NYC DATA SCIENCE ACADEMY

# Introduction to Natural Language Processing

Data Science Bootcamp

# Outline

- **Background**
  - Regular Expressions
  - Working with words
  - Natural Language ToolKit (NLTK)
  - Creating a corpus
  - Parts of Speech
  - Text Classification
    - Sentiment Analysis of Tweets
  - Latent Dirichlet Allocation

# What is NLP?

Natural Language Processing

Computational Linguistics

Speech Recognition

Computational Psycholinguistics

# Foundation of Linguistics

| | |
|---|---|
| Phonetics and Phonology | The study of sounds |
| Morphology | The study of the meaningful components of words |
| Syntax | The study of the structural relationship between words |
| Semantics | The study of meaning |
| Pragmatics | The study of how language is used to accomplish goals |
| Discourse | The study of linguistic units larger than a single utterance |

# Challenges of Natural Language: Ambiguity

All of the models and algorithms in NLP are aiming at reducing ambiguity.

Examples:

- POS tagging
- Probabilistic Parsing

# Challenges of Natural Language: How do we reduce ambiguity?

- Text Normalization
  - We will talk more about how to do this today, but first...

# Outline

- Background
- **Regular Expressions**
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- Parts of Speech
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# What is regular expression?

The standard notation for characterizing text sequences.

# Regular Expressions: Basics

| | | |
|---|---|---|
| [Pp]erson | | Person, person |
| [0123456789] | | Any digit |
| [A-Z] | | Any uppercase letter |
| [a-z] | | Any lowercase letter |
| [0-9] | | Any single digit |
| [^1] | | Negation: Not a one |
| /a/ | matches any “a” but not “A” | My name is Andrew. |
| /name␣is/ | | My name is Andrew. |

# Regular Expressions: ? * + .

| | | |
|---|---|---|
| colou?r | Optional previous character | color, colour |
| oo*h! | 0 or more of previous character | oh!, ooh!, oooh!, ooooh!, etc… |
| o+h! | 1 or more of previous character | oh!, ooh!, oooh!, ooooh!, etc… |
| baa+ | | baa, baaa, baaaa, etc... |
| beg.n | | begin, begun, beg3n, beg'n, etc... |

# Regular Expressions: Anchors

Unlike most regular expression notation, anchors specify a position rather than a pattern

| | |
|---|---|
| \.$ | The end. |
| ^[A-Z] | New York |

# Example

Find all instances of “an” in a text.

| | |
|---|---|
| 'an ' | Misses caps |
| ' \b[Aa]n\b' | What we want |
| '[Aa]n' | Also returns words like “canal” and “anova” |

What types of error are we trying to fix?

# What types of error are we trying to fix?

Type I Error (False Positive)

Matching strings we don't want : "canal" and "anova"

Type II Error (False Negative)

Not matching the ones we want : when we miss caps

# Outline

- Background
- Regular Expressions
- **Working with words**
- Natural Language ToolKit (NLTK)
- Creating a corpus
- Parts of Speech
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# How many words?

So, uh, yeah. I think we coul-, no, should do this.

- Fragments, pauses, fillers

That boy is not like the other boys.

- lemma - same stem
  - boy and boys
- Wordform - inflections
  - boy and boys

NOTE: The term tokenization can be used as a synonym for **text segmentation**.

However, in **lexical analysis**, tokenization refers to breaking up a stream of text into meaningful units (usually words or **clitics**). In this case, it is a specific type of text segmentation.

# Common Tokenization Problems

- boy boys boy's
- Uppercase upper case upper-case
- I'm you're what're should've
- state-of-the-art
- New York City (one token or three)
- M.B.A.
- New York-based (where should we split?)

# Normalization

- N.Y.C. and NYC and New York City and New York
- can’t and cannot

Powerful but not very efficient!

# Common Normalization Methods

- Case Folding
- Lemmatization

# Lemmatization

- am, are, is, was, been → be
- book, books, book's, books' → book

The books' bindings are beautiful. → *The book binding be beautiful.*

Very useful for translation algorithms!

# Stemming

In morphology:

- Stems ⟶ Base meaning
- Affixes ⟶ Modifier

produc = produce, produces, production, productive, productively

# Porter Stemmer

The Porter algorithm is one of the most common stemmers in English. There are other stemmers which can be used but they will vary depending on the end result desired. No stemming algorithm is perfect and all of them will present both Type I and Type II errors.

*For the purposes, of this class we will be using this algorithm.

To see more details about how this algorithm stems:

http://snowball.tartarus.org/algorithms/porter/stemmer.html

Alternative stemming algorithms include: Snowball, Lancaster, ISRI, Regexp, etc…

# Outline

- Background
- Regular Expressions
- Working with words
- **Natural Language ToolKit (NLTK)**
- Creating a corpus
- Parts of Speech
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# Prepare Our Environment

```
from bs4 import BeautifulSoup
# Python 2 and 3
try:
    from urllib.request import urlopen
except ImportError:
    from urllib2 import urlopen
import re
import nltk
```

# NLTK

```
In [36]: nltk.download('all')
```

```
[nltk_data] Downloading collection u'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /Users/andrew/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package alpino to
[nltk_data]    |     /Users/andrew/nltk_data...
[nltk_data]    |   Unzipping corpora/alpino.zip.
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     /Users/andrew/nltk_data...
[nltk_data]    |   Unzipping corpora/biocreative_ppi.zip.
[nltk_data]    | Downloading package brown to
[nltk_data]    |     /Users/andrew/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
```

# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- **Creating a corpus**
- Parts of Speech
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# Project Gutenberg - The Great Gatsby

http://gutenberg.net.au/ebooks02/0200041.txt

Ï»¿

**Project Gutenberg Australia**

*a treasure-trove of literature*

treasure found hidden with no evidence of ownership

```
Title:      The Great Gatsby
Author:     F. Scott Fitzgerald
* A Project Gutenberg of Australia eBook *
eBook No.:  0200041.txt
Language:   English
Date first posted:          January 2002
Date most recently updated: July 2008

This eBook was produced by: Colin Choat

Project Gutenberg of Australia eBooks are created from printed editions
which are in the public domain in Australia, unless a copyright notice
is included. We do NOT keep any eBooks in compliance with a particular
paper edition.

Copyright laws are changing all over the world. Be sure to check the
copyright laws for your country before downloading or redistributing this
file.

This eBook is made available at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg of Australia License which may be viewed online at
http://gutenberg.net.au/licence.html

To contact Project Gutenberg of Australia go to http://gutenberg.net.au


Title:      The Great Gatsby
Author:     F. Scott Fitzgerald



Then wear the gold hat, if that will move her;
```

# Creating a Corpus

Scrape the text for the novel, clean it up using regex, and put into a list.

```
url = "http://gutenberg.net.au/ebooks02/0200041.txt"   # URL of book
text = urllib2.urlopen(url).read()                     # Read in the HTML
soup = BeautifulSoup(text, 'html.parser')              # Parse HTML
cleantext = BeautifulSoup.get_text(soup)               # Remove HTML and JS
cleantext = re.sub( '\s+', ' ', cleantext ).strip()    # Remove all whitespace
cleantext = cleantext.lower()                          # All lowercase
cleantext = re.sub( '[.:\',\-!;"()?]', "", cleantext).strip()  # Remove punctuation
corpus = cleantext.split(" ")                          # Tokenize text
corpus
```

# Creating a Corpus

```
Out[142]: [u'\xef\xbb\xbf',
           u'project',
           u'gutenberg',
           u'australia',
           u'a',
           u'treasuretrove',
           u'of',
           u'literature',
           u'treasure',
           u'found',
           u'hidden',
           u'with',
           u'no',
           u'evidence',
           u'of',
           u'ownership',
           u'title',
           u'the',
```

# Creating a Corpus

Remove text at beginning and end of corpus that doesn't belong to the novel.

```
for x in range(0, len(corpus)):
    if corpus[x] == "chapter":
        break_number_1 = x
        break

for x in range((len(corpus)-1), 0, -1):
    if corpus[x] == "end":
        break_number_2 = x + 1
        break

corpus = corpus[break_number_1 : break_number_2]
corpus
```

# Creating a Corpus

```
Out[143]: [u'chapter',
           u'1',
           u'in',
           u'my',
           u'younger',
           u'and',
           u'more',
           u'vulnerable',
           u'years',
           u'my',
           u'father',
           u'gave',
           u'me',
           u'some',
           u'advice',
           u'that',
           u'ive',
           u'been'
```

# Creating a Corpus

```
stemmer = nltk.stem.PorterStemmer()                          # Create our stemmer
stemmed_corpus = [stemmer.stem(word) for word in corpus]     # Apply stemmer
```

```
Out[15]: [u'chapter',
          u'1',
          u'in',
          u'my',
          u'younger',
          u'and',
          u'more',
          u'vulner',
          u'year',
          u'my',
          u'father',
          u'gave',
          u'me',
          u'some',
          u'advic',
          u'that',
          u'ive',
          u'been',
          u'turn',
          u'over',
          u'in',
          u'my',
          u'mind',
          u'ever',
          u'sinc',
          u'whenev',
          u'you',
          u'feel',
          u'like',
```

# Creating a Corpus

Remember that no stemming algorithm is perfect. What are some problems that you see below?

```
to_be_stemmed = ["monkeys","possesses", "possess", "dogs", "eating",
                 "constitutional", "worthy", "relatable", "had", "was", "ties",
                "ive"]
[stemmer.stem(word) for word in to_be_stemmed]
```

```
Out[6]: [u'monkey',
         u'possess',
         u'possess',
         u'dog',
         u'eat',
         u'constitut',
         u'worthi',
         u'relat',
         u'had',
         u'wa',
         u'tie',
         u'ive']
```

# Hands-on Session

- Please go to the **"Exercise: Lemmatization"** in the lecture code.

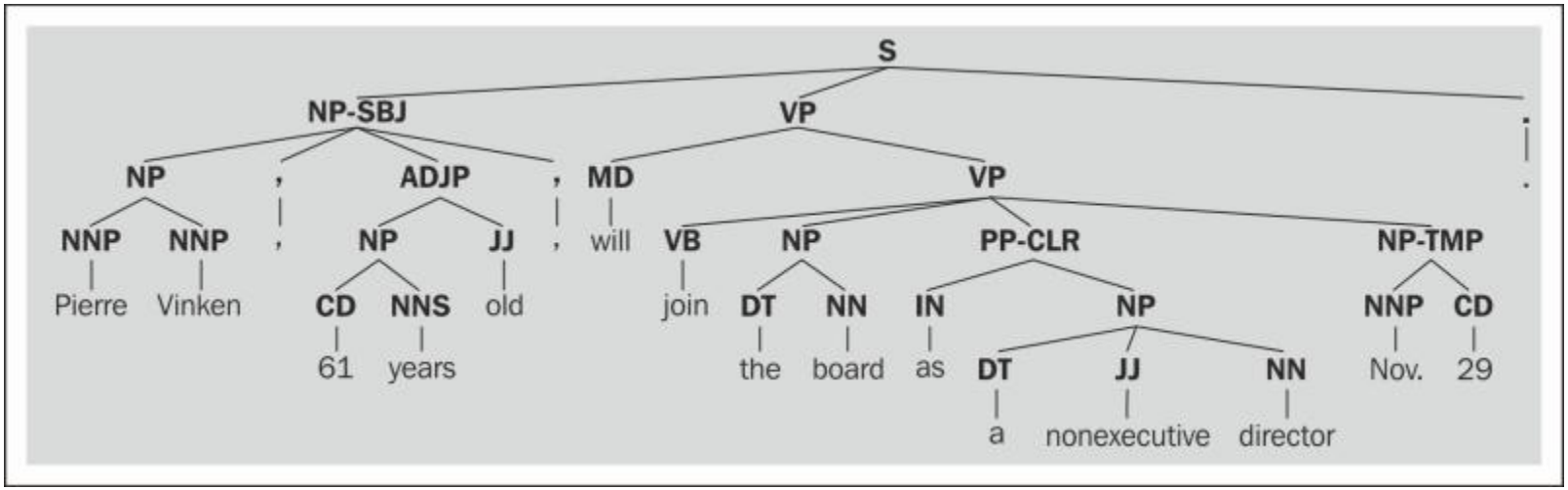
# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- **Parts of Speech**
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# What is a POS tagger?

The syntactic structure of language lends words to perform a specific structural element in a sentence or longer discourse.

In the field of syntax, this inherent structure is broken down using syntax trees.




Source: https://www.safaribooksonline.com/library/view/python-3-text/9781782167853/ch06s10.html

# What problems are inherent when analyzing syntax?

- A word can present itself as various POS in the tree
- Different languages allow for different possibilities in their trees due to differences of information encoded in the lexicon
- Manipulation of syntax by native speakers
- Garden path sentences

# POS

In order to look up definitions for all of the parts of speech, you can run this command.

```
nltk.help.upenn_tagset()
```

Note: There is linguistic vocabulary that is beyond the scope of this class in this help lookup. You can use a book like *Speech and Language Processing* or wikipedia as a good resource for learning more.

# Just so you can get an idea…

| | | |
|---|---|---|
| 1. | CC | Coordinating conjunction |
| 2. | CD | Cardinal number |
| 3. | DT | Determiner |
| 4. | EX | Existential *there* |
| 5. | FW | Foreign word |
| 6. | IN | Preposition or subordinating conjunction |
| 7. | JJ | Adjective |
| 8. | JJR | Adjective, comparative |
| 9. | JJS | Adjective, superlative |
| 10. | LS | List item marker |
| 11. | MD | Modal |
| 12. | NN | Noun, singular or mass |
| 13. | NNS | Noun, plural |
| 14. | NNP | Proper noun, singular |
| 15. | NNPS | Proper noun, plural |
| 16. | PDT | Predeterminer |
| 17. | POS | Possessive ending |
| 18. | PRP | Personal pronoun |
| 19. | PRP$ | Possessive pronoun |
| 20. | RB | Adverb |
| 21. | RBR | Adverb, comparative |
| 22. | RBS | Adverb, superlative |
| 23. | RP | Particle |
| 24. | SYM | Symbol |
| 25. | TO | *to* |
| 26. | UH | Interjection |
| 27. | VB | Verb, base form |
| 28. | VBD | Verb, past tense |
| 29. | VBG | Verb, gerund or present participle |
| 30. | VBN | Verb, past participle |
| 31. | VBP | Verb, non-3rd person singular present |
| 32. | VBZ | Verb, 3rd person singular present |
| 33. | WDT | Wh-determiner |
| 34. | WP | Wh-pronoun |
| 35. | WP$ | Possessive wh-pronoun |
| 36. | WRB | Wh-adverb |

# POS

First tokenize the text.

```
sentence = """At eight o'clock on Thursday morning Arthur didn't feel very good."""
tokens = nltk.word_tokenize(sentence)
tokens
```

```
Out[38]: ['At',
          'eight',
          "o'clock",
          'on',
          'Thursday',
          'morning',
          'Arthur',
          'did',
          "n't",
          'feel',
          'very',
          'good',
          '.']
```

# POS

Now we can tag the tokens.

```
tagged = nltk.pos_tag(tokens)

tagged[0:6]
```

```
Out[39]: [('At', 'IN'),
          ('eight', 'CD'),
          ("o'clock", 'JJ'),
          ('on', 'IN'),
          ('Thursday', 'NNP'),
          ('morning', 'NN')]
```

# POS

One more example.

```
text = nltk.word_tokenize("And now I present to you the best professor on earth",
                                    language="english")
tagged = nltk.pos_tag(text)
tagged
```

```
Out[43]: [('And', 'CC'),
          ('now', 'RB'),
          ('I', 'PRP'),
          ('present', 'VBP'),
          ('to', 'TO'),
          ('you', 'PRP'),
          ('the', 'DT'),
          ('best', 'JJS'),
          ('professor', 'NN'),
          ('on', 'IN'),
          ('earth', 'JJ')]
```

## Hands-on Session

- Please go to the **"Exercise: Lemmatization with POS Tag"** in the lecture code.

# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- **Parts of Speech**
  - Chunking
- Text Classification
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# Chunking

Defining patterns in POS helps, for example, to find phrases in a corpus -- especially noun phrases.

# Chunking

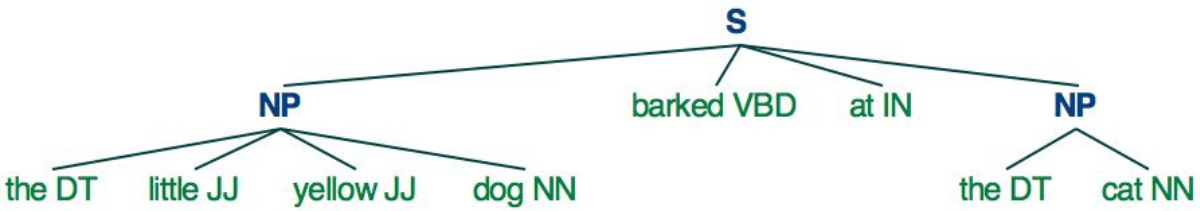
```
# a simple sentence with POS tags
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked",
          "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]
pattern = "NP: {<DT>?<JJ>*<NN>}"          # define a tag pattern of an NP chunk
NPChunker = nltk.RegexpParser(pattern)     # create a chunk parser
result = NPChunker . parse(sentence)        # parse the example sentence
print result                                # or draw graphically using result.draw()
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

The example is from: http://www.nltk.org/book/ch07.html

# Chunking

```
result.draw()
```



# Regex Chunking

Pros:

- More control over patterns matched

Cons:

- Difficult to hard code in every rule

# Hands-on Session

- Please go to the **"Exercise: Syntax Tree/ Chunking"** in the lecture code.

# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- Parts of Speech
- **Text Classification**
  - Sentiment Analysis of Tweets
- Latent Dirichlet Allocation

# Text Classification

- Assigning a subject
- Spam Detection
- Age/Gender of author
- Who is the author?
- Sentiment Analysis

# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- Parts of Speech
- Text Classification
  - **Sentiment Analysis of Tweets**
- Latent Dirichlet Allocation

# Sentiment Analysis

Let's say we have some tweets.

```
pos_tweets = [('I love this book', 'positive'),
              ('This food is amazing', 'positive'),
              ('I feel great this morning', 'positive'),
              ('I am so excited about the party', 'positive'),
              ('He is my best friend', 'positive')]

neg_tweets = [('I do not like this book', 'negative'),
              ('This food is horrible', 'negative'),
              ('I feel tired this morning', 'negative'),
              ('I am not looking forward to the party', 'negative'),
              ('He is my enemy', 'negative')]
```

# Sentiment Analysis

Let's clean up our tweets.

```
tweets = []
for (words, sentiment) in pos_tweets + neg_tweets:
    words_filtered = [e.lower() for e in words.split() if len(e) >= 3]
    tweets.append((words_filtered, sentiment))
tweets
```

```
Out[33]: [(['love', 'this', 'book'], 'positive'),
          (['this', 'food', 'amazing'], 'positive'),
          (['feel', 'great', 'this', 'morning'], 'positive'),
          (['excited', 'about', 'the', 'party'], 'positive'),
          (['best', 'friend'], 'positive'),
          (['not', 'like', 'this', 'book'], 'negative'),
          (['this', 'food', 'horrible'], 'negative'),
          (['feel', 'tired', 'this', 'morning'], 'negative'),
          (['not', 'looking', 'forward', 'the', 'party'], 'negative'),
          (['enemy'], 'negative')]
```

# Sentiment Analysis

And make some test tweets.

```
test_tweets = [
    (['feel', 'happy', 'this', 'morning'], 'positive'),
    (['larry', 'friend'], 'positive'),
    (['not', 'like', 'that', 'man'], 'negative'),
    (['house', 'not', 'great'], 'negative'),
    (['your', 'song', 'annoying'], 'negative')]
```

# Sentiment Analysis

Now let's extract the features we'll use in this algorithm.

```
def get_words_in_tweets(tweets):
    all_words = []
    for (words, sentiment) in tweets:
      all_words.extend(words)
    return all_words

def get_word_features(wordlist):
    wordlist = nltk.FreqDist(wordlist)
    word_features = wordlist.keys()
    return word_features

word_features = get_word_features(get_words_in_tweets(tweets))
```

# Sentiment Analysis

Finalize the extraction.

```
def extract_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

# Sentiment Analysis

Create our training set and train our classifier.

```
training_set = nltk.classify.apply_features(extract_features, tweets)

classifier = nltk.NaiveBayesClassifier.train(training_set)
```

# Sentiment Analysis

Now let's check how it does. *Keep in mind that this is a very simple classifier.*

```
tweet = 'Larry is my friend'
print classifier.classify(extract_features(tweet.split()))
```

```
positive
```

# Sentiment Analysis

Internally we can check how the classifier is working.

```
print extract_features(tweet.split())
```

```
{'contains(looking)': False, 'contains(feel)': False, 'contains(the)': False, 'contains(part
y)': False, 'contains(about)': False, 'contains(great)': False, 'contains(horrible)': False,
'contains(this)': False, 'contains(best)': False, 'contains(friend)': True, 'contains(enem
y)': False, 'contains(forward)': False, 'contains(excited)': False, 'contains(tired)': False,
'contains(like)': False, 'contains(love)': False, 'contains(book)': False, 'contains(amazin
g)': False, 'contains(food)': False, 'contains(not)': False, 'contains(morning)': False}
```

# Hands-on Session

- Please go to the **"Exercise: Classify the Testing Set"** in the lecture code.

# Outline

- Background
- Regular Expressions
- Working with words
- Natural Language ToolKit (NLTK)
- Creating a corpus
- Parts of Speech
- Text Classification
  - Sentiment Analysis of Tweets
- **Latent Dirichlet Allocation**

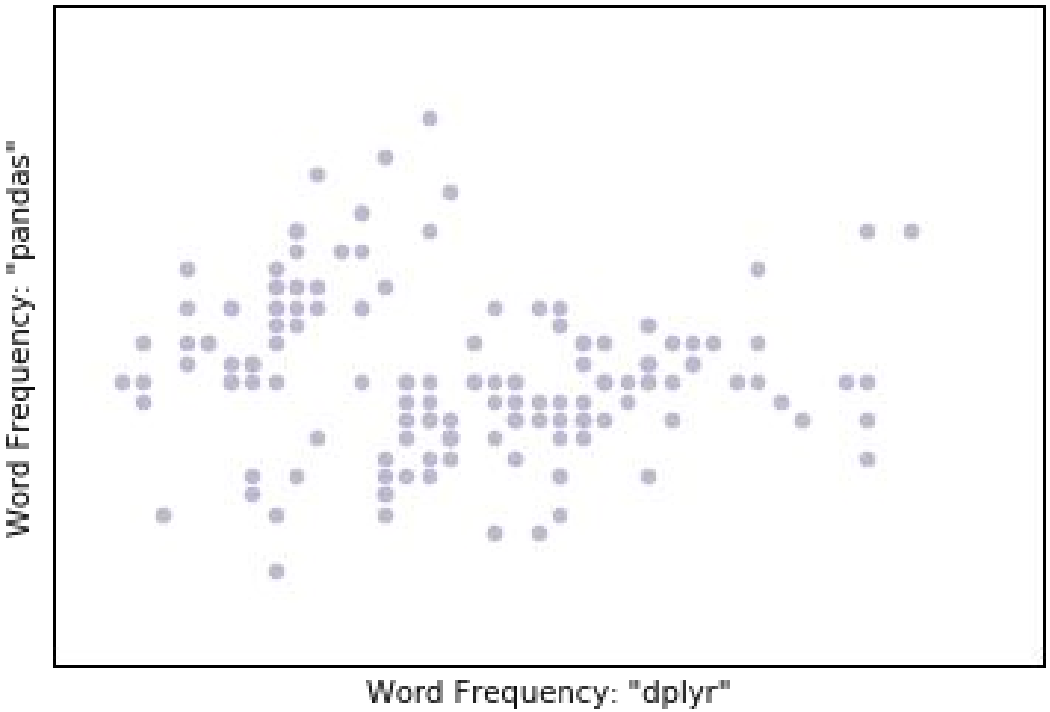
# Latent Dirichlet Allocation

In natural language processing, **latent Dirichlet allocation** (LDA) is an unsupervised method that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar.

For example, a **topic** is a common latent variable produced by LDA and used to characterize a document.

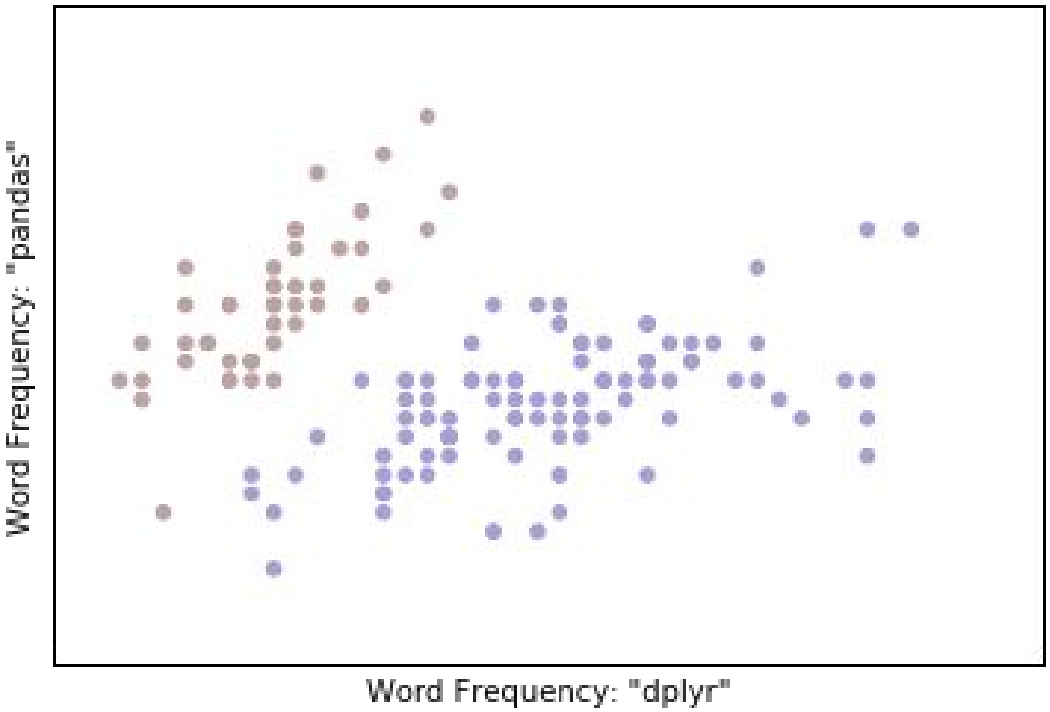
# Latent Dirichlet Allocation

Assume each dot below is an document in our corpus, and the scatter plot visualize the term frequency of two words: **pandas** and **dplyr**.



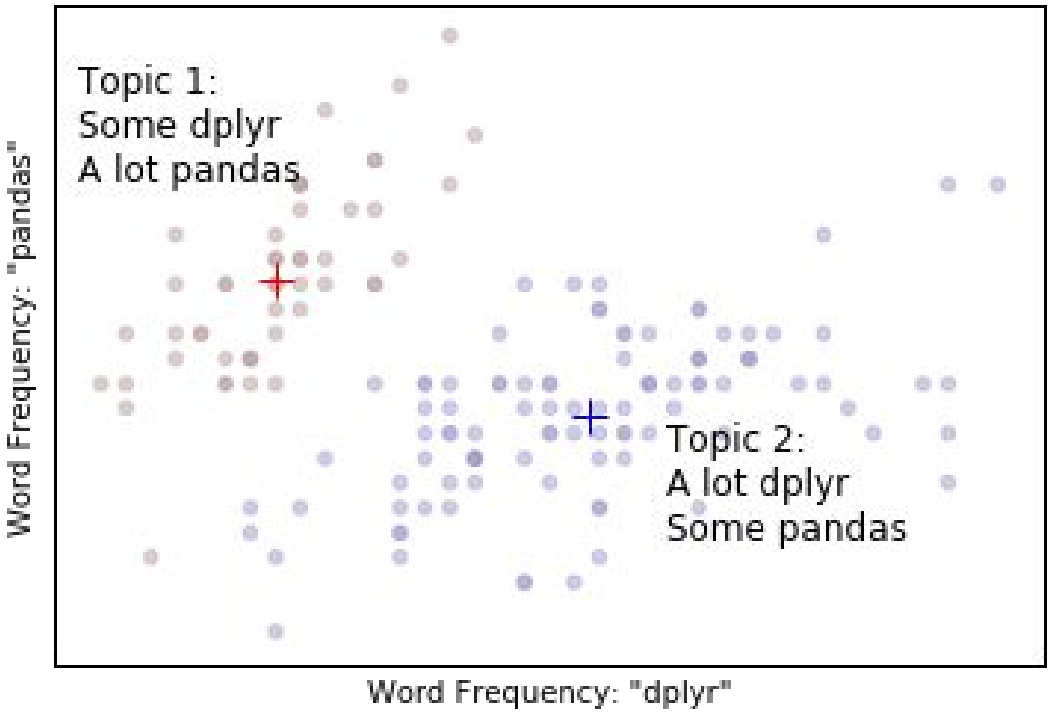
# Latent Dirichlet Allocation

We need to than specify how many topics we think there are, for example: two. The LDA algorithm will group the documents into two parts.



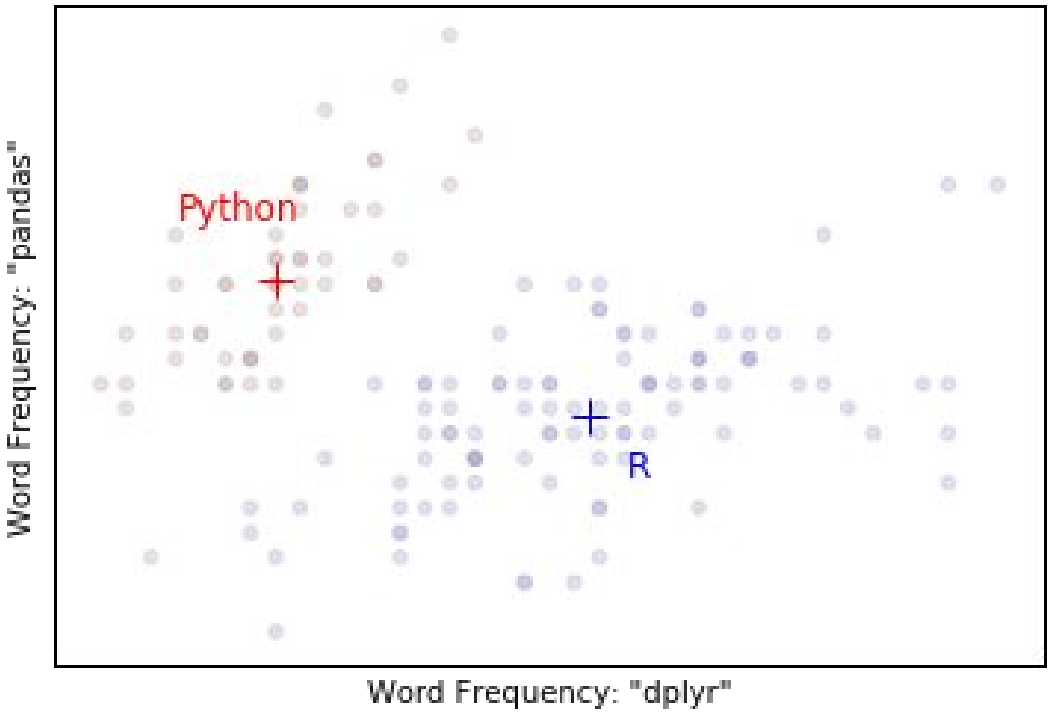
# Latent Dirichlet Allocation

LDA will also specify the two topics. Each topic is characterised by (essentially) their term frequency.



# Latent Dirichlet Allocation

Then it's up to us (data scientists!) to decide what actual topics should be.



# Latent Dirichlet Allocation

One thing that we haven't been very precise, is that each document is **NOT** assigned to a unique topic (as in the left figure). Instead, LDA will assign a mixture of topics to a document (as in the right figure):