Need for Mathematical Language

Some Technical Background

- The Need for Mathematical Language
- Our Software for Tackling Machine Learning
- Probability
- Linear Combinations, Weighted Sums, and Dot Products
- A Geometric View: Points in Space
- Notation and the Plus-One Trick
- Getting Groovy, Breaking the Straight-Jacket, and Nonlinearity
- NumPy versus “All the Maths”
- Floating-Point Issues

The Need for Mathematical Language

- It is very difficult to talk about machine learning without discussing some mathematics.
- The following concepts from the mathematical world are used in machine learning:
  - ✓Algebra
  - ✓Probability
  - ✓Geometry
  - ✓Symbol

Our Software for Tackling Machine Learning

- The Python standard library includes **itertools**, **collections**, and **functools**.
- Members of the Python number-crunching and data science stack are **numpy**, **pandas**, **matplotlib**, **pytorch**, and **seaborn**.
- **numpy** is a Python package that stands for 'Numerical Python'.
- It is the core library for scientific computing, which contains a powerful n-dimensional array object.
- **pandas** is a Python package providing fast, flexible, and expressive data structures.
- It is designed to make working with relational or labeled data both easy and intuitive.

Our Software for Tackling Machine Learning (continued)

- It has two primary data structures, namely, **Series** and **DataFrame**.
- **Series** is one-dimensional, while **DataFrame** is two-dimensional.
- **pytorch** is built based on python and torch library which supports computations of tensors on Graphical Processing Units.
- The reason to use the number-crunching tools is because they form the foundation of, or work well with, **scikit-learn**.
- **sklearn** is a great environment for playing with the ideas of machine learning.
- It implements many different learning algorithms and evaluation strategies and gives a uniform interface to run them.

Our Software for Tackling Machine Learning (continued)

» The **tabula-py** library can read a table from PDF.

» It also enables you to convert a PDF file into a CSV/TSV/JSON file.

2.4 Probability

- 2.4.1 Primitive Events
- 2.4.2 Independence
- 2.4.3 Conditional Probability
- 2.4.4 Distributions

2.4 Probability (continued)

- Probability theory is the mathematical framework that allows us to analyze chance events in a logically sound manner.
- NumPy can be used to generate evenly weighted random events with **np.random.randint**.
- **randint** is designed to mimic Python's indexing semantics, which means that we include the starting point and we exclude the ending point.
- **np.histogram** is used to count up how many times each event occurred.
- It is designed around plotting buckets of continuous values.

2.4 Probability (continued)

- Samples can be drawn from a binomial distribution with specified parameters, **n** trials and **p** probability of success.
- Where, **n** is an **integer >= 0** and **p** is in the interval **[0,1]**.

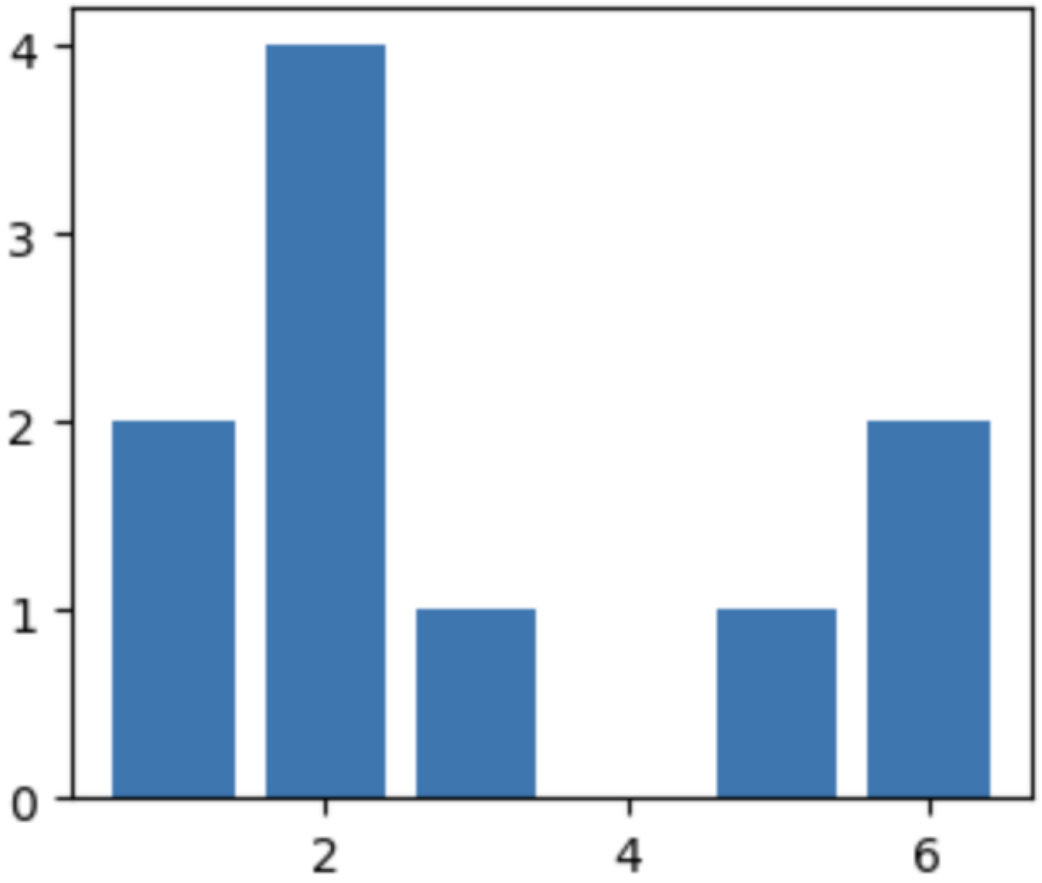
2.4 Probability (continued)

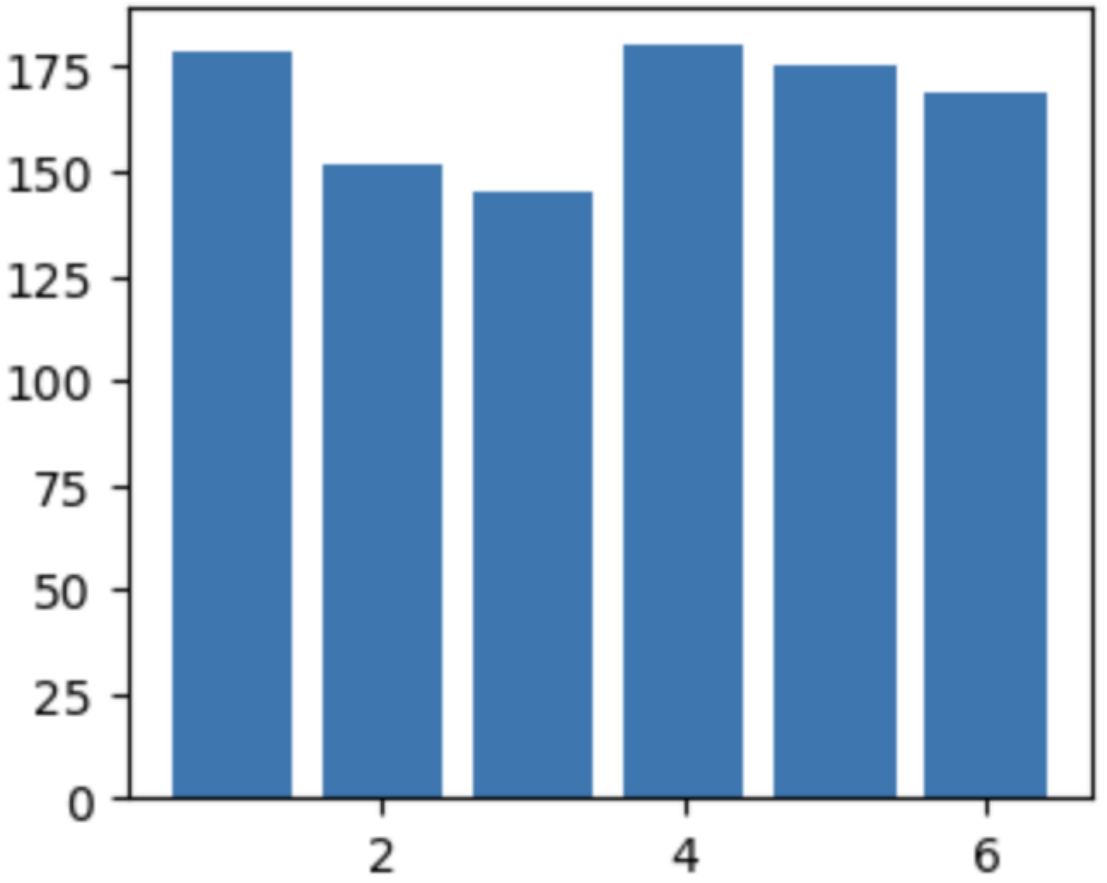
```
import numpy as np
import matplotlib.pyplot as plt

#np.random.randint(1,7)
fewrolls = np.random.randint(1,7, size=10)
manyrolls = np.random.randint(1,7, size=1000)


fewcounts = np.histogram(fewrolls, bins=np.arange(.5, 7.5))[0]
manycounts = np.histogram(manyrolls, bins=np.arange(.5, 7.5))[0]
  #General pattern:
  #For values 1 to N, use 0.5 to N+0.5
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8,3))
ax1.bar(np.arange(1,7), fewcounts)
ax2.bar(np.arange(1,7), manycounts)
```

2.4 Probability (continued)





2.4.1 Primitive Events

- An event is an outcome or defined collection of outcomes of a random experiment.
- We can get probabilities of compound events by either counting primitive events or adding up the probabilities of primitive events.

2.4.1 Primitive Events (continued)

- Here are some important aspects of probability:
  - ✓The sum of the probabilities of all possible primitive events in a universe is 1.
  - ✓The probability of an event not occurring is 1 minus the probability of it occurring.
  - ✓There are nonprimitive events.
  - ✓A compound event is a combination of primitive events.
  - ✓Compound events are recursive.
  - ✓We can create a compound event from other compound events.
  - ✓The two compound events overlap because they share primitive events.

2.4.2 Independence

- Two events are independent if knowing one event occurred doesn't change the probability of the other event.
- For example, if we flip a coin in the air and get the outcome as Head, then again if we flip the coin but this time we get the outcome as Tail.
- In both cases, the occurrence of both events is independent of each other.
- Independent probabilities work both ways: they are an if-and-only-if.

2.4.3 Conditional Probability

» Conditional probability is defined as the likelihood of an event or outcome occurring, based on the occurrence of a previous event or outcome.

2.4.4 Distributions

- The mapping between events and probabilities refers to a **probability distribution**.

  Example: Coin flip

  Event: getting **Heads**
  Probability: **0.5**
  Event: getting **Tails**
  Probability: **0.5**

  We've mapped:
  Heads → 0.5
  Tails → 0.5

- When a group of events shares a common probability value—like the different faces on a fair die—we call it a **uniform distribution**.

2.4.4 Distributions

» When a group of events shares a common probability value—like the different faces on a fair die—we call it a **uniform distribution**.

Example: A fair die has outcomes 1, 2, 3, 4, 5, 6

Each one has the **same probability**:

P(1) = 1/6

P(2) = 1/6

P(3) = 1/6

P(4) = 1/6

P(5) = 1/6

P(6) = 1/6

2.4.4 Distributions

» **Poisson distribution** describes the distribution of binary data from an infinite sample.

- A **Poisson distribution** is used when:
  - We count **how many times something happens**
  - In a **fixed interval** (time, space, area)
  - Events happen **independently**

examples: Each moment is basically **yes/no** (event happened or not), but we count **how many total events occur**.

- Number of emails you receive **per hour**
- Number of cars passing a toll booth **per minute**
- Number of typos on a page

Key idea:

Poisson = counting rare or random events over time or space.

2.4.4 Distributions

- **Gaussian distribution** (also known as **normal distribution**) is a continuous probability distribution wherein values lie in a symmetrical fashion around the mean.
- This is the famous **bell-shaped curve.**

Examples:

- Heights of students in a class
- Exam scores
- Measurement errors

Most values:

- Cluster around the **average (mean)**
- Fewer values appear as you move away from the mean

2.4.4 Distributions (continued)

- The normal distribution has the following characteristics:
  - ✓ Its midpoint has the most likely value.
  - ✓ It is symmetric.
  - ✓ It can be mirrored about its midpoint.
  - ✓ As we get further from the midpoint, the values fall off more and more quickly.

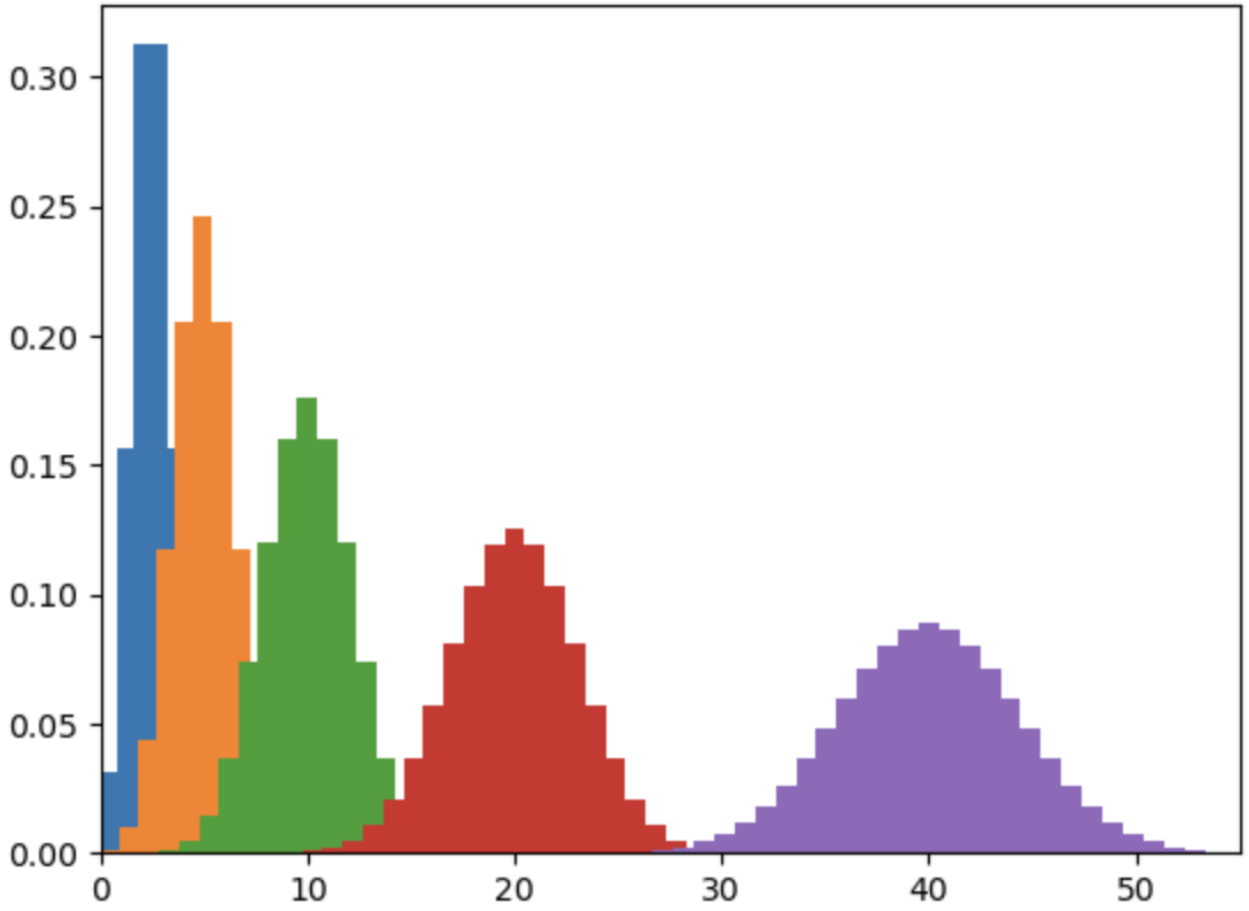
2.4.4 Distributions (continued)

Write a Python code to plot a probability distribution graph that illustrates the probability of flipping a coin 5 times.

```
Import scipy.stats as ss


b = ss.distributions.binom
for flips in [5, 10, 20, 40, 80]:
        success = np.arange(flips)
        ourdistb = b.pmf(success, flips, .5)
        plt.hist(success, flips, weights=ourdistb)
plt.xlim(0, 55)
```

2.4.4 Distributions (continued)



2.4.4 Distributions (continued)

Write a Python code to plot a probability distribution graph that illustrates the probability of flipping a coin 6 times using subplots.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as ss

# Binomial distribution
b = ss.distributions.binom

# Flip counts we want to display
flips_list = [5, 10, 20, 40, 80, 100]

# Create a 3x3 grid of subplots
fig, axes = plt.subplots(3, 3, figsize=(10, 8))
axes = axes.flatten() # make indexing easier
```

2.4.4 Distributions (continued)

```
for i, flips in enumerate(flips_list):
    # Possible number of successes (heads)
    success = np.arange(0, flips + 1)

    # Binomial probabilities
    probs = b.pmf(success, flips, 0.5)

    # Plot on the i-th subplot
    axes[i].bar(success, probs)
    axes[i].set_title(f"{flips} Flips")
    axes[i].set_xlabel("Number of Heads")
    axes[i].set_ylabel("Probability")

plt.tight_layout()
plt.show()
```

2.5 Linear Combinations, Weighted Sums, and Dot Products

- 2.5.1 Weighted Average
- 2.5.2 Sums of Squares
- 2.5.3 Sum of Squared Errors