12. Linear algebra and statistical distributions LING 471

Plan for next week

- Assignment 3 is due end of Tuesday next week
- Haotian will give two lectures next week

Learning outcomes

- Define **matrices** and **vectors**
- Create matrices and vectors in **NumPy**
- Perform basic **matrix and vector operations** in NumPy
- Describe a bar plot and a histogram
- Describe probability density
- Describe the Gaussian/normal distribution

Some caveats

- We will be covering a fair amount of **mathematical concepts** today
- Why?
  - This is a course about computational linguistics (and computational methods)
  - Most modern computational linguistics and methods revolve around machine learning
  - Machine learning is all math
- Most of the math requires multiple pre-reqs to understand deeply
  - As before, you will be responsible for concepts and using pre-built libraries, not implementing all of this yourself

Linear algebra: Matrices and vectors

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

(a) Column vector

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

(b) Row vector

Figure 1: Examples of vectors.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 8 \end{bmatrix}$$

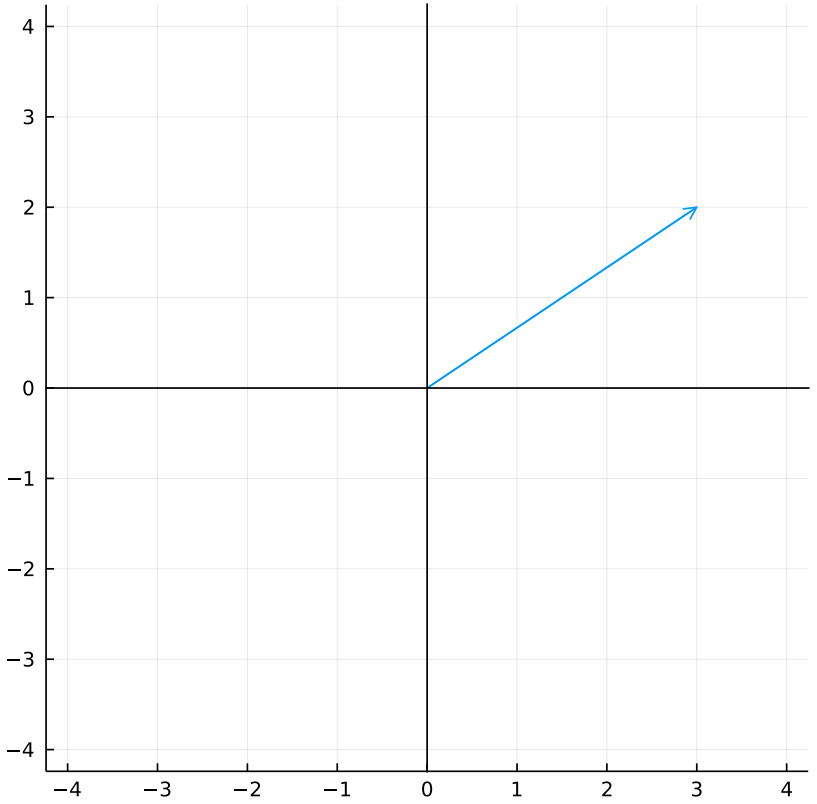
Figure 2: Example of a matrix.

Linear algebra

- A branch of mathematics that deals with vectors and matrices
- **Vector**: a **row** or **column** of individuals values ("**scalars**")
  - Like a row or column from a data frame!
- **Matrix**: one or more vectors together
  - Like a table/data frame!

Graphical representation of a vector

- When your vector has two numbers, it is common to think of it as a point on a graph
  - With an arrow pointing to it
- For example, $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ can be a line from (0, 0) to (3, 2)
- Often useful to think about linear algebra with simple 2D examples



Linear algebra

- An abstract branch of mathematics
- Often, working just with ideas of matrices and vectors rather than evaluating specific operations
- General goals: **perform mathematics** on **entire matrices and vectors**
  - **Multiply** them
  - Prove **theorems** about complex expressions involving them
  - Etc.

Some notes on notation

- I will be using Householder notation in these slides
- Matrices are represented with capital Roman letters: A, B, X, Y etc.
- Vectors are represented with lower-case Roman letters: a, b, x, y etc.
- Scalars (individuals numbers/elements) are represented with lower-case Greek letters: $\alpha, \beta, \chi, \psi$, etc.
- The Greek letters are supposed to be similar in shape to the Roman letters, e.g.,
  - Matrix: A
  - Vector of A: a
  - Scalar in A or a: α

Doing linear algebra in Python: NumPy

- Python does not have the ability to handle some data types built in
  - For example, vectors and matrices
- So, we use a library called NumPy to provide that functionality
- I will show some example code in the slides
  - Note that it is typical to import with `import numpy as np`

Making a matrix in NumPy

- NumPy has a general concept of an "array"
  - An array can be a vector, a matrix, or have even more dimensions
- Matrices are made as lists of lists in NumPy
  - Each sublist is a row, by default

```
np.array([[1, 2], [3, 4]))
# array([[1, 2],
#        [3, 4]])
np.array([[1, 3], [2, 4]])
# array([[1, 3],
#        [2, 4]])
```

Matrix/vector transpose

- An operation you can perform on a matrix or vector
  - Can think of vector as a matrix w/ 1 row or 1 column
- Makes columns rows and vice-versa
- Current mathematical notation: a superscript T
  - X^T

```
A = np.array([[1, 2], [3, 4]))
# A = array([[1, 2],
                [3, 4]])
A_T = A.T
# A_T = array([[1, 3],
                  [2, 4]])
```

Dot product

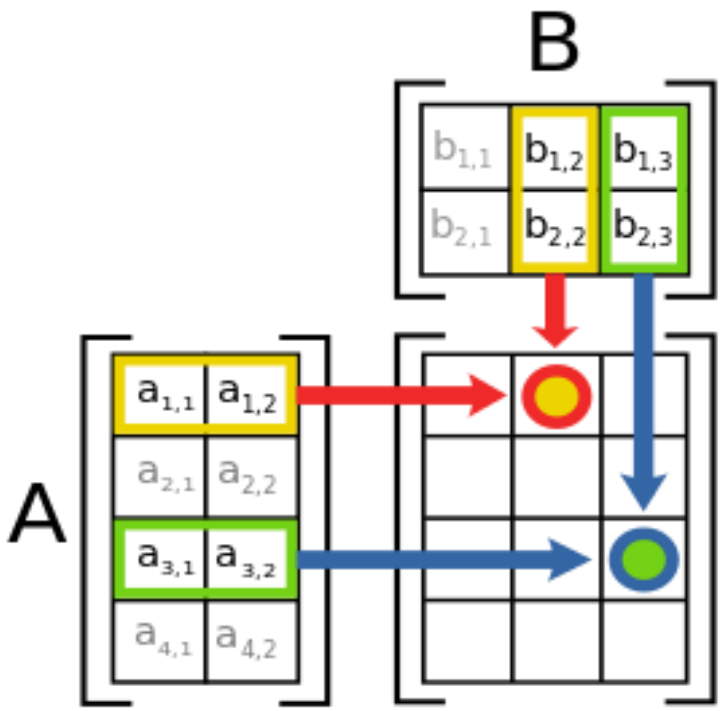
- Operation on two vector of the same length
- Notation options:
  - $a \cdot b$
  - $a^T b$
- Multiply corresponding elements in the two vectors and then sum the results

- $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
- $b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$
- $a \cdot b = 1 * 4 + 2 * 5 + 3 * 6$
- $= 4 + 10 + 18 = 32$

Dot product in NumPy

- This is kind of annoying to do…
- There is a function `numpy.dot`, but it is annoying to use
- An array also has a `dot` method, but it is identical to `numpy.dot`
- Probably best option: `@` operator (does matrix multiplication)

```
a = np.array([[1], [2], [3]])
b = np.array([[4], [5], [6]])
np.dot(a.T, b) # array([[32]])
a.T.dot(b) # array([[32]])
a.T @ b # array([[32]])
```



Matrix multiplication

- For two matrices A and B, with dimensions NxM and MxK (row x column):
  - The product is an NxK matrix
  - Each cell is the dot product of the i-th row of A and the i-th column of B
  - N and K can be different, but M must match
- Notation: AB (just put the matrix names next to each other)

Matrix multiplication example

- $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
- $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$
- $AB = \begin{bmatrix} 1*5+2*7 & 1*6+2*8 \\ 3*5+4*7 & 3*6+4*8 \end{bmatrix} = \begin{bmatrix} 5+14 & 6+16 \\ 15+28 & 18+32 \end{bmatrix}$
- $AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Matrix multiplication in NumPy

- You rarely ever want to do matrix multiplication by hand
  - And when you do, you rarely ever go above 2x2 or 3x3 matrices!
- NumPy does this for us

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
A @ B
# array([[19, 22],
#        [43, 50]])
```

What to know about matrix multiplication

- There is a specified, structured method to multiply two matrices together
  - And you can treat a vector as a matrix
- The dimensions have to match between the matrices (the inner two dimensions match)
  - Nx**M** matrix and **M**xP matrix can be multiplied
  - Nx**M** matrix and **Q**xP matrix can**not** be multiplied

Scalar multiplicative inverse

- What do we do if we want to invert (reverse) multiplying by a number?
- E.g., if we multiply 2 by 3 and get 6, what number do we multiply 6 by to get back to 2?
- We multiply by the reciprocal of 3, that is, 1/3
- For single real numbers except for 0, a reciprocal is guaranteed to exist

A formal definition

- The multiplicative inverse of a real number χ is the real number ψ such that $\chi \cdot \psi = 1$
- If we solve for ψ, the reciprocal, we see that $\psi = 1/\chi$
- So, thinking back to our example of 3, we see that the reciprocal is 1/3, and

$$3\ \frac{1}{3} = \frac{3}{3} = 1$$

The identity matrix

- We want to extend our formal definition of inverses to matrices, but what kind of matrix could represent 1, and what would that mean?
- Well, when you multiply a number by 1, you get that number back
  - i.e., $1\alpha = \alpha$
  - We want a matrix that does this…
- The matrix with the same property is called the **identity matrix**, denoted I
- The identity matrix is square (n x n) and has 1s along the top-left to lower-right diagonal and 0s elsewhere
- For 2 dimensions, $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- For 3 dimensions, $I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Etc.

Identity matrix example

- $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
- $B = I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- $AB = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 1 \\ 3 \cdot 1 + 4 \cdot 0 & 3 \cdot 0 + 4 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Matrix multiplicative inverse

- For a matrix A, we are defining the multiplicative inverse of A, denoted A^{-1}, to be a matrix such that $AA^{-1} = I$
  - How abstract!
  - Note, we are saying A^{-1} is the inverse of A
- We aren't going to go through the math of determining the inverse
  - In fact, we rarely ever have to do this on actual matrices
  - But it is useful to think about this conceptually
- Note, however, that not all matrices have an inverse!

Matrix inverses in NumPy

- NumPy will calculate the inverse of a matrix for you if possible
- The function is `numpy.linalg.inv`
- It will give you an error if a matrix cannot be inverted

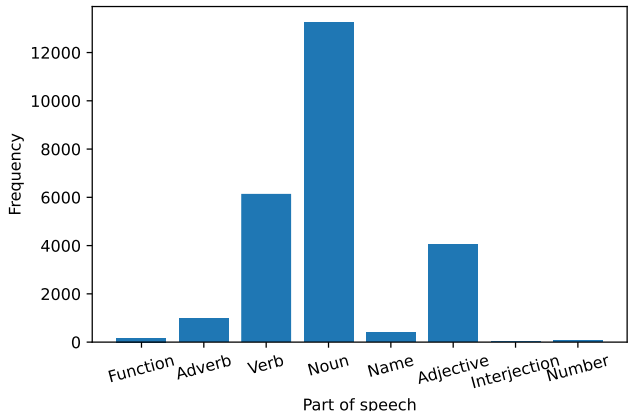
```
A = np.array([[1, 2], [3, 4]])

Ainv = np.linalg.inv(A)

# array([[-2. ,  1. ],

          [ 1.5, -0.5]])

A @ Ainv

# array([[1.0000000e+00,
0.0000000e+00],

       [8.8817842e-16,
1.0000000e+00]])

# The above is effectively 1s and 0s
```

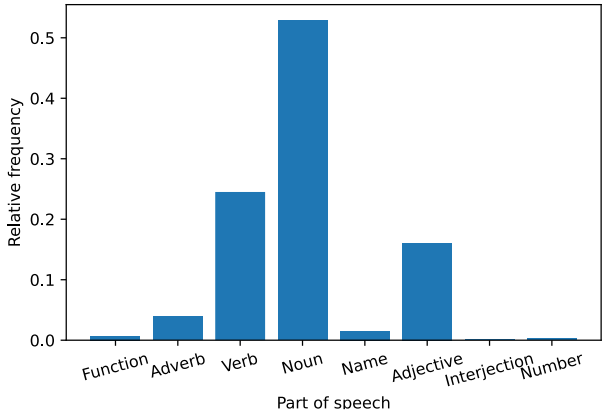
Programming exercises

- Assign the following vectors and matrices
- $a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- $b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$
- $A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- $B = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$

- Compute the following
  1. The dot product, $a \cdot b$
  2. The dot product $b \cdot a$
  3. The matrix-vector product Aa
  4. The matrix-vector product Bb
  5. The matrix-matrix product AB
  6. The matrix-matrix product BA
  7. The inverse of A
     a) AA^{-1}
  8. The inverse of B
     a) BB^{-1}

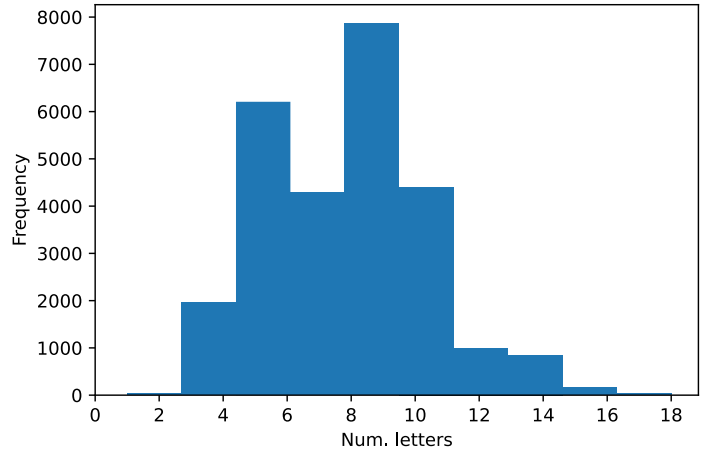
Probability mass, density, and distributions

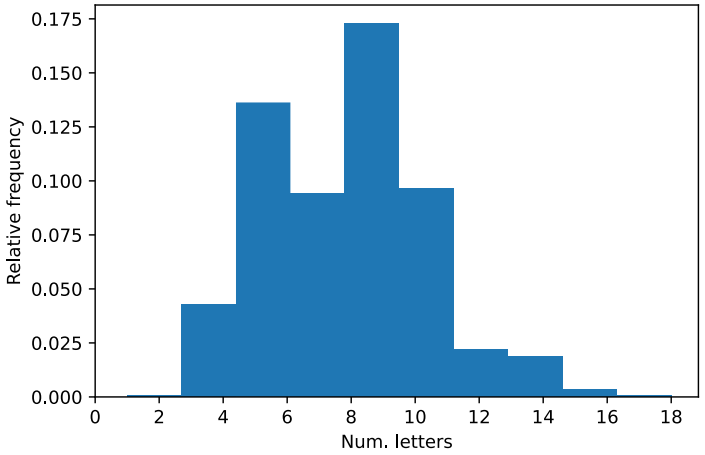




Bar plots

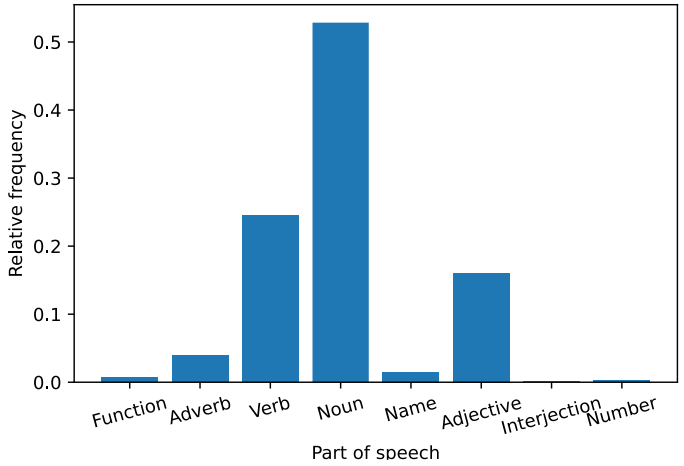
- Last week, we talked about frequency of occurrence and relative frequency
- We can visualize **category** frequencies with **bar plots**
- Either raw or relative frequency can be visualized
- When using relative frequency, we are approximating a **probability mass function**

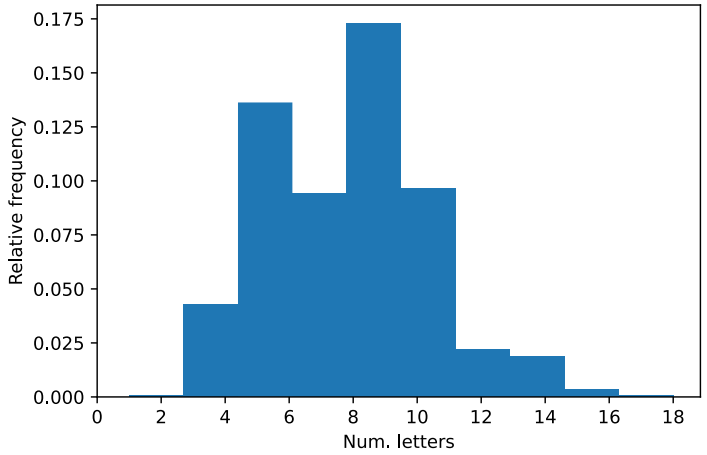




Histograms

- We can also visualize the frequency of non-categorical data with a histogram
- Must first create discrete bins (basically, numerical categories)
- When using relative frequencies, we are approximating the **probability density function**





Probability mass vs. density

- Probability **mass** is for **categorical** outcomes
  - Part of speech, coin flips, rolling dice, etc.
- Probability **density** is for **continuous** outcomes
  - Word/review length (sort of), word/utterance duration, height, etc.
- Notice the different spacing between bars for bar plots and histograms

Probability distributions

- Recall that we said the probabilities of mutually exclusive events need to sum to 1
- For a particular sample space, we need to **distribute** those probabilities (or relative frequencies)
  - I.e., make a **probability distribution**
- Not all possible distributions will distribute the probability evenly or **uniformly**
  - See: bar plots and histograms on previous slides!

Notes on probability distributions

- Centuries ago, mathematicians noticed similar shapes kept appearing when analyzing data
- These commonly occurring shapes were named and formalized
  - Examples: uniform, Gaussian/normal (named for Gauss), Bernoulli/binomial (named for Bernoulli)
- However, not all distributions resemble known functions
  - We will often approximate them with known functions, though

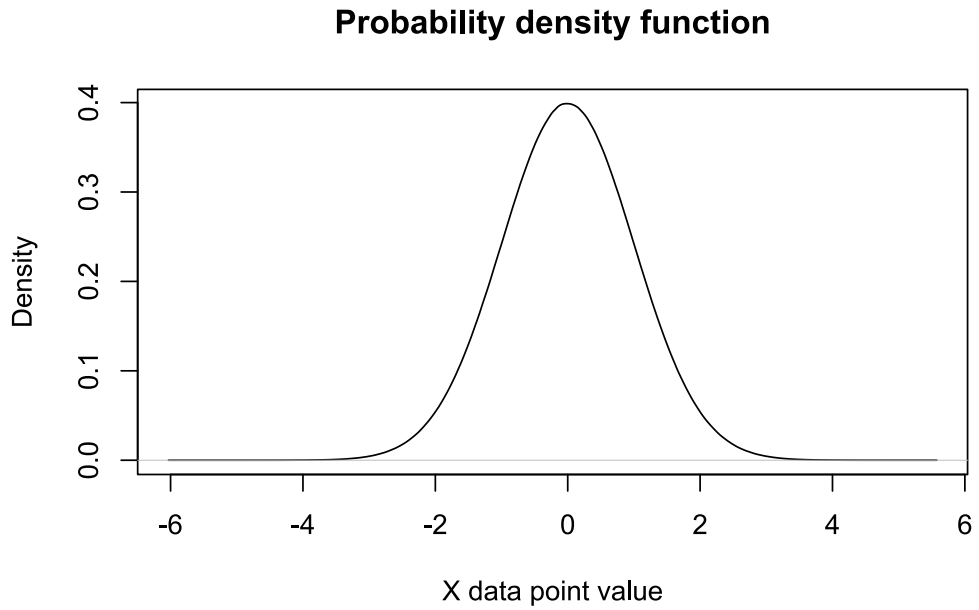
Continuous variables and distributions

Continuous random variables

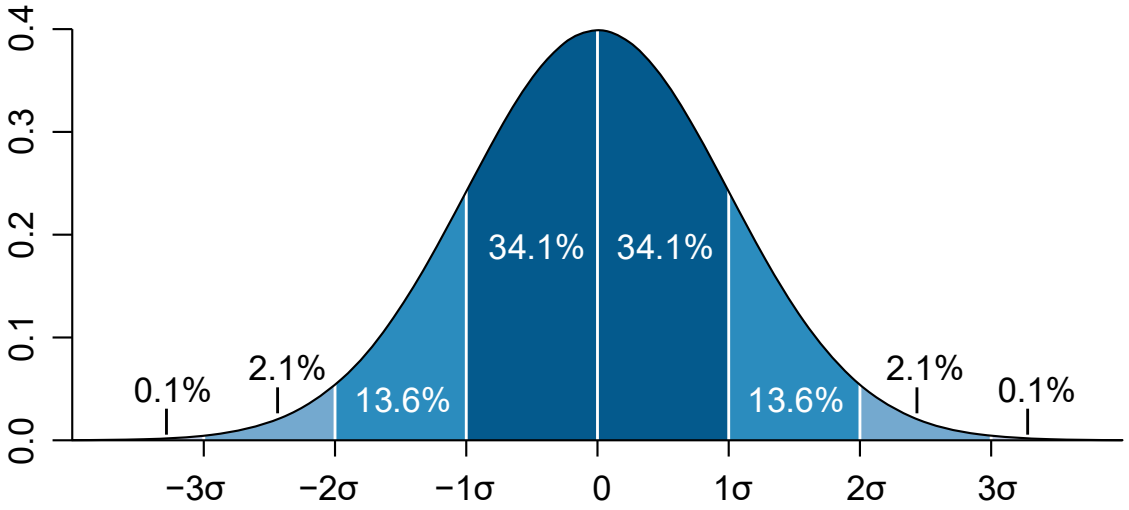
- Some random variables are **continuous**
  - Bascially, your values should theoretically be able to "reach" infinity or have infinite possible values (even if you can't realistically observe some of them)
    - E.g., age: 25.31415 years old
    - E.g., a word could, theoretically, have infinite letters
      - And number of letters is a proxy for how long it takes to say a word
  - Contrast with a discrete variable like a coin flip (either H or T, no in between, no infinite possible values)

Probability density functions (PDFs)

- A continuous random variable has a probability **density**
- Area under curve must sum to 1
- Each specific point is a density, not a probability!
  - Probability requires calculating the area under a region (integrating)
- Most common PDF use for continuous variables is **normal/Gaussian distribution**



The normal/Gaussian distribution

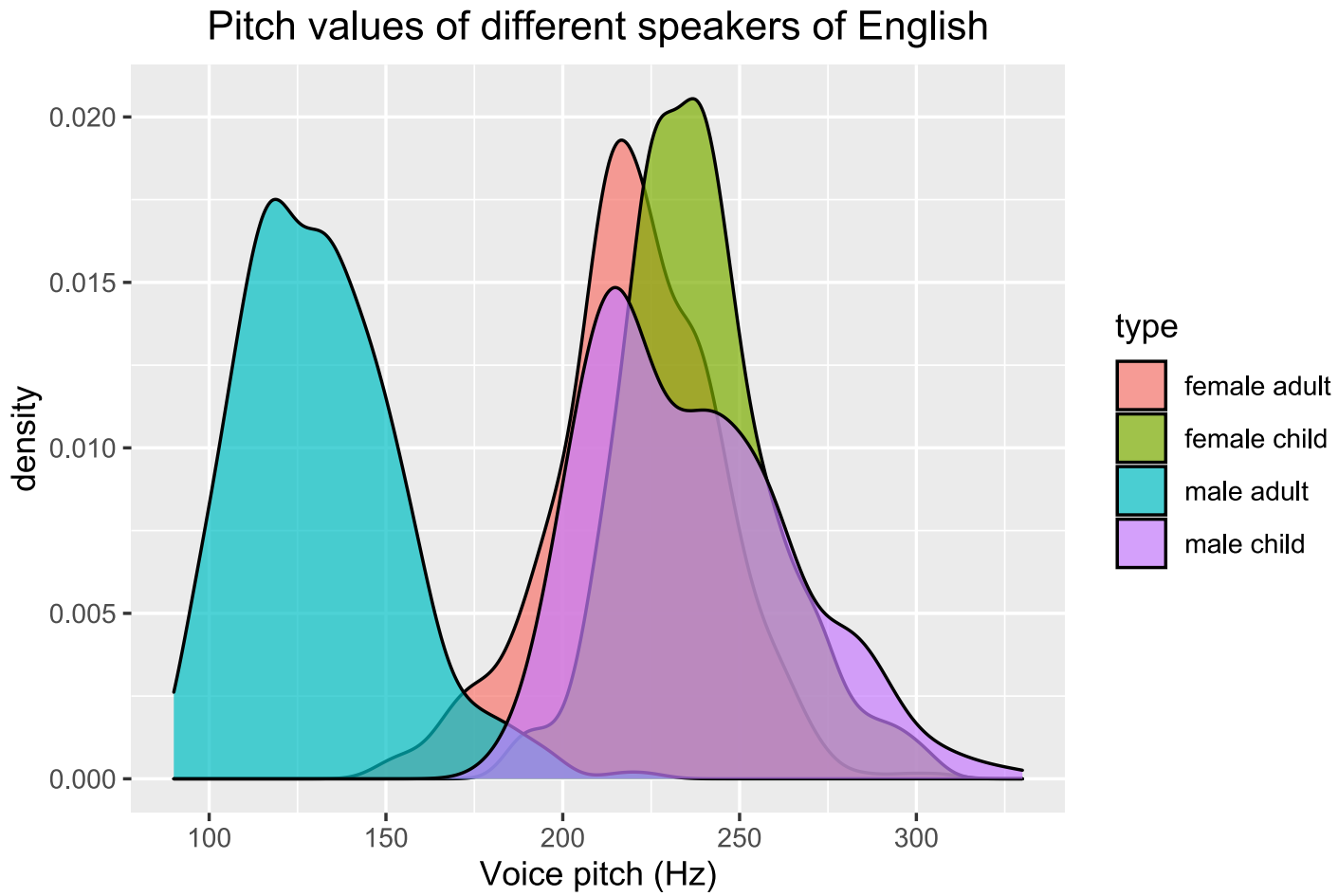


- A lot of data in the world is normally distributed
  - Or can reasonably be approximated as such
- Has two **parameters**

1. Mean (signified μ or m)
   - Indicates where the middle of the distribution is; most typical/likely value
2. Standard deviation (signified σ or s)
   - Indicates how much **spread** or **variation** there is in the distribution

Normal distribution notes

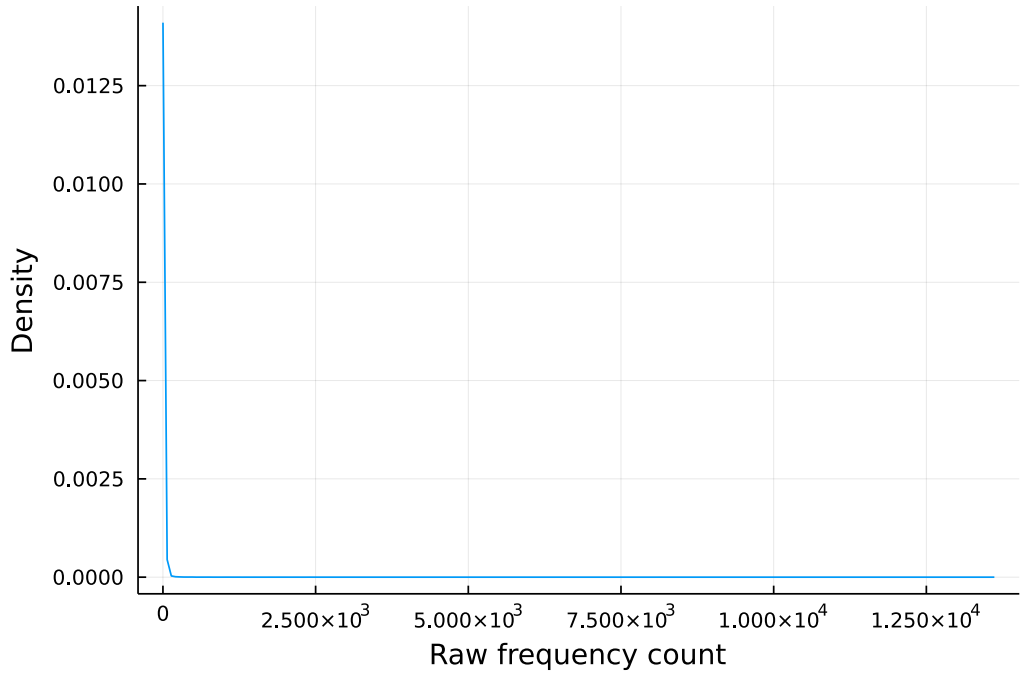
- Greek letters used for population, Roman letters for sample
- Sometimes, variance (var) is used instead of standard deviation
  - $var = \sigma^2$
- Etymology of "normal"
  - Gauss used it to refer to orthogonality (right angles)
  - Today, "normal distribution" doesn't really break down into "normal" + "distribution"
  - Rather, "normal" is used as a label (with no other meaning)

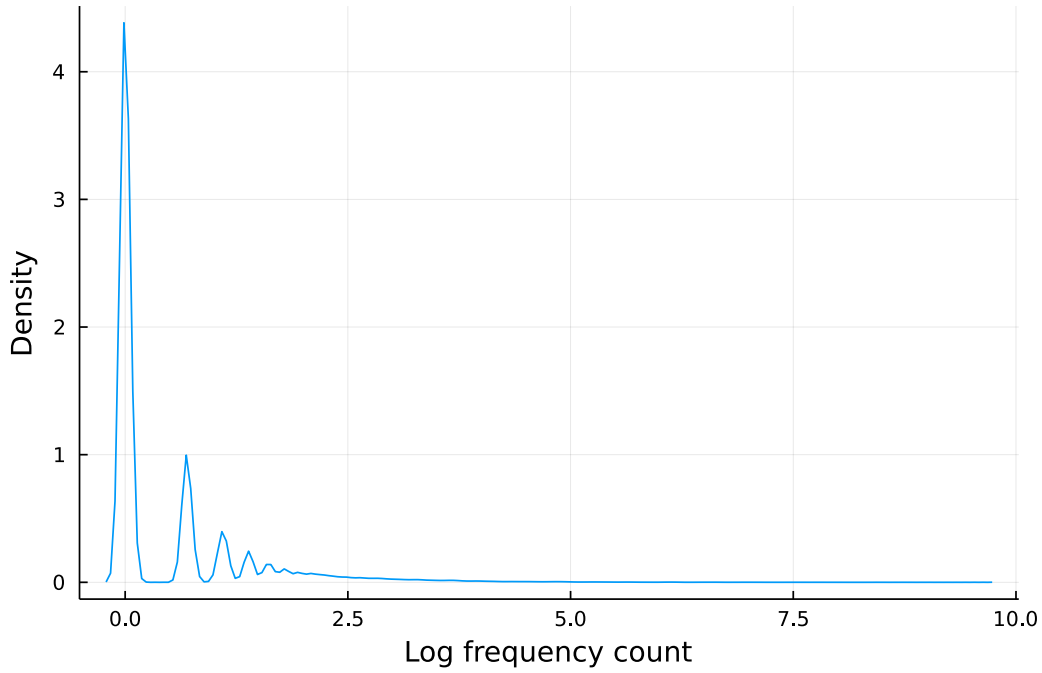


Language data and the Gaussian distribution

- Is language data reasonably normal?
  - It depends…
- Some of it is!
  - See pitch plot here
- Some of it isn't!
  - See frequency data

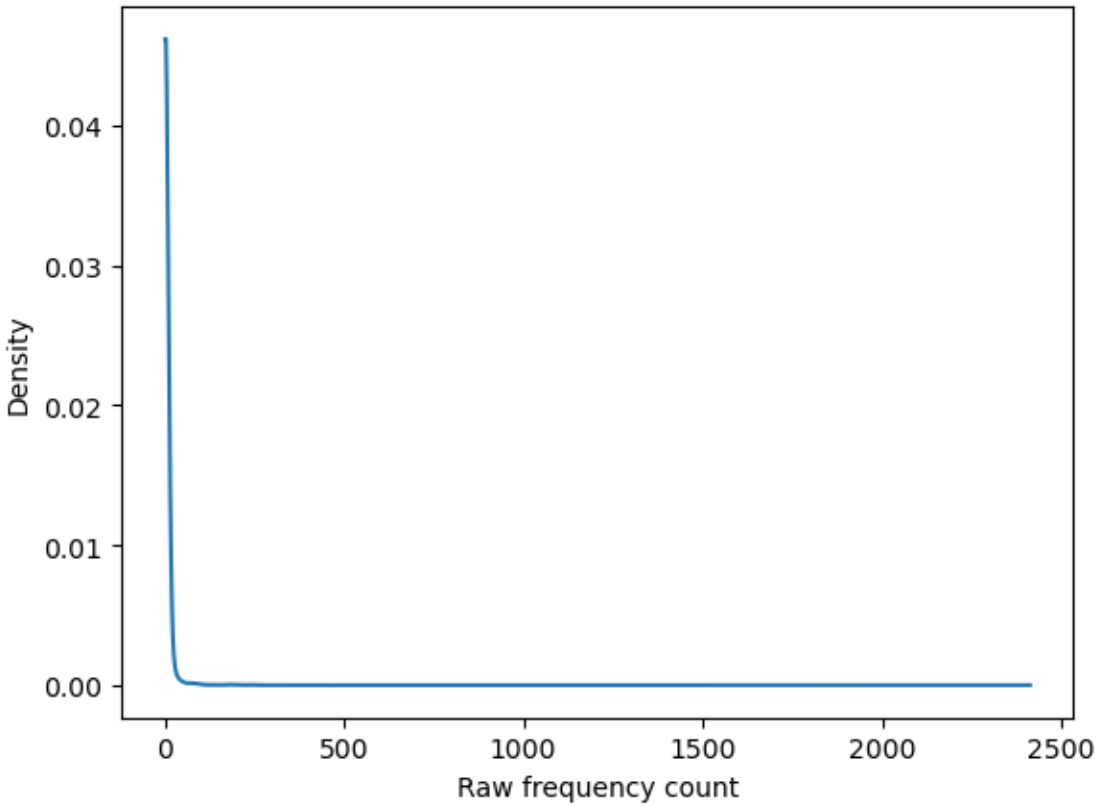
Probability density of frequency counts in *Ulysses*

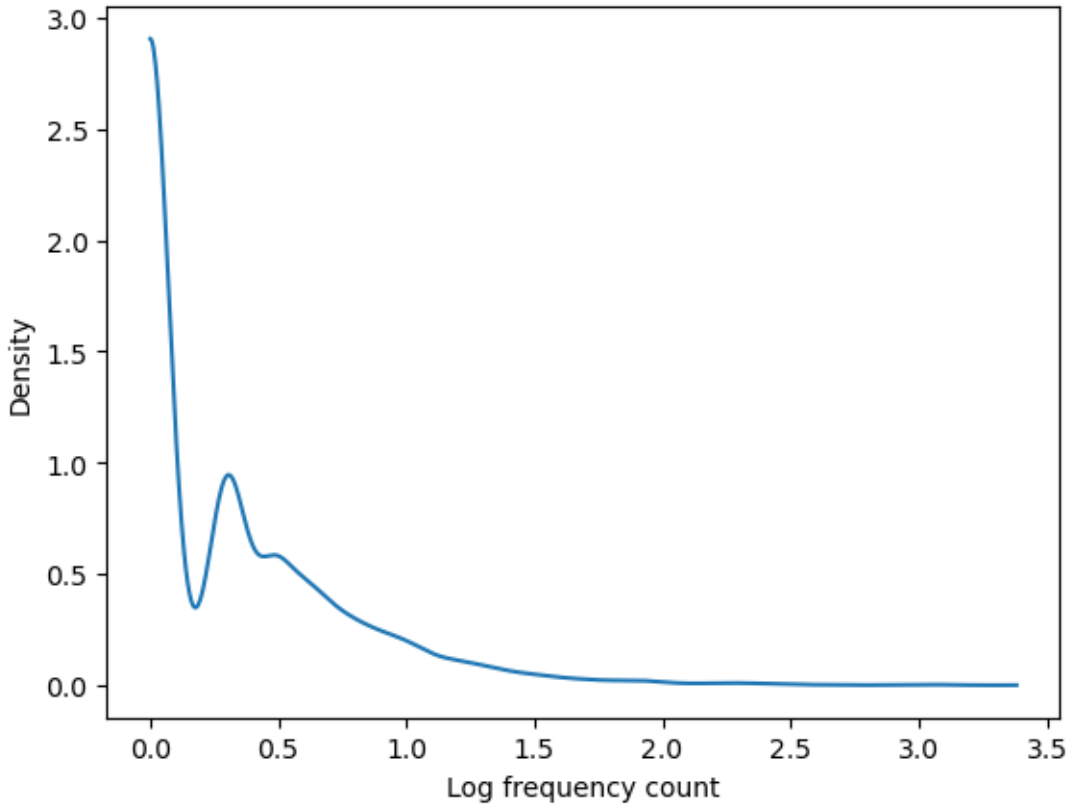




Clearly not Gaussian…

Probability density of frequency counts in our class corpus





Normal distribution features

- Mean is calculated as the average of all possible outcomes
- Mean is most likely value
- About 67% of data is within 1 standard deviation from the mean
- About 95% of data is within 2 standard deviations from the mean

- How to calculate Gaussian probability density
  - $pdf(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
- If you write a function to do this, you'll want to split this up a bit
  - Usually, you want to use someone else's code to do this…