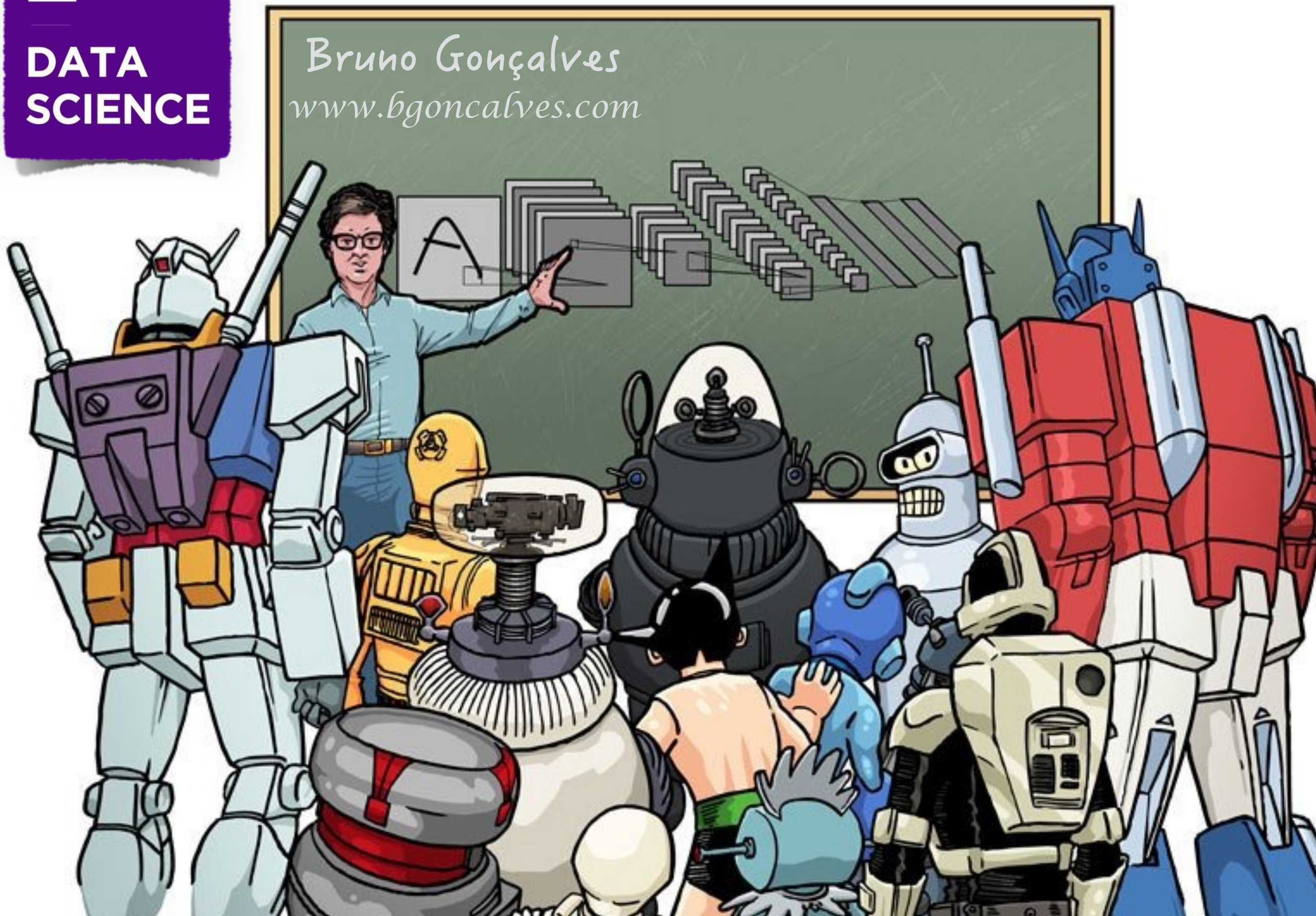
NYU

DATA SCIENCE

# Machine(s) Learning Data Science

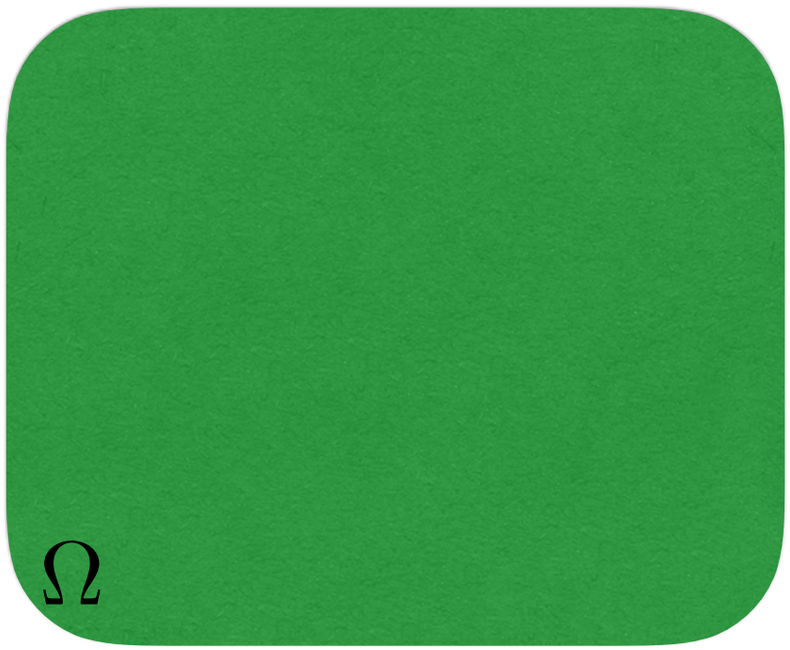
Bruno Gonçalves

www.bgoncalves.com



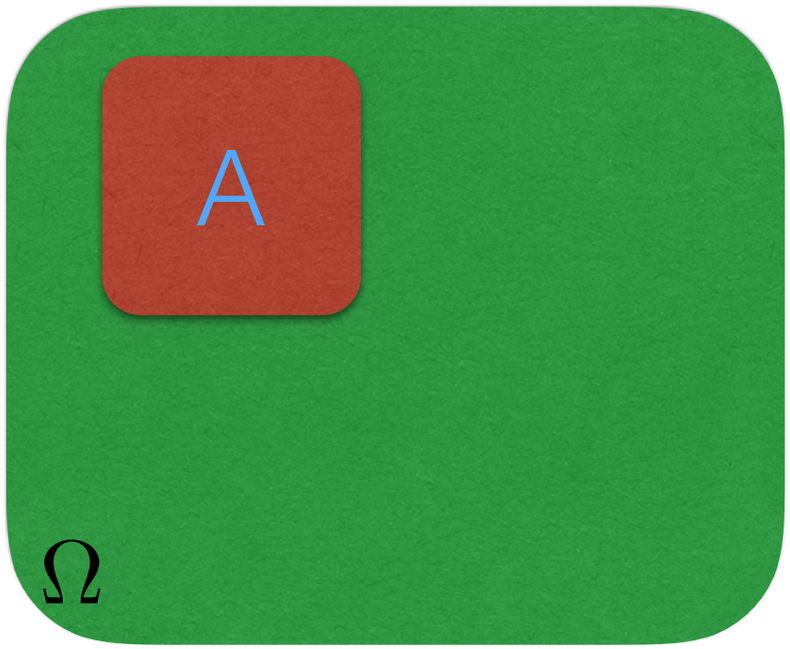
# Probability

# Probability



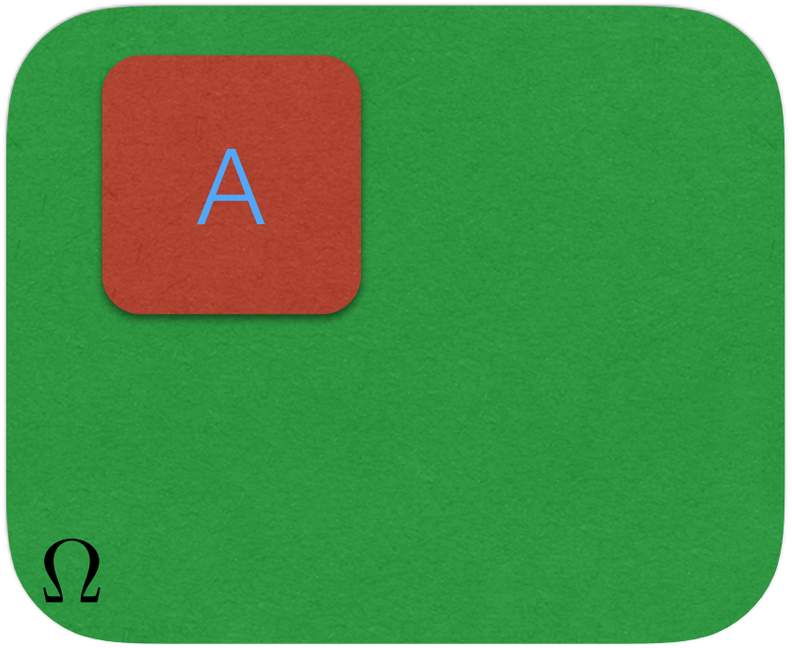
# Probability

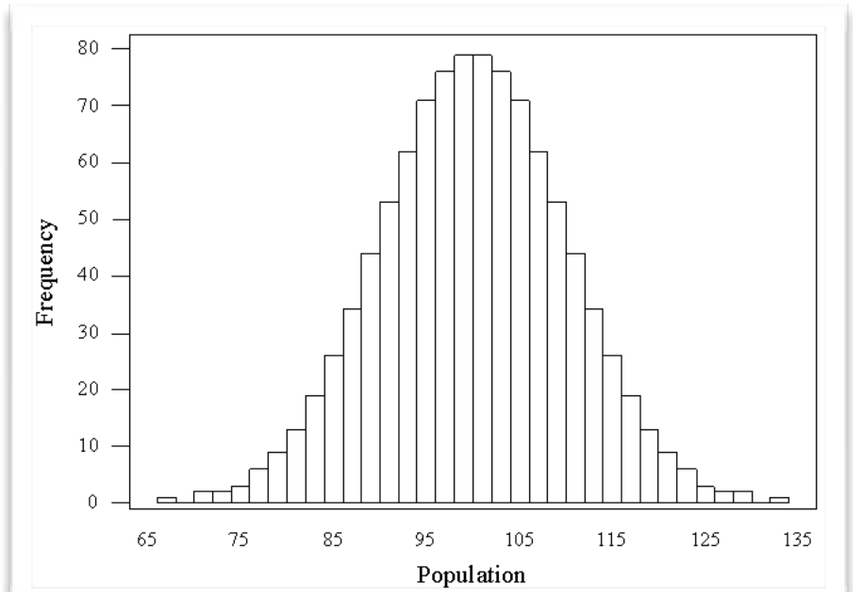
$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$




# Probability

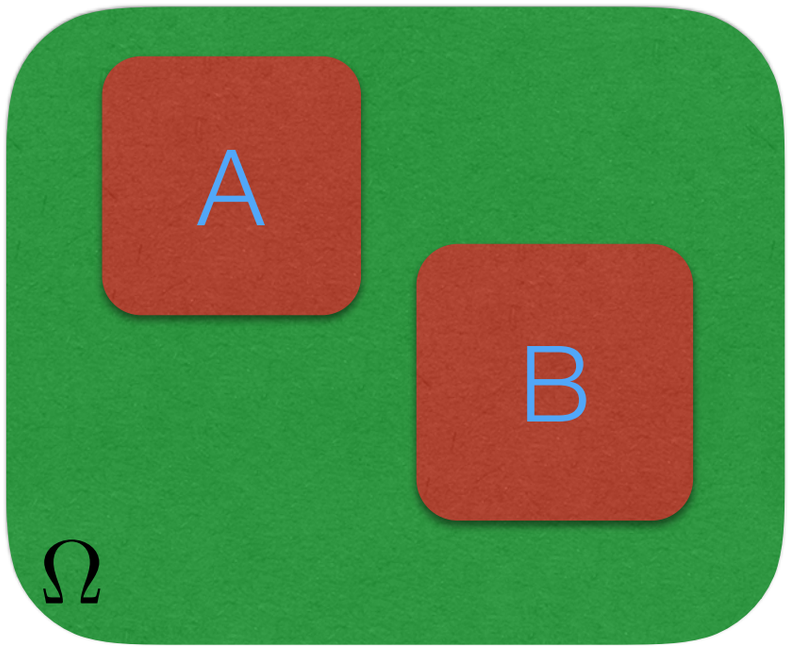
$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$

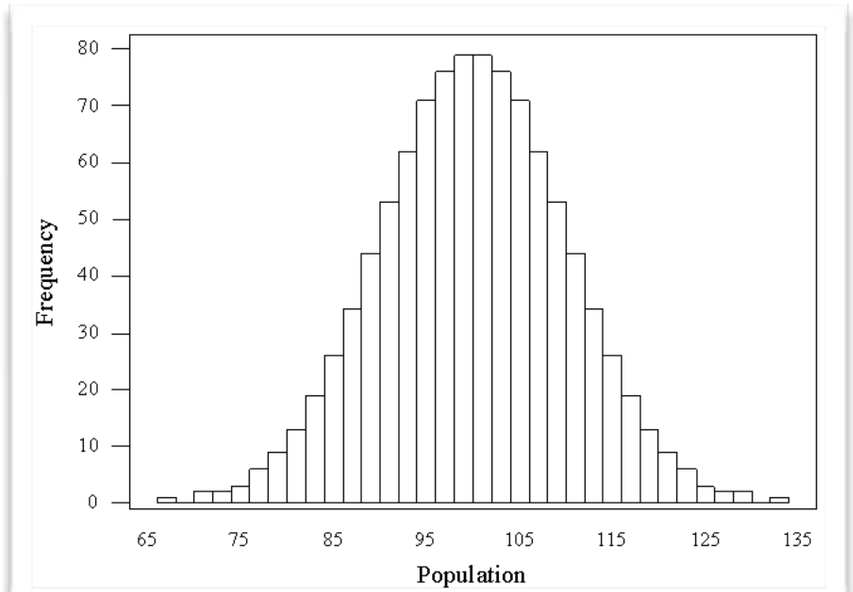


# Probability

$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$

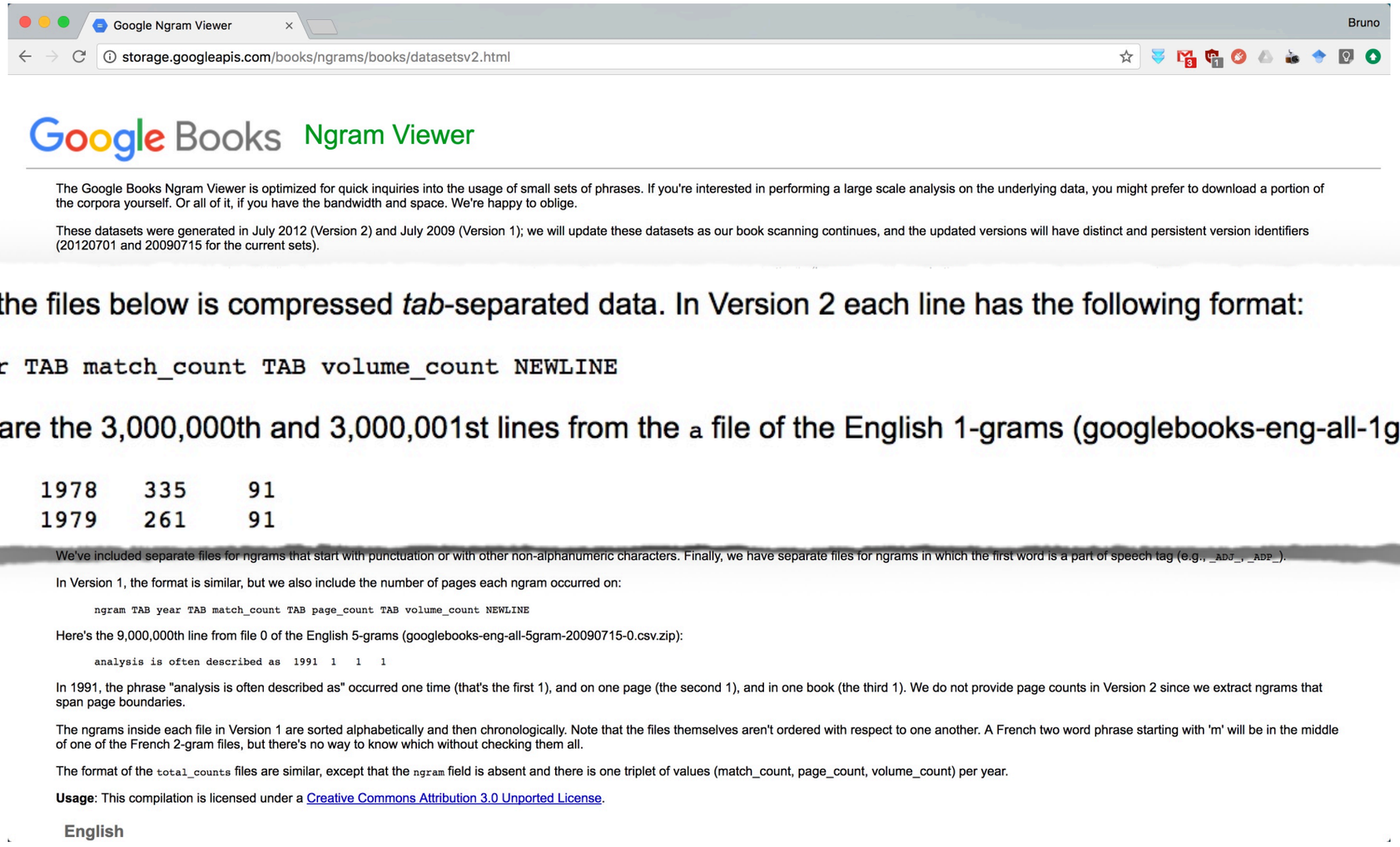



$$P(A \text{ or } B) = P(A) + P(B)$$

# Character Probabilities

http://storage.googleapis.com/books/ngrams/books/datasetsv2.html



Google Ngram Viewer

storage.googleapis.com/books/ngrams/books/datasetsv2.html

Bruno

Google Books Ngram Viewer

The Google Books Ngram Viewer is optimized for quick inquiries into the usage of small sets of phrases. If you're interested in performing a large scale analysis on the underlying data, you might prefer to download a portion of the corpora yourself. Or all of it, if you have the bandwidth and space. We're happy to oblige.

These datasets were generated in July 2012 (Version 2) and July 2009 (Version 1); we will update these datasets as our book scanning continues, and the updated versions will have distinct and persistent version identifiers (20120701 and 20090715 for the current sets).

**File format:** Each of the files below is compressed *tab*-separated data. In Version 2 each line has the following format:

```
ngram TAB year TAB match_count TAB volume_count NEWLINE
```

As an example, here are the 3,000,000th and 3,000,001st lines from the `a` file of the English 1-grams (googlebooks-eng-all-1gram-20120701-a.gz):

```
circumvallate   1978   335    91
circumvallate   1979   261    91
```

We've included separate files for ngrams that start with punctuation or with other non-alphanumeric characters. Finally, we have separate files for ngrams in which the first word is a part of speech tag (e.g., `_ADJ_`, `_ADP_`).

In Version 1, the format is similar, but we also include the number of pages each ngram occurred on:

```
ngram TAB year TAB match_count TAB page_count TAB volume_count NEWLINE
```

Here's the 9,000,000th line from file 0 of the English 5-grams (googlebooks-eng-all-5gram-20090715-0.csv.zip):

```
analysis is often described as  1991  1   1   1
```

In 1991, the phrase "analysis is often described as" occurred one time (that's the first 1), and on one page (the second 1), and in one book (the third 1). We do not provide page counts in Version 2 since we extract ngrams that span page boundaries.

The ngrams inside each file in Version 1 are sorted alphabetically and then chronologically. Note that the files themselves aren't ordered with respect to one another. A French two word phrase starting with 'm' will be in the middle of one of the French 2-gram files, but there's no way to know which without checking them all.

The format of the `total_counts` files are similar, except that the `ngram` field is absent and there is one triplet of values (match_count, page_count, volume_count) per year.

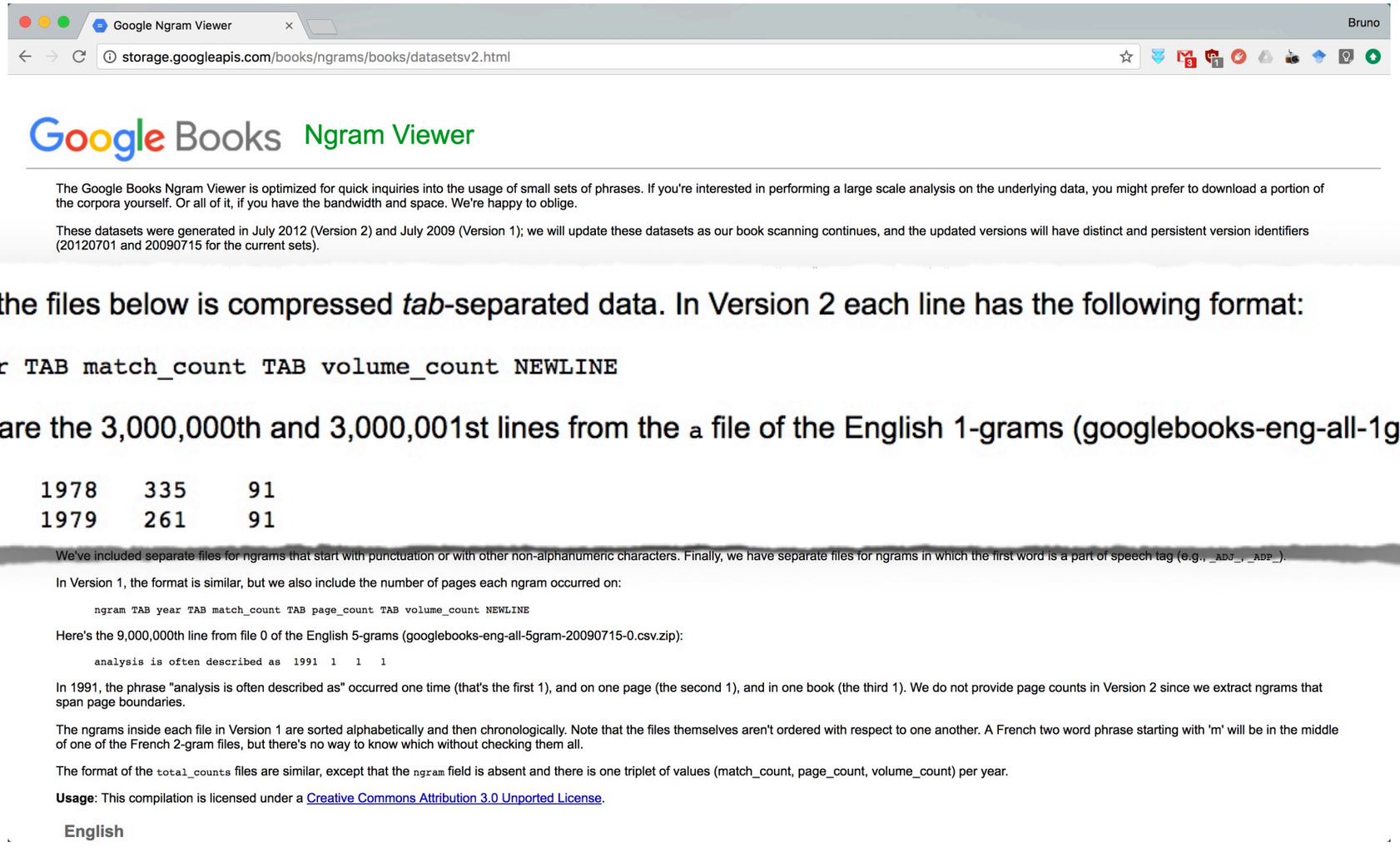
**Usage**: This compilation is licensed under a Creative Commons Attribution 3.0 Unported License.

English

# Character Probabilities

http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

As an example, let's calculate the probability of each letter occurring in the English language using Google Books 1-gram dataset.



Google Ngram Viewer

storage.googleapis.com/books/ngrams/books/datasetsv2.html

Bruno

Google Books Ngram Viewer

The Google Books Ngram Viewer is optimized for quick inquiries into the usage of small sets of phrases. If you're interested in performing a large scale analysis on the underlying data, you might prefer to download a portion of the corpora yourself. Or all of it, if you have the bandwidth and space. We're happy to oblige.

These datasets were generated in July 2012 (Version 2) and July 2009 (Version 1); we will update these datasets as our book scanning continues, and the updated versions will have distinct and persistent version identifiers (20120701 and 20090715 for the current sets).

**File format:** Each of the files below is compressed *tab*-separated data. In Version 2 each line has the following format:

```
ngram TAB year TAB match_count TAB volume_count NEWLINE
```

As an example, here are the 3,000,000th and 3,000,001st lines from the `a` file of the English 1-grams (googlebooks-eng-all-1gram-20120701-a.gz):

```
circumvallate   1978   335    91
circumvallate   1979   261    91
```

We've included separate files for ngrams that start with punctuation or with other non-alphanumeric characters. Finally, we have separate files for ngrams in which the first word is a part of speech tag (e.g., `_ADJ_`, `_ADP_`).

In Version 1, the format is similar, but we also include the number of pages each ngram occurred on:

```
ngram TAB year TAB match_count TAB page_count TAB volume_count NEWLINE
```

Here's the 9,000,000th line from file 0 of the English 5-grams (googlebooks-eng-all-5gram-20090715-0.csv.zip):

```
analysis is often described as  1991  1   1   1
```

In 1991, the phrase "analysis is often described as" occurred one time (that's the first 1), and on one page (the second 1), and in one book (the third 1). We do not provide page counts in Version 2 since we extract ngrams that span page boundaries.

The ngrams inside each file in Version 1 are sorted alphabetically and then chronologically. Note that the files themselves aren't ordered with respect to one another. A French two word phrase starting with 'm' will be in the middle of one of the French 2-gram files, but there's no way to know which without checking them all.

The format of the `total_counts` files are similar, except that the `ngram` field is absent and there is one triplet of values (match_count, page_count, volume_count) per year.

**Usage**: This compilation is licensed under a Creative Commons Attribution 3.0 Unported License.

English

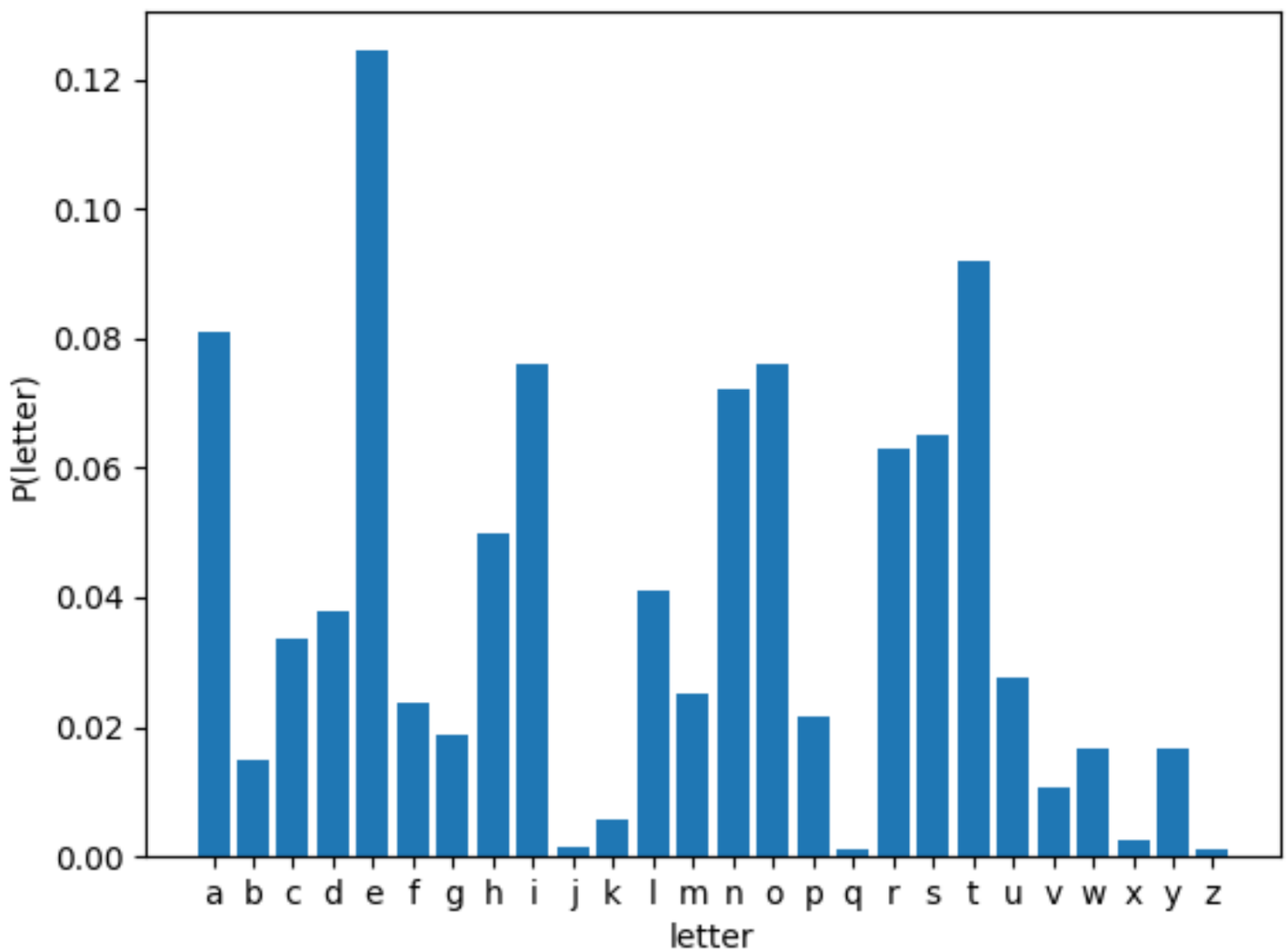
# Character Probabilities

```
pos = dict(zip(characters, range(len(characters))))
counts = np.zeros(len(characters), dtype='uint64')

line_count = 0

for filename in sys.argv[1:]:
    for line in gzip.open(filename, "rt"):
        fields = line.lower().strip().split()

        count = int(fields[2])
        word = fields[0]

        if "_" in word:
            continue

        letters = letter_regex.findall(word)

        if len(letters) != len(word):
            continue

        for letter in letters:
            if letter not in pos:
                continue

            counts[pos[letter]] += count

total = np.sum(counts)
pos = list(pos.items())
pos.sort(key=lambda x: x[1])

for key, value in enumerate(pos):
    print(value[0], counts[key]/total)
```

build_lang_table.py

# Character Probabilities

```
pos = dict(zip(characters, range(len(characters))))
counts = np.zeros(len(characters), dtype='uint64')
```

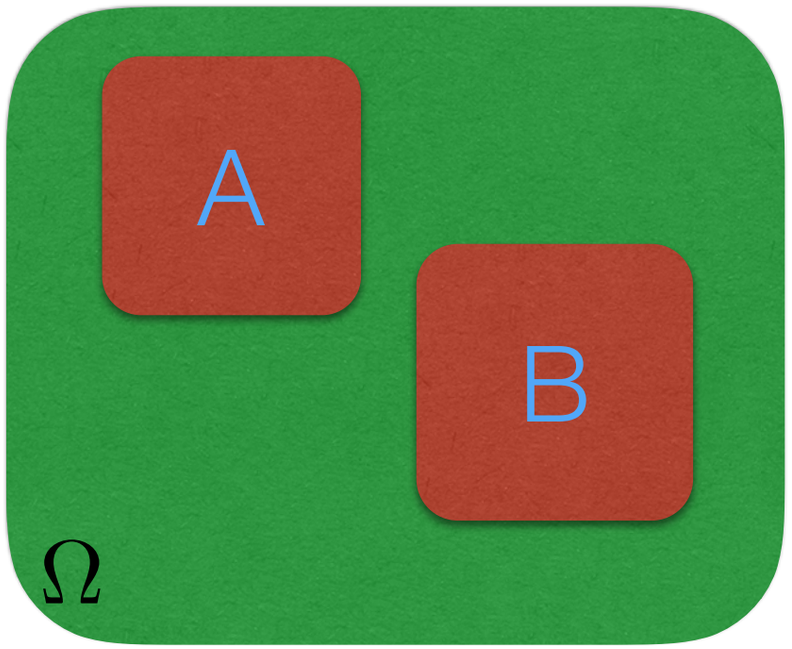


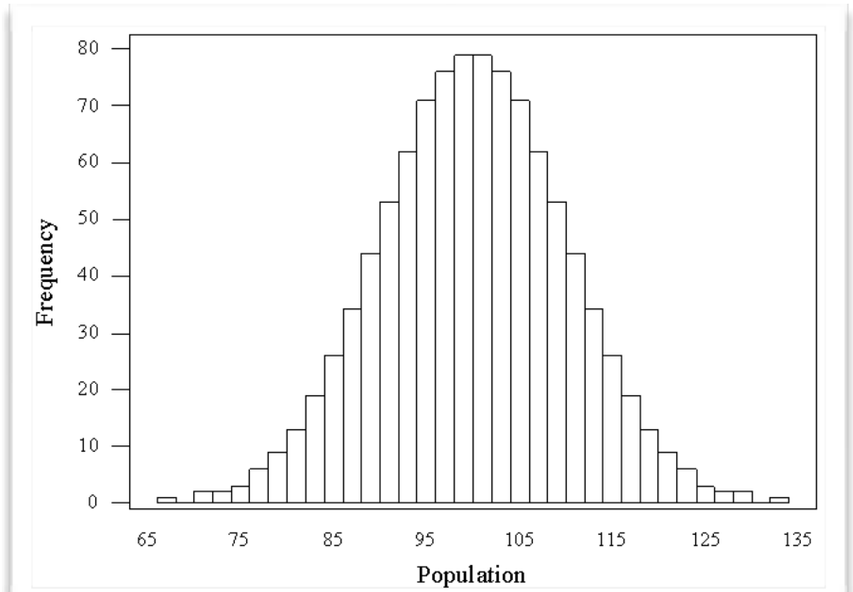

```
for key, value in enumerate(pos):
    print(value[0], counts[key]/total)
```

build_lang_table.py

# Probability

$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$



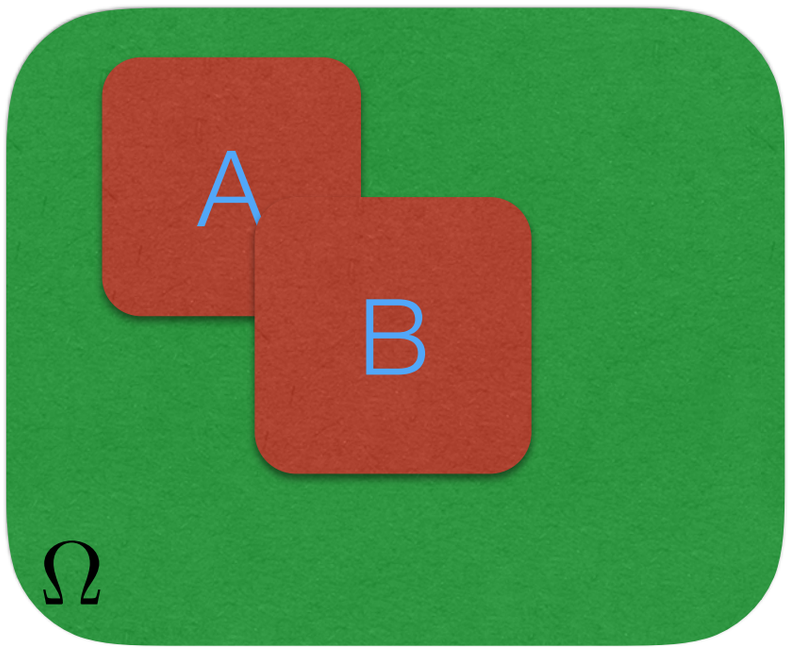




$$P(A \text{ or } B) = P(A) + P(B)$$

# Probability

$$P(A) = \text{"Area" of A}$$

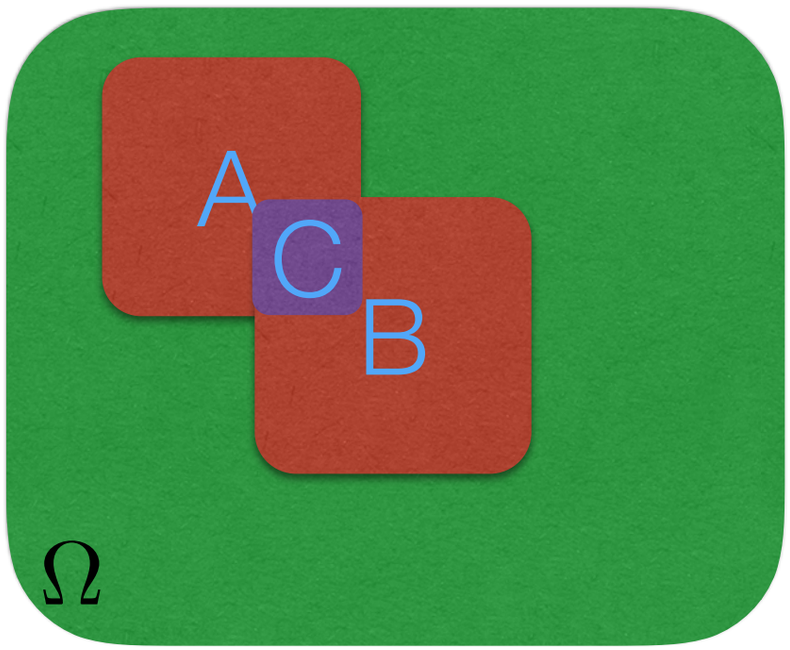
$$P(\Omega) = 1 \text{ (Normalization)}$$




$$P(A \ or \ B) = P(A) + P(B)$$

# Probability

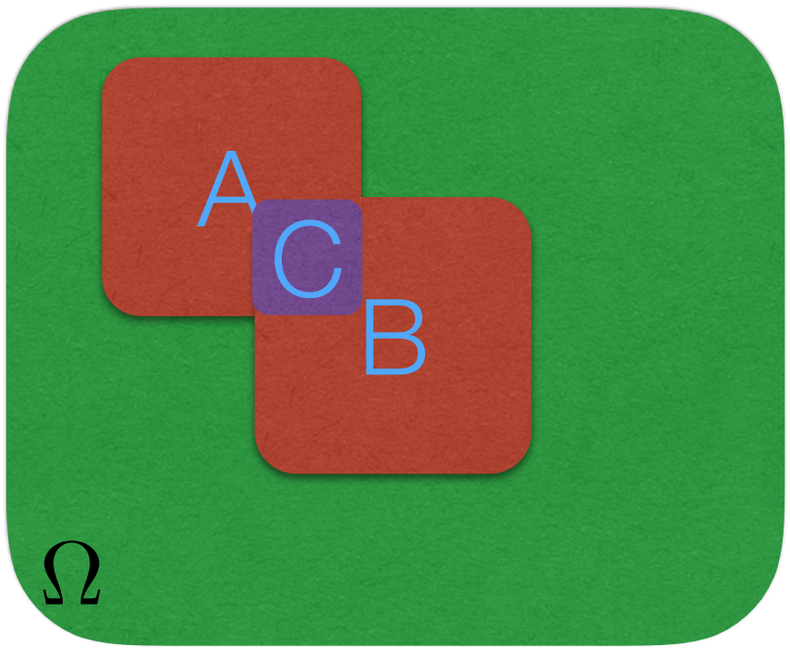
$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$




$$P(A \ or \ B) = P(A) + P(B) - P(A \ and \ B)$$

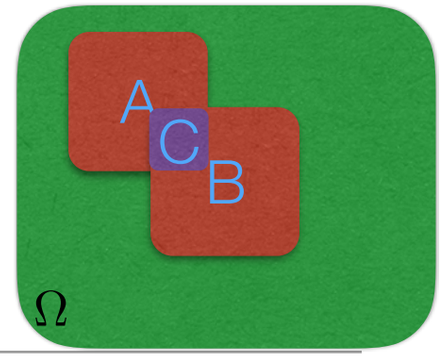
# Probability

$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$



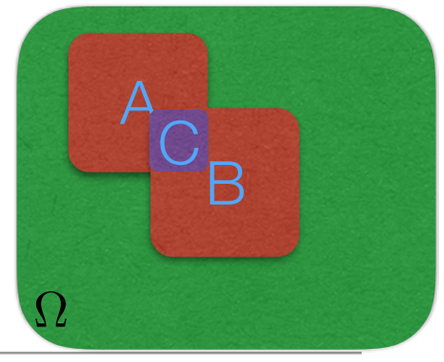

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

# Probability




$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

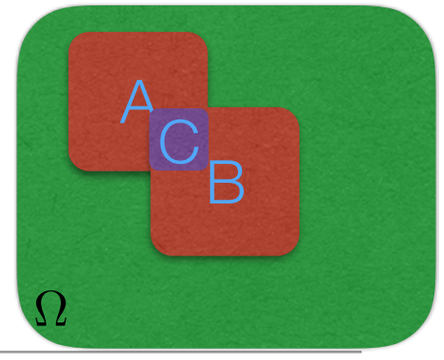
# Probability




$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

What's the probability that
I'm in B given that I'm in A?

# Probability

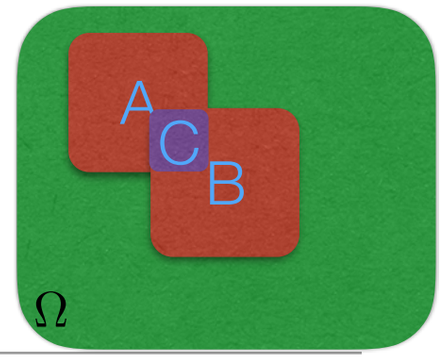



$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

What's the probability that
I'm in B given that I'm in A?

$$P(B|A)$$

# Probability



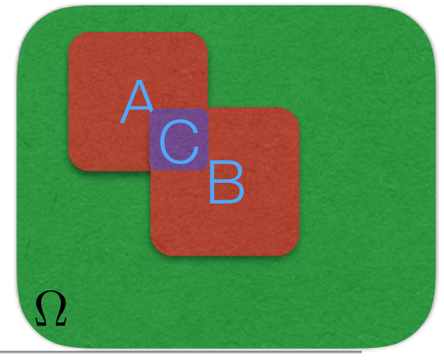

$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

What's the probability that I'm in B given that I'm in A?

What fraction of A is occupied by B?

$$P(B|A)$$

# Probability



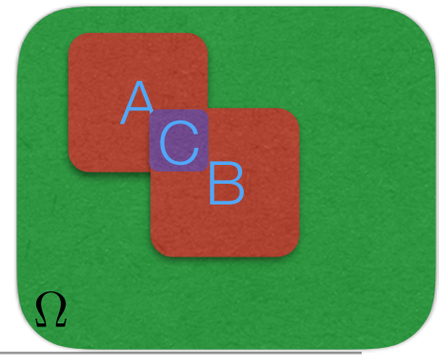

$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

What's the probability that I'm in B given that I'm in A?

What fraction of A is occupied by B?

$$P(B|A) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(B|A)\,P(A)$$

# Probability




$$P(A) = \text{"Area" of A}$$
$$P(\Omega) = 1 \text{ (Normalization)}$$
$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$
$$P(C) = P(A \text{ and } B) = \text{overlap of A and B}$$

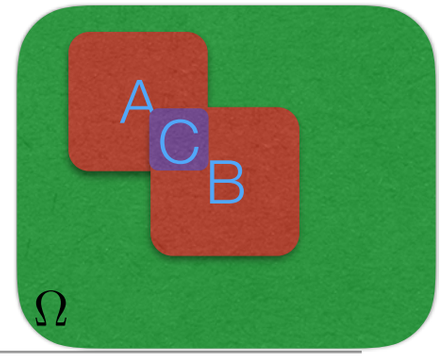
What's the probability that I'm in B given that I'm in A?

What fraction of A is occupied by B?

$$P(B|A) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(B|A)\,P(A)$$

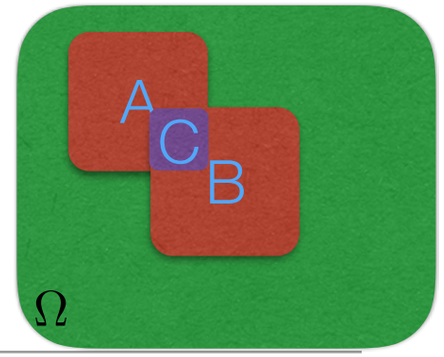
$$P(C) = P(C)$$

$$P(C) = P(C)$$




$$P(A|B) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(A|B)\,P(B)$$

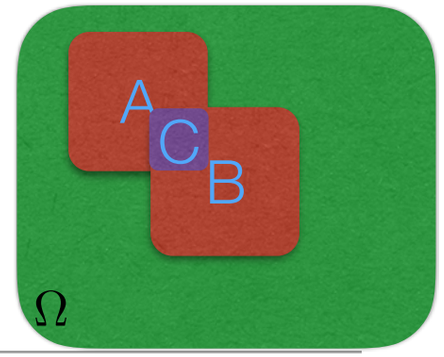
$$P(C) = P(C)$$




$$P(B|A)$$

$$P(A|B) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(A|B)\,P(B)$$

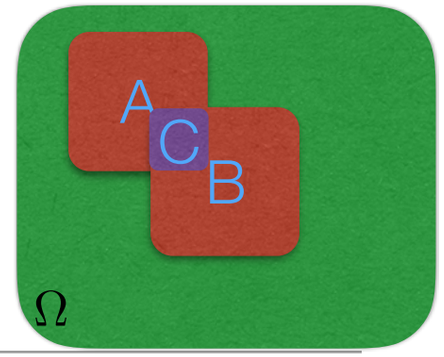
$$P(C) = P(C)$$




$$P(B|A) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(B|A)\,P(A)$$

$$P(A|B) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(A|B)\,P(B)$$

$$P(C) = P(C)$$

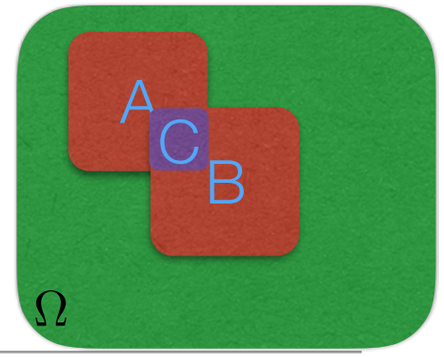



$$P(B|A) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(B|A)\,P(A)$$

$$P(A|B) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(A|B)\,P(B)$$

$$P(B|A)\,P(A) = P(A|B)\,P(B)$$

$$P(C) = P(C)$$




$$P(B|A) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(B|A)\,P(A)$$

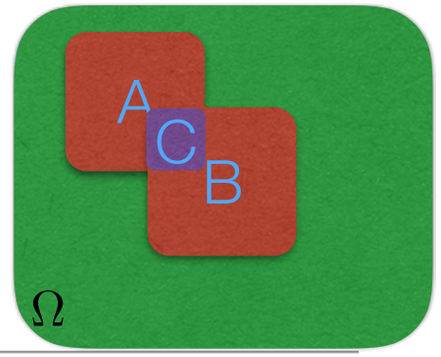
$$P(A|B) = \frac{P(C)}{P(A)} \rightarrow P(C) = P(A|B)\,P(B)$$

$$P(B|A)\,P(A) = P(A|B)\,P(B)$$

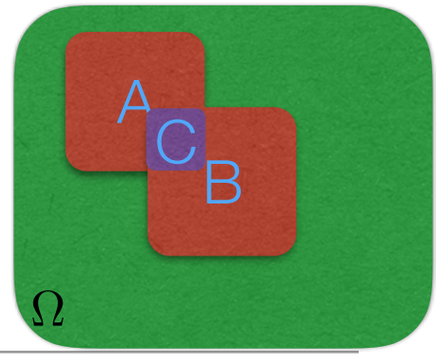
$$P(B|A) = \frac{P(A|B)\,P(B)}{P(A)}$$

Bayes Theorem

# Medical Tests

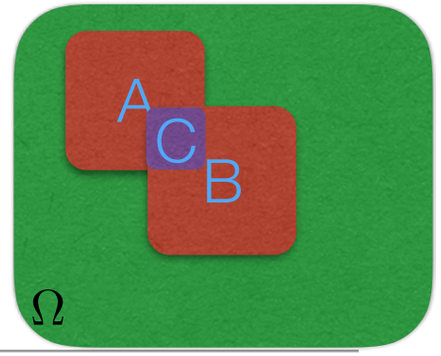


# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

# Medical Tests

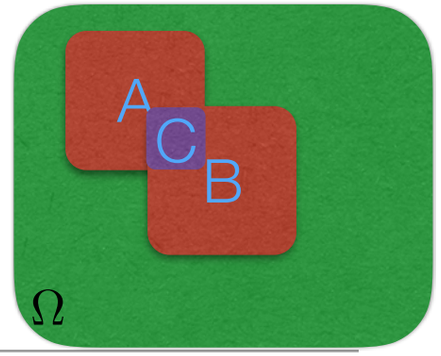



Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

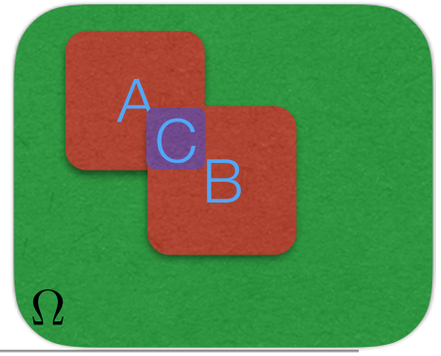
Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

Total Probability:

$$\begin{aligned} P\left(positive\ test\right) = & P\left(positive\ test|disease\right)P\left(disease\right) \\ & + P\left(positive\ test|no\ disease\right)P\left(no\ disease\right) \end{aligned}$$

# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

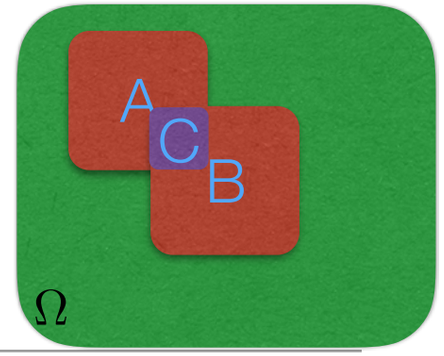
Total Probability:

$$\begin{aligned} P\left(positive\ test\right) = & P\left(positive\ test|disease\right)P\left(disease\right) \\ & + P\left(positive\ test|no\ disease\right)P\left(no\ disease\right) \end{aligned}$$

Finally:

$$P\left(disease|positive\ test\right) = 0.0098$$

# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

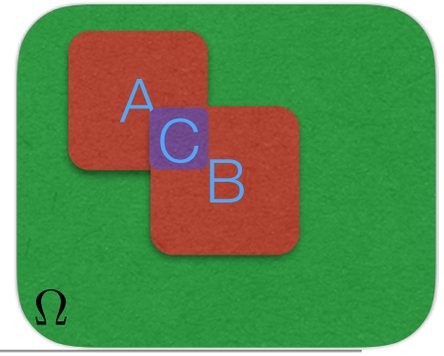
Total Probability:

$$P\left(positive\ test\right) = P\left(positive\ test|disease\right)P\left(disease\right) + P\left(positive\ test|no\ disease\right)P\left(no\ disease\right)$$

Finally:

$$P\left(disease|positive\ test\right) = 0.0098$$

# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

Total Probability:

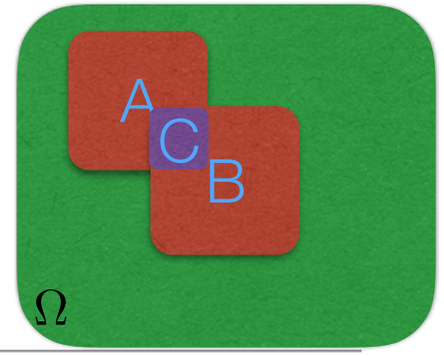
$$P\left(positive\ test\right) = P\left(positive\ test|disease\right)P\left(disease\right) + P\left(positive\ test|no\ disease\right)P\left(no\ disease\right)$$

Finally:

$$P\left(disease|positive\ test\right) = 0.0098$$

Base Rate Fallacy

# Medical Tests




Your doctor thinks you might have a rare disease that affects 1 person in 10,000. A test that is 99% accurate comes out positive. What's the probability of you having the disease?

Bayes Theorem:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

Total Probability:

$$P\left(positive\ test\right) = P\left(positive\ test|disease\right)P\left(disease\right) + P\left(positive\ test|no\ disease\right)P\left(no\ disease\right)$$

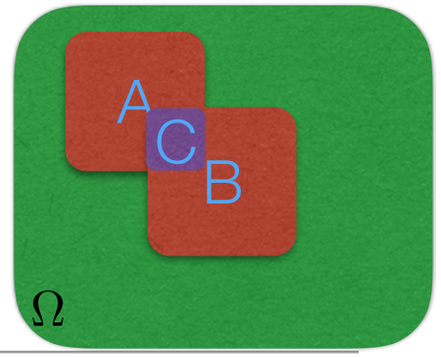
Finally:

$$P\left(disease|positive\ test\right) = 0.0098$$

Base Rate Fallacy

Low Base Rate Value
+
Non-zero False Positive Rate

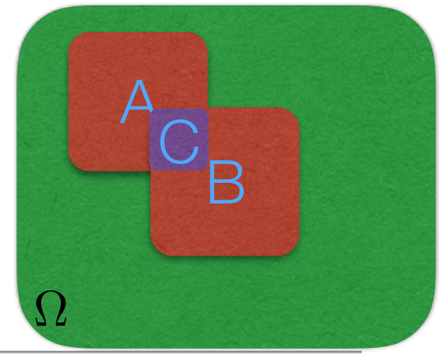
# Medical Tests




Consider a population of 1,000,000 individuals. The numbers we should expect are:

| | disease | no disease |
|---|---|---|
| **positive** | 99 | 9,999 |
| **negative** | 1 | 989,901 |

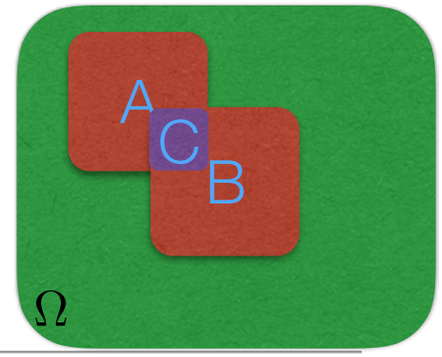
# Medical Tests




Consider a population of 1,000,000 individuals. The numbers we should expect are:

| | disease | no disease | |
|---|---|---|---|
| **positive** | 99 | 9,999 | 10,098 |
| **negative** | 1 | 989,901 | 989,902 |
| | 100 | 999,900 | 1,000,000 |

# Medical Tests

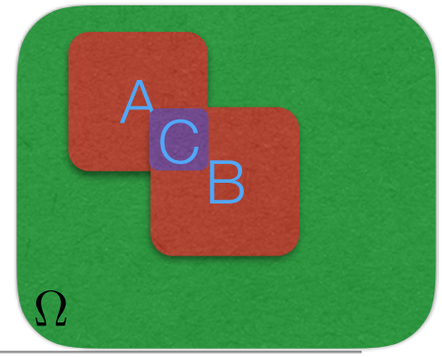



Consider a population of 1,000,000 individuals. The numbers we should expect are:

Marginals

| | disease | no disease | |
|---|---|---|---|
| **positive** | 99 | 9,999 | 10,098 |
| **negative** | 1 | 989,901 | 989,902 |
| Marginals | 100 | 999,900 | 1,000,000 |

# Medical Tests

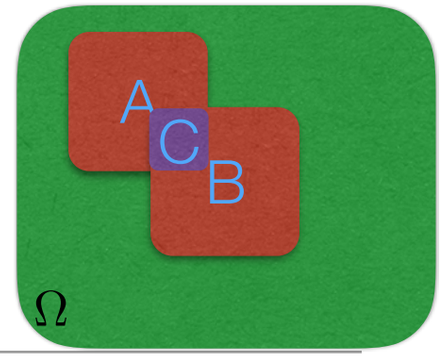



Consider a population of 1,000,000 individuals. The numbers we should expect are:

Marginals

| | disease | no disease | |
|---|---|---|---|
| positive | 99 | 9,999 | 10,098 |
| negative | 1 | 989,901 | 989,902 |
| Marginals → | 100 | 999,900 | 1,000,000 |

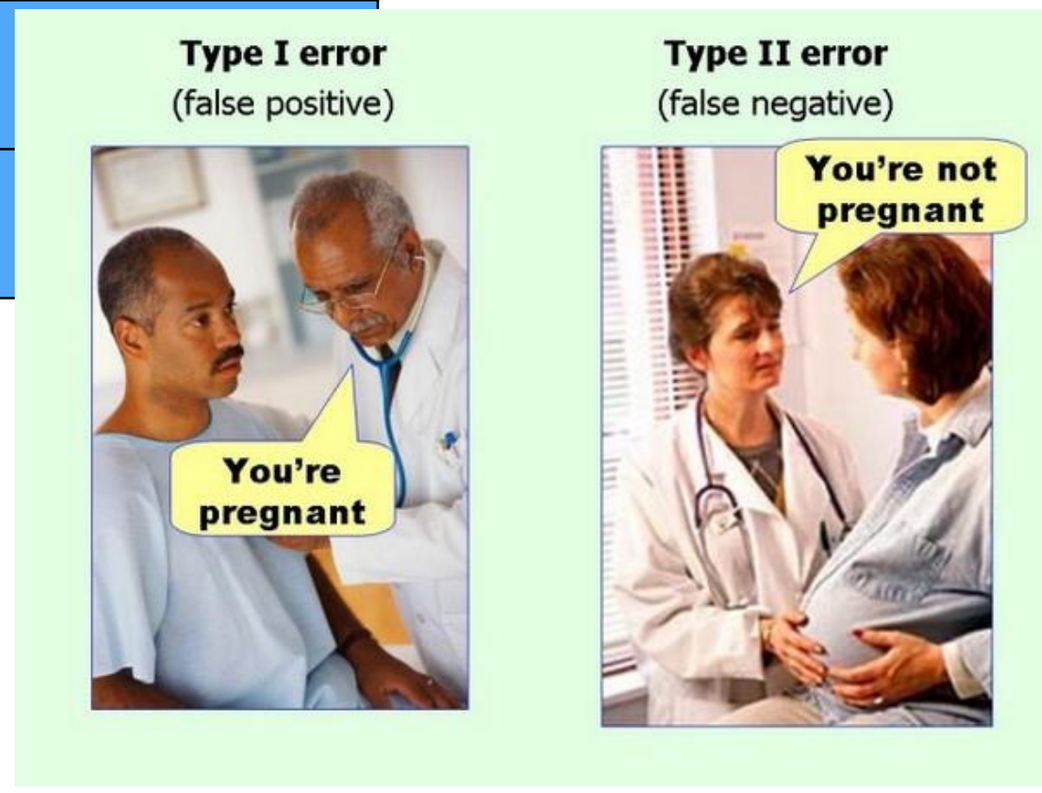
# Medical Tests




Consider a population of 1,000,000 individuals. The numbers we should expect are:

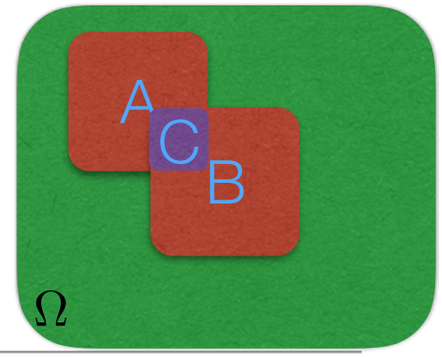
Marginals

| | disease | no disease | |
|---|---|---|---|
| positive | 99 | 9,999 | 10,098 |
| negative | 1 | 989,901 | |
| Marginals → | 100 | 999,900 | |




*@bgoncalves*

# Medical Tests




Consider a population of 1,000,000 individuals. The numbers we should expect are:

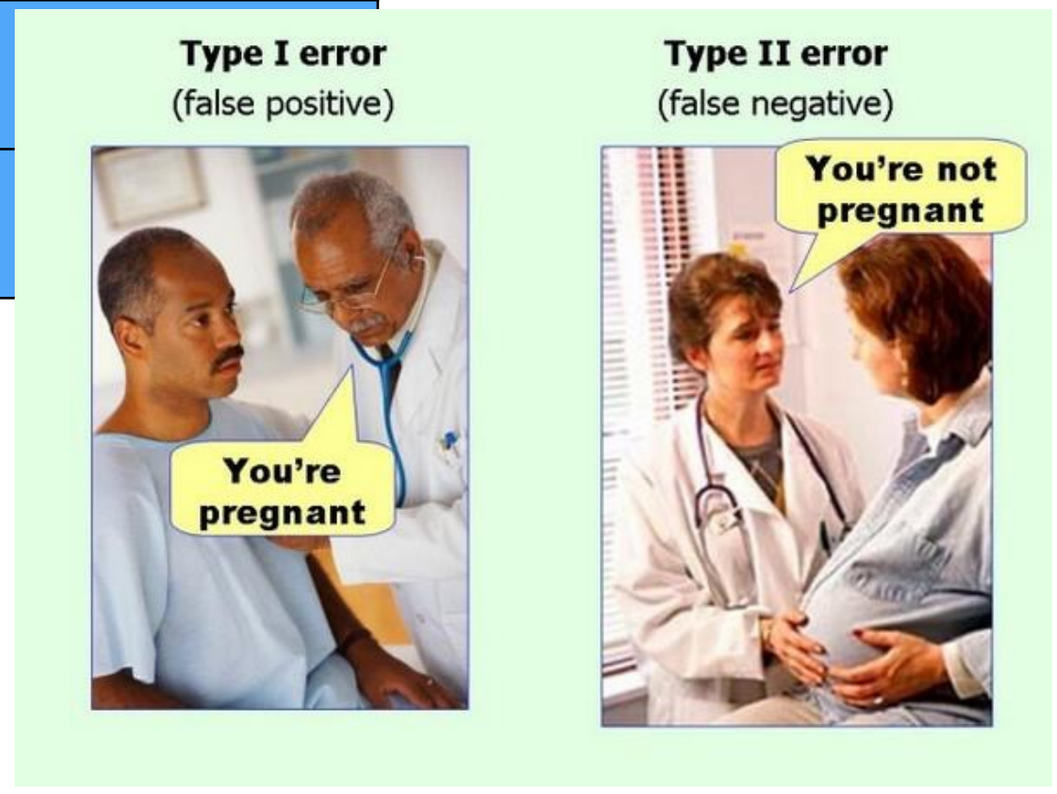
Marginals

| | disease | no disease | |
|---|---|---|---|
| positive | 99 | 9,999 | 10,098 |
| negative | 1 | 989,901 | |
| Marginals → | 100 | 999,900 | |

$$P\left(disease|positive\ test\right) = \frac{TP}{TP + FP} = 0.0098$$

$$P\left(no\ disease|negative\ test\right) = \frac{TN}{TN + FN} = 0.99999$$

*@bgoncalves*



# (Confusion Matrix)

| Feature / Test | positive | negative |
|---|---|---|
| **positive** | TP | FP |
| **negative** | FN | TN |

# (Confusion Matrix)

| Feature / Test | positive | negative |
|---|---|---|
| **positive** | TP | FP |
| **negative** | FN | TN |

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$specificity = \frac{TN}{FP + TN}$$

# (Confusion Matrix)

| Feature / Test | positive | negative |
|---|---|---|
| **positive** | TP | FP |
| **negative** | FN | TN |

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$specificity = \frac{TN}{FP + TN}$$

$$precision = \frac{TP}{TP + FP}$$

$$sensitivity = \frac{TP}{TP + FN}$$

# (Confusion Matrix)

| Feature / Test | positive | negative |
|---|---|---|
| **positive** | TP | FP |
| **negative** | FN | TN |

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$specificity = \frac{TN}{FP + TN}$$

$$precision = \frac{TP}{TP + FP}$$

$$sensitivity = \frac{TP}{TP + FN}$$

harmonic mean

$$F1 = \frac{2TP}{2TP + FP + FN}$$

# A second Test

# A second Test

Bayes Theorem still looks the same:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

# A second Test

Bayes Theorem still looks the same:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

but now the probability that we have the disease has been updated:

$$P^{\dagger}\left(disease\right) = 0.0098$$

# A second Test

Bayes Theorem still looks the same:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

but now the probability that we have the disease has been updated:

$$P^{\dagger}\left(disease\right) = 0.0098$$

So this time we find:

$$P^{\dagger}\left(disease|positive\ test\right) = 0.4949$$

# A second Test

Bayes Theorem still looks the same:

$$P\left(disease|positive\ test\right) = \frac{P\left(positive\ test|disease\right)P\left(disease\right)}{P\left(positive\ test\right)}$$

but now the probability that we have the disease has been updated:

$$P^{\dagger}\left(disease\right) = 0.0098$$

So this time we find:

$$P^{\dagger}\left(disease|positive\ test\right) = 0.4949$$

Each test is providing new evidence, and Bayes theorem is simply telling us how to use it to update our beliefs.

# Bayesian Coin Flips

*@bgoncalves*

# Bayesian Coin Flips

- Biased coin with unknown probability of heads (**p**)

# Bayesian Coin Flips

- Biased coin with unknown probability of heads (p)
- Perform N flips and update our belief after each flip using Bayes Theorem

$$P\left(p|heads\right) = \frac{P\left(heads|p\right)P\left(p\right)}{P\left(heads\right)}$$

$$P\left(p|tails\right) = \frac{P\left(tails|p\right)P\left(p\right)}{P\left(tails\right)}$$

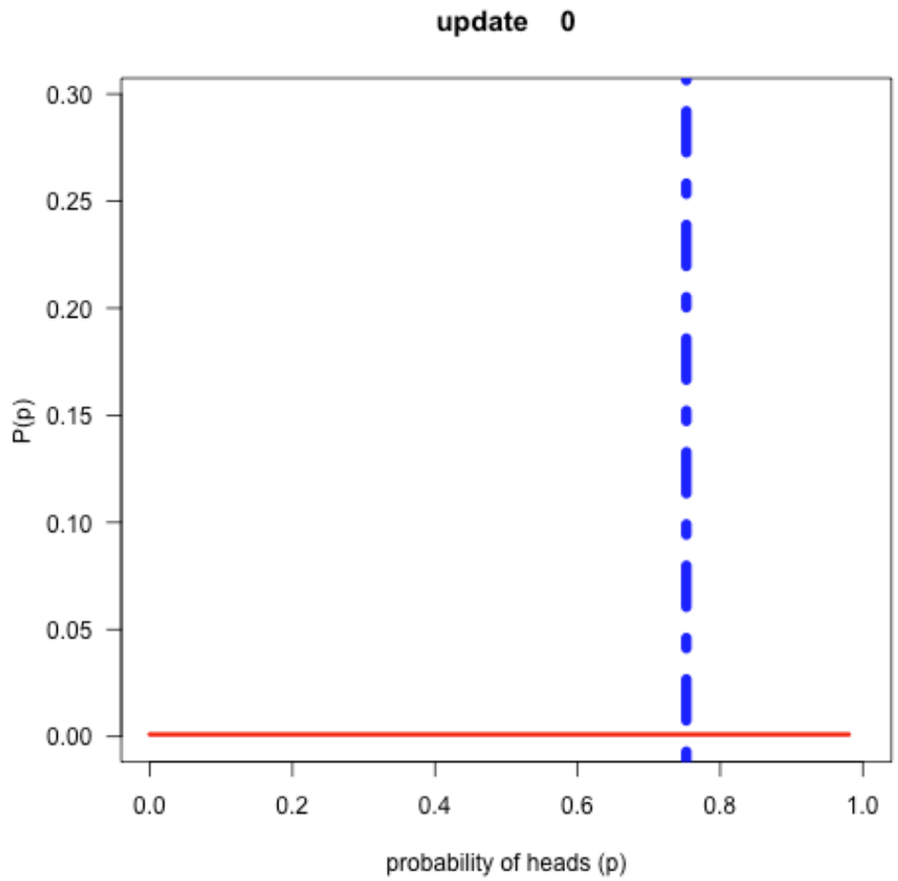
# Bayesian Coin Flips

http://youtu.be/GTx0D8VY0CY

- Biased coin with unknown probability of heads (p)
- Perform N flips and update our belief after each flip using Bayes Theorem

$$P\left(p|heads\right) = \frac{P\left(heads|p\right)P\left(p\right)}{P\left(heads\right)}$$

$$P\left(p|tails\right) = \frac{P\left(tails|p\right)P\left(p\right)}{P\left(tails\right)}$$



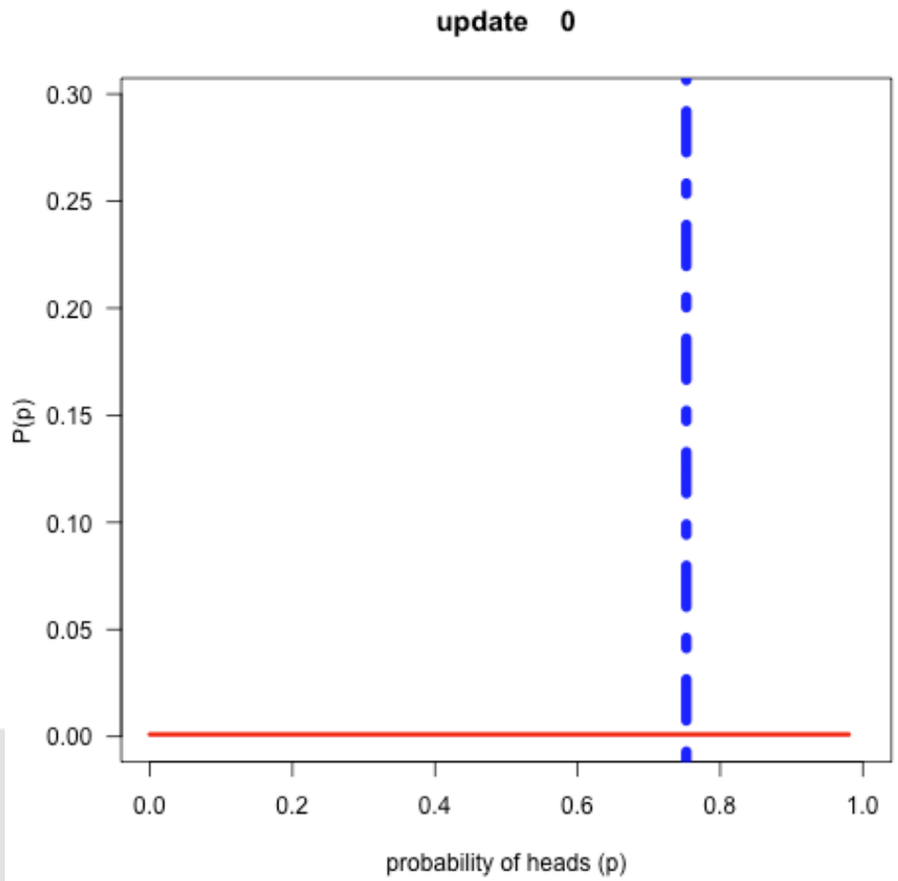
# Bayesian Coin Flips

http://youtu.be/GTx0D8VY0CY

- Biased coin with unknown probability of heads (p)
- Perform N flips and update our belief after each flip using Bayes Theorem

$$P\left(p|heads\right) = \frac{P\left(heads|p\right)P\left(p\right)}{P\left(heads\right)}$$

$$P\left(p|tails\right) = \frac{P\left(tails|p\right)P\left(p\right)}{P\left(tails\right)}$$




```
# Uninformative prior
prior = np.ones(bins, dtype='float')/bins
likelihood_heads = np.arange(bins)/float(bins)
likelihood_tails = 1-likelihood_heads
flips = np.random.choice(a=[True, False], size=flips, p=[0.75, 0.25])

for coin in flips:
    if coin:  # Heads
        posterior = prior * likelihood_heads
    else:  # Tails
        posterior = prior * likelihood_tails

    # Normalize
    posterior /= np.sum(posterior)

    # The posterior is now the new prior
    prior = posterior
```

coins.py

# 3 Types of Machine Learning

# 3 Types of Machine Learning

- Supervised Learning
    - Predict output given input
    - Training set of known inputs and outputs is provided

# 3 Types of Machine Learning

- Supervised Learning
    - Predict output given input
    - Training set of known inputs and outputs is provided
- Unsupervised Learning
    - Autonomously learn an good representation of the dataset
    - Find clusters in input

# 3 Types of Machine Learning

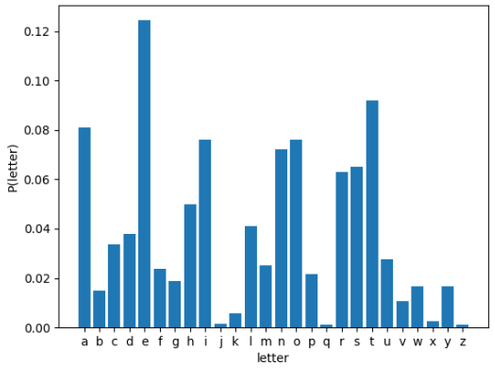
- Supervised Learning
    - Predict output given input
    - Training set of known inputs and outputs is provided
- Unsupervised Learning
    - Autonomously learn an good representation of the dataset
    - Find clusters in input
- Reinforcement Learning
    - Learn sequence of actions to maximize payoff
    - Discount factor for delayed rewards

# 3 Types of Machine Learning

- Supervised Learning
  - Predict output given input
  - Training set of known inputs and outputs is provided
- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input
- ~~Reinforcement Learning~~
  - ~~Learn sequence of actions to maximize payoff~~
  - ~~Discount factor for delayed rewards~~

# Language Detection

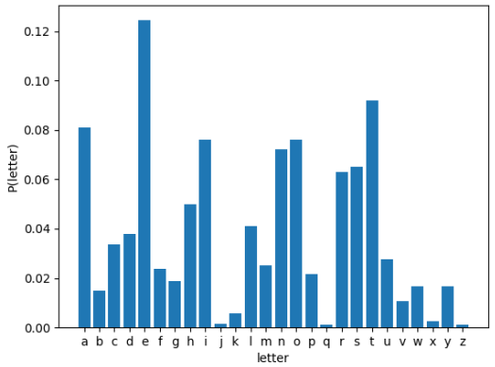
# Language Detection




- Previously, we measured the probability distribution of letters in the english language. In effect, we calculated:

$$P\left(letter|english\right)$$

# Language Detection



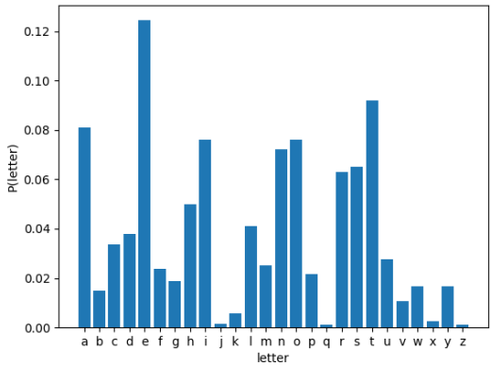

- Previously, we measured the probability distribution of letters in the english language. In effect, we calculated:

$$P\left(letter|english\right)$$

- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

$$P\left(letter|language\right)$$

# Language Detection




- Previously, we measured the probability distribution of letters in the english language. In effect, we calculated:

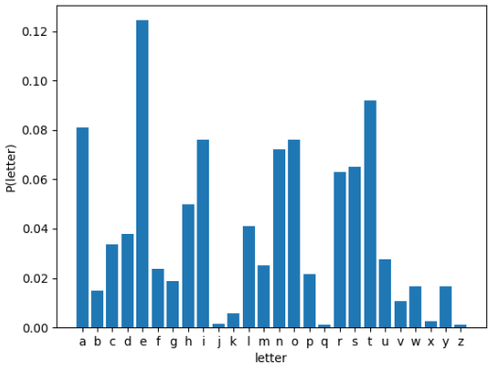
$$P\left(letter|english\right)$$

- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

$$P\left(letter|language\right)$$

- In the repository you'll find table_langs.dat where this conditional probability is calculated for 5 different languages: English, French, German, Italian and Spanish.

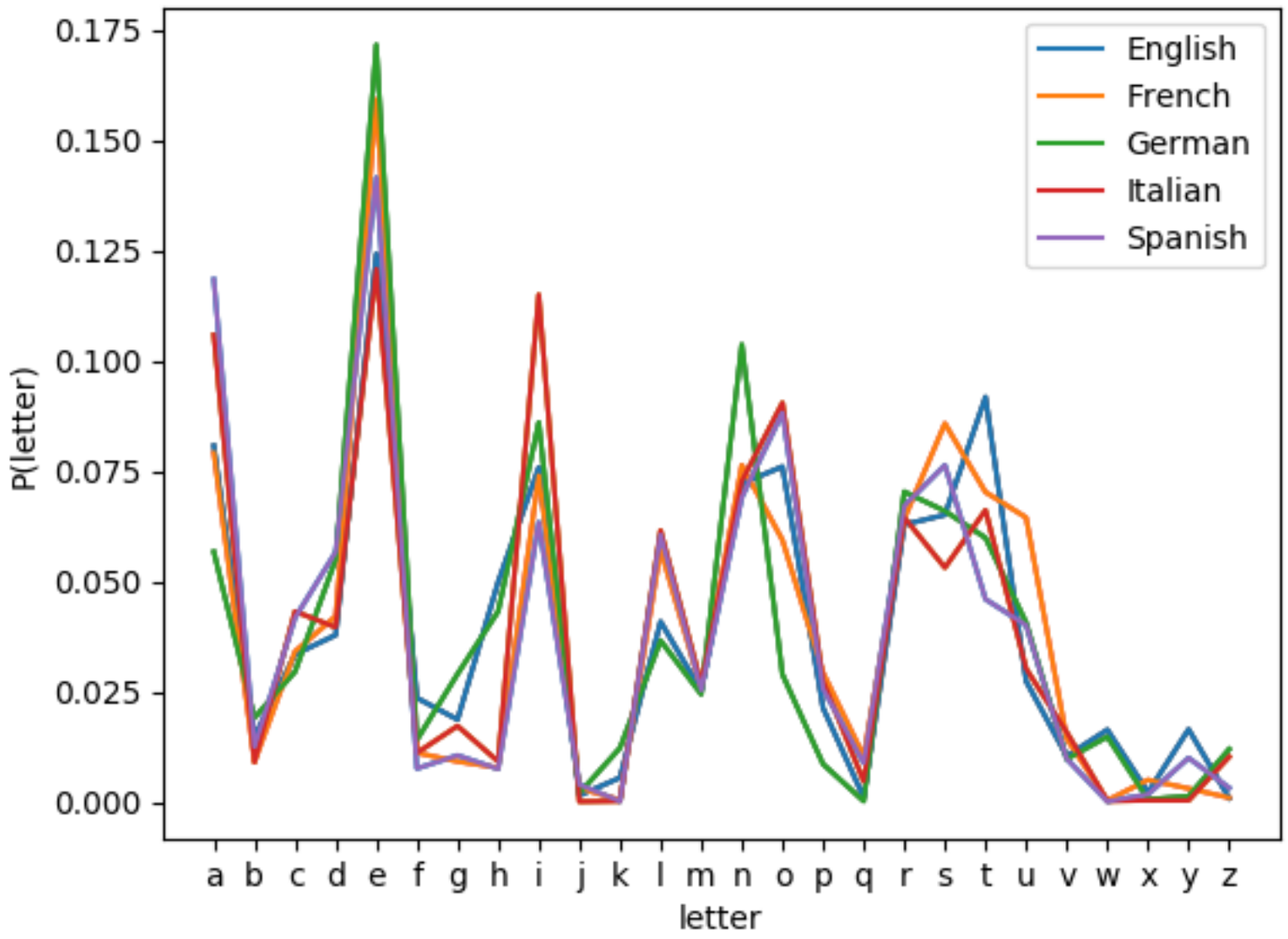
# Language Detection




- Previously, we measured the probability distribution of letters in the english language. In effect, we c

- The probab

- In the repos




ve do this for

a few other

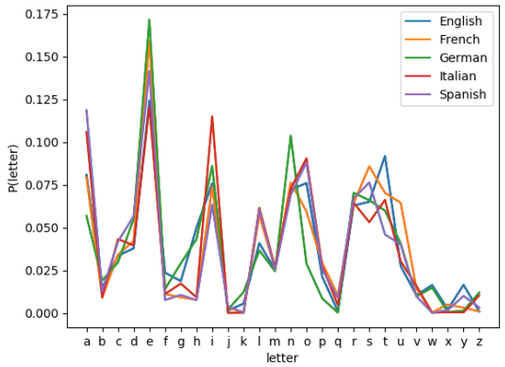
calculated fc

ty is

d Spanish.

plot_langs.py

# Language Detection




- A few minutes ago we measured the probability distribution of letters in the english language. In effect, we calculated

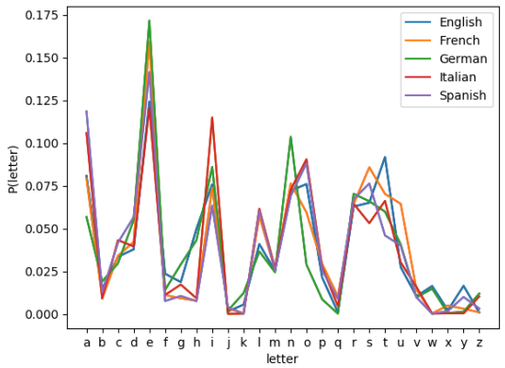
$$P\left(letter|english\right)$$

- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

$$P\left(letter|language\right)$$

- In the repository you'll find table_langs.dat where this conditional probability is calculated for 5 different languages: English, French, German, Italian and Spanish.

# Language Detection




- A few minutes ago we measured the probability distribution of letters in the english language. In effect, we calculated

$$P\left(letter|english\right)$$

- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

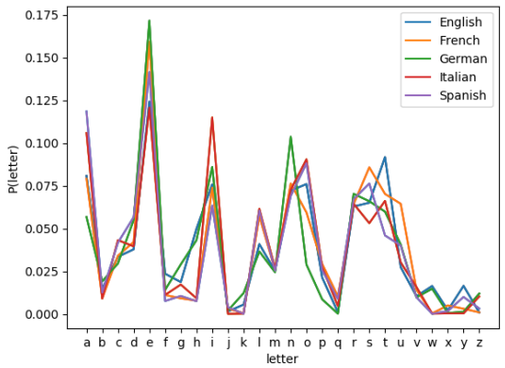
$$P\left(letter|language\right)$$

- In the repository you'll find table_langs.dat where this conditional probability is calculated for 5 different languages: English, French, German, Italian and Spanish.

- Using these conditional probabilities, and Bayes Theorem, we can easily build a language detector. For that we just need to calculate:

$$P\left(language|text\right)$$

# Language Detection




- A few minutes ago we measured the probability distribution of letters in the english language. In effect, we calculated

$$P\left(letter|english\right)$$

- The probability of seeing a specific letter given that the text is in English. If we do this for a few other languages we can have a table of the form:

$$P\left(letter|language\right)$$

- In the repository you'll find table_langs.dat where this conditional probability is calculated for 5 different languages: English, French, German, Italian and Spanish.

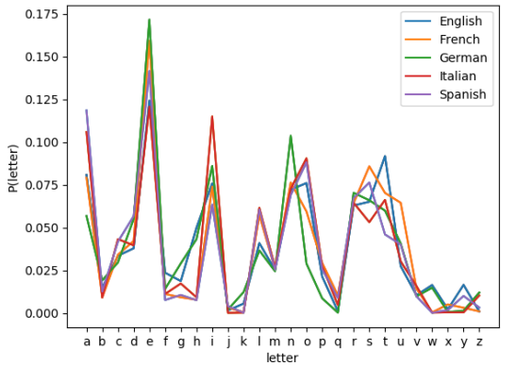
- Using these conditional probabilities, and Bayes Theorem, we can easily build a language detector. For that we just need to calculate:

$$P\left(language|text\right)$$

- We can rewrite it as a:

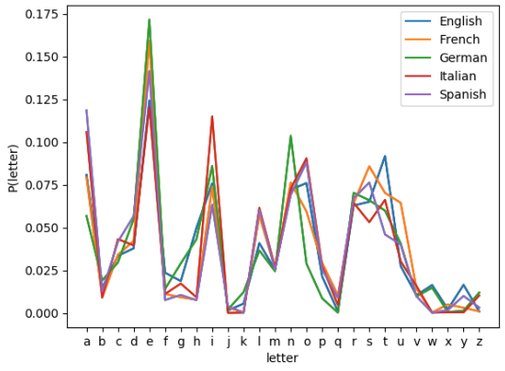
$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

# Naive Bayes Classifier




$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

# Naive Bayes Classifier

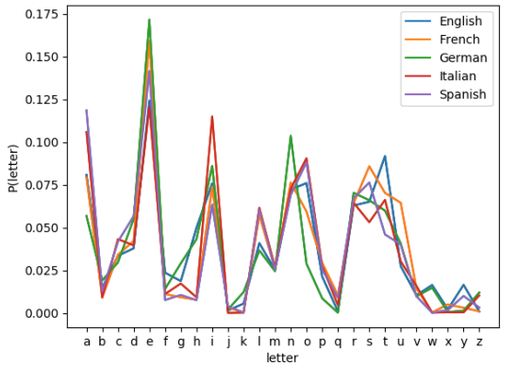



$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

- If we treat each letter independently, we obtain:

$$P\left(language|letter_1, letter_2, \cdots, letter_n\right) = \prod_i P\left(language|letter_i\right)$$

# Naive Bayes Classifier



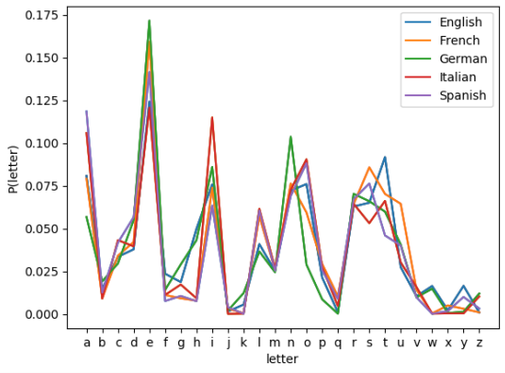

$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

- If we treat each letter independently, we obtain:

$$P\left(language|letter_1, letter_2, \cdots, letter_n\right) = \prod_i P\left(language|letter_i\right)$$

- This is known as the Naive Bayes Approach and is an obvious oversimplification as it completely ignores correlations present in the sequence of letters.

# Naive Bayes Classifier




$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

- If we treat each letter independently, we obtain:

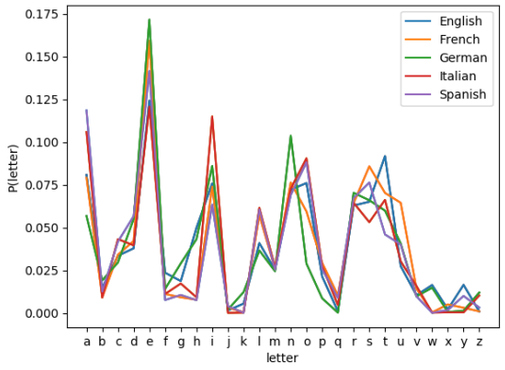
$$P\left(language|letter_1, letter_2, \cdots, letter_n\right) = \prod_i P\left(language|letter_i\right)$$

- This is known as the Naive Bayes Approach and is an obvious oversimplification as it completely ignores correlations present in the sequence of letters.

- All we have to do is apply Bayes Theorem to our original table:

$$P\left(language|letter\right) = \frac{P\left(letter|language\right)P\left(language\right)}{P\left(letter\right)}$$

# Naive Bayes Classifier




$$P\left(language|letter_1, letter_2, \cdots, letter_n\right)$$

- If we treat each letter independently, we obtain:

$$P\left(language|letter_1, letter_2, \cdots, letter_n\right) = \prod_i P\left(language|letter_i\right)$$

- This is known as the Naive Bayes Approach and is an obvious oversimplification as it completely ignores correlations present in the sequence of letters.

- All we have to do is apply Bayes Theorem to our original table:

$$P\left(language|letter\right) = \frac{P\left(letter|language\right)P\left(language\right)}{P\left(letter\right)}$$

- And if we assume that all languages are equally probable (non-informative prior)

$$P\left(letter\right) = \frac{1}{N_{langs}}$$

# Naive Bayes Classifier - Numerical Considerations

# Naive Bayes Classifier - Numerical Considerations

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P\left(A|B_i\right)$$

# Naive Bayes Classifier - Numerical Considerations

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P\left(A|B_i\right)$$

- which implies multiplying many small numbers. To avoid numerical complications, it is best to use, instead:

$$\sum_i \log P\left(A|B_i\right)$$

# Naive Bayes Classifier - Numerical Considerations

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P\left(A|B_i\right)$$

- which implies multiplying many small numbers. To avoid numerical complications, it is best to use, instead:

$$\sum_i \log P\left(A|B_i\right)$$

- Which is commonly referred to as the "Log-Likelihood". Our expression then becomes:

$$\mathcal{L}\left(language|letter_1, letter_2, \cdots, letter_n\right) = \sum_i \log\left[\frac{P\left(letter_i|language\right)P\left(language\right)}{P\left(letter_i\right)}\right]$$

# Naive Bayes Classifier - Numerical Considerations

- Naive Bayes approaches (and many others) use terms of the form:

$$\prod_i P\left(A|B_i\right)$$

- which implies multiplying many small numbers. To avoid numerical complications, it is best to use, instead:

$$\sum_i \log P\left(A|B_i\right)$$

- Which is commonly referred to as the "Log-Likelihood". Our expression then becomes:

$$\mathcal{L}\left(language|letter_1, letter_2, \cdots, letter_n\right) = \sum_i \log\left[\frac{P\left(letter_i|language\right)P\left(language\right)}{P\left(letter_i\right)}\right]$$

- Or more simply:

$$\mathcal{L}\left(language|text\right) = \sum_i \log\left[\frac{P\left(letter_i|language\right)P\left(language\right)}{P\left(letter_i\right)}\right]$$

$$\mathcal{L}\left(language|text\right) = \sum_i \mathcal{L}\left(language|letter_i\right)$$

# Naive Bayes Classifier - Normalization

# Naive Bayes Classifier - Normalization

- The final question to answer is, how can we compare the different results?

# Naive Bayes Classifier - Normalization

- The final question to answer is, how can we compare the different results?

- Intuitively, we expect that the language with the highest likelihood is, well, the most likely one.

# Naive Bayes Classifier - Normalization

- The final question to answer is, how can we compare the different results?

- Intuitively, we expect that the language with the highest likelihood is, well, the most likely one.

- Can we quantify how certain we are that we have the correct answer?

# Naive Bayes Classifier - Normalization

- The final question to answer is, how can we compare the different results?

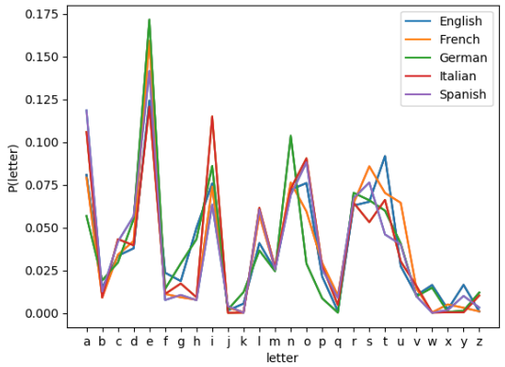
- Intuitively, we expect that the language with the highest likelihood is, well, the most likely one.

- Can we quantify how certain we are that we have the correct answer?

- We just need to remember that likelihoods actually represent probabilities and renormalize them!

$$P\left(language|text\right) = \frac{\exp\left[\mathcal{L}\left(language|text\right)\right]}{\sum_{language}\exp\left[\mathcal{L}\left(language|text\right)\right]}$$

# Language Detection




```python
import pandas as pd
import numpy as np
from collections import Counter

def load_data():
    P_letter_lang = pd.read_csv('table_langs.dat', sep=' ', header=0, index_col = 0)

    langs = list(P_letter_lang.columns)

    P_letter = P_letter_lang.mean(axis=1)
    P_letter /= P_letter.sum()

    P_lang_letter = np.array(P_letter_lang)/(P_letter_lang.shape[1]*P_letter.T[:,None])

    L_lang_letter = np.log(P_lang_letter.T)

    return langs, P_letter, L_lang_letter

def detect_lang(langs, P_letter, L_lang_letter, text):
    counts = np.zeros(26, dtype='int')
    pos = dict(zip(P_letter.index, range(26)))

    text_counts =  Counter(text).items()

    for letter, count in text_counts:
        if letter in pos:
            counts[pos[letter]] += count

    L_text = np.dot(L_lang_letter, counts)
    index = np.argmax(L_text)
    lang_text = langs[index]
    prob = np.exp(L_text[index])/np.sum(np.exp(L_text))*100

    return lang_text, prob, L_text
```

detect.py

# Data Normalization

# Data Normalization

- The range of raw data values can vary widely.

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some suitable way

# Data Normalization

- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some suitable way

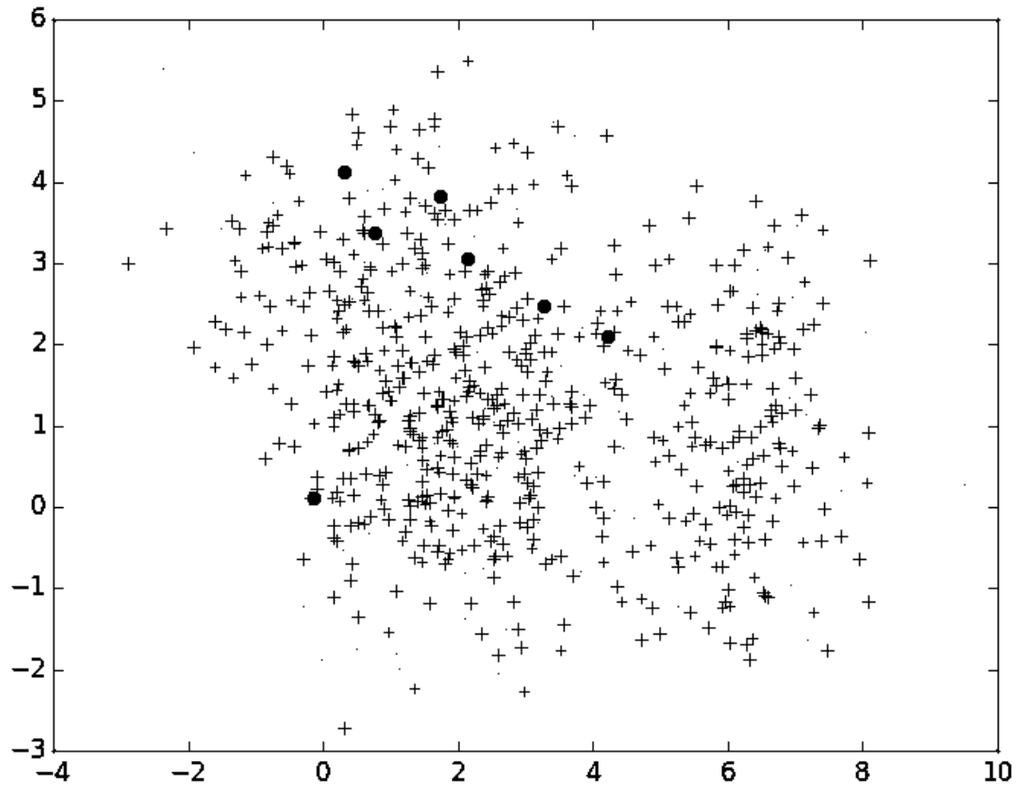
# Unsupervised Learning

- Extracting patters from data
  - K-Means
  - Agglomerative Clustering
  - PCA
  - Expectation Maximization
  - Gaussian Mixture Models

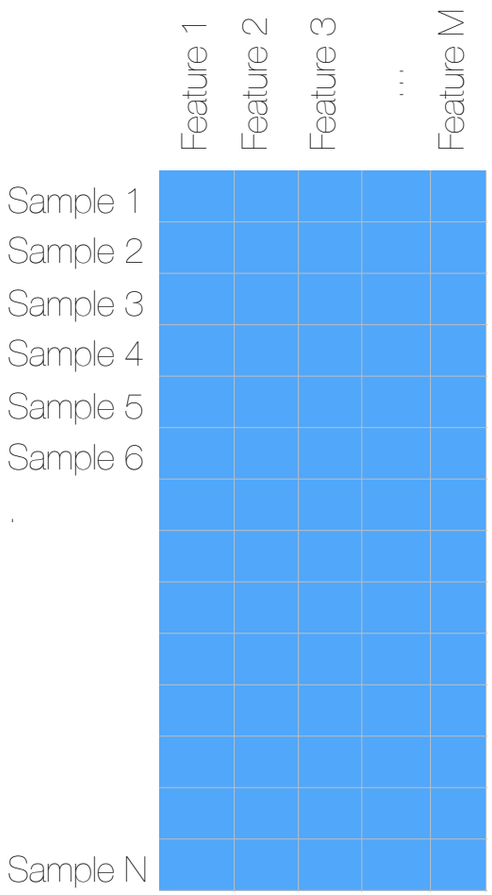
# Unsupervised Learning

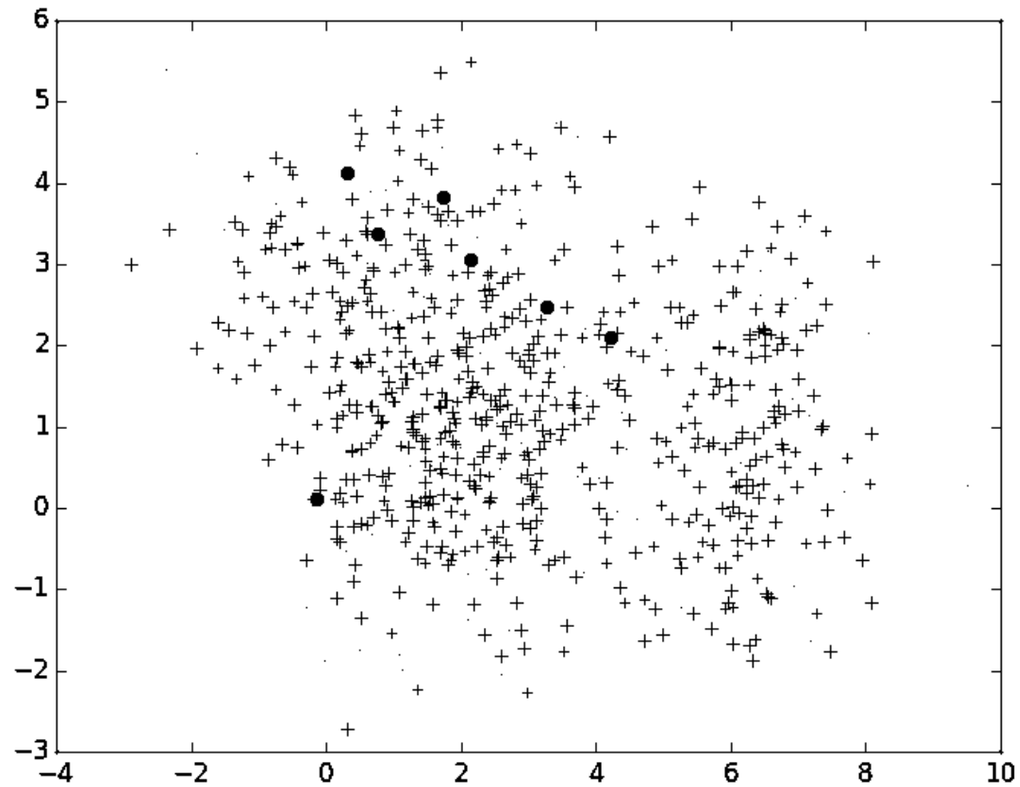
- Extracting patters from data
  - K-Means
  - Agglomerative Clustering
  - PCA
  - Expectation Maximization
  - Gaussian Mixture Models

# Clustering

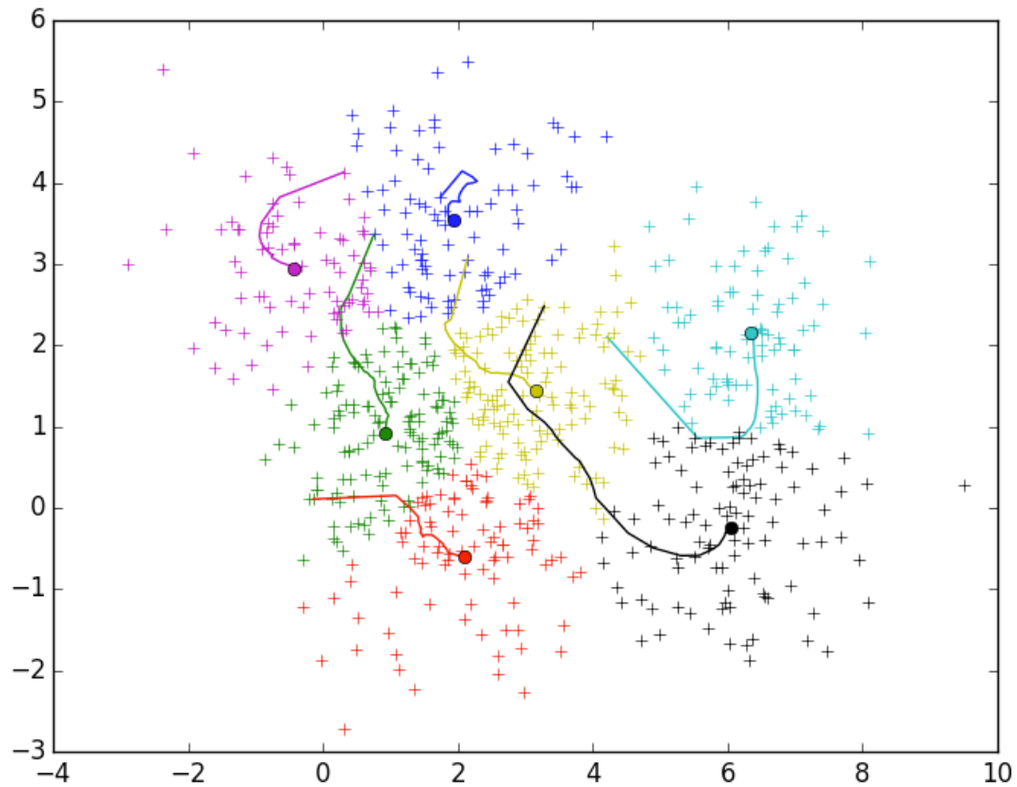


# Clustering

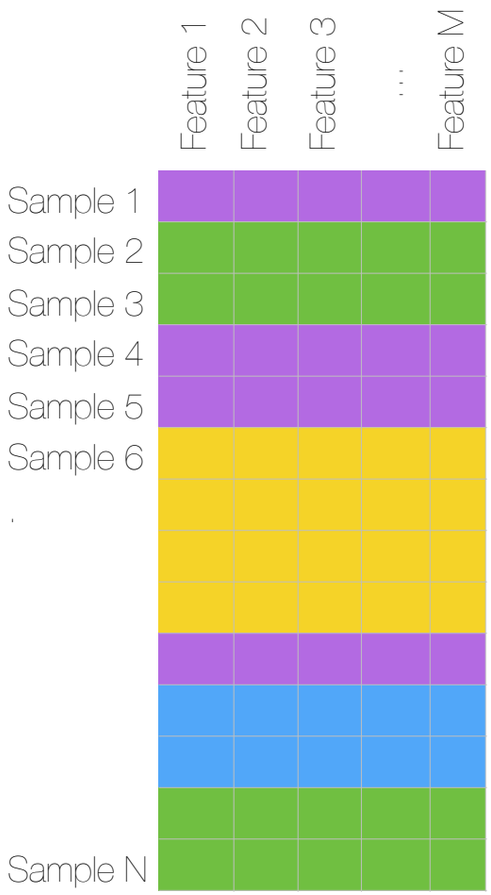

Feature 1
Feature 2
Feature 3
. . .
Feature M
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Sample 6
Sample N

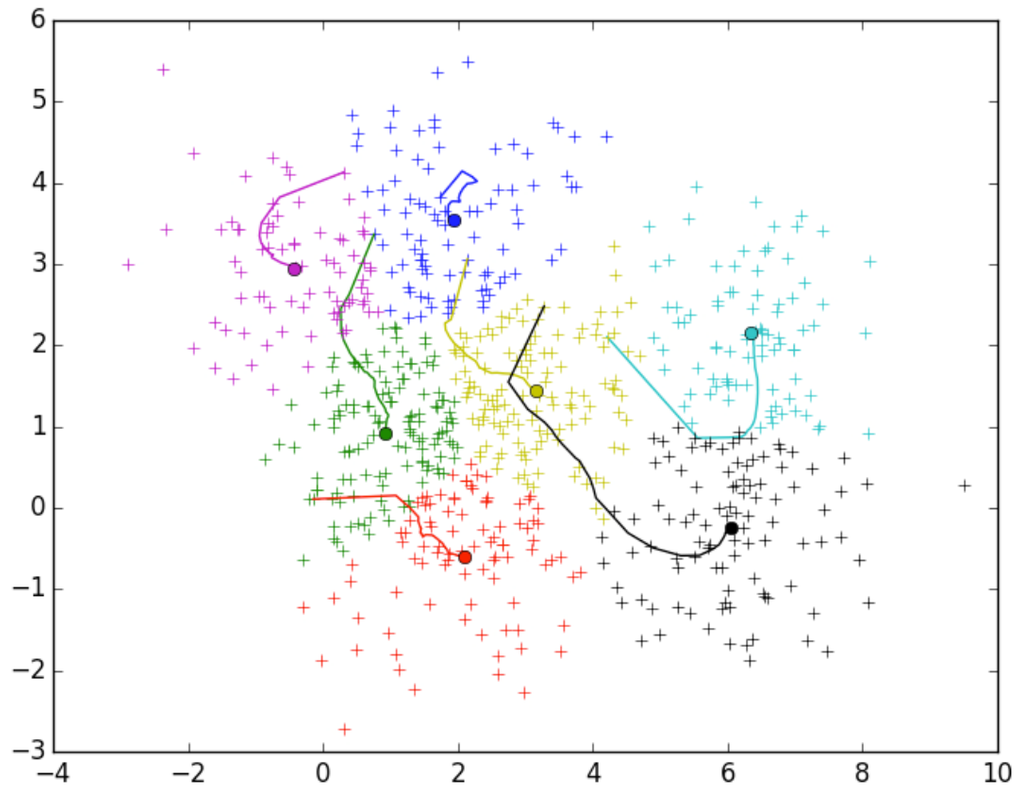

6
5
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8
10

# Clustering

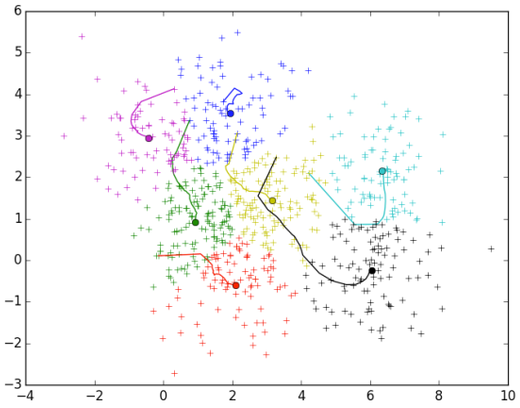


# Clustering

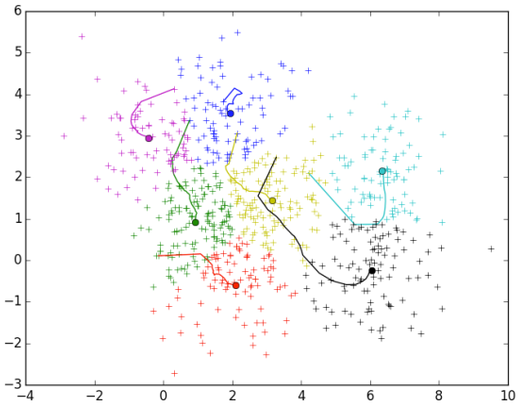

Feature 1
Feature 2
Feature 3
. . .
Feature M
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Sample 6
.
Sample N


6
5
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8
10

# K-Means

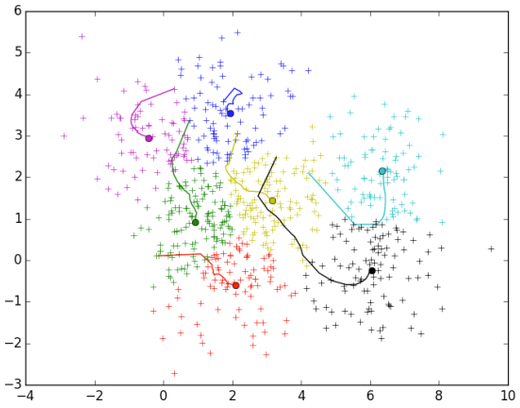


# K-Means




- Choose k randomly chosen points to be the centroid of each cluster

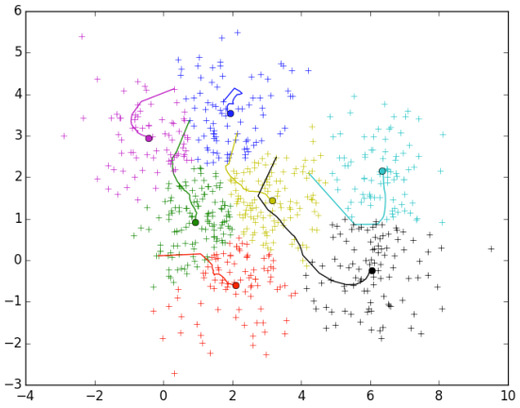
# K-Means




- Choose k randomly chosen points to be the centroid of each cluster

- Assign each point to belong the cluster whose centroid is closest

# K-Means

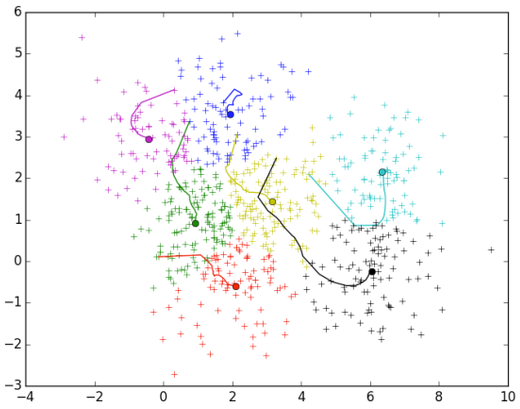



- Choose k randomly chosen points to be the centroid of each cluster

- Assign each point to belong the cluster whose centroid is closest

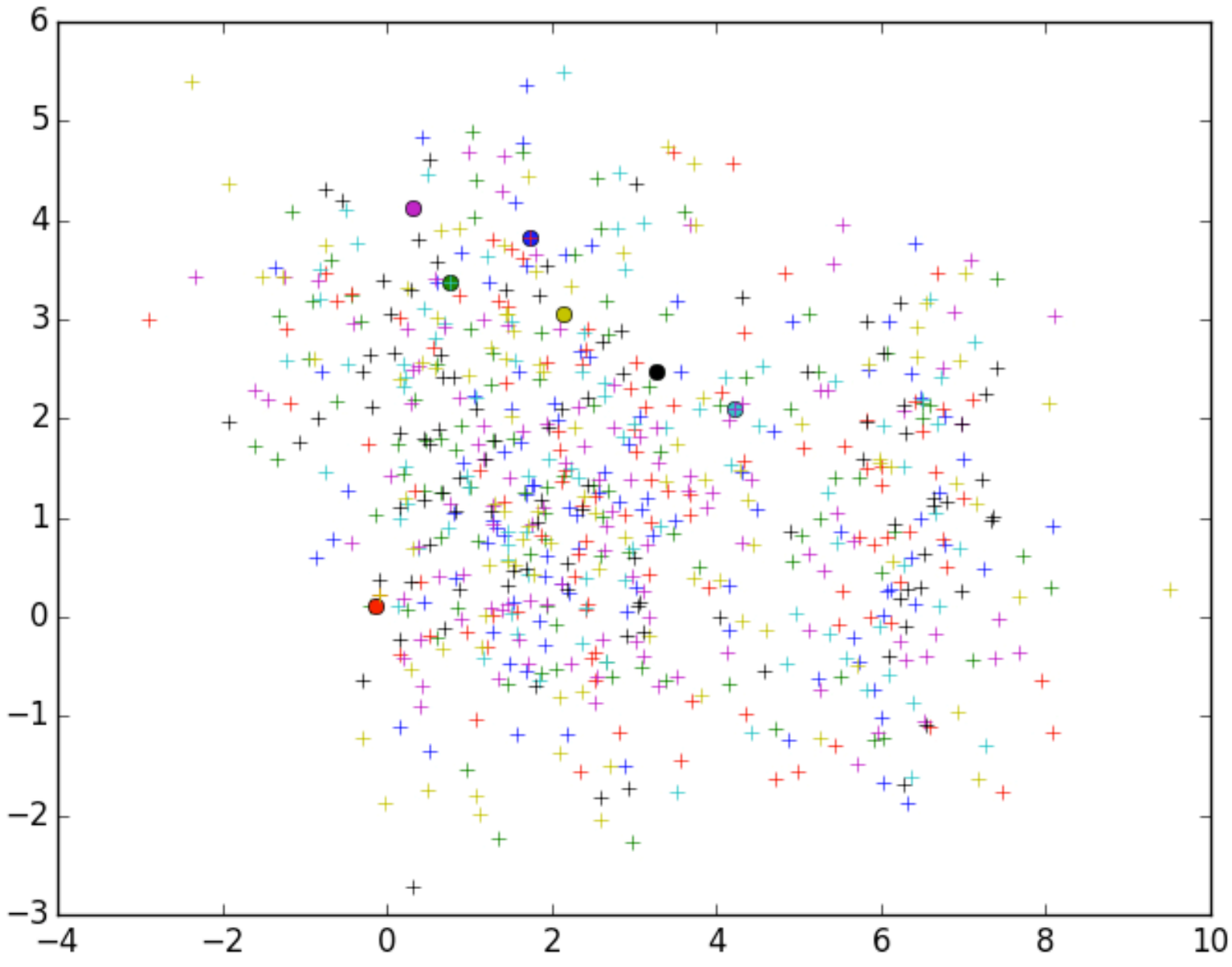
- Recompute the centroid positions (mean cluster position)

# K-Means

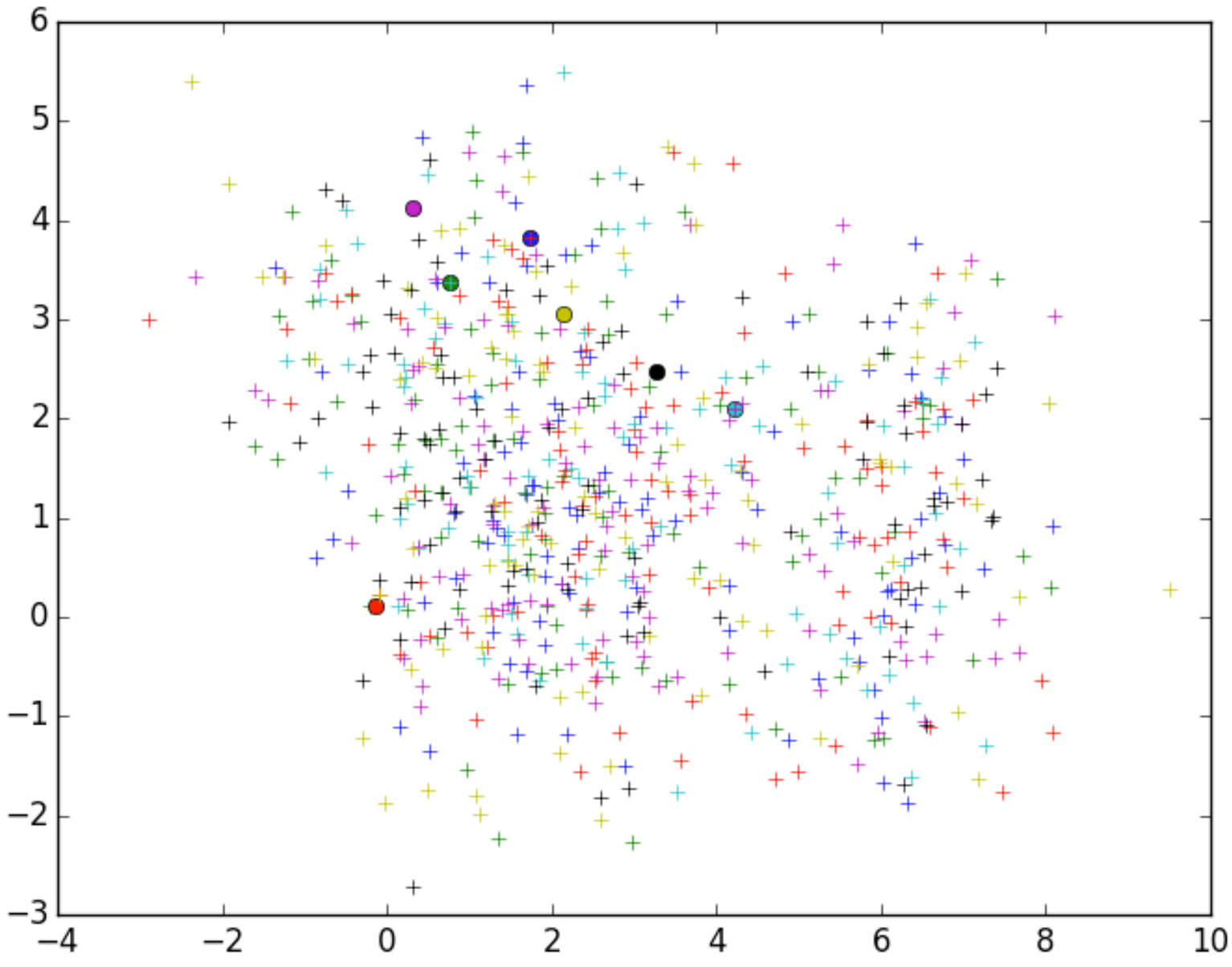



- Choose k randomly chosen points to be the centroid of each cluster
- Assign each point to belong the cluster whose centroid is closest
- Recompute the centroid positions (mean cluster position)
- Repeat until convergence

# K-Means

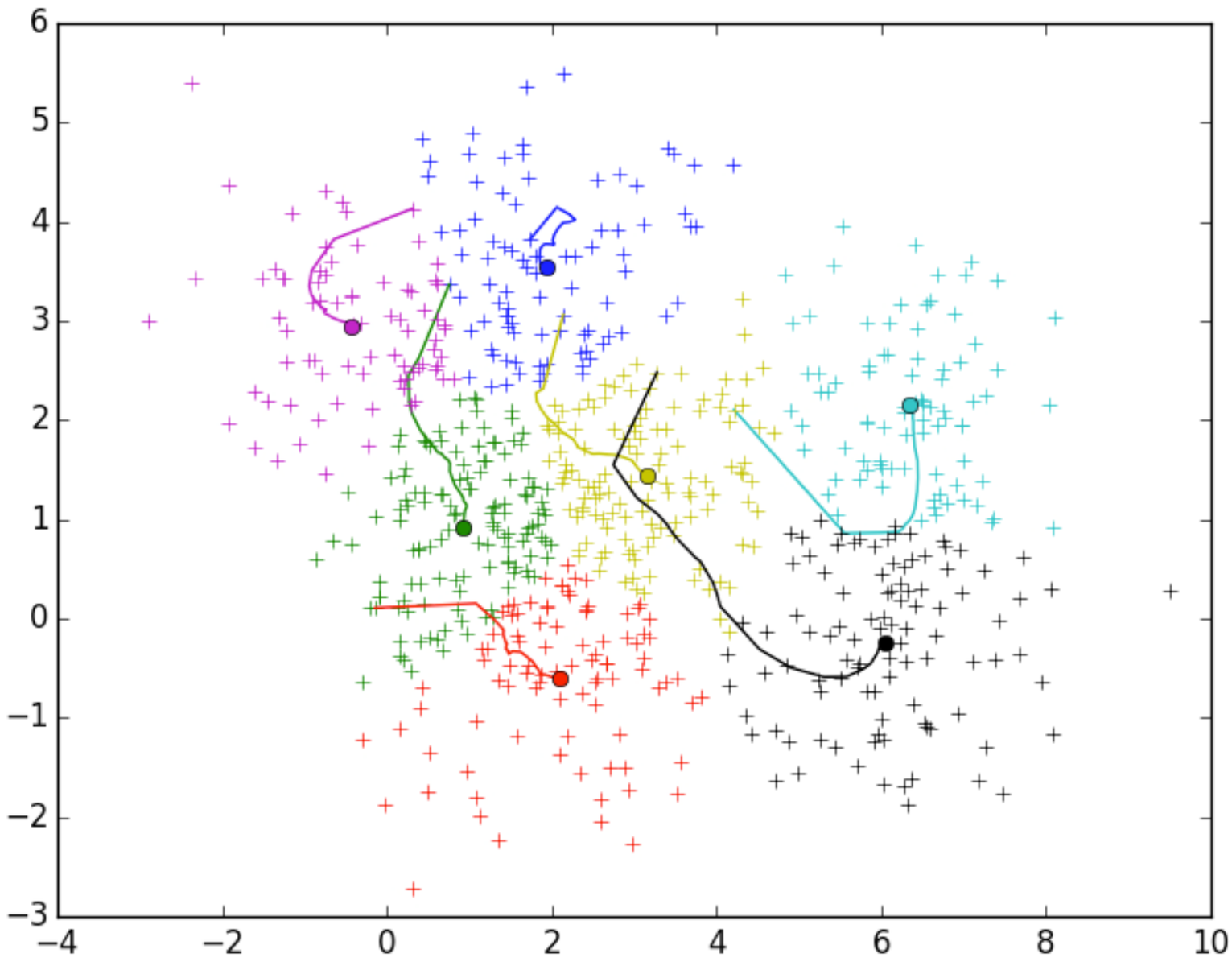


# K-Means



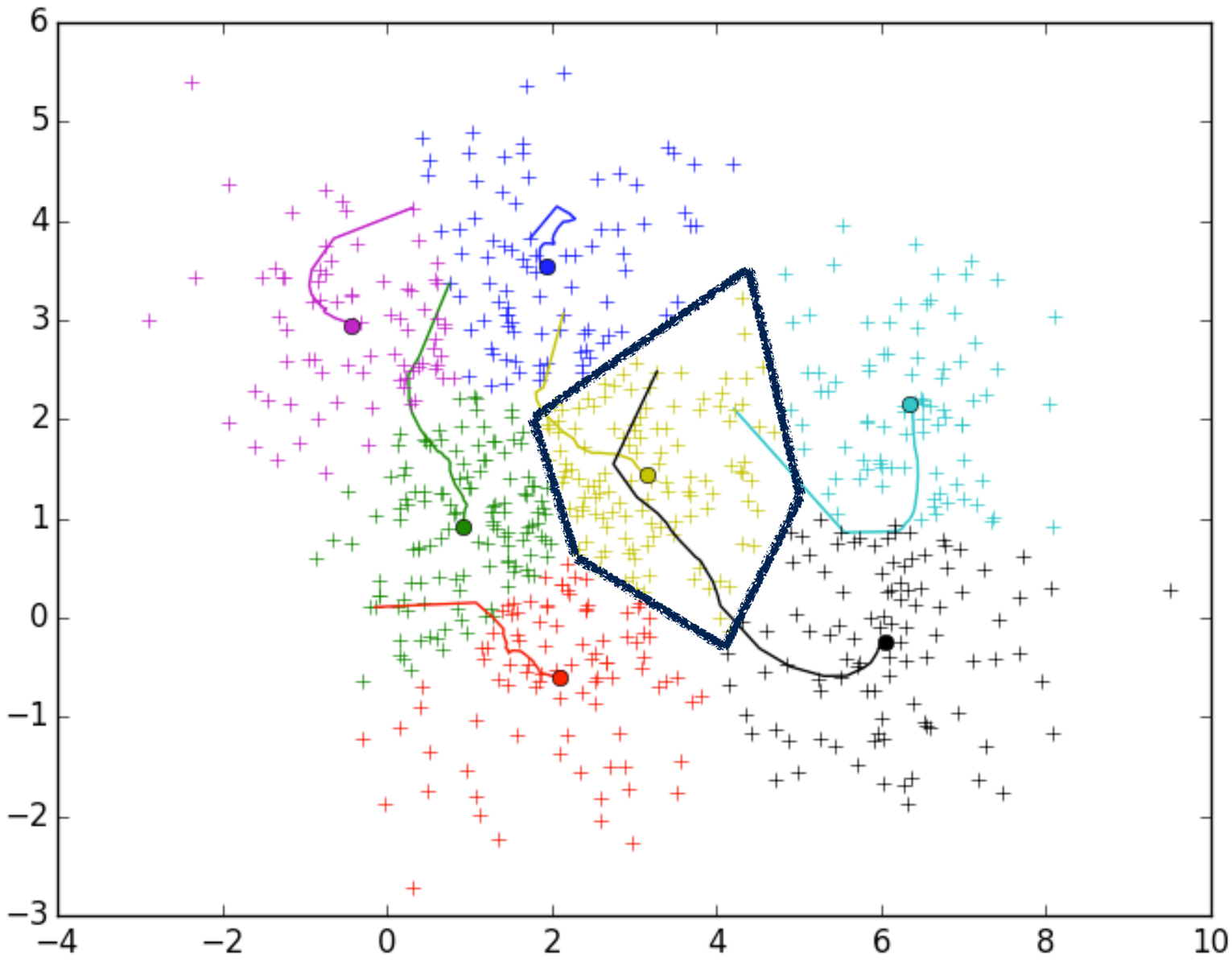
# K-Means: Structure

Voronoi Tesselation



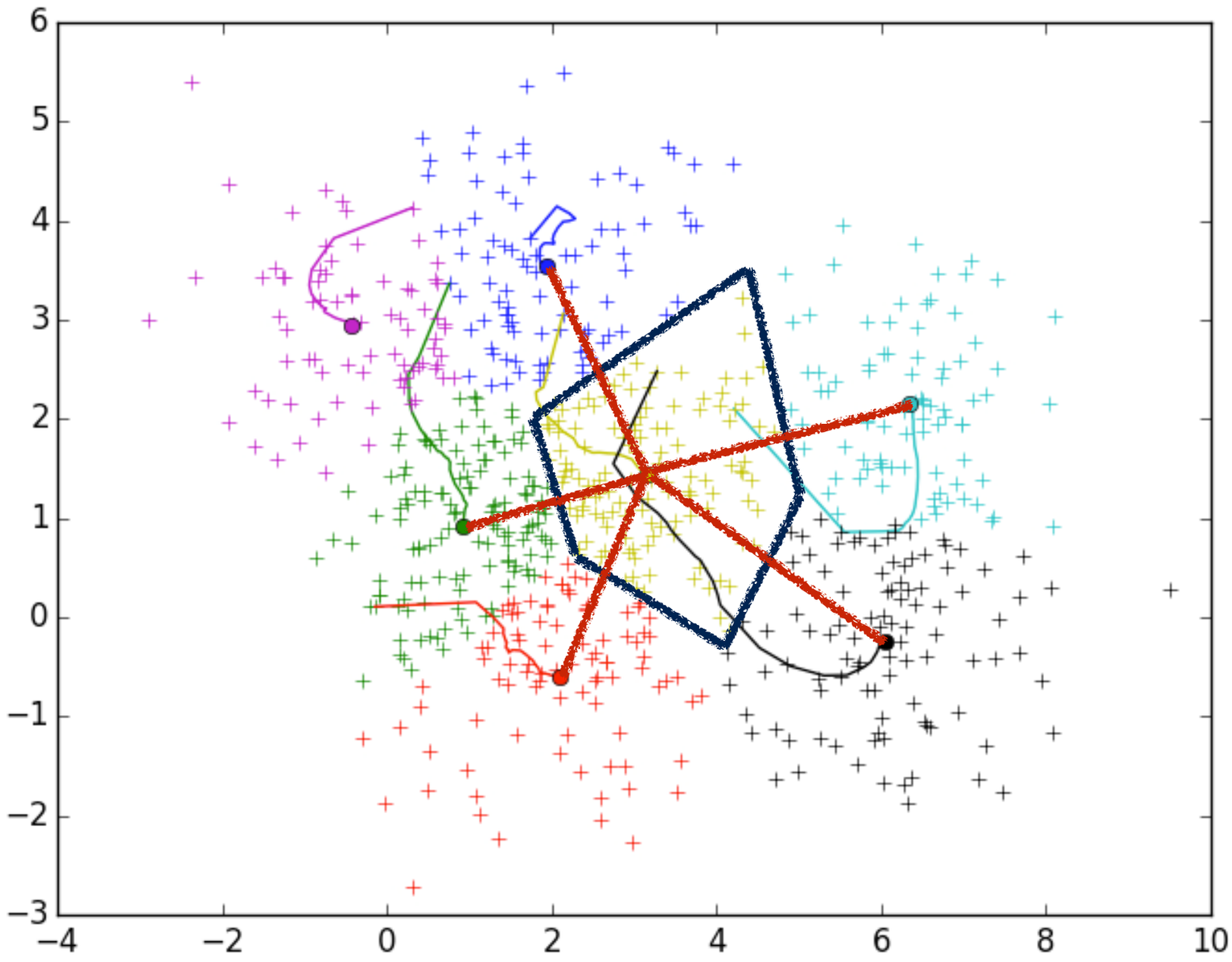
# K-Means: Structure

Voronoi Tesselation



# K-Means: Structure

Voronoi Tesselation



# K-Means: Convergence

# K-Means: Convergence

- How to quantify the "quality" of the solution found at each iteration, $n$?

# K-Means: Convergence

- How to quantify the "quality" of the solution found at each iteration, $n$?

- Measure the "Inertia", the square intra-cluster distance:

$$I_n = \sum_{i=0}^{N} \|x_i - \mu_i\|^2$$

where $\mu_i$ are the coordinates of the centroid of the cluster to which $x_i$ is assigned.

# K-Means: Convergence

- How to quantify the "quality" of the solution found at each iteration, $n$?

- Measure the "Inertia", the square intra-cluster distance:

$$I_n = \sum_{i=0}^{N} \|x_i - \mu_i\|^2$$

where $\mu_i$ are the coordinates of the centroid of the cluster to which $x_i$ is assigned.

- Smaller values are better

- Can stop when the relative variation is smaller than some value

$$\frac{I_{n+1} - I_n}{I_n} < tol$$

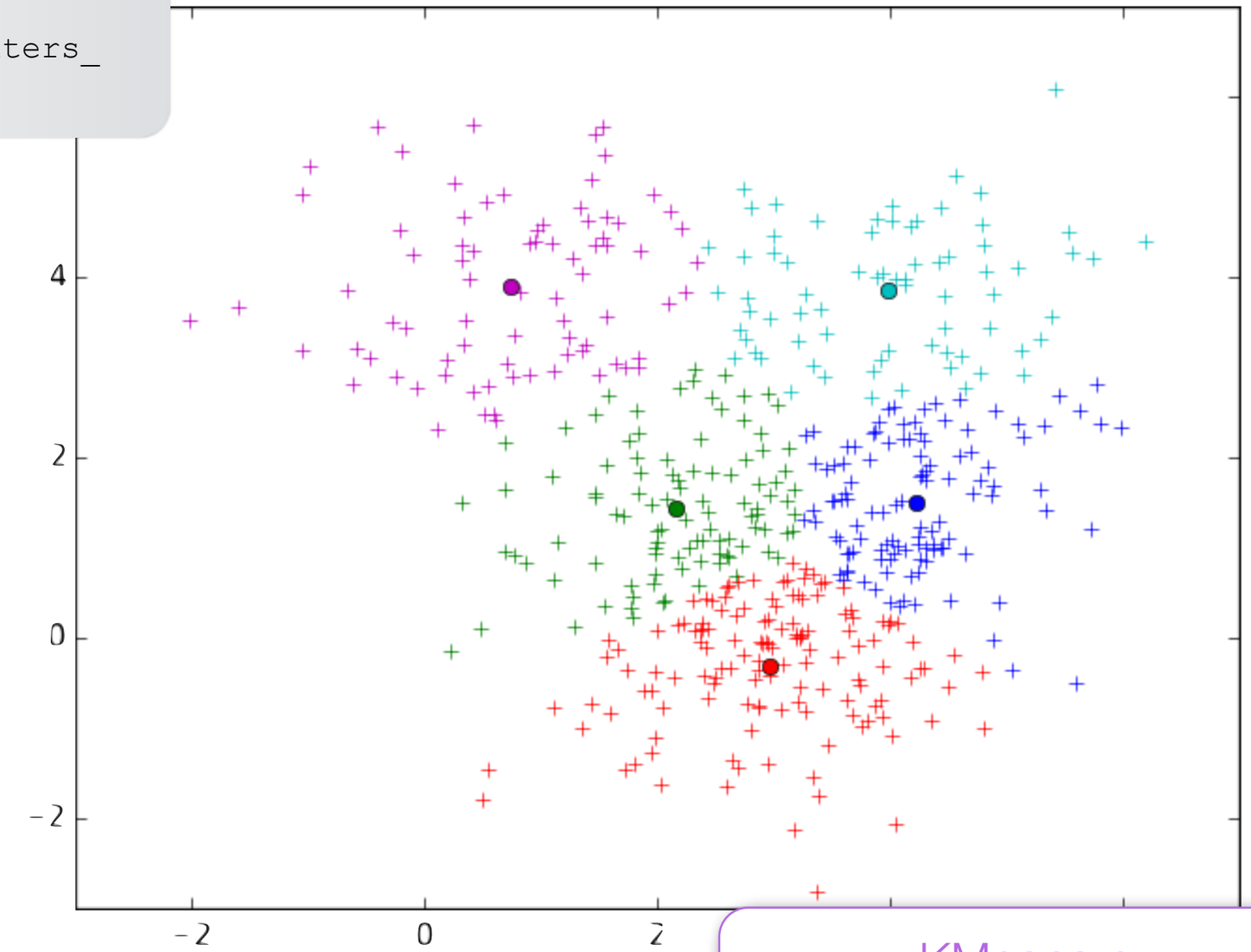
# K-Means: sklearn

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=nclusters)
kmeans.fit(data)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

KMeans.py

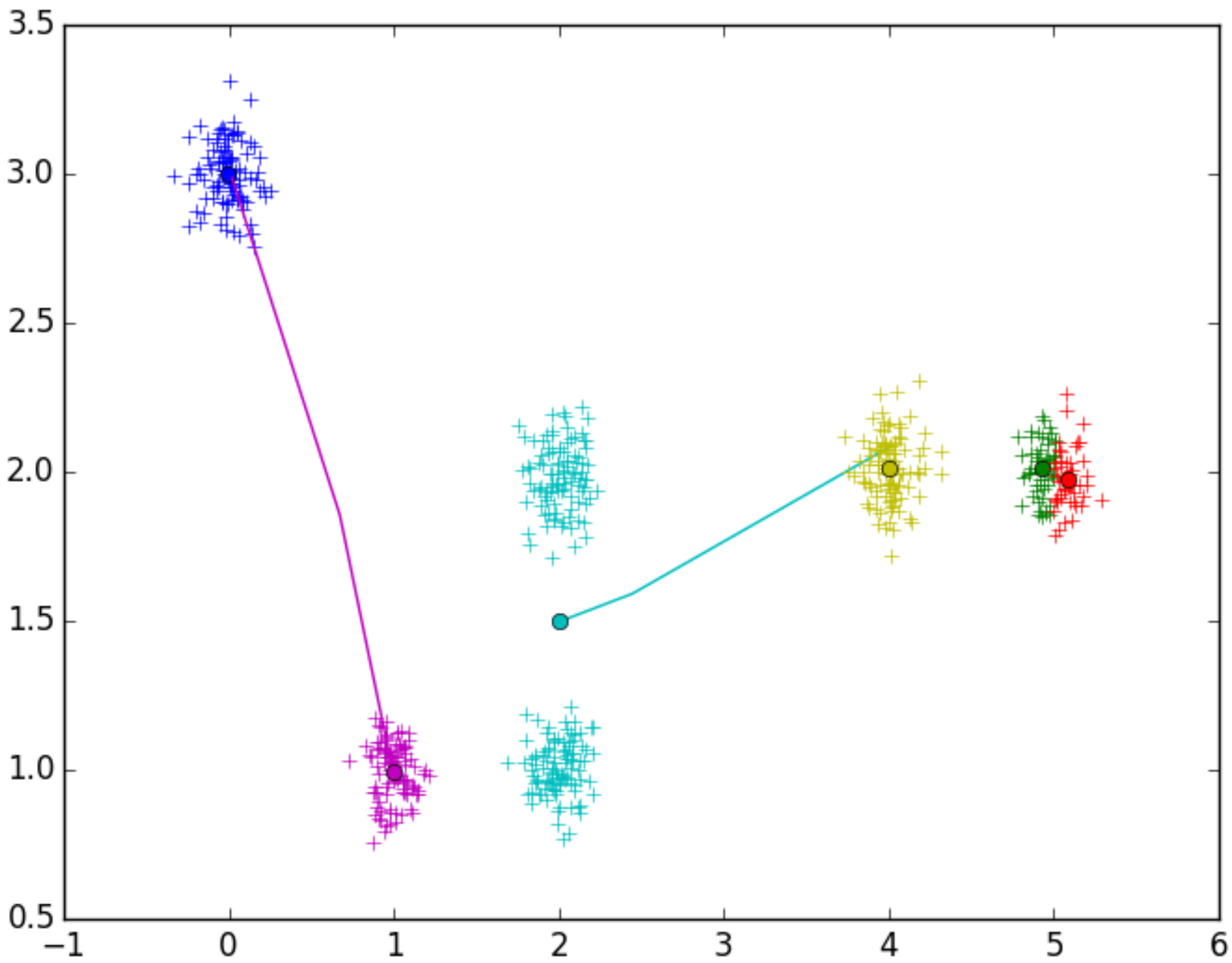
# K-Means: sklearn

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=nclusters)
kmeans.fit(data)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

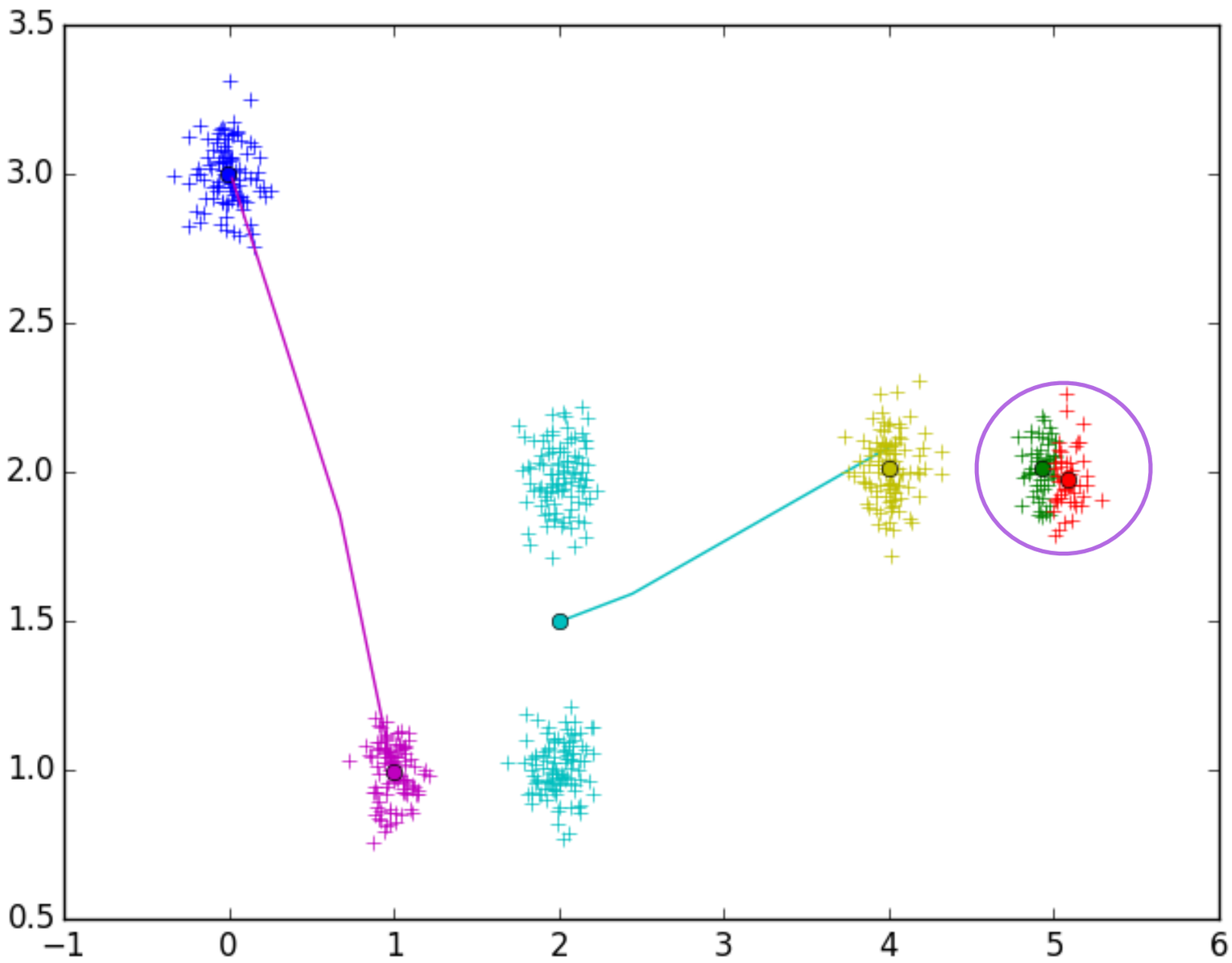



KMeans.py

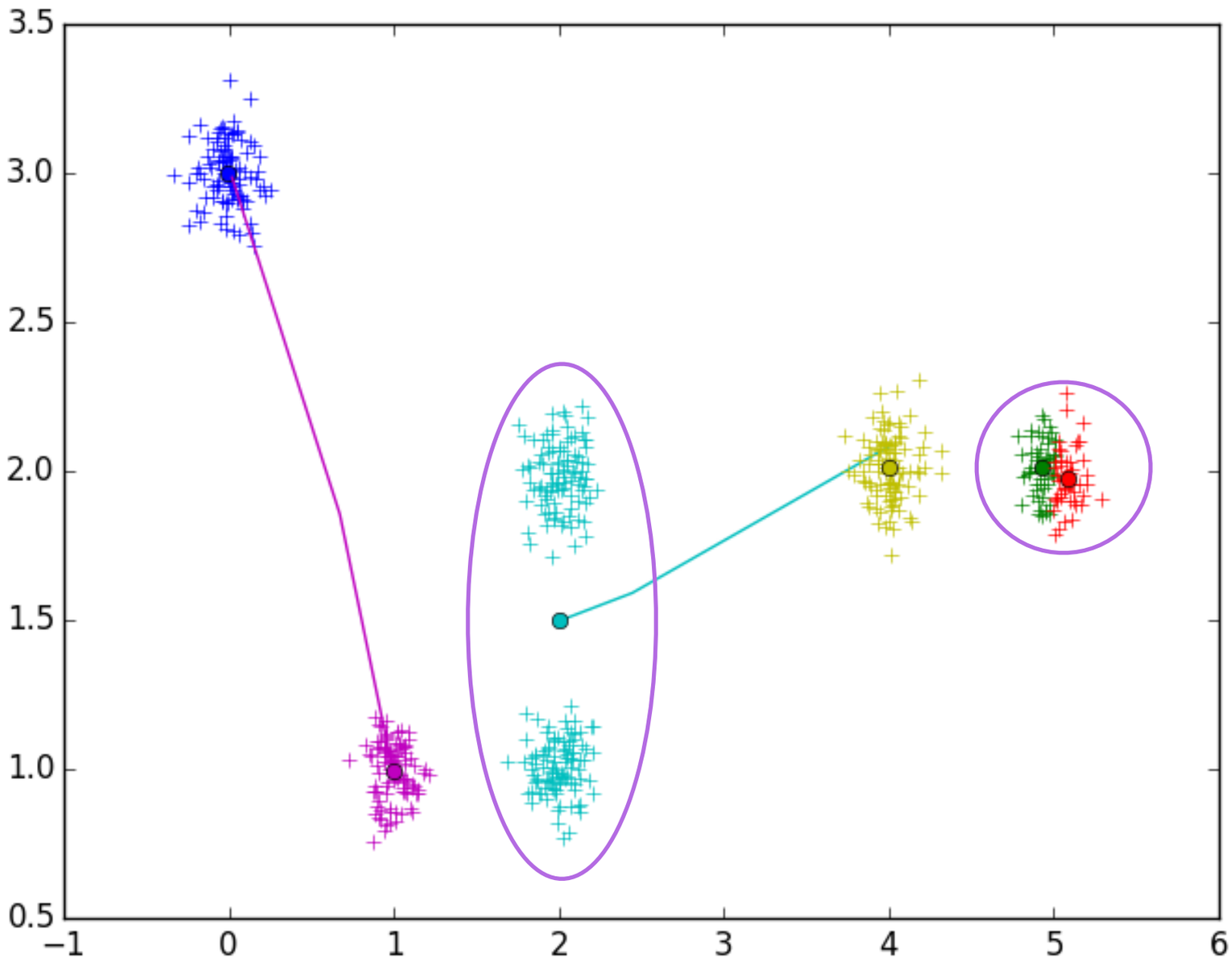
# K-Means: Limitations



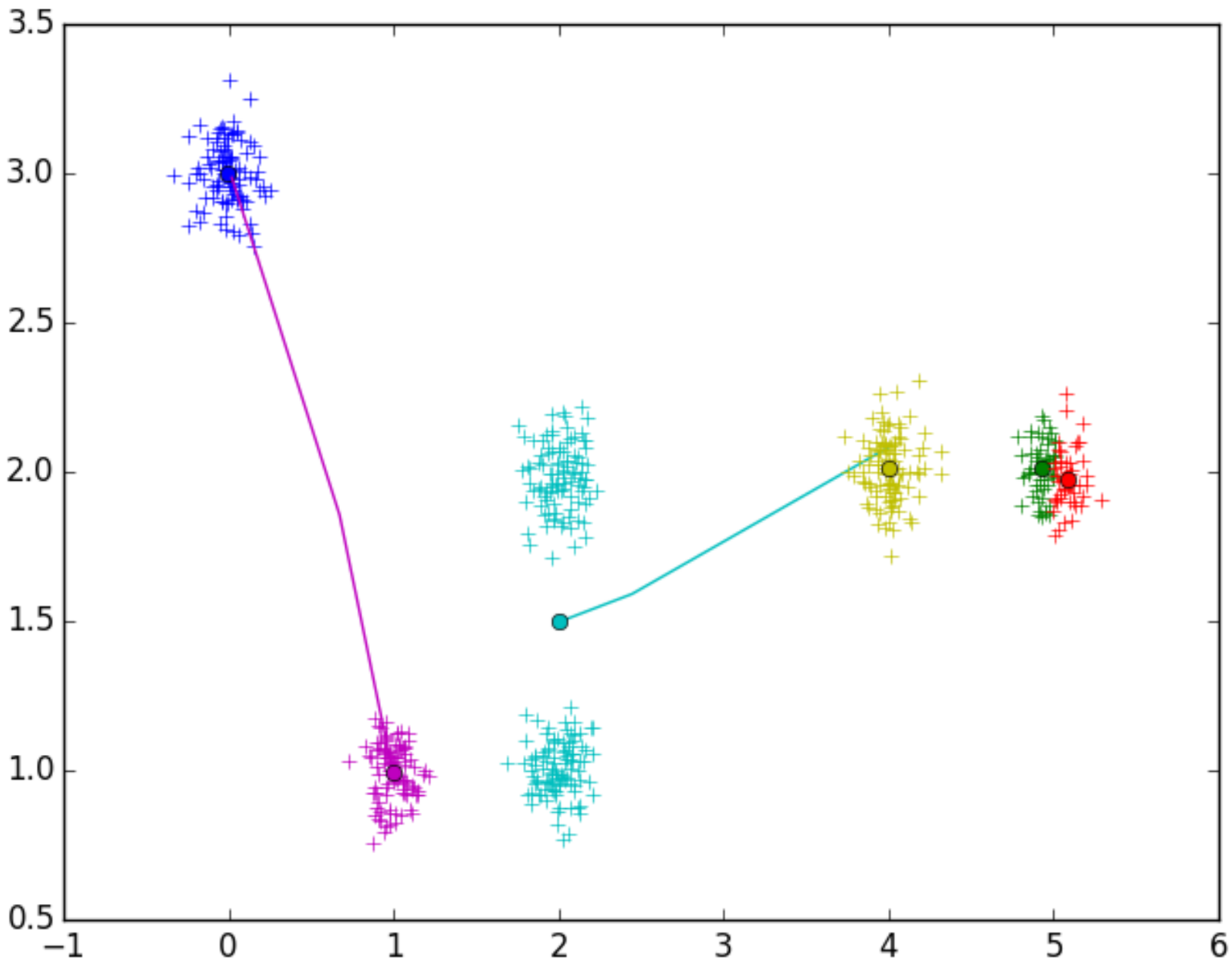
# K-Means: Limitations



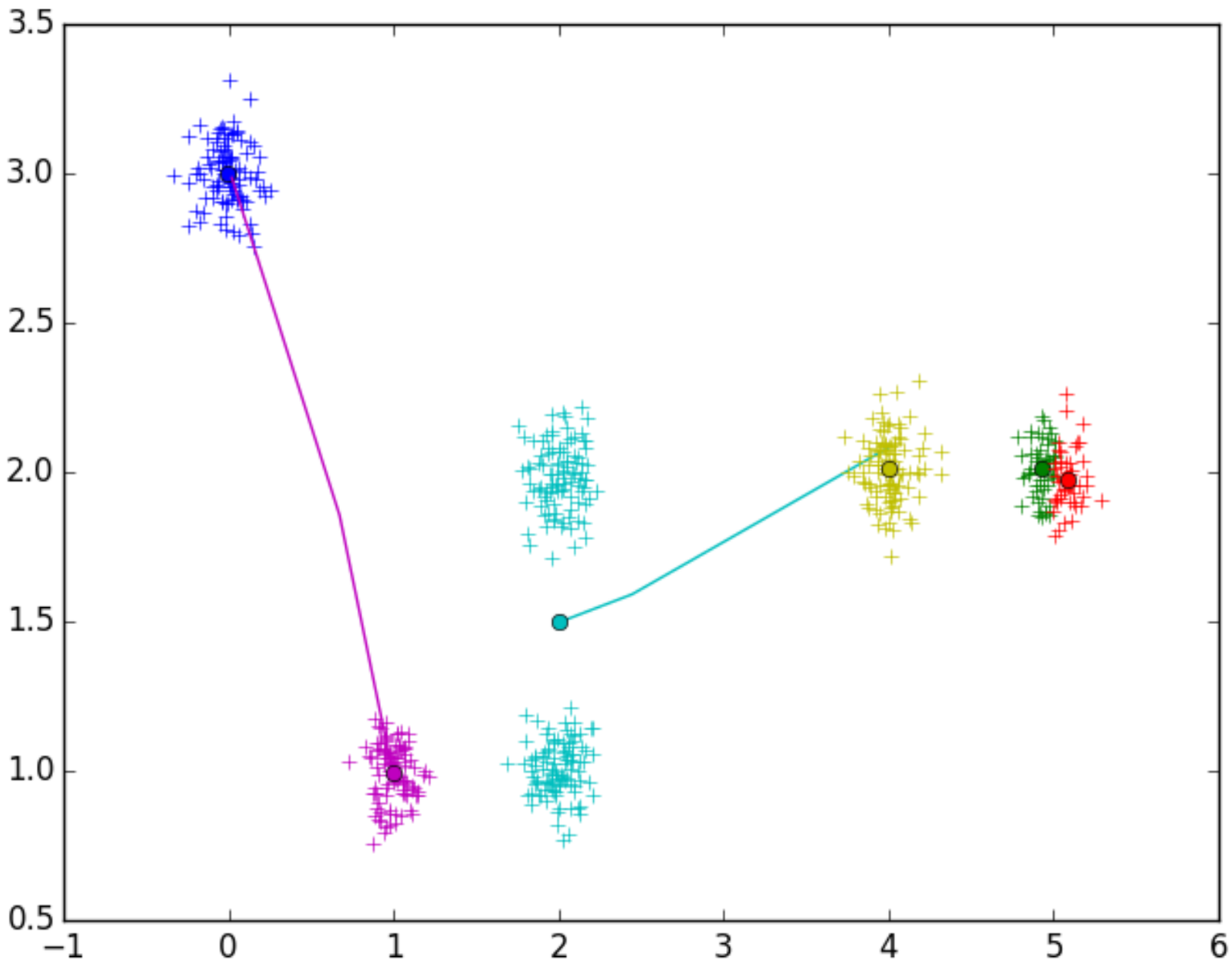
# K-Means: Limitations



# K-Means: Limitations

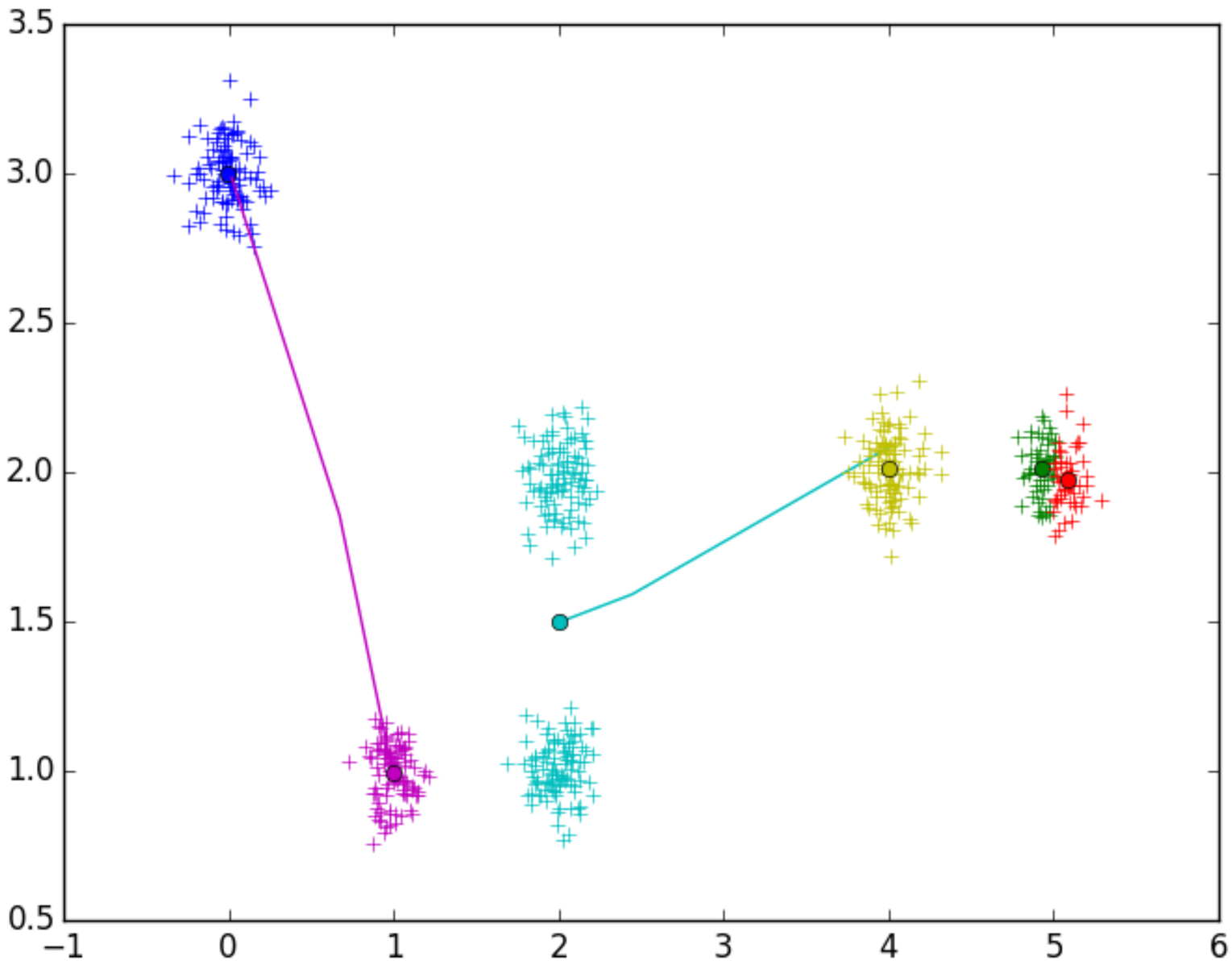


# K-Means: Limitations



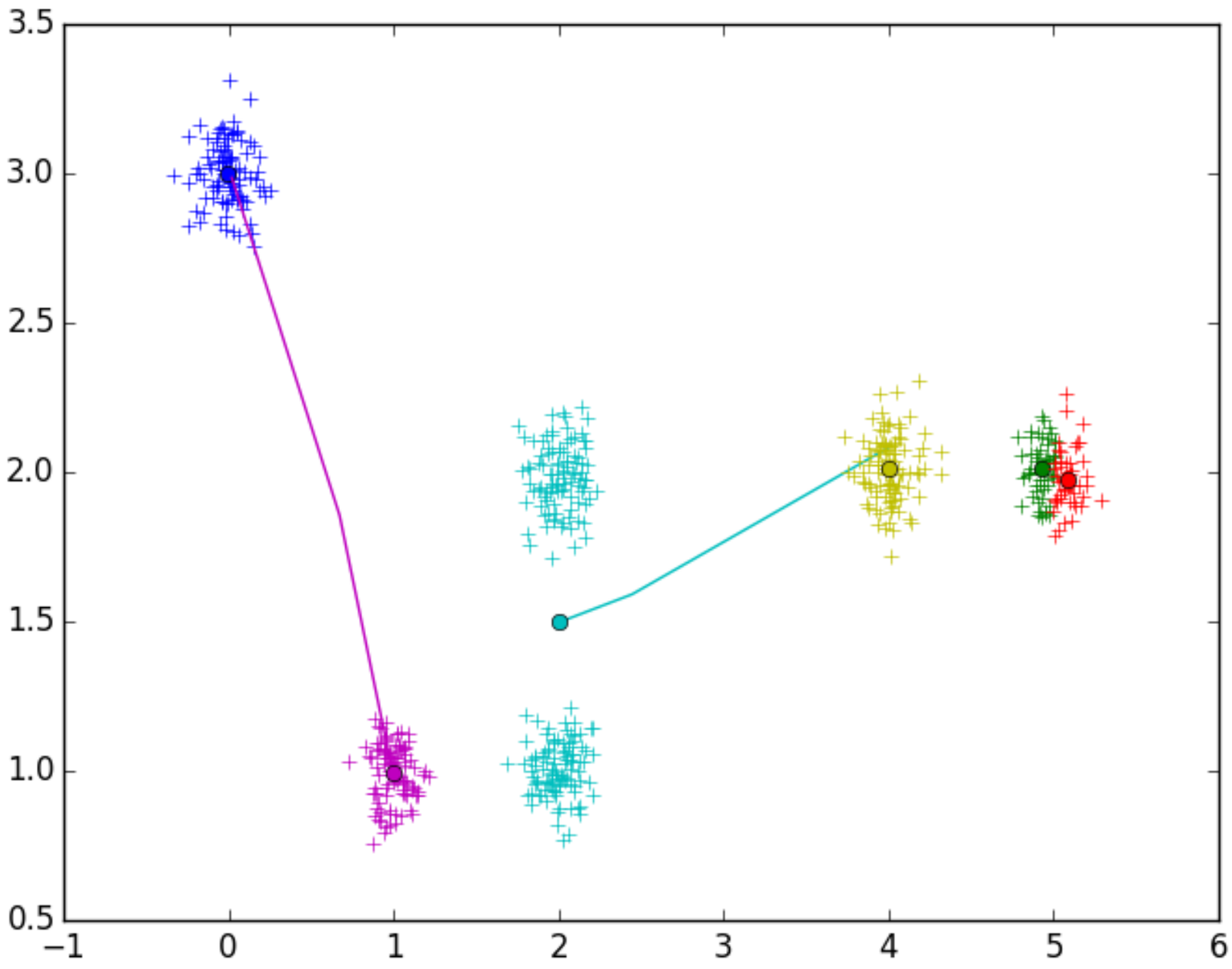

- No guarantees about Finding "Best" solution

# K-Means: Limitations




- No guarantees about Finding "Best" solution
- Each run can find different solution

# K-Means: Limitations




- No guarantees about Finding "Best" solution
- Each run can find different solution
- No clear way to determine "k"

# Silhouettes

# Silhouettes

- For each point $x_i$ define $a_c\left(x_i\right)$ as:

$$a_c\left(x_i\right) = \frac{1}{N_c}\sum_{j \in c} \|x_i - x_j\|$$

the average distance between point $x_i$ and every other point within cluster $c$

# Silhouettes

- For each point $x_i$ define $a_c\left(x_i\right)$ as:

$$a_c\left(x_i\right) = \frac{1}{N_c}\sum_{j \in c} \|x_i - x_j\|$$

the average distance between point $x_i$ and every other point within cluster $c$

- Let $b\left(x_i\right)$ be:

$$b\left(x_i\right) = \min_{c \neq c_i} a_c\left(x_i\right)$$

the minimum value of $a_c\left(x_i\right)$ excluding $c_i$

# Silhouettes

- For each point $x_i$ define $a_c\left(x_i\right)$ as:

$$a_c\left(x_i\right) = \frac{1}{N_c}\sum_{j \in c}\left\|x_i - x_j\right\|$$

the average distance between point $x_i$ and every other point within cluster $c$

- Let $b\left(x_i\right)$ be:

$$b\left(x_i\right) = \min_{c \neq c_i} a_c\left(x_i\right)$$

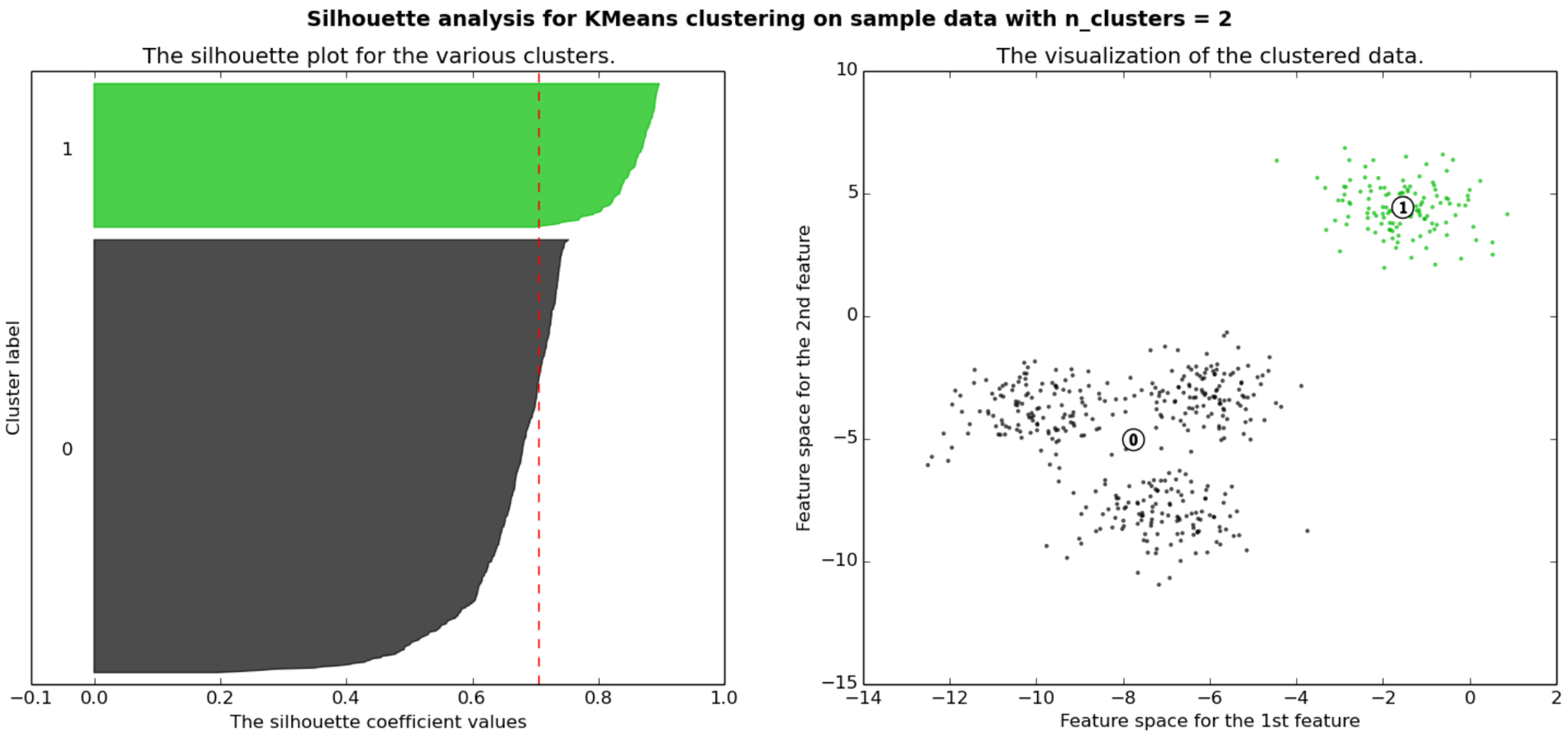
the minimum value of $a_c\left(x_i\right)$ excluding $c_i$

- The silhouette of $x_i$ is then:

$$s\left(x_i\right) = \frac{b\left(x_i\right) - a_{c_i}\left(x_i\right)}{\max\left\{b\left(x_i\right), a_{c_i}\left(x_i\right)\right\}}$$

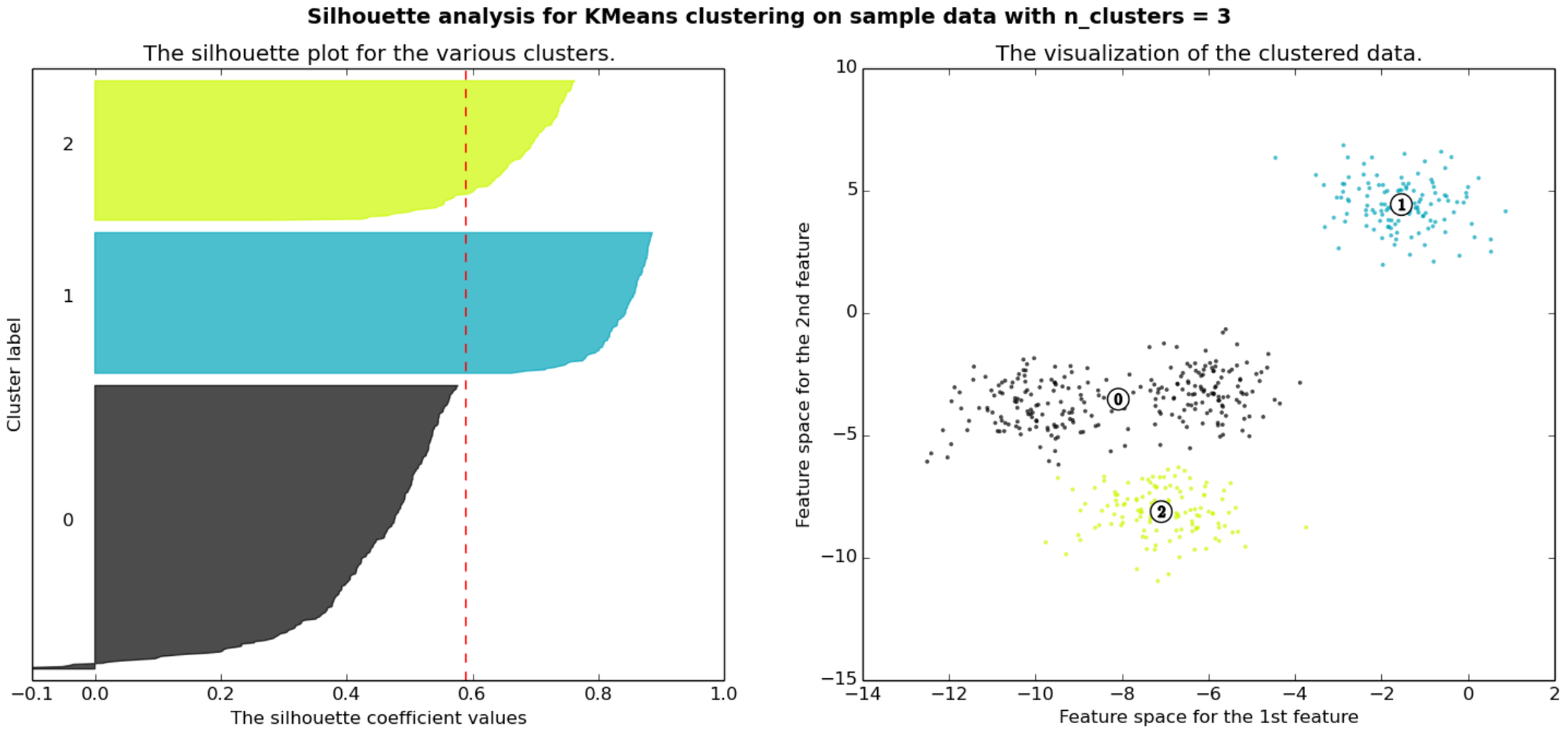
# Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



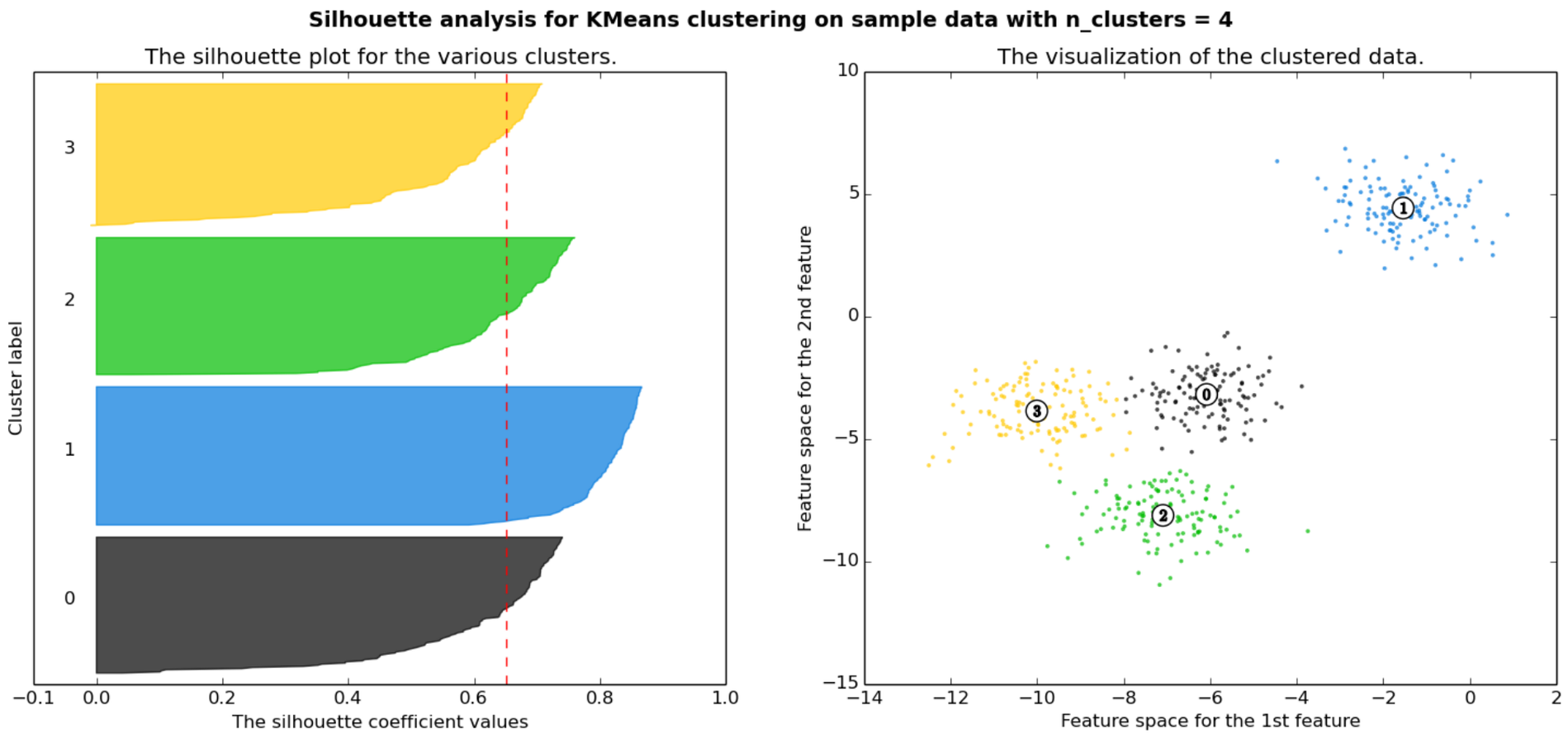
# Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



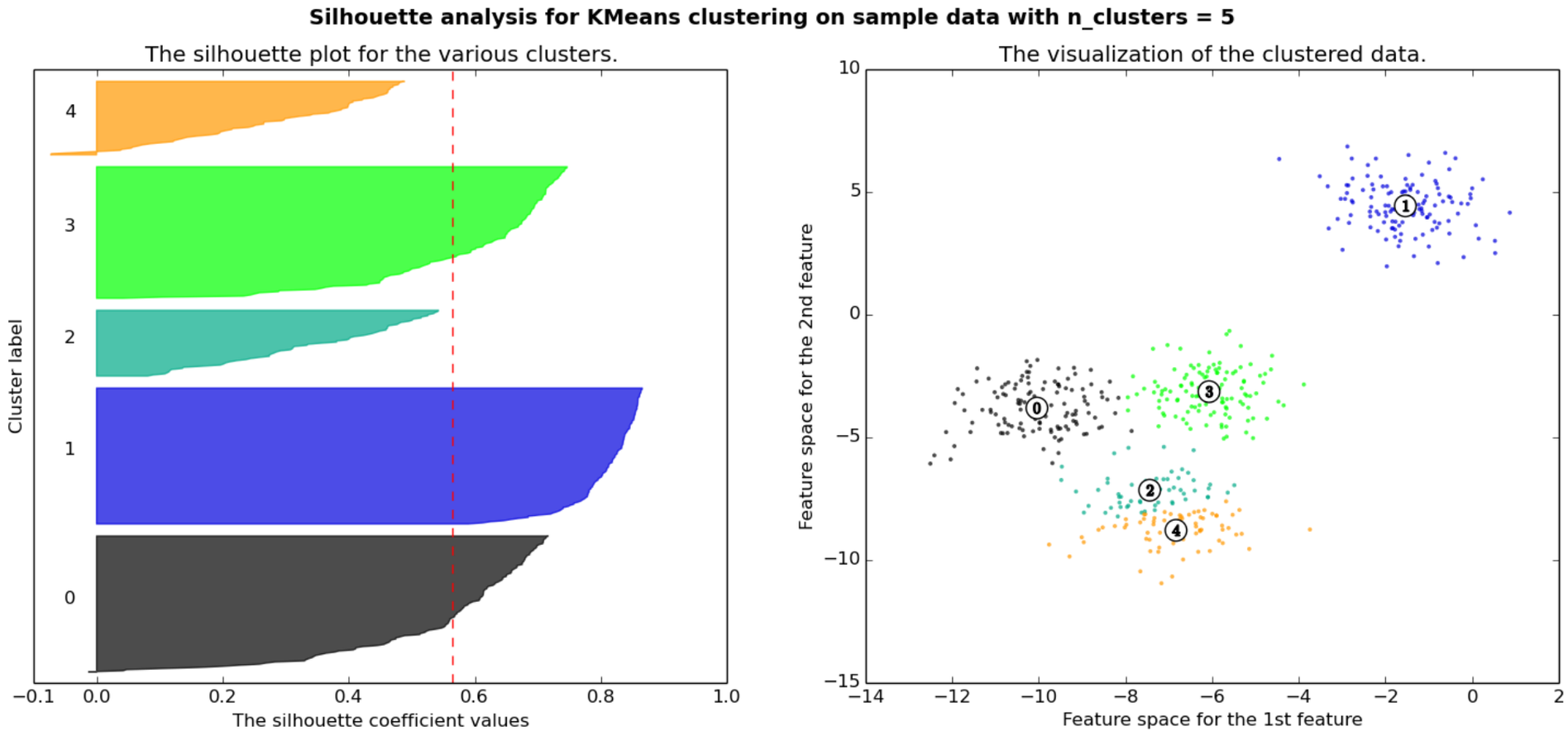
# Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



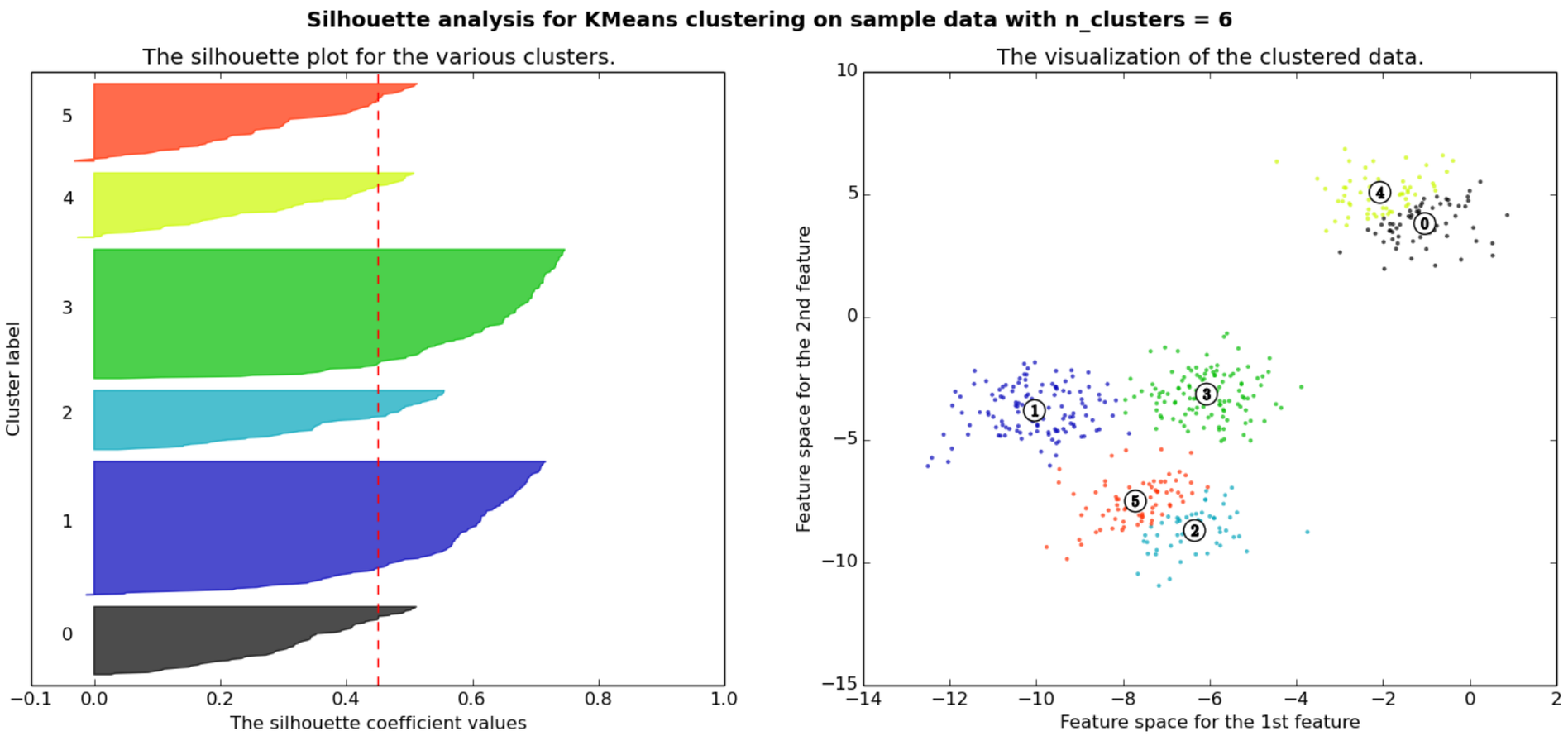
# Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



# Silhouettes

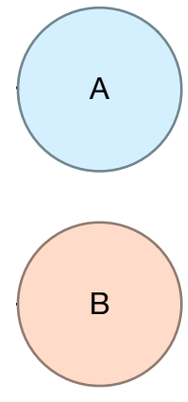
http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



# Agglomerative Clustering

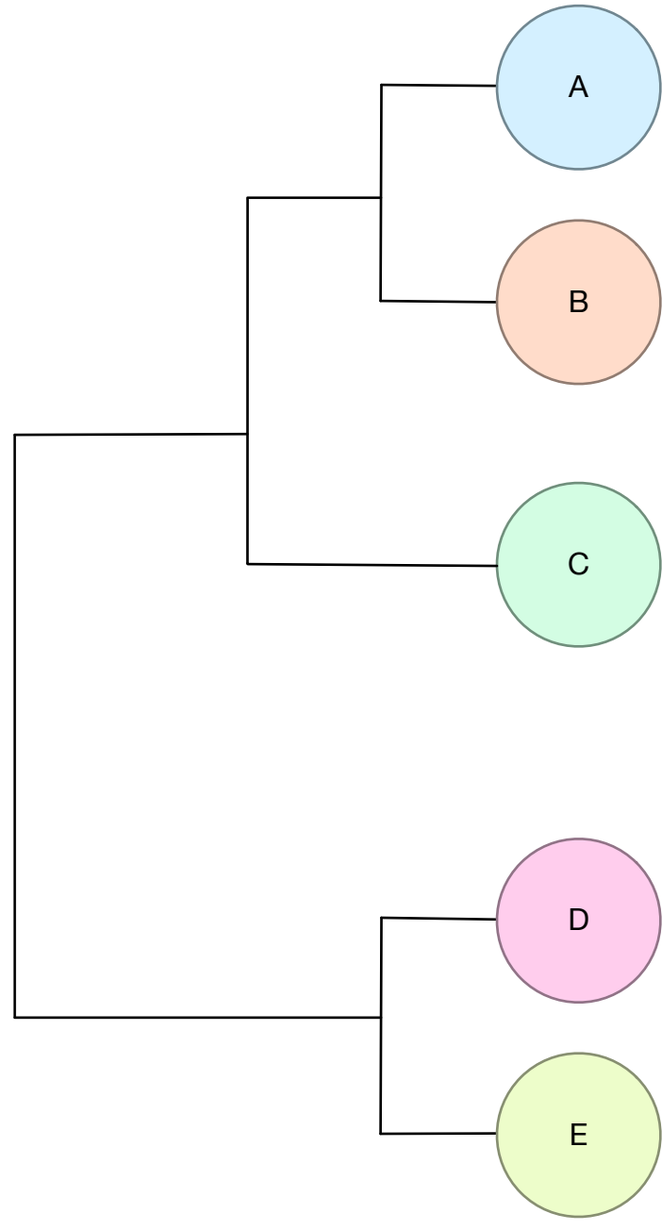
# Agglomerative Clustering

- Every point starts as it's own cluster.



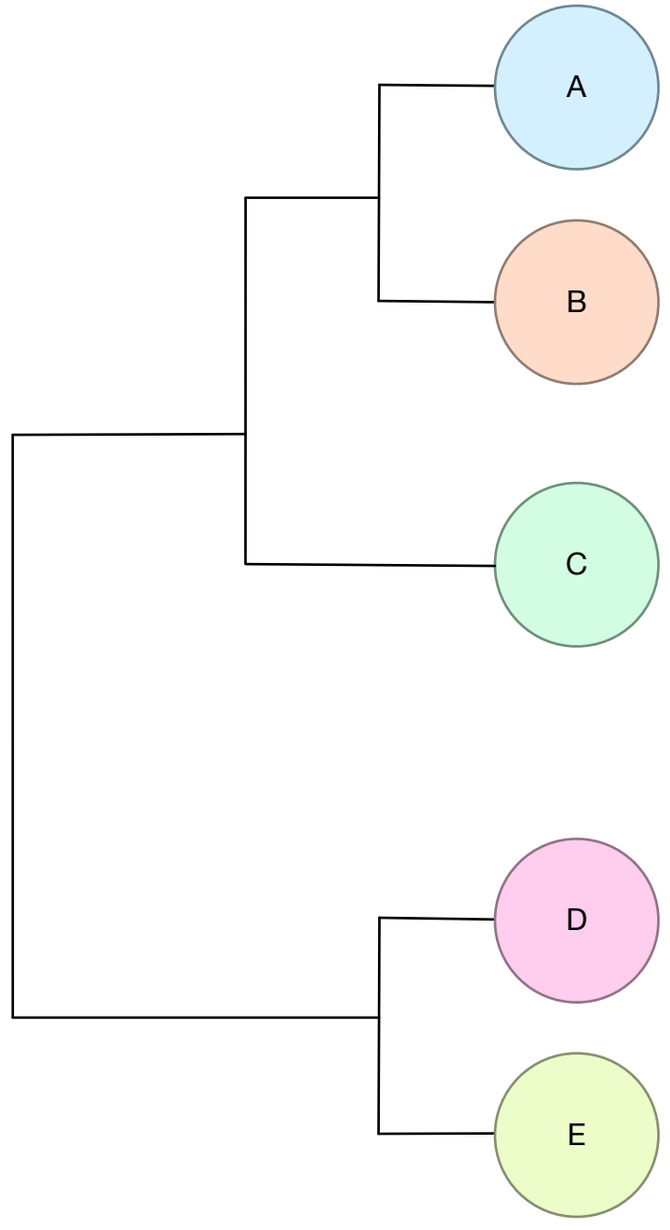
# Agglomerative Clustering

- Every point starts as it's own cluster.
- Merge the two clusters that are nearest to each other
- Repeat…



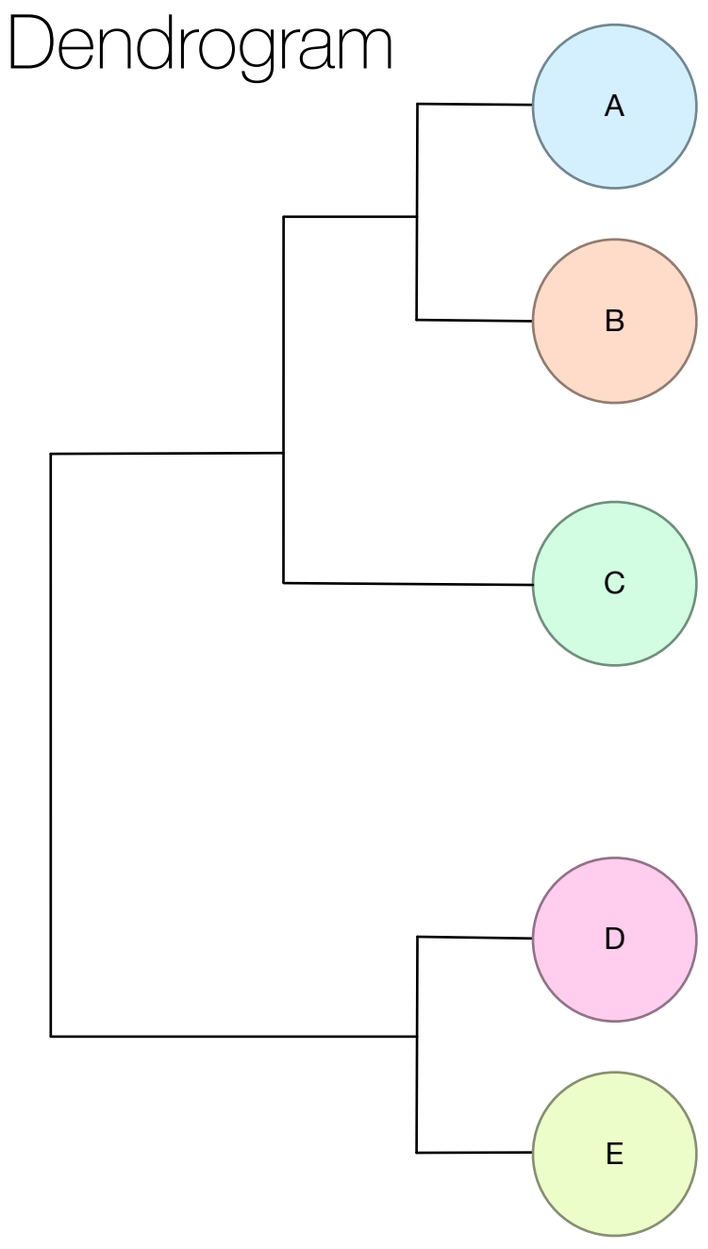
# Agglomerative Clustering

- Every point starts as it's own cluster.
- Merge the two clusters that are nearest to each other
- Repeat…
- Choose the number of clusters based on a cutoff in the Dendrogram



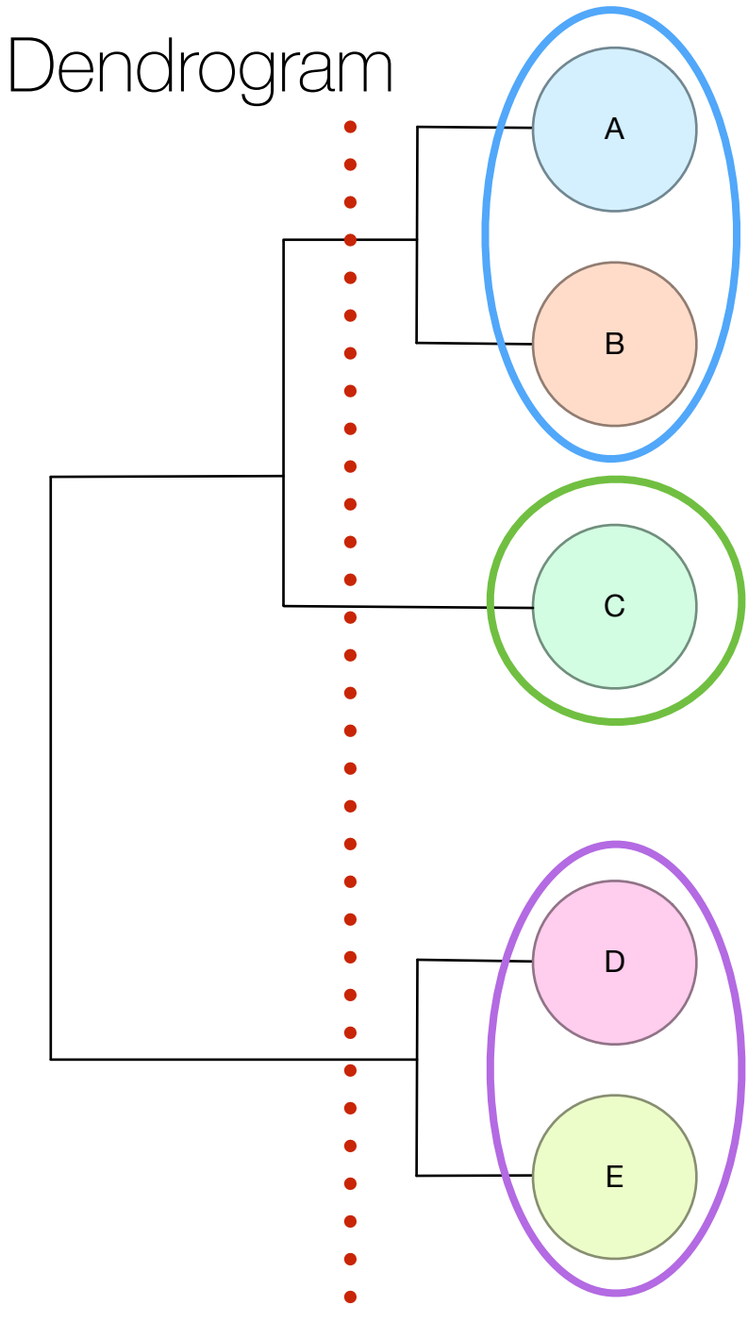
# Agglomerative Clustering

- Every point starts as it's own cluster.
- Merge the two clusters that are nearest to each other
- Repeat…
- Choose the number of clusters based on a cutoff in the Dendrogram
- Different algorithms based on different ways of computing the distances.



# Agglomerative Clustering

- Every point starts as it's own cluster.
- Merge the two clusters that are nearest to each other
- Repeat…
- Choose the number of clusters based on a cutoff in the Dendrogram
- Different algorithms based on different ways of computing the distances.



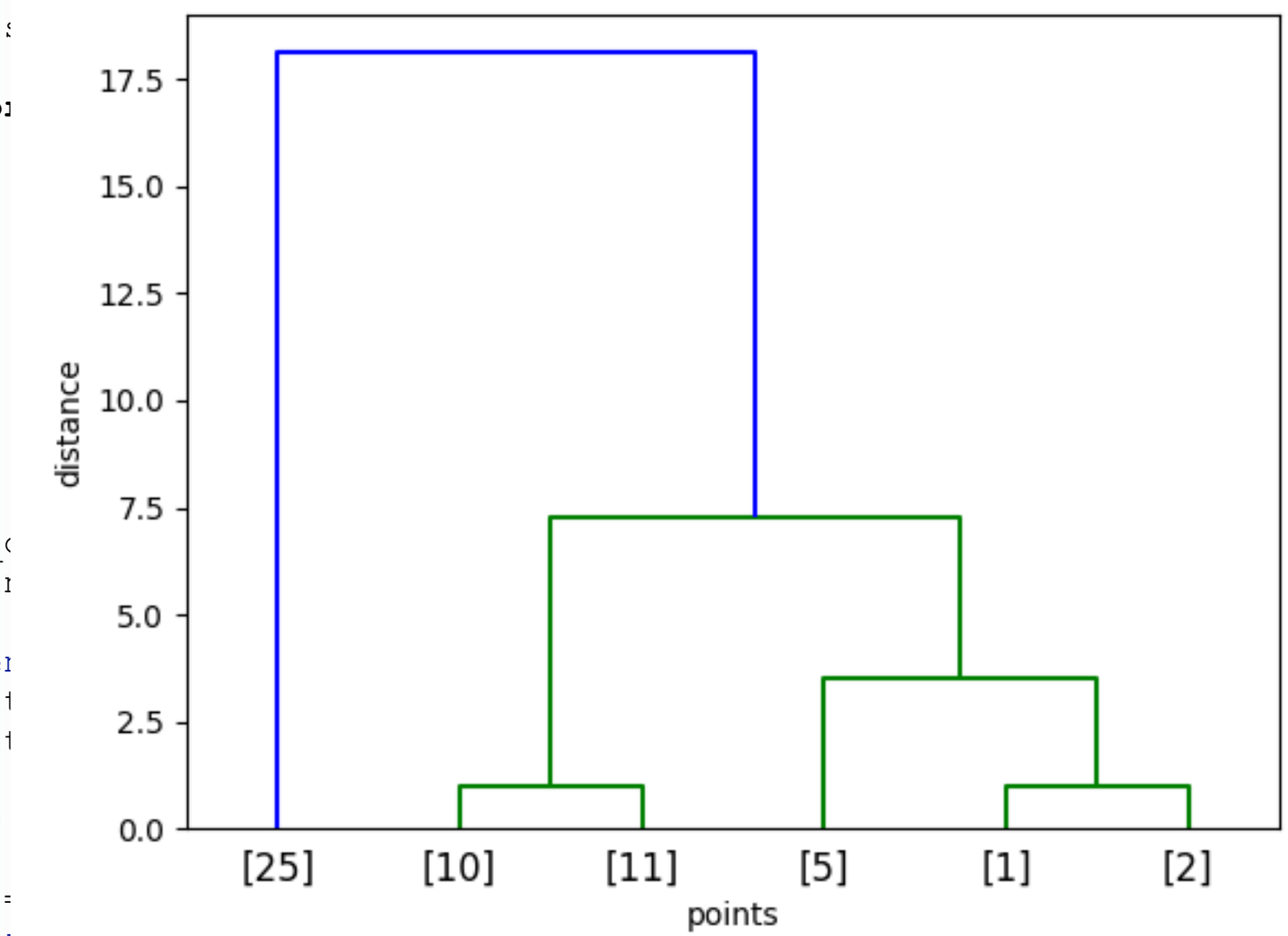
Agg

```python
import numpy as np
from numpy import linalg

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

def plot_dendrogram(points, children):
    N, M = children.shape
    new_points = list(points)
    dist = np.zeros(N)

    for i in range(N):
        node_i, node_j = children[i, :]

        pos_i = new_points[node_i]
        pos_j = new_points[node_j]

        new_pos = (pos_i+pos_j)/2.
        new_points.append(new_pos)

        dist[i] = linalg.norm(pos_i-pos_j)

    n_obs = np.arange(2, N+2)
    link_matrix = np.column_stack([children, dist, n_obs]).astype(float)

    dendrogram(link_matrix, distance_sort=True, labels=points)
    plt.xlabel('points')
    plt.ylabel('distance')

points = np.array([[1,2,5,10,11,25]]).T

model = AgglomerativeClustering()
model.fit(points)

plot_dendrogram(points, model.children_)
plt.savefig('agglomerative.png')
```

agglomerative.py

# Agg

```
import numpy as np
from numpy import linalg

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering

def pl
    N,
    ne
    di

    fo

    n_
    li

    de
    pl
    pl

points

model =
model.

plot_dendrogram(points, model.children_)
plt.savefig('agglomerative.png')
```




agglomerative.py

# Expectation Maximization

# Expectation Maximization

- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

# Expectation Maximization

- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

- Two steps for each iteration

# Expectation Maximization

- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

- Two steps for each iteration
  - Expectation: Calculate the likelihood of the data given current parameter estimate

# Expectation Maximization

- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

- Two steps for each iteration
  - Expectation: Calculate the likelihood of the data given current parameter estimate
  - Maximization: Find the parameter values that maximize the likelihood

# Expectation Maximization

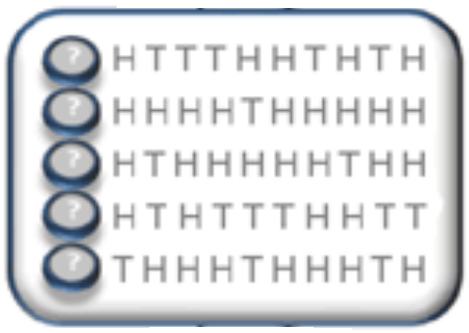
- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

- Two steps for each iteration
  - **Expectation:** Calculate the likelihood of the data given current parameter estimate
  - **Maximization:** Find the parameter values that maximize the likelihood

- Stop when the relative variation of the parameter estimates is smaller than some value

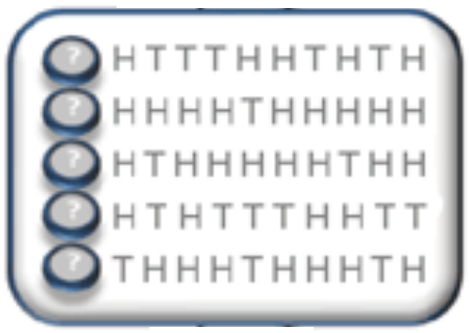
# Expectation Maximization

Nature BioTech 26, 897 (2008)



# Expectation Maximization

Nature BioTech 26, 897 (2008)

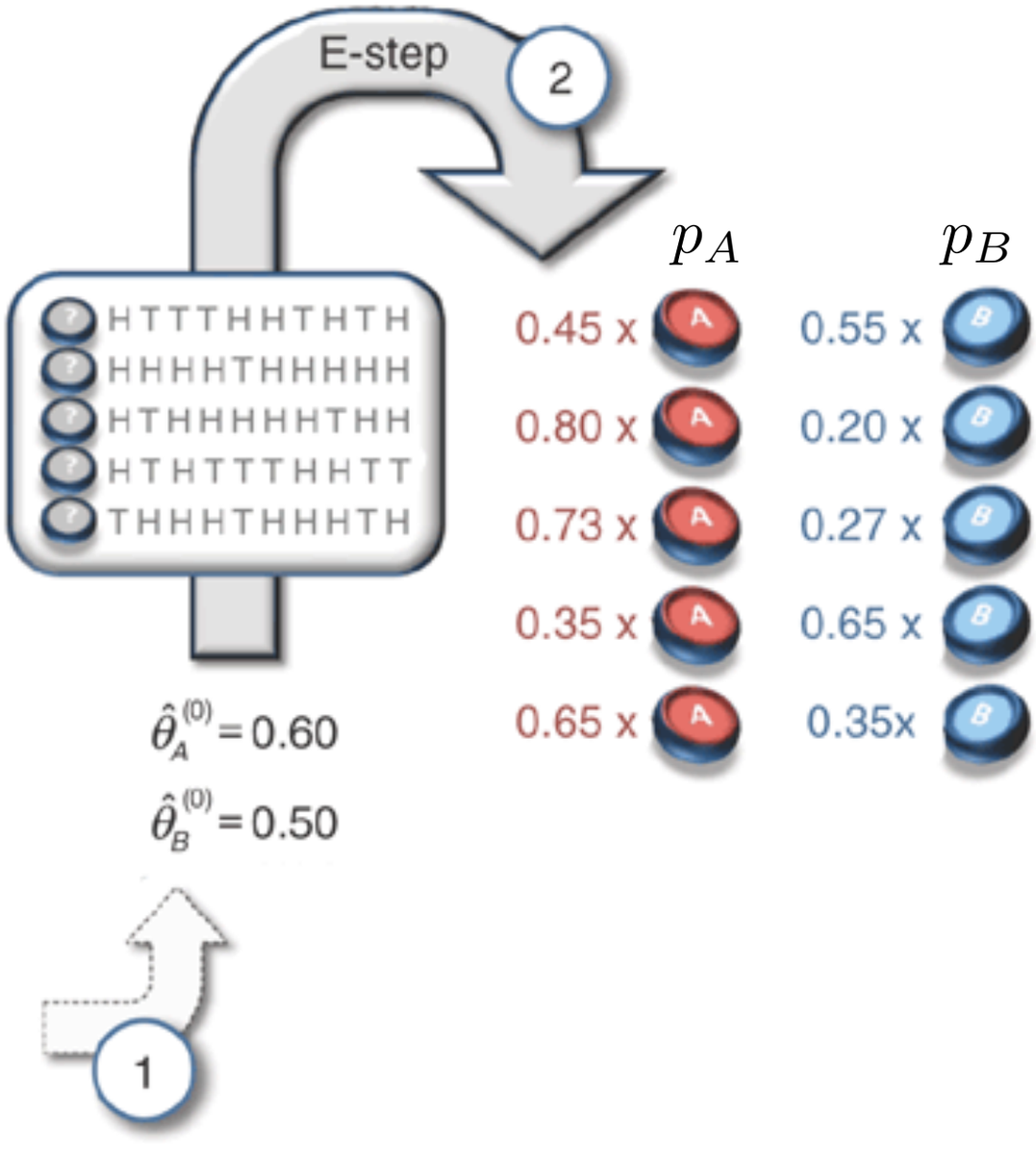



$\hat{\theta}_A^{(0)} = 0.60$

$\hat{\theta}_B^{(0)} = 0.50$

# Expectation Maximization

Nature BioTech 26, 897 (2008)

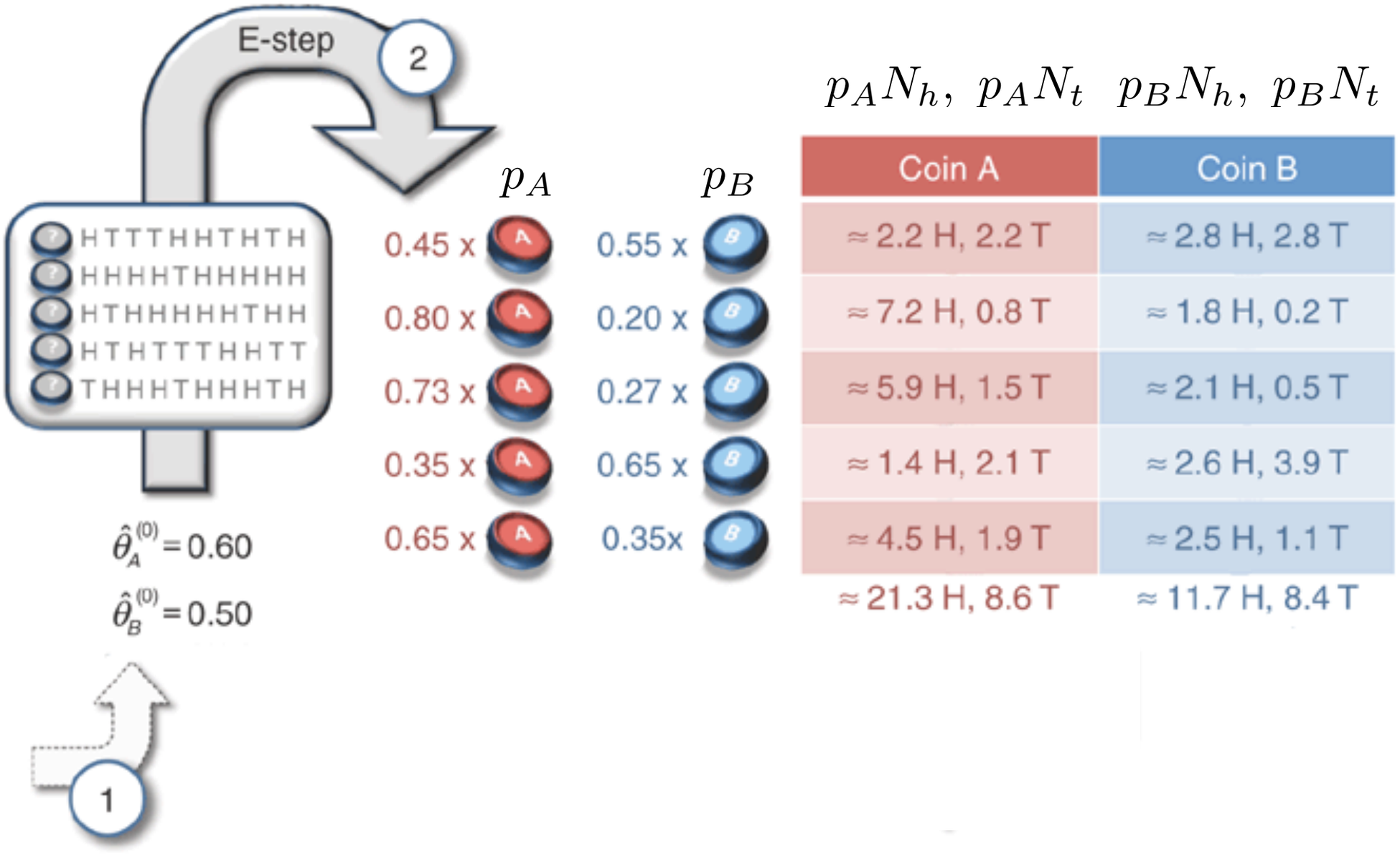



*@bgoncalves*

$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

# Expectation Maximization

Nature BioTech 26, 897 (2008)

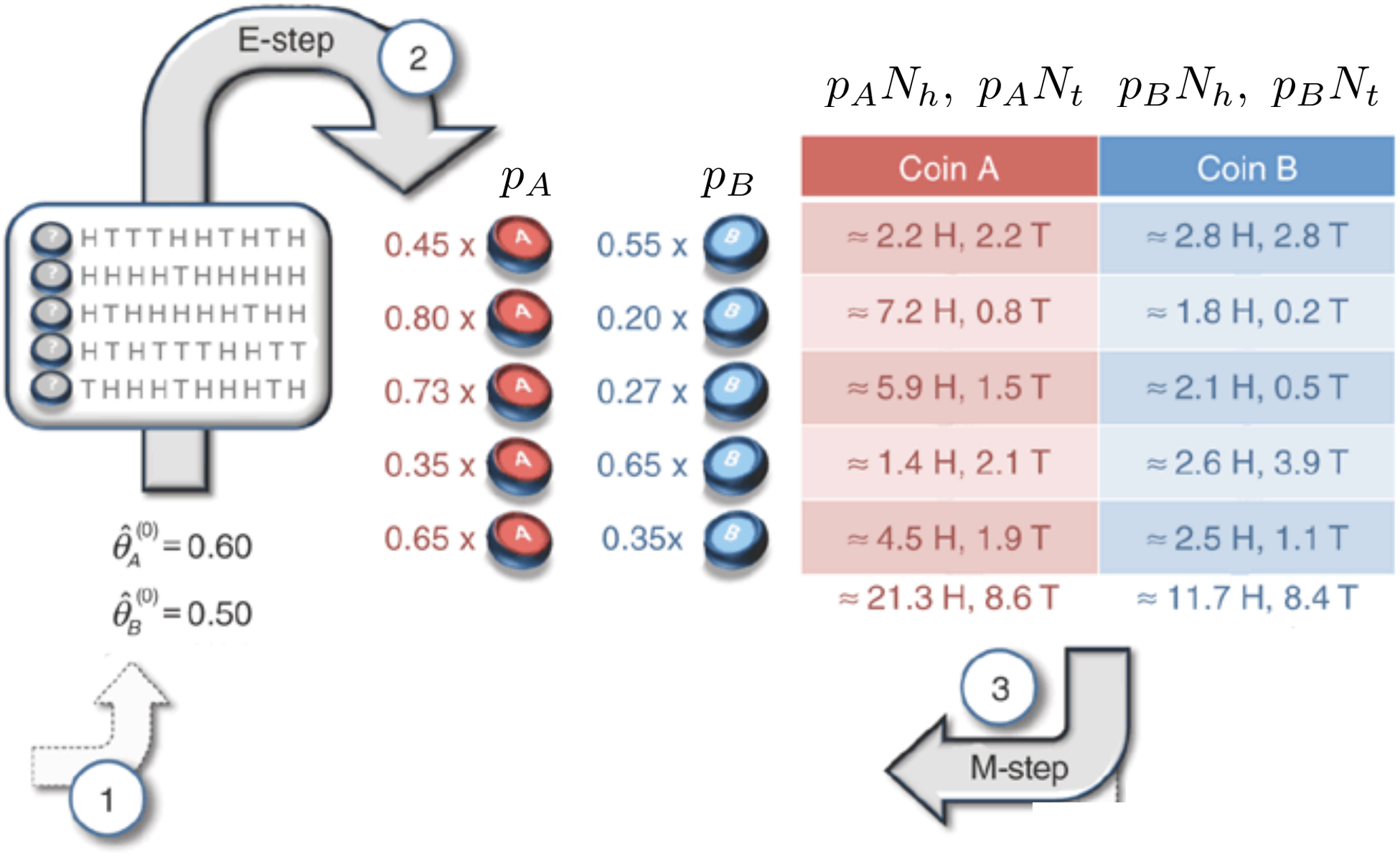



*@bgoncalves*

$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

# Expectation Maximization

Nature BioTech 26, 897 (2008)

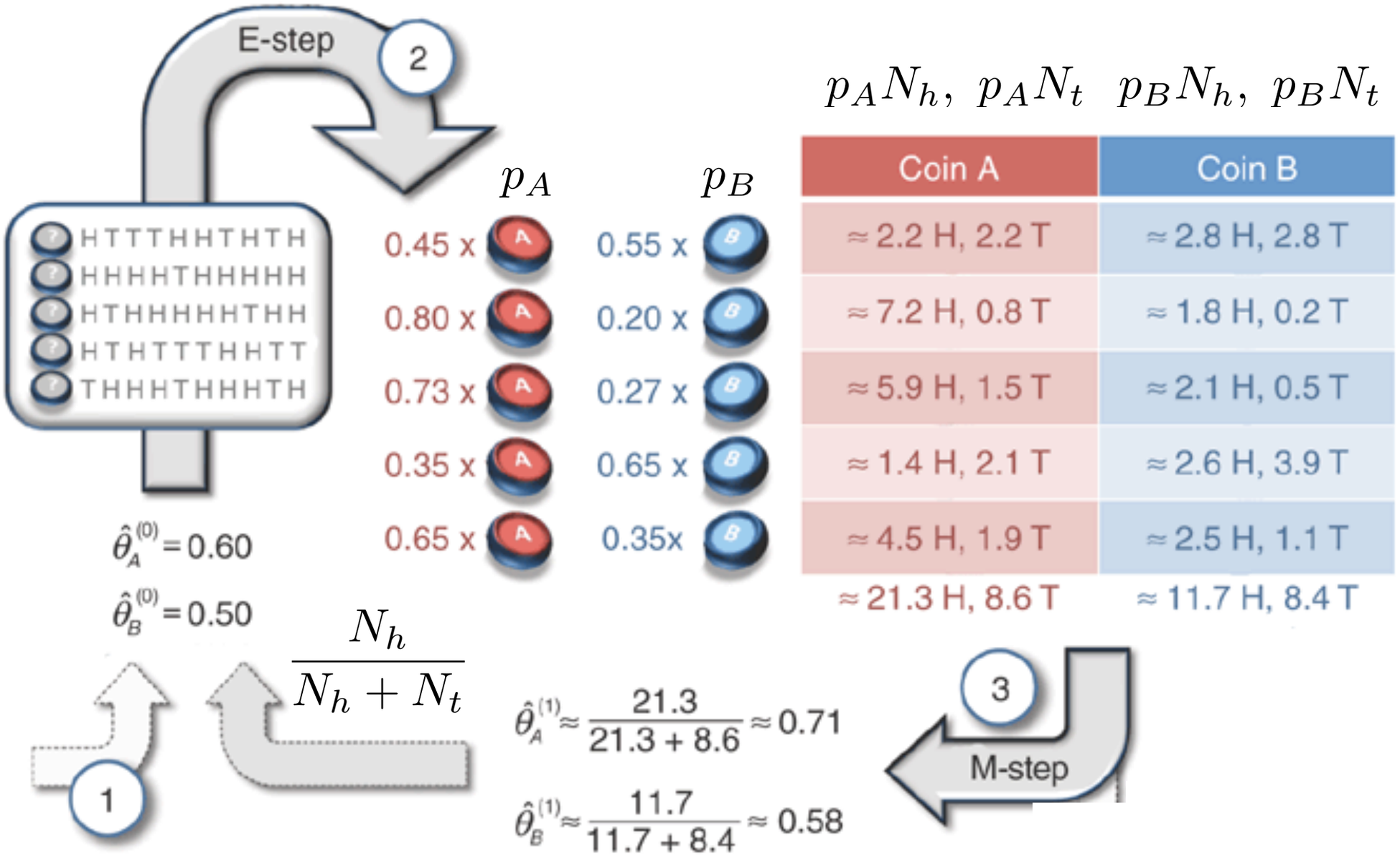



*@bgoncalves*

$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

# Expectation Maximization

Nature BioTech 26, 897 (2008)

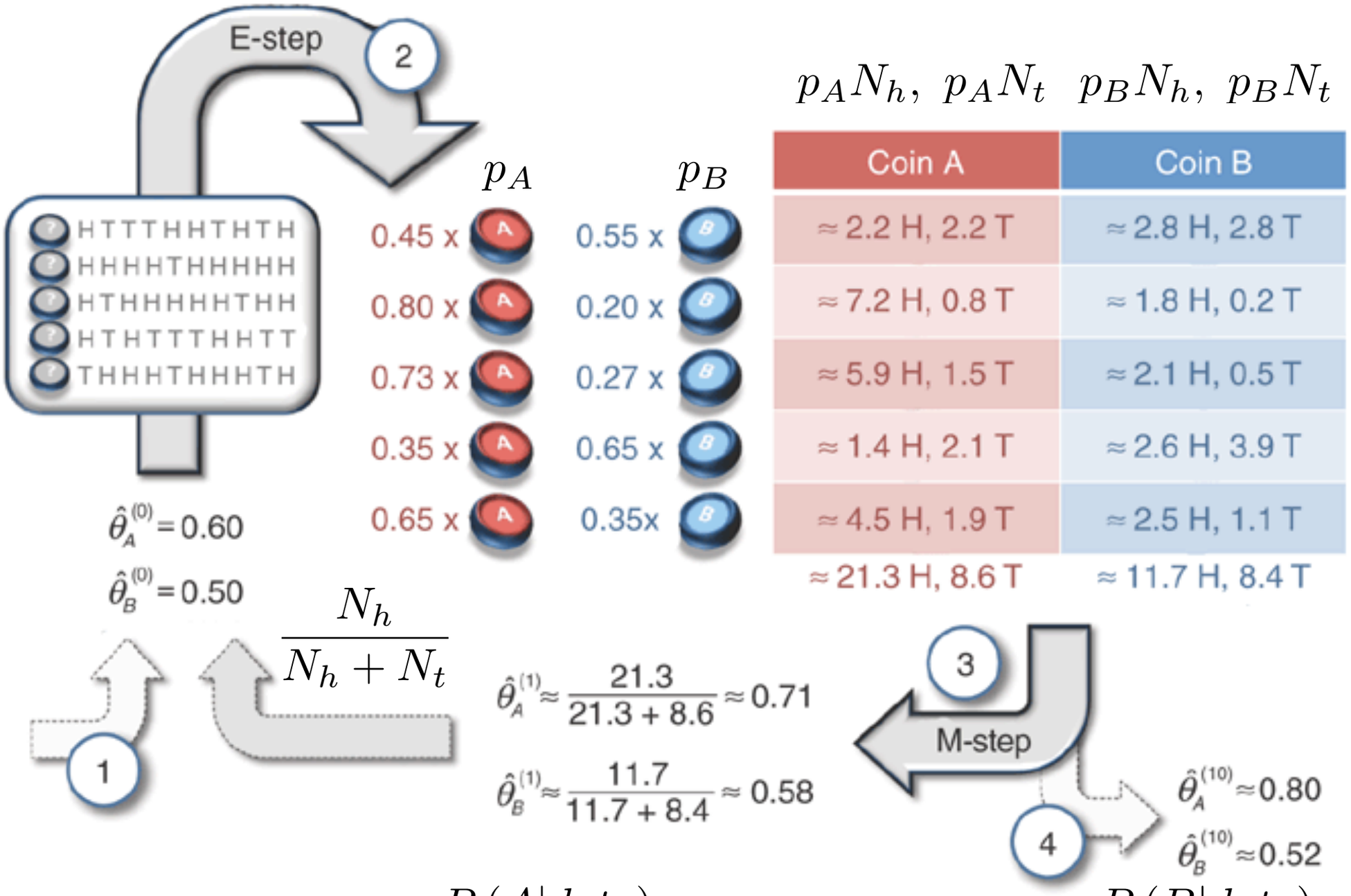



*@bgoncalves*

$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

# Expectation Maximization

Nature BioTech 26, 897 (2008)




*@bgoncalves*

$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

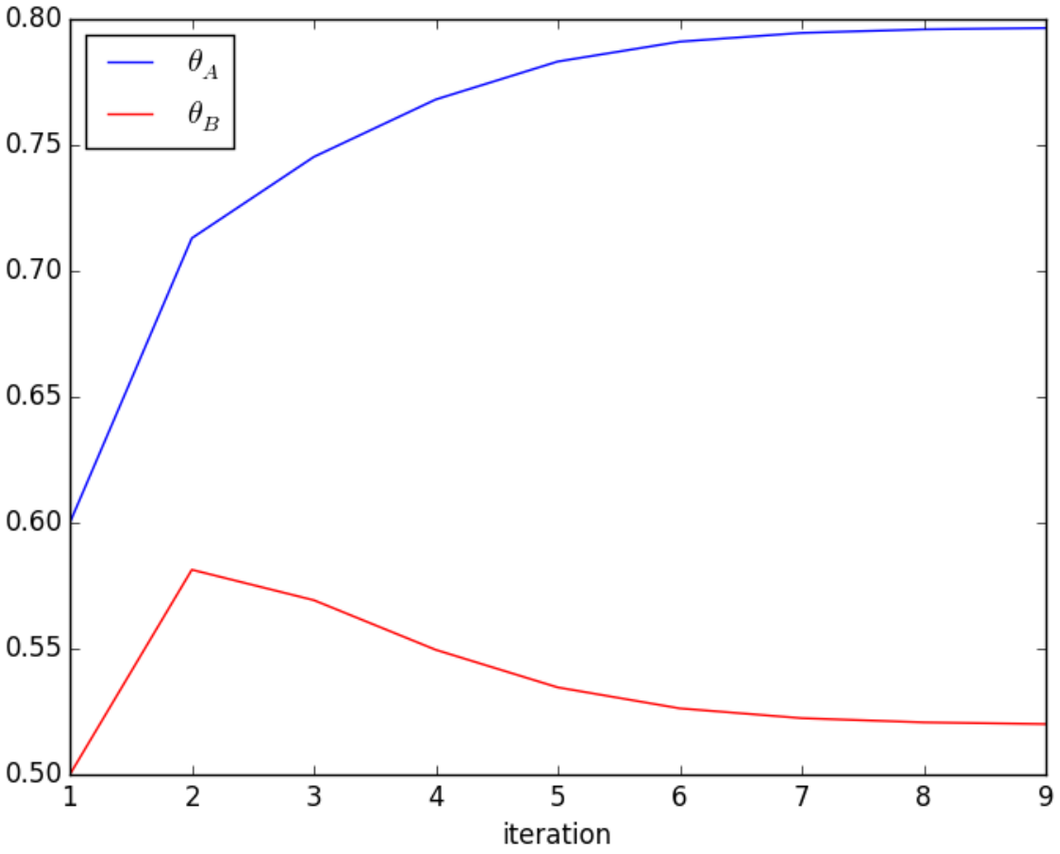
# Expectation Maximization

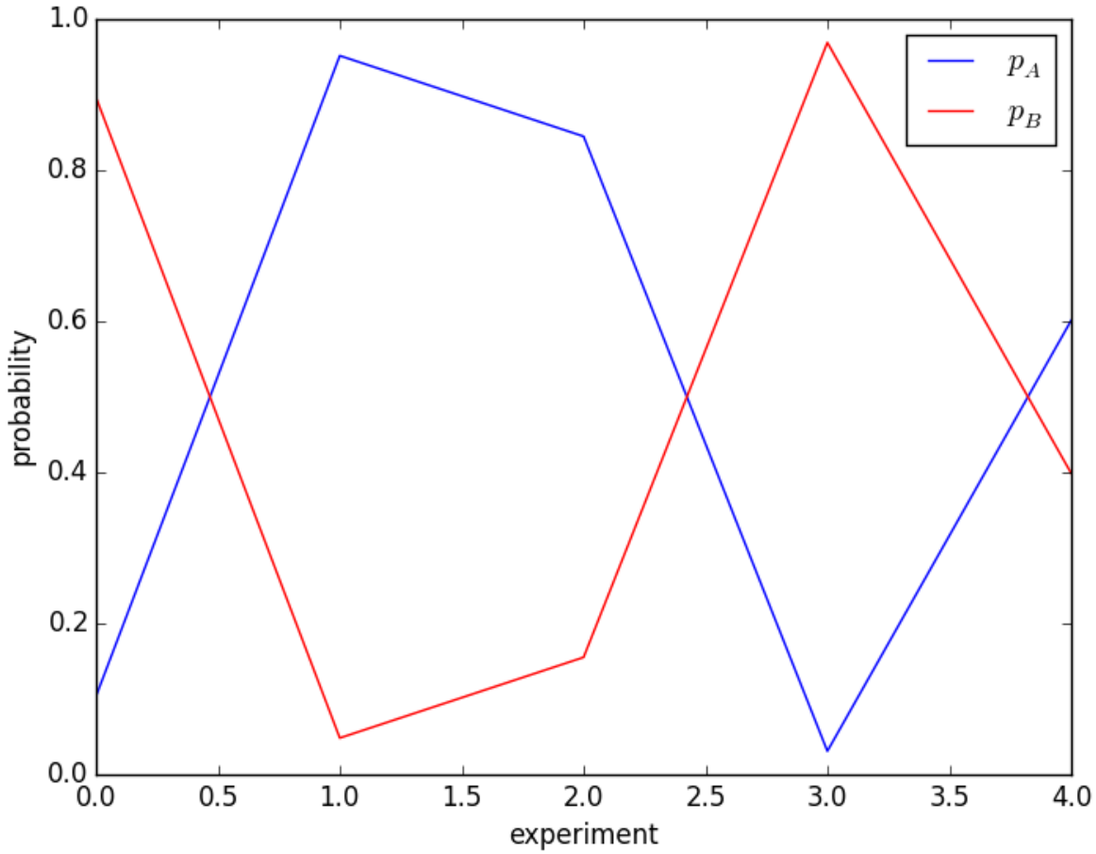
```
while (improvement > delta):
    expectation_A = np.zeros((5, 2), dtype=float)
    expectation_B = np.zeros((5, 2), dtype=float)

    for i in range(0, len(experiments)):
        e = experiments[i]  # i'th experiment
        ll_A = get_mn_likelihood(e, np.array([tA[-1], 1-tA[-1]]))
        ll_B = get_mn_likelihood(e, np.array([tB[-1], 1-tB[-1]]))

        weightA = ll_A/(ll_A + ll_B)
        weightB = ll_B/(ll_A + ll_B)

        expectation_A[i] = np.dot(weightA, e)
        expectation_B[i] = np.dot(weightB, e)

    tA.append(sum(expectation_A)[0] / sum(sum(expectation_A)))
    tB.append(sum(expectation_B)[0] / sum(sum(expectation_B)))

    improvement = max(abs(np.array([tA[-1], tB[-1]]) - np.array([tA[-2], tB[-2]])))
```



EM.py

# Expectation Maximization



# Principle Component Analysis

*@bgoncalves*

# Principle Component Analysis

- Finds the directions of maximum variance of the dataset

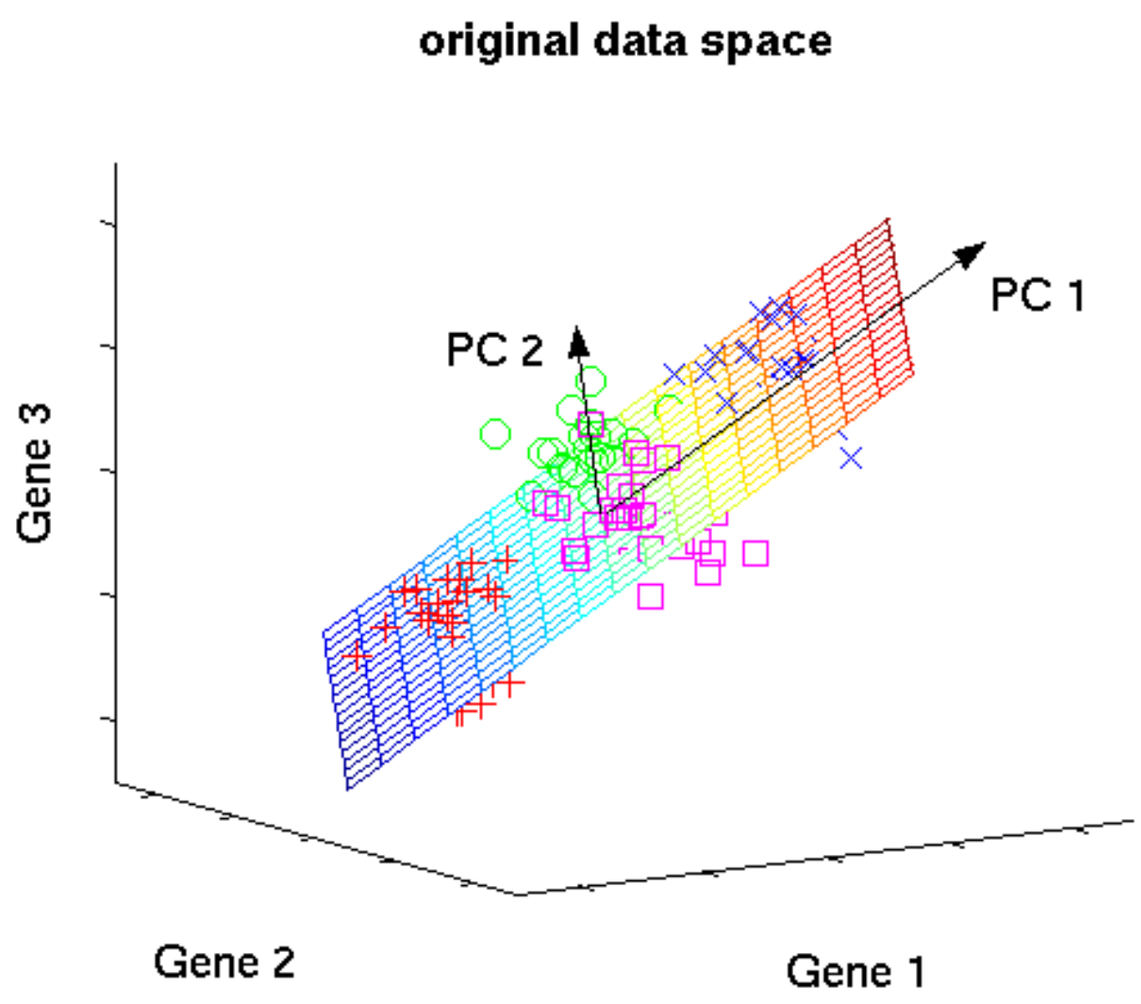
# Principle Component Analysis

- Finds the directions of maximum variance of the dataset

- Useful for dimensionality reduction

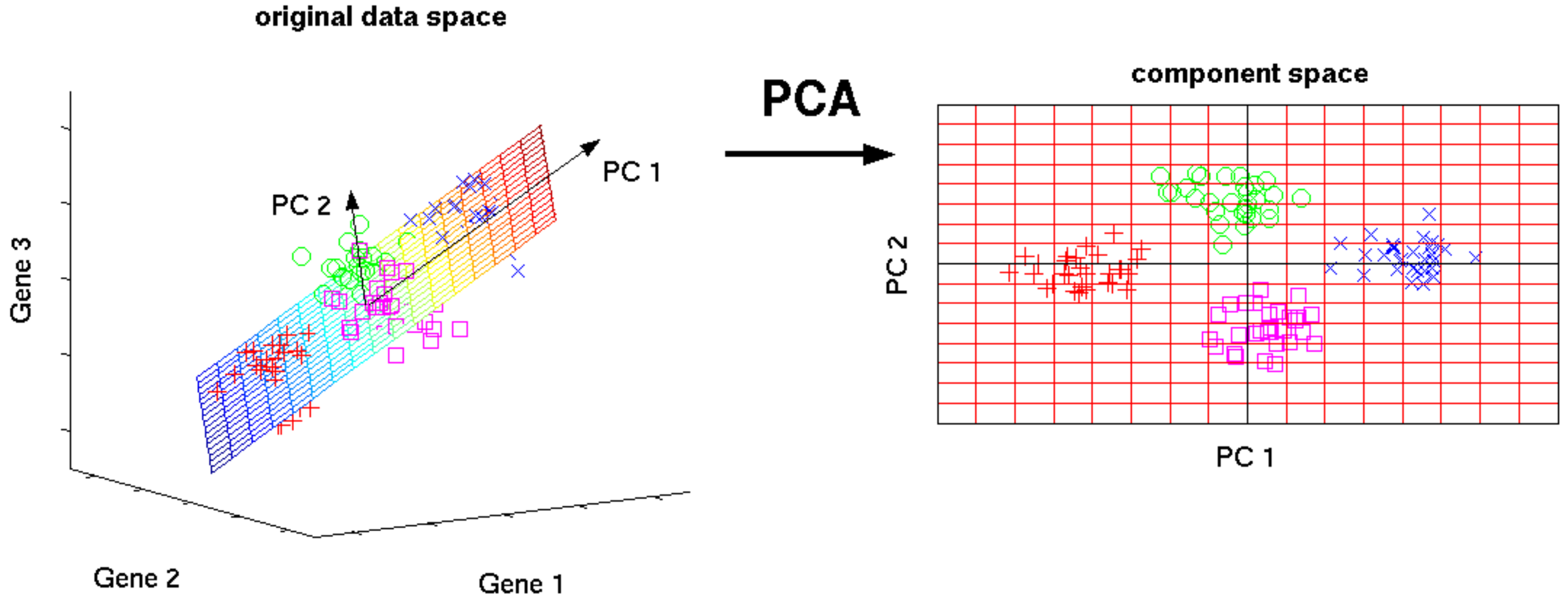
# Principle Component Analysis

- Finds the directions of maximum variance of the dataset
- Useful for dimensionality reduction
- Often used as preprocessing of the dataset



# Principle Component Analysis

- Finds the directions of maximum variance of the dataset
- Useful for dimensionality reduction
- Often used as preprocessing of the dataset



# Principle Component Analysis

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

$$T = AW$$

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

- and corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$

Generalization of Eigenvalue/ Eigenvector decomposition for non-square matrices.

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

- and corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$

Generalization of Eigenvalue/ Eigenvector decomposition for non-square matrices.

- So we can write:

$$T = U\Sigma W^T W \equiv U\Sigma$$

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

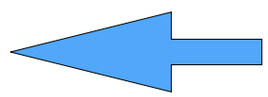
$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

- and corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$



Generalization of Eigenvalue/ Eigenvector decomposition for non-square matrices.

- So we can write:

$$T = U\Sigma W^T W \equiv U\Sigma$$

- Showing that the Principle Component projection corresponds to the left singular vectors of A scaled by the respective singular values

# Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

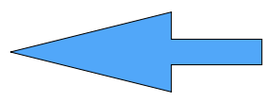
$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

- and corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$



Generalization of Eigenvalue/ Eigenvector decomposition for non-square matrices.

- So we can write:

$$T = U\Sigma W^T W \equiv U\Sigma$$

- Showing that the Principle Component projection corresponds to the left singular vectors of A scaled by the respective singular values $\Sigma$

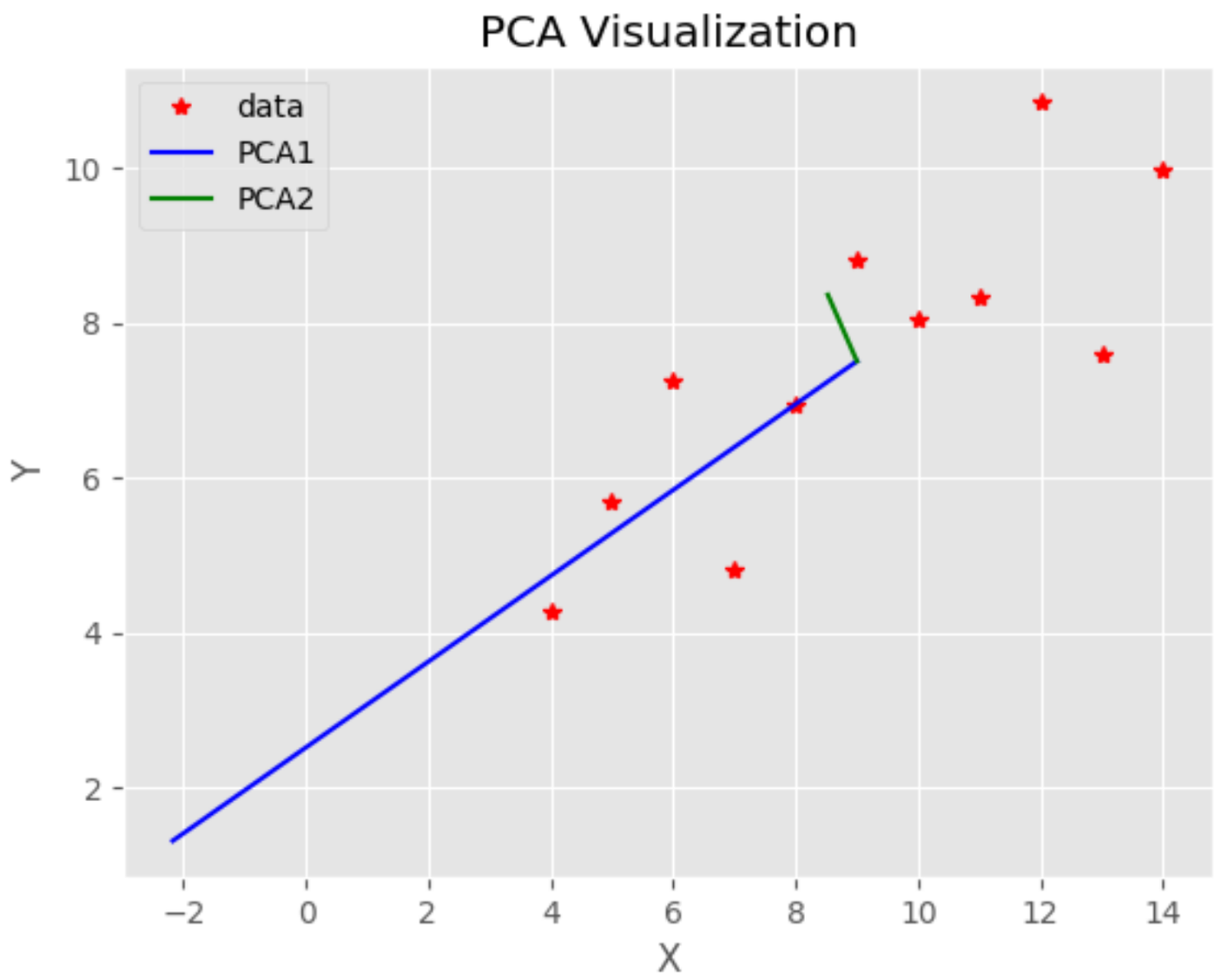
- Columns of T are ordered in order of decreasing variance.

Princ

```python
import sys
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt(sys.argv[1])

x = data.T[0]
y = data.T[1]

pca = PCA()
pca.fit(data)

meanX = np.mean(x)
meanY = np.mean(y)

plt.style.use('ggplot')
plt.plot(x, y, 'r*')

plt.plot([meanX, meanX+pca.components_[0][0]*pca.explained_variance_[0]],
         [meanY, meanY+pca.components_[0][1]*pca.explained_variance_[0]], 'b-')
plt.plot([meanX, meanX+pca.components_[1][0]*pca.explained_variance_[1]],
         [meanY, meanY+pca.components_[1][1]*pca.explained_variance_[1]], 'g-')
plt.title('PCA Visualization')
plt.legend(['data', 'PCA1', 'PCA2'], loc=2)
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('PCA.png')
plt.close()

transform = pca.transform(data)
plt.plot(transform.T[0], transform.T[1], 'r*')
plt.title('PCA Transform Visualization')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.savefig('PCATransform.png')
plt.close()
```

PCA.py

# Principle Component Analysis



# Principle Component Analysis