http://github.com/bmtgoncalves/NLPGotham

Natural Language Processing From Scratch

Bruno Gonçalves
www.bgoncalves.com

Teaching machines to read!

- How can computers represent, analyze and understand a piece of text?
- Computers are really good at crunching numbers but not so much when it comes to words.
- Perhaps can we substitute words by numbers?
    - Unfortunately, computers assume that numbers are sequential.
- Vectors work much better!
    - Each word corresponds to a unique dimension.

$$v_{after} = (0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \cdots)^T$$

One-hot encoding

1	a
2	about
3	above
4	after
5	again
6	against
7	all
8	am
9	an
10	and
11	any
12	are
13	aren't
14	as
…	…

Bag of Words

$$v_{after} = (0,0,0,1,0,0,\cdots)^T$$
$$v_{above} = (0,0,1,0,0,0,\cdots)^T$$

- What about full texts instead of single words?

- The vector representation of a text is simply the vector sum of all the words it contains:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

$$v_{text} = (2,4,1,2,2,1,1,2,6,1,5,1,2,1,1,1,1,4,1)^T$$

0	had	10	lamb
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whose
5	to	15	go
6	snow	16	the
7	everywhere	17	little
8	mary	18	white
9	fleece		

Bag of Words

$$v_{after} = (0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \cdots)^T$$

- What about full texts instead of single words?
- The vector representation of a text is simply the vector sum of all the words it contains:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

0	had	10	lamb
1	went	11	as
2	and	12	that
3	a	13	sure
4	was	14	whose
5	to	15	go
6	snow	16	the
7	everywhere	17	little
8	mary	18	white
9	fleece		

Bag of Words

$$v_{after} = (0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \cdots)^T$$

- What about full texts instead of single words?

- The vector representation of a text is simply the vector sum of all the words it contains:

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

$$v_{text} = (2, 4, 1, 2, 2, 1, 1, 2, 6, 1, 5, 1, 2, 1, 1, 1, 1, 4, 1)^T$$

had	2	lamb	5
went	4	as	1
and	1	that	2
a	2	sure	1
was	2	whose	1
to	1	go	1
snow	1	the	1
everywhere	2	little	4
mary	6	white	1
fleece	1		

- In practice it's much more convenient to use a dictionary instead of an actual vector

- This is known as a Bag of Words, and word order is discarded.

Bag of Words

```
import re
from collections import Counter

word_regex = re.compile(r"\w+", re.U)

text = """Mary had a little lamb, little lamb,
    little lamb, Mary had a little lamb
    whose fleece was white as snow.
    And everywhere that Mary went
    Mary went, Mary went, everywhere
    that Mary went
    The lamb was sure to go."""

# extract all words from the text, ignoring punctuation and case
words = word_regex.findall(text.lower())

# Count how many times each word appears
counts = Counter(words) # This is essentially our "bag of words"

items = list(counts.items())

# Extract word dictionary and vector representation
word_dict = dict([[items[i][0], i] for i in range(len(items))])
text_vector = [items[i][1] for i in range(len(items))]

print(text_vector)
print(word_dict)
```

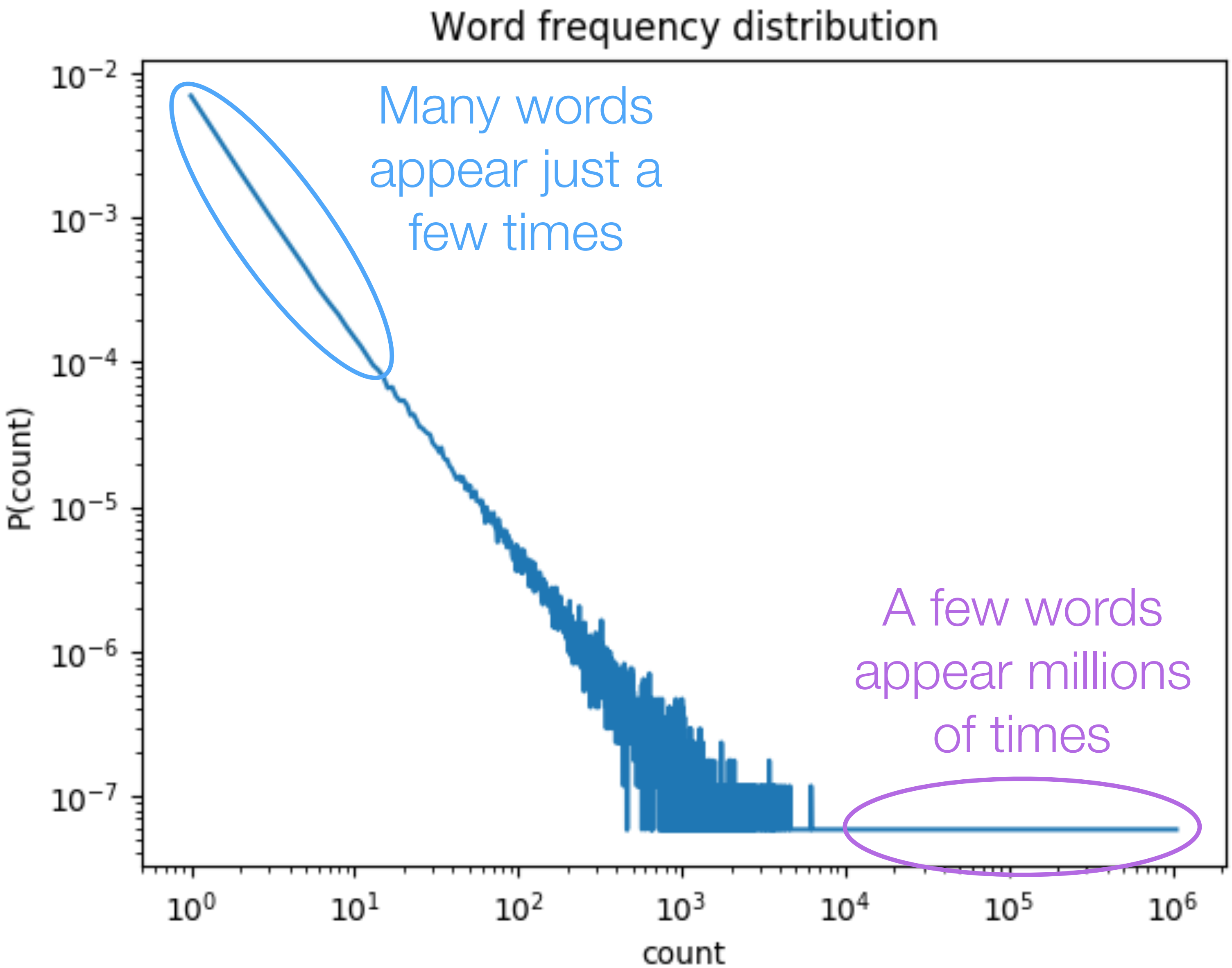
@bgoncalves

bag_of_words.py

Stopwords

- Some words are much more common than others.
- While most words are very rare.
- The most common words in a corpus of 17M words:
- These are known as "stopwords" words that carry little meaning and can be discarded.
- After removing the most common words we go from 17M to 9M.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644



Stop

```python
from collections import Counter
import gzip
import matplotlib.pyplot as plt
import numpy as np

data = []

for line in gzip.open("text8.gz"):
    data.extend(line.strip().split())

counts = Counter(data)

sorted_counts = sorted(list(counts.items()), key=lambda x:x[1], reverse=True)

for word, count in sorted_counts[:100]:
    print(word, count)


dist = Counter(counts.values())
dist = list(dist.items())
dist.sort(key=lambda x:x[0])
dist = np.array(dist)

norm = np.dot(dist.T[0], dist.T[1])

plt.loglog(dist.T[0], dist.T[1]/norm)
plt.xlabel("count")
plt.ylabel("P(count)")
plt.title("Word frequency distribution")
plt.savefig("freq.png")

stopwords = set([word for word, count in sorted_counts[:100]])

clean_data = []

for word in data:
    if word not in stopwords:
        clean_data.append(word)

print("Original size:", len(data))
print("Clean size:", len(clean_data))
```

most_common_words.py

Stopwords

- In practice, stopwords aren't simply the most common words but rather curated lists of common and non-informative words.
- Computational Linguists have published lists of stop words that can easily be found online, and that were curated for different languages and purposes.
- Stopwords in 40 languages: https://www.ranks.nl/stopwords
- NLTK also includes stopwords from the 14 languages listed here: http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/ plus Romanian (http://arlc.ro/resources/) and Kasakh.

TF-IDF

- We already saw that some words are much more common than others

- The number of times that a word appears in a document is known as the "Term Frequency" (TF)

- After the removal of stopwords, the Term Frequency is a good indicator of what words are most important. A book on Python Programming will likely have words like "code", "script", "print", "error", etc much more frequently than a book on Football.

- TF gives us an idea of how popular a specific term is within a document, but how can we compare across documents within a corpus?

- The Inverse Document Frequency (IDF) tells us how unusual it is for a document to include that word. The idea is that words that appear in more documents are less meaningful.

the	1061396
of	593677
and	416629
one	411764
in	372201
a	325873
to	316376
zero	264975
nine	250430
two	192644

TF-IDF

- Mathematically there are several possible definitions for both TF and IDF

TF	IDF
$N_{t,d}$	$\log\left(\frac{N}{N_t}\right)$
$\frac{N_{t,d}}{\sum_{t'} N_{t',d}}$	$\log\left(1+\frac{N}{N_t}\right)$
$1+\log\left(N_{t,d}\right)$	

Number of time term t occurs in document d

Number of documents

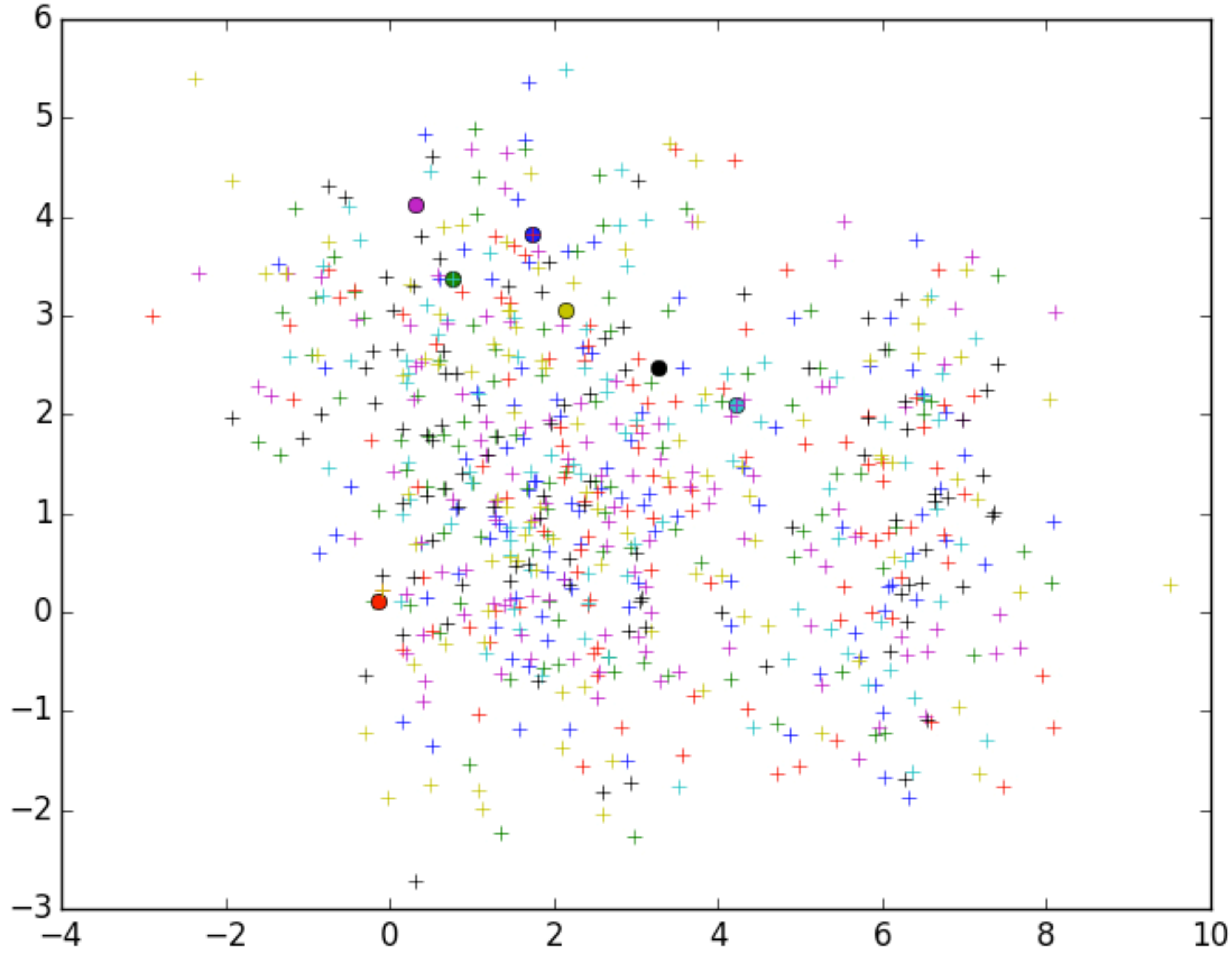
Number of documents in which term t appears

- TF-IDF is the product of these two quantities and is useful to find terms that are important for the specific document (high TF) and uncommon in the corpus as a whole (large IDF/ small DF).
- In particular, a term that occurs in every document is meaningless when it comes to distinguish between documents.
- Stopwords, are naturally weighed down due to appearing in all documents.

Document Clustering

- Documents represented by their TF-IDF vectors can be grouped using an unsupervised clustering algorithm
- K-Means is a popular choice:
    - 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

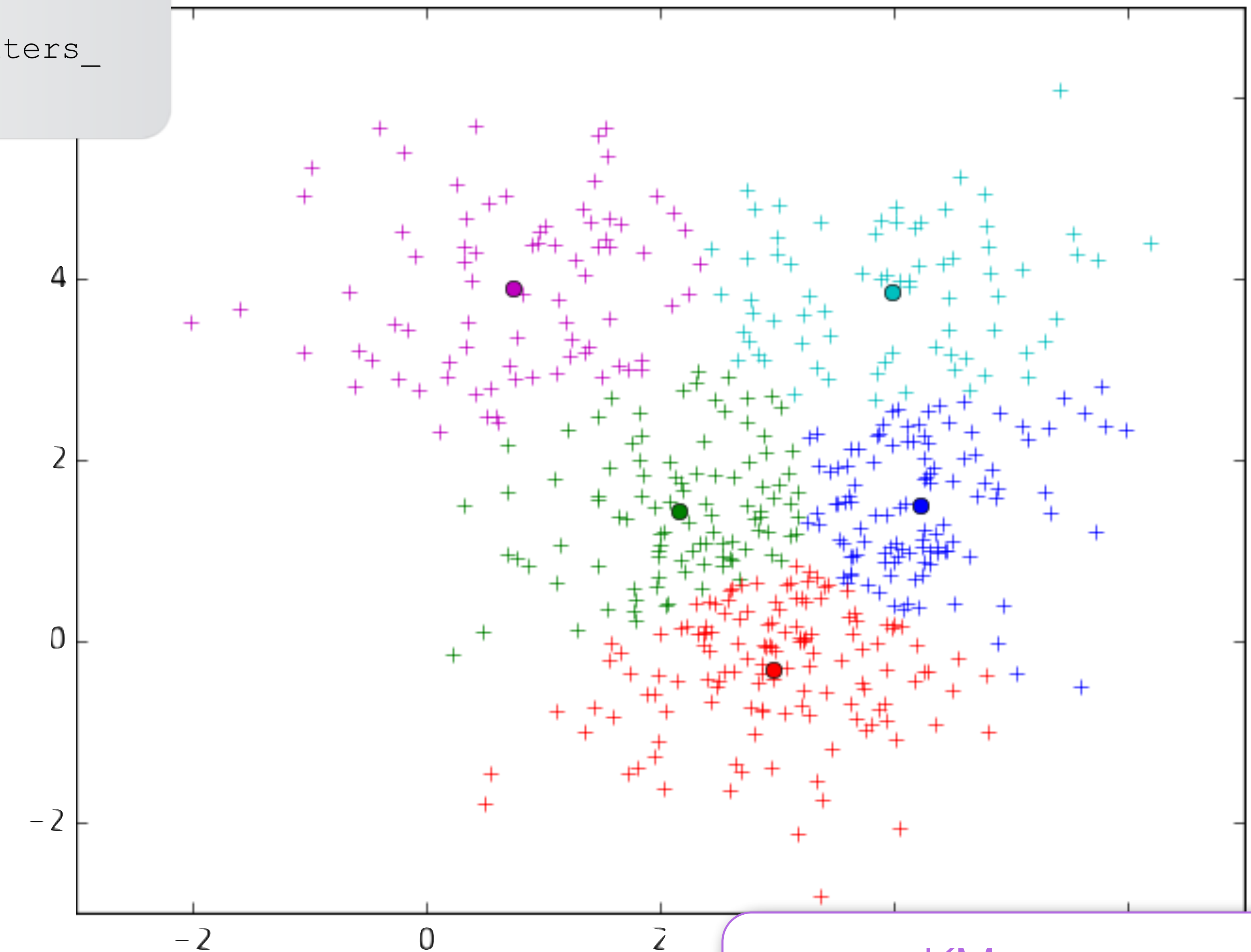
K-Means



http://youtu.be/M0sCjfRP32k

K-Means: sklearn

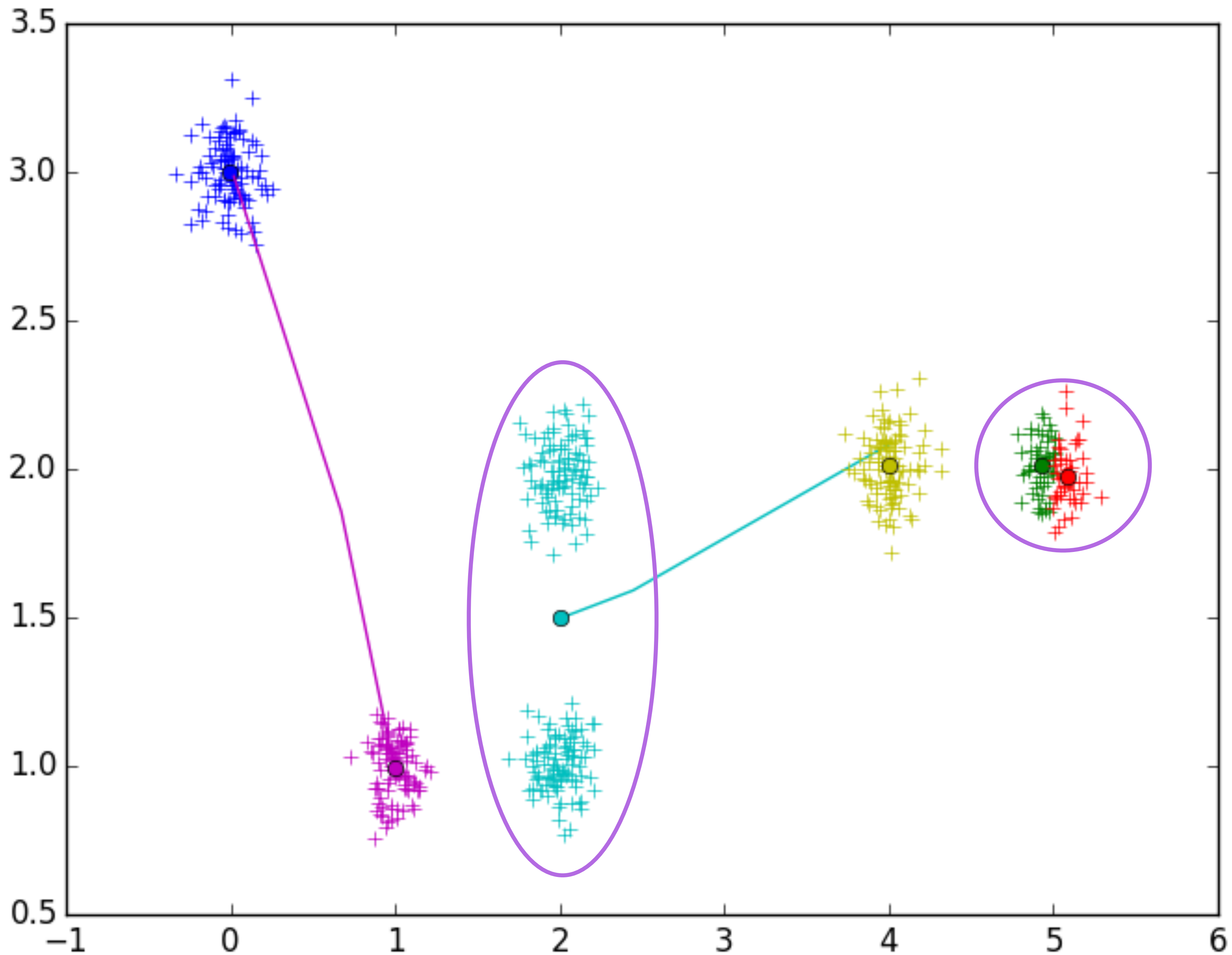
```
from sklearn.cluster import KMeans

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

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



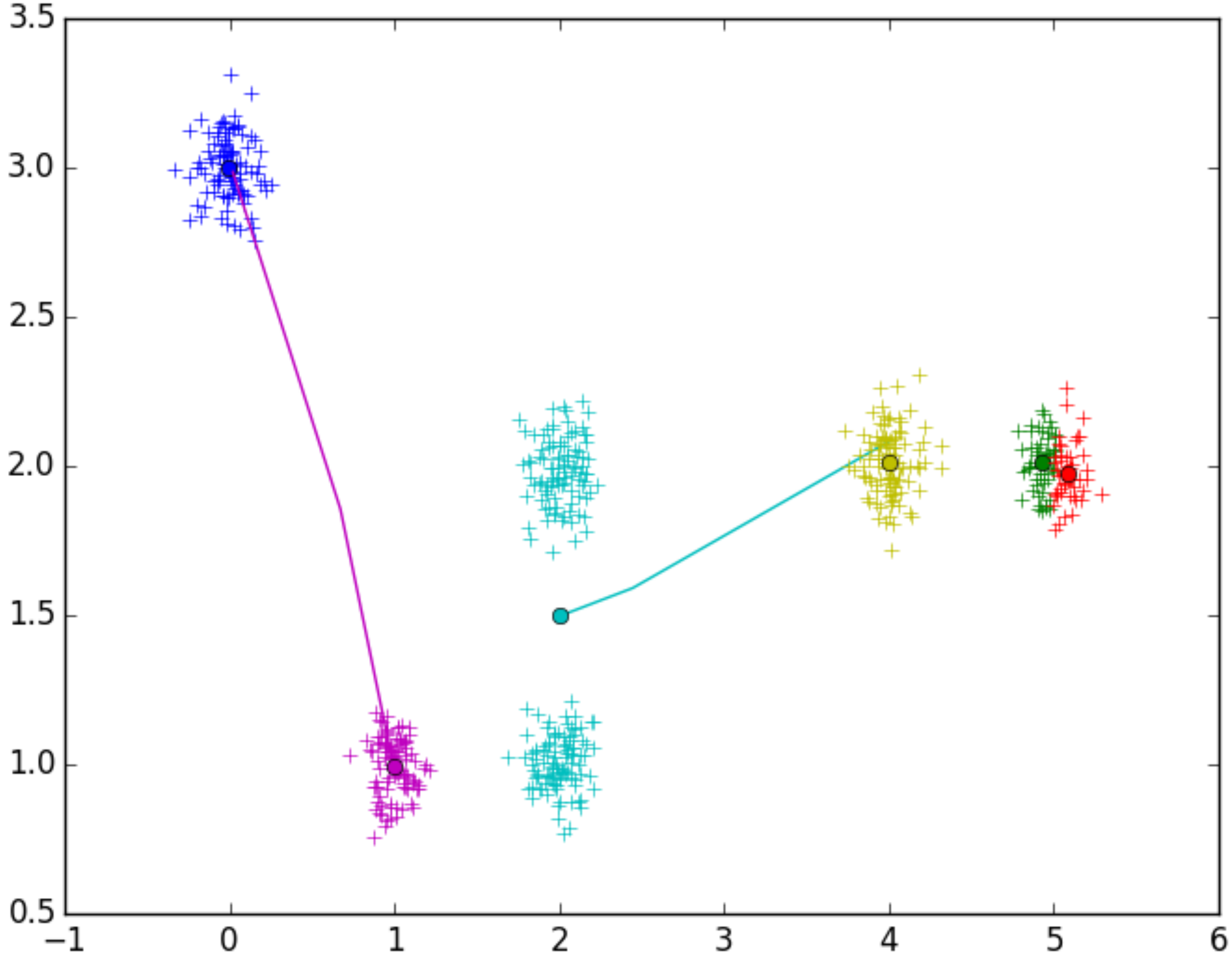
KMeans.py

@bgoncalves

K-Means: Limitations



K-Means: Limitations



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