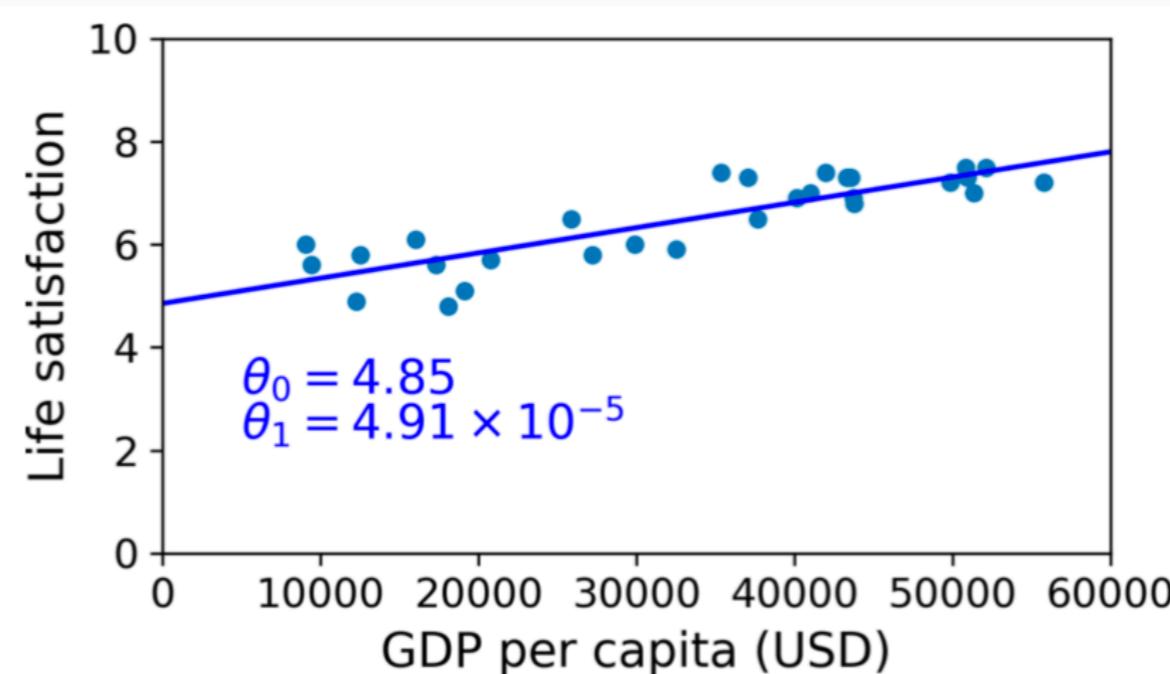
Training a linear model using Scikit-Learn

- A few lines of code

```
# Select a linear model
model = sklearn.linear_model.LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[22587]] # Cyprus's GDP per capita
print(model.predict(X_new)) # outputs [[ 5.96242338]]
```

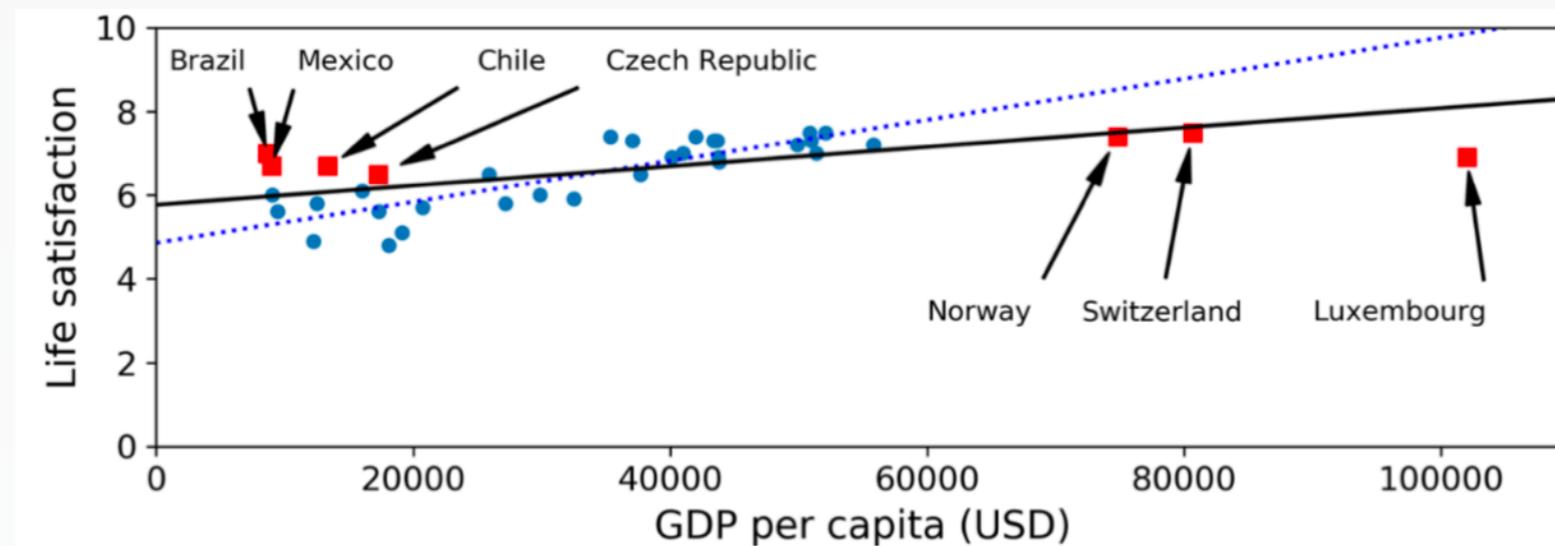


Summary of steps

- Study the data (data preparation)
- Select a model or learning algorithm
- Train on the available data and assess performance
- Apply the model to make predictions on new cases

Challenges of machine learning

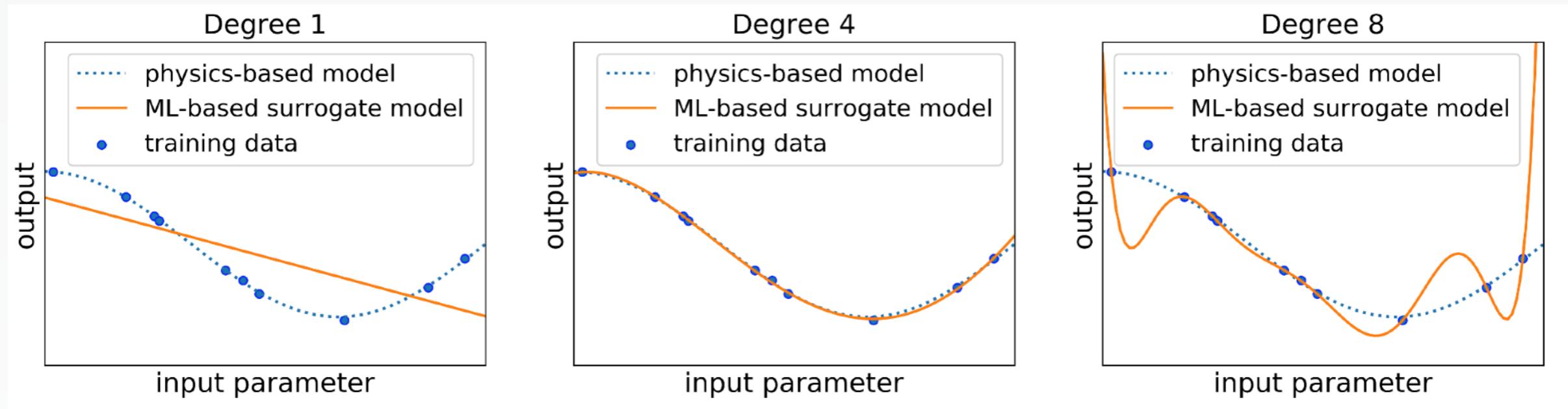
- Insufficient quantity of training data
- Non-representative and poor-quality training data



- Irrelevant features: coming up with a good set of features

Challenges of machine learning

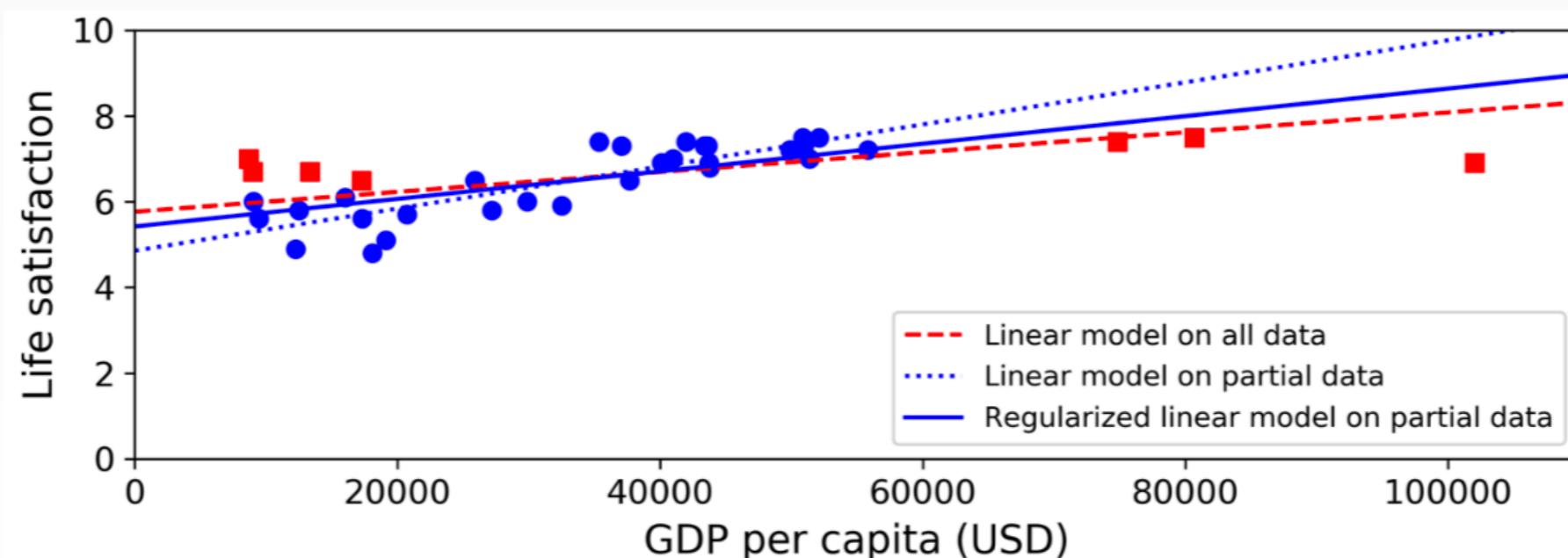
- Overfitting the training data
 - When the model is too complex relative to the amount and noisiness of data



- How to alleviate this problem?
 - Simplify the model
 - Gather more training data
 - Reduce noise and outliers (i.e., data preparation)

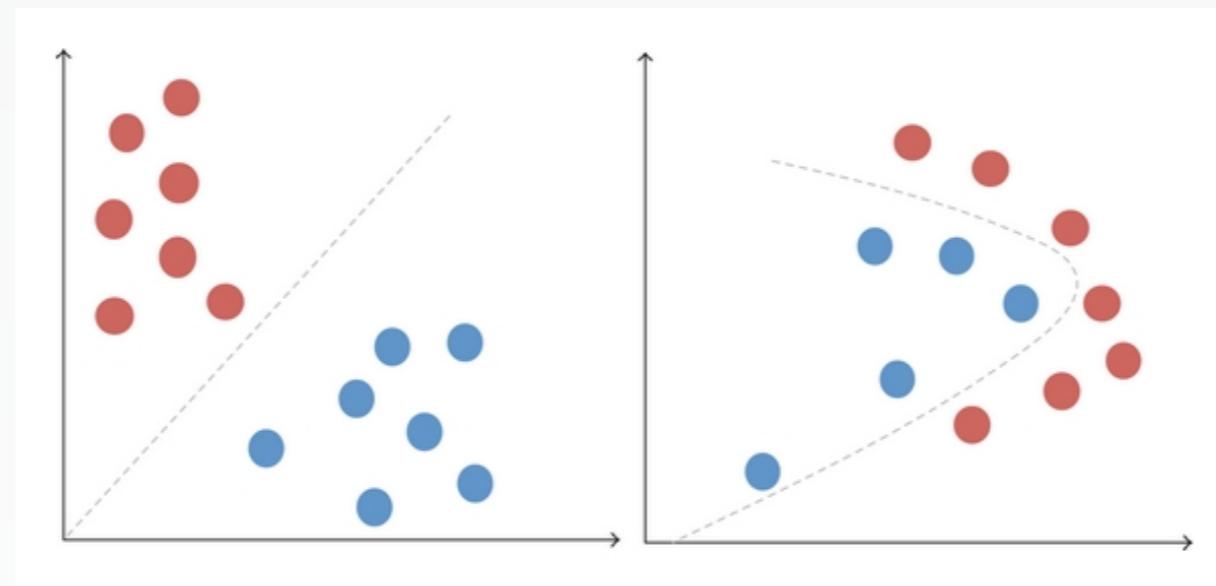
Example of reducing overfitting

- Use a subset of data and regularization
 - Regularization: balance between fitting the data perfectly and keeping the model simple enough
 - Amount of regularization is controlled by a hyperparameter (must be set prior to learning and remains constant during training)
 - Tuning hyperparameters is extremely important



Challenges of machine learning

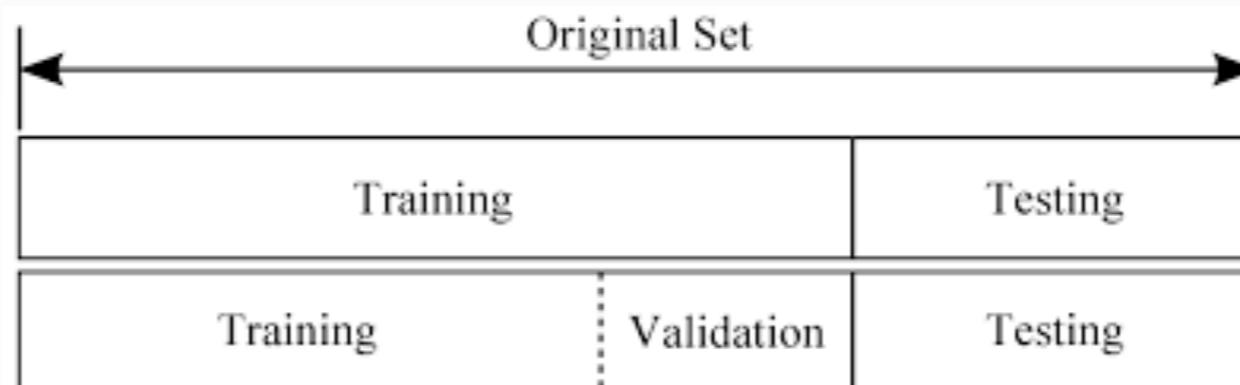
- Underfitting the training data
 - When the model is too simple



- How to solve this problem?
 - Select a more powerful model
 - Reduce constraints on the model such as regularization

Testing and validation

- The only way to know how a model will generalize to new cases is to actually try it out on new cases
- Split the available data into two sets: training set and testing set
 - The error on the test set is called the generalization error or out-of-sample error
 - If training error is low and generalization error is high, then overfitting occurs
- How to choose suitable models and their hyperparameters?



Basics of Python

<https://github.com/farhad-pourkamali/machine-learning>

Review of linear algebra and NumPy

<https://github.com/farhad-pourkamali/machine-learning>

Part 1: <https://youtu.be/xSO0qOyvevc>

Part 2: <https://youtu.be/SeBRHg9ZrSs>

Introduction

- NumPy offers mathematical functions, random number generators, linear algebra routines, and more
 - “de-facto” standards of array computing
- 1D (vector) and 2D arrays (matrix)

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbb{R}^n$$

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & & a_{nm} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

Creating arrays

```
[1] import numpy as np
    print(np.__version__)

⇒ 1.18.5

[2] np.zeros(3)

⇒ array([0., 0., 0.])

[3] np.zeros((4,5))

⇒ array([[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]])
```

```
[4] print(np.zeros(3).shape, np.zeros((4,5)).shape)

⇒ (3,) (4, 5)
```

```
[5] np.zeros((4,5)).ndim # number of array dimensions (each dimension is called axis)

⇒ 2
```

- Notice the difference between 1D and 2D arrays

Creating arrays

```
[6] np.ones((3,4))
```

```
↳ array([[1., 1., 1., 1.],
          [1., 1., 1., 1.],
          [1., 1., 1., 1.]])
```

```
[7] np.array([[1,2,3,4], [10, 20, 30, 40]]) # convert regular python array
```

```
↳ array([[ 1,  2,  3,  4],
          [10, 20, 30, 40]])
```

```
[8] type(np.array([[1,2,3,4], [10, 20, 30, 40]]))
```

```
↳ numpy.ndarray
```

```
[9] np.arange(1,5)
```

```
↳ array([1, 2, 3, 4])
```

```
[10] np.arange(1, 5, 0.5) # step parameter
```

```
↳ array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

```
[11] print(np.linspace(0, 4, 8))
```

```
↳ [0.           0.57142857 1.14285714 1.71428571 2.28571429 2.85714286
    3.42857143 4.           ]
```

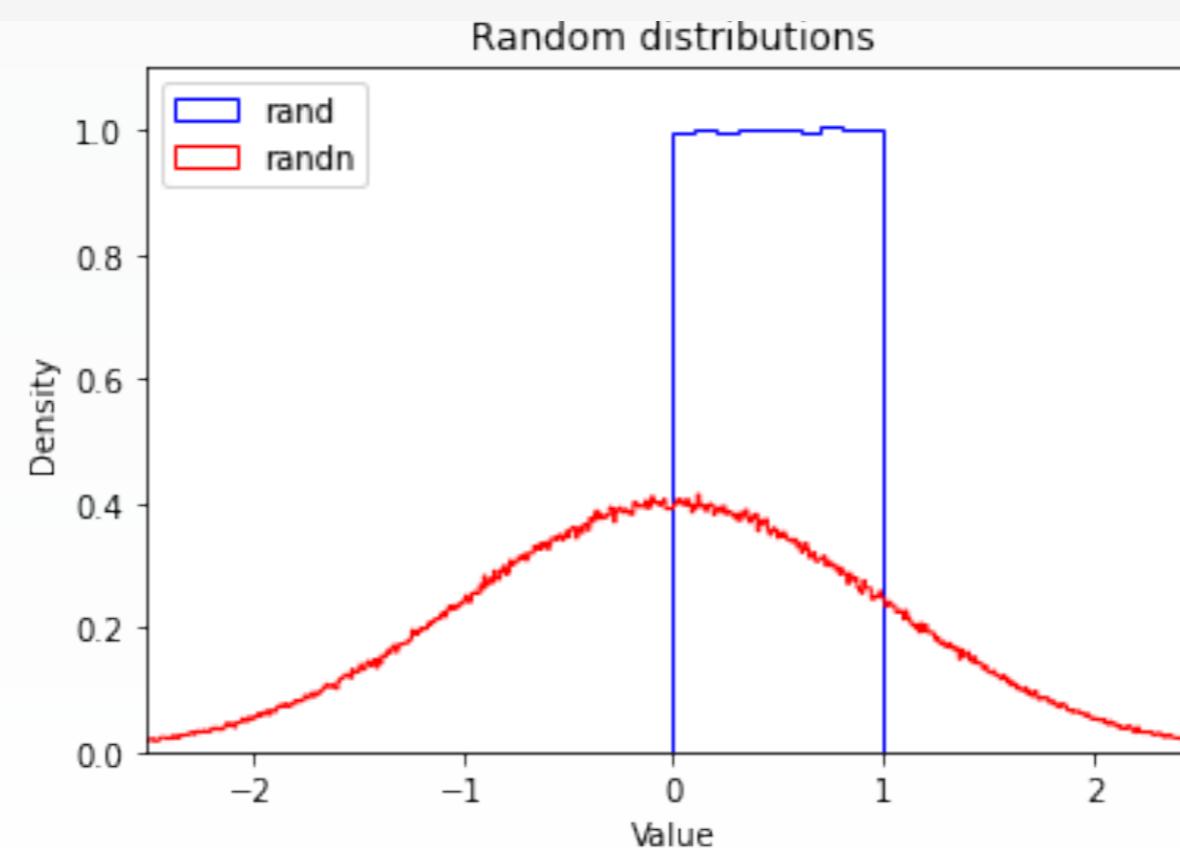
```
[12] 4/7
```

```
↳ 0.5714285714285714
```

Creating random numbers

- Two popular choices are uniform distribution between 0 and 1, and a univariate normal distribution

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(np.random.rand(1000000), density=True, bins=10, histtype="step", color="blue", label="rand")
plt.hist(np.random.randn(1000000), density=True, bins=1000, histtype="step", color="red", label="randn")
plt.axis([-2.5, 2.5, 0, 1.1])
plt.legend(loc = "upper left")
plt.title("Random distributions")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()
```



Array data

- NumPy's ndarrays are efficient because their elements must have the same type
 - Data types: <https://numpy.org/doc/stable/user/basics.types.html>

```
c = np.arange(1, 5)
print(c.dtype, c)

int64 [1 2 3 4]

c = np.arange(1.0, 5.0)
print(c.dtype, c)

float64 [1. 2. 3. 4.]

z = np.arange(3, dtype=np.uint8)
print(z.dtype)

uint8
```

Array data

- NumPy's ndarrays are efficient because their elements must have the same type

```
import numpy as np
import time

class Timer:
    """Record multiple running times."""
    def __init__(self):
        self.times = []
        self.start()

    def start(self):
        """Start the timer."""
        self.tik = time.time()

    def stop(self):
        """Stop the timer and record the time in a list."""
        self.times.append(time.time() - self.tik)
        return self.times[-1]
```

```
n = 10000
a = np.ones(n)
b = np.ones(n)

c = np.zeros(n)
timer = Timer()
for i in range(n):
    c[i] = a[i] + b[i]
f'{timer.stop():.5f} sec'
```

'0.01340 sec'

```
timer.start()
d = a + b
f'{timer.stop():.5f} sec'
```

'0.00019 sec'

Reshaping an array

```
g = np.arange(18)
```

```
g.shape = (6, 3)
print(g)
print("Rank:", g.ndim)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]
 [15 16 17]]
```

```
Rank: 2
```

```
g2 = g.reshape(3,6)
print(g2)
print("Rank:", g2.ndim)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
```

```
Rank: 2
```

```
g2[1, 2] = 1000
g2
```

```
array([[ 0,    1,    2,    3,    4,    5],
       [ 6,    7, 1000,    9,   10,   11],
       [12,   13,   14,   15,   16,   17]])
```

Reshaping an array

- The reshape function returns a new array object pointing at the same data.

```
g2 = g.reshape(3,6)
print(g2)
print("Rank:", g2.ndim)
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
Rank: 2
```

```
g2[1, 2] = 1000
g2
```

```
array([[ 0,    1,    2,    3,    4,    5],
       [ 6,    7, 1000,    9,   10,   11],
       [12,   13,   14,   15,   16,   17]])
```

```
g # pointing at the same data
```

```
array([[ 0,    1,    2],
       [ 3,    4,    5],
       [ 6,    7, 1000],
       [ 9,   10,   11],
       [12,   13,   14],
       [15,   16,   17]])
```

Conditional operators

```
m = np.array([20, -5, 30, 40])
```

```
m<25
```

```
array([ True,  True, False, False])
```

```
m[m<25]
```

```
array([20, -5])
```