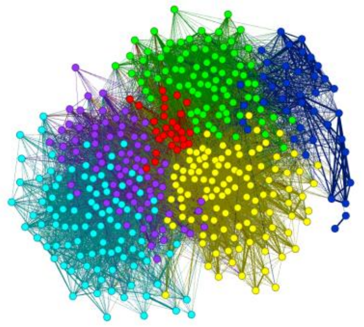
Data Analytics

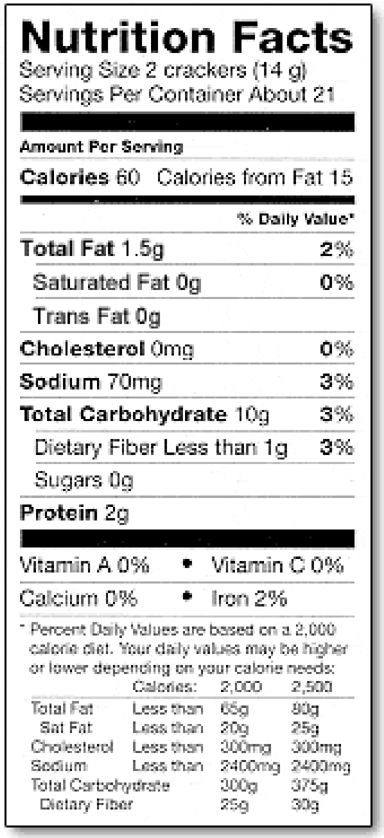
Gates



Outline

- Text Mining Overview
- Sentiment Analysis
- Topic Modeling
- Twitter Mining with Python3

80% of data around the world is unstructured (Forbes magazine)



Nutrition Facts
Serving Size 2 crackers (14 g)
Servings Per Container About 21

Amount Per Serving
Calories 60 Calories from Fat 15

	% Daily Value*
Total Fat 1.5g	2%
Saturated Fat 0g	0%
Trans Fat 0g	
Cholesterol 0mg	0%
Sodium 70mg	3%
Total Carbohydrate 10g	3%
Dietary Fiber Less than 1g	3%
Sugars 0g	
Protein 2g	

Vitamin A 0% • Vitamin C 0%
Calcium 0% • Iron 2%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your calorie needs:

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2400mg	2400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Text is Everywhere

Problems with Text Data

- High dimensionality
- Large number of features
- Noisy data
- Redundant data
- Schemaless
- Different ways to represent it
- Machines have a hard time processing it
- Lots of it
- Typically partial and biased

Text Mining

Text mining focuses on generating useful knowledge from text data.

Tasks

- Identify relevant documents (information retrieval)
- Cluster documents
- Summarize ideas
- Determine **sentiment**
- Identify **themes/topics**

Modeling Text

- We do not want to treat each character separately – instead we **TOKENIZE**

TOKENIZATION OPTIONS

- **Bag of words** (phrases, sentences, graphs) model
- Each **document** is represented as a **set of terms and weights**.
- The **weight** measures the importance of the word.
  - It can be frequency, inverse document frequency

Bag of Words

Document1:
Bob loves to eat ice cream. Sally loves to eat chocolate more.

Document 2:
Some popular desserts are ice cream, chocolate, cakes, pies, and donuts. I love chocolate.

Word List for Bag of Words Model: (in sorted order)
and, are, cakes, chocolate, desserts, donuts, eat, I, ice cream, Bob, love, loves, more, pie, popular, Sally, some, to

	Document 1	Document 2
and	0	1
are	0	1
cakes	0	1
chocolate	1	2
desserts	0	1
donuts	0	1
eat	2	0
I	0	1
ice	1	1
cream	1	1
Bob	1	0
love	0	1
loves	2	0
Sally	1	0

Possible Problems

	Document 1	Document 2
~~and~~	~~0~~	~~1~~
~~are~~	~~0~~	~~1~~
cakes	0	1
chocolate	1	2
desserts	0	1
donuts	0	1
eat	2	0
I	0	1
ice	1	1
cream	1	1
Lisa	1	0
love	2	1
…		

Remove common stopwords

lemmatize

Modeling Text – Determining Weights

- The **weight** measures the **importance** of the word.

Common options:

- **Frequency** (previous example)
  - Not always the best because common words like ***the*** and ***a*** are not information rich words.

- **Inverse document frequency (IDF).**

$$IDF(t) = log(\frac{|D|}{|D(t)|})$$

t = term
tf = term frequency (number of times term appear in a single document)
idf = inverse document frequency
D = document set (collection of documents)
|D| = number of documents in D
|D(t)| = number of documents in D containing t

A Closer Look and TF-IDF

TF: Term Frequency: measures how frequently a term occurs in a document.

- Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones.
- Thus, the term frequency is often **divided by the document length** as a way of normalization:

  Normalized TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

IDF: Inverse Document Frequency: measures how **important** a term is.

- While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones; by computing the following:

  IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

RE: http://www.tfidf.com/

TF-IDF

$$TF\_IDF(t, D_i) = tf(t, D_i) \times IDF(t)$$

Example 1: Consider a **document** containing **100 words** wherein the word ***cat*** appears 3 times.

The **term frequency** (i.e., normalized tf) for *cat* is then (3 / 100) = 0.03.

Now, assume we have **10 million documents** and the word ***cat* appears in 1000 of these**.

Then, the **inverse document frequency** (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4.

Thus, the **Tf-idf weight** is the product of these quantities: 0.03 * 4 = 0.12.

http://www.tfidf.com/

Example 2

Example Query - Hoyas

Suppose Doc A contains the term “Hoyas” 25 times out of 5000 words.

Term Freq **TF**(“Hoyas”, DocA)=25/5000 = .005

The word “Hoyas” is in 1000 of the documents in the corpus (document set).
|D(t)| = 1000

The Document set has 1 million documents in it.
|D| = 1,000,000

Inverse Doc Freq **IDF**(“Hoyas”) = log_e(1000000/1000) = 3

TF-IDF(“Hoyas”, Document Set) = .005 x 3 = .015

Text Pre-processing Options

Because our goal is to makes sense of text, **pre-processing** is vital.

Stop-word removal

- Removing words that have no information content
- **Stop words** include words like a, the are, and

Tokenizing words, phrase, sentences.

- Standardizing and deciding on the **unit of text**.

Stemming and/or **Lemmatization**

- Take different forms of a word and reduce them to a single form.
- **sing, singing, sings becomes sing**

Ref: http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Text Pre-processing Example

Example Paragraph:

The Chicago Cubs won. Now win the World Series.

1) Remove stop word: the

2) Stem/Lemmatize: won to win

3) Tokenize

Unit words: word tokens (most common) = {win, Chicago, Cubs, now, world, series}

Unit phrase: phrase tokens = {Chicago Cubs, World Series}

Unit sentence: sentence tokens = {Chicago cubs win, now win world series}

Tokenizers and Parsing

Think about how you might read in documents, preprocess them, and parse/tokenize them.

Tokenizing/Parsing: Separating or breaking down streams of text into meaningful units (usually words) but can also include symbols, phrases, or any other goal.

- This can include the identification of specific elements.
- Interpreters and compilers also first parse code to evaluate syntax and grammar.

RE:

http://aircconline.com/acii/V3N1/3116acii04.pdf

http://www.nltk.org/api/nltk.tokenize.html

Install NLTK: http://pypi.python.org/pypi/nltk

Python NLTK Tokenizer

```
# Get nltk for anaconda
import nltk
sentence = "The Hoyas have the cutest mascot of all colleges"
tokens = nltk.word_tokenize(sentence)
print(tokens)
#The output will be ['The', 'Hoyas', 'have', 'the', 'cutest',
#'mascot', 'of', 'all', 'colleges']
```

Scrape Example with nltk and requests

```
import requests
import nltk

txt=requests.get("http://www.georgetown.edu")
JustTheText=txt.text
#print(JustTheText[0:100])
GUtokens=nltk.word_tokenize(JustTheText)
#print(GUtokens)
#print(type(GUtokens))

count=0
FindThisWordL="research"
FindThisWordC="Research"
for word in GUtokens:
   if ((FindThisWordL in word) or (FindThisWordC in word)):
      print(word)
      count=count+1
```

Example 1:
Comparing Books (via text)
and Visualizing the Comparisons

- The following slides will show a collection of code that will explore methods for comparing books written by different authors.
- This example will use several Python3 Libraries.
- This example will include three visualization methods.

Python 3 Libraries Used

```
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
from sklearn.manifold import MDS
from mpl_toolkits.mplot3d import Axes3D
from scipy.cluster.hierarchy import ward, dendrogram
```

Getting the Data

For this example-set, I will use British books.

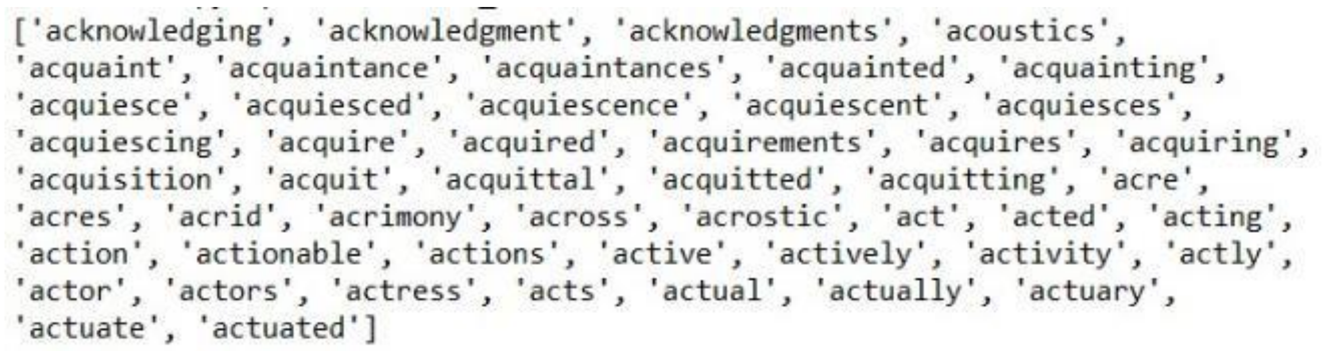
Here is the list of books I use:

filenames = [
'../DATA/**Austen_Emma**.txt', '../DATA/**Austen_Pride**.txt',
'../DATA/**Austen_Sense**.txt', '../DATA/**CBronte_Jane**.txt',
'../DATA/**CBronte_Professor**.txt', '../DATA/**Dickens_Bleak**.txt',
'../DATA/**Dickens_David**.txt', '../DATA/**Dickens_Hard**.txt‘
]

You can download this data from here:
https://de.dariah.eu/tatom/datasets.html#datasets

Vectorize and Convert to Various Data Structures

vectorizer = CountVectorizer(input='filename')
#Using input='filename' means that fit_transform will
#expect a list of file names
#dtm is document term matrix (a sparse matrix)
dtm = vectorizer.fit_transform(filenames) print(type(dtm))
#vocab is a vocabulary list of each word that appears
vocab = vectorizer.get_feature_names() *# change to a list*
dtm = dtm.toarray() *# convert to a regular array*
print(list(vocab)[500:550])



```
['acknowledging', 'acknowledgment', 'acknowledgments', 'acoustics',
'acquaint', 'acquaintance', 'acquaintances', 'acquainted', 'acquainting',
'acquiesce', 'acquiesced', 'acquiescence', 'acquiescent', 'acquiesces',
'acquiescing', 'acquire', 'acquired', 'acquirements', 'acquires', 'acquiring',
'acquisition', 'acquit', 'acquittal', 'acquitted', 'acquitting', 'acre',
'acres', 'acrid', 'acrimony', 'across', 'acrostic', 'act', 'acted', 'acting',
'action', 'actionable', 'actions', 'active', 'actively', 'activity', 'actly',
'actor', 'actors', 'actress', 'acts', 'actual', 'actually', 'actuary',
'actuate', 'actuated']
```

Counting Words

##Ways to count the word "house" in Emma (file 0 in the list of files)

house_idx = list(vocab).index('house') *#index of "house"* **print(house_idx)**

print(dtm[0, house_idx])

#There are 95 occurrences of “house” in Emma

#Counting "house" in Pride and Prejudice (107 in Pride)

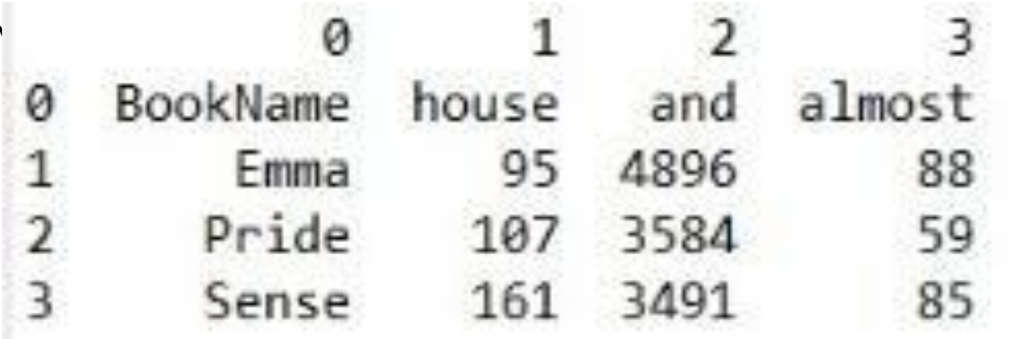
print(dtm[1,house_idx])

print(list(vocab)[house_idx]) #this will print “house”

print(dtm) #prints the *doc term matrix*

Create a word count table

```
##Create a table of word counts to compare Emma and Pride and Prejudice
columns=["BookName", "house","and","almost"]
MyList=["Emma"]
MyList2=["Pride"]
MyList3=["Sense"]
for someword in ["house","and", "almost"]:
    EmmaWord = (dtm[0, list(vocab).index(someword)])
    MyList.append(EmmaWord)
    PrideWord = (dtm[1, list(vocab).index(someword)])
    MyList2.append(PrideWord)
    SenseWord = (dtm[2, list(vocab).index(someword)])
    MyList3.append(SenseWord)
#print(MyList)
#print(MyList2)
df2=pd.DataFrame([columns, MyList,MyList2, M
print(df2)
```



```
          0      1     2       3
0  BookName  house   and  almost
1      Emma     95  4896      88
2     Pride    107  3584      59
3     Sense    161  3491      85
```

Measuring the “distance” between documents

1) **Euclidean distance**
2) **Cosine Similarity**

- The **cosine similarity** between two vectors (or two documents on the Vector Space) is a measure that calculates the cosine of the angle between them.

- This metric is a measurement of **orientation** and not magnitude.

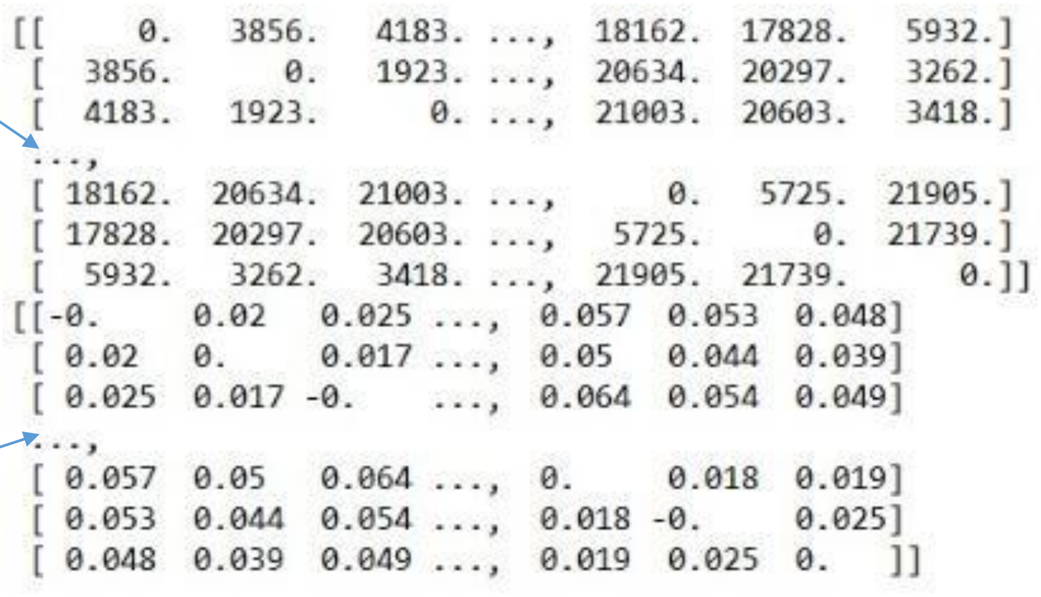
- It can be seen as a **comparison between documents on a normalized space** because we’re not taking into the consideration only the magnitude of each word count (tf-idf) of each document, but the **angle between the documents.**

RE: http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

Distance in Python

#Euclidean Distance

```
dist = euclidean_distances(dtm)
print(np.round(dist,0))
#The dist between Emma and Pride is 3856
```



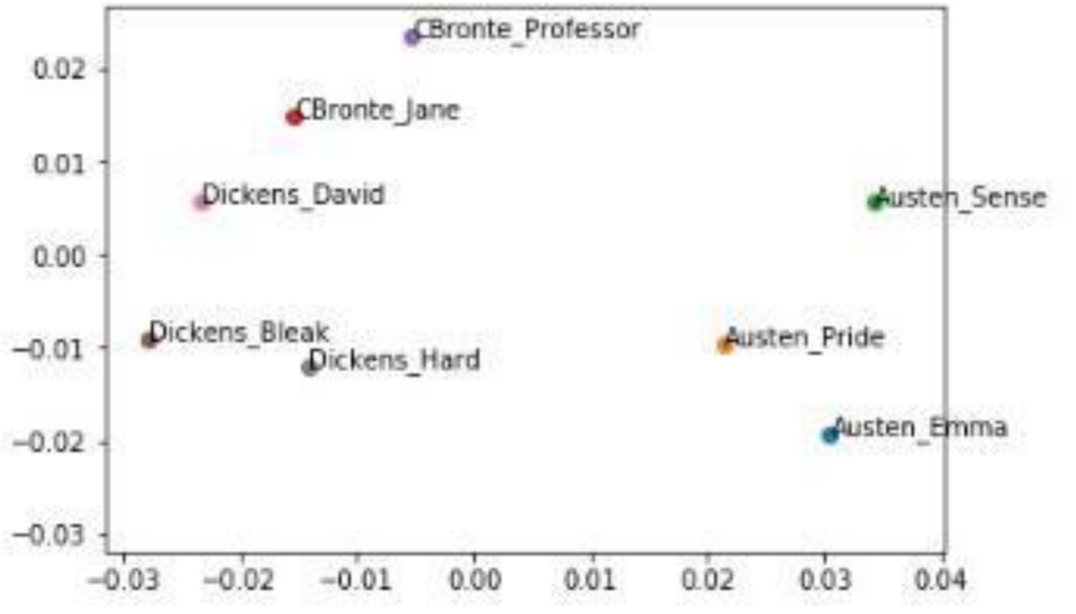
```
[[     0.   3856.   4183. ...,  18162.  17828.   5932.]
 [  3856.      0.   1923. ...,  20634.  20297.   3262.]
 [  4183.   1923.      0. ...,  21003.  20603.   3418.]
 ...,
 [ 18162.  20634.  21003. ...,      0.   5725.  21905.]
 [ 17828.  20297.  20603. ...,   5725.      0.  21739.]
 [  5932.   3262.   3418. ...,  21905.  21739.      0.]]
[[-0.     0.02   0.025 ...,  0.057  0.053  0.048]
 [ 0.02   0.     0.017 ...,  0.05   0.044  0.039]
 [ 0.025  0.017 -0.    ...,  0.064  0.054  0.049]
 ...,
 [ 0.057  0.05   0.064 ...,  0.     0.018  0.019]
 [ 0.053  0.044  0.054 ...,  0.018 -0.     0.025]
 [ 0.048  0.039  0.049 ...,  0.019  0.025  0.   ]]
```

#Cosine Similarity

```
cosdist = 1 - cosine_similarity(dtm)
print(np.round(cosdist,3))  #cos dist should be .02
```

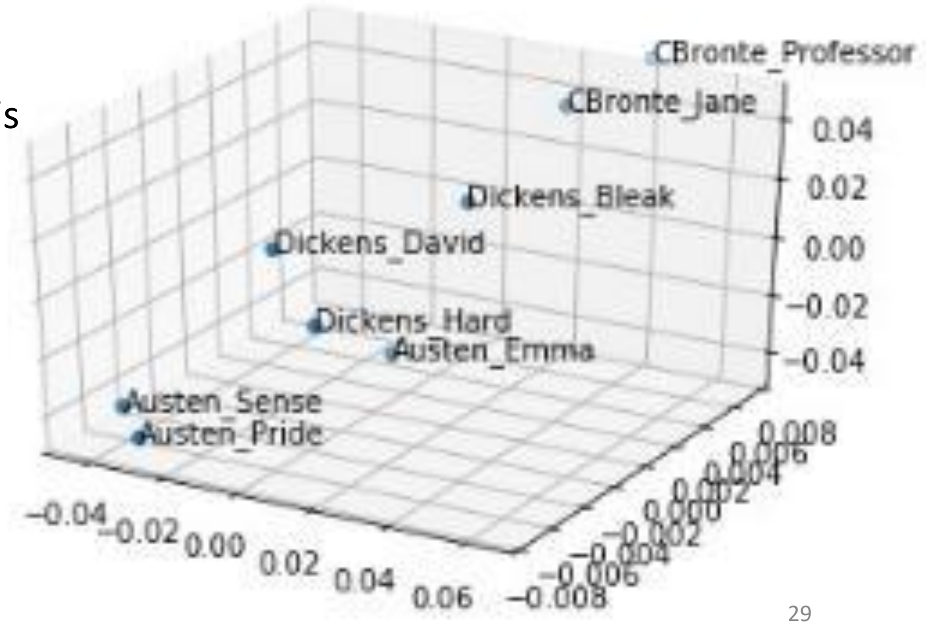
Visualizing Document Distances 2D Cartesian

```
## This type of visualization is called **multidimensional scaling (MDS)**

mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)
## "precomputed" means we will give the dist (as cosine sim)
pos = mds.fit_transform(cosdist)  # shape (n_components, n_samples)
xs, ys = pos[:, 0], pos[:, 1]
names=["Austen_Emma", "Austen_Pride", "Austen_Sense", "CBronte_Jane",
    "CBronte_Professor", "Dickens_Bleak",
    "Dickens_David", "Dickens_Hard"]

for x, y, name in zip(xs, ys, names):
  plt.scatter(x, y)
  plt.text(x, y, name)

plt.show()
```



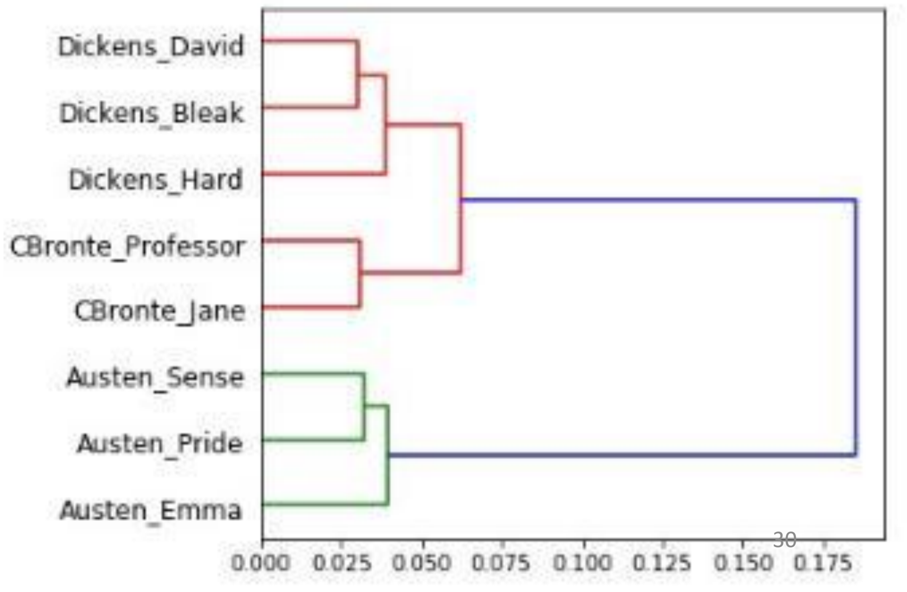
Visualization Cartesian 3D

```
mds = MDS(n_components=3, dissimilarity="precomputed", random_state=1)
pos = mds.fit_transform(cosdist)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(pos[:, 0], pos[:, 1], pos[:, 2])
for x, y, z, s in zip(pos[:, 0], pos[:, 1], pos[:, 2], names):
   ax.text(x, y, z, s)

ax.set_xlim3d(-.05,.07) #stretch out the x axis
ax.set_ylim3d(-.008,.008) #stretch out the x axis
ax.set_zlim3d(-.05,.05) #stretch out the z axis
plt.show()
```



Visualization with Hierarchical Clustering

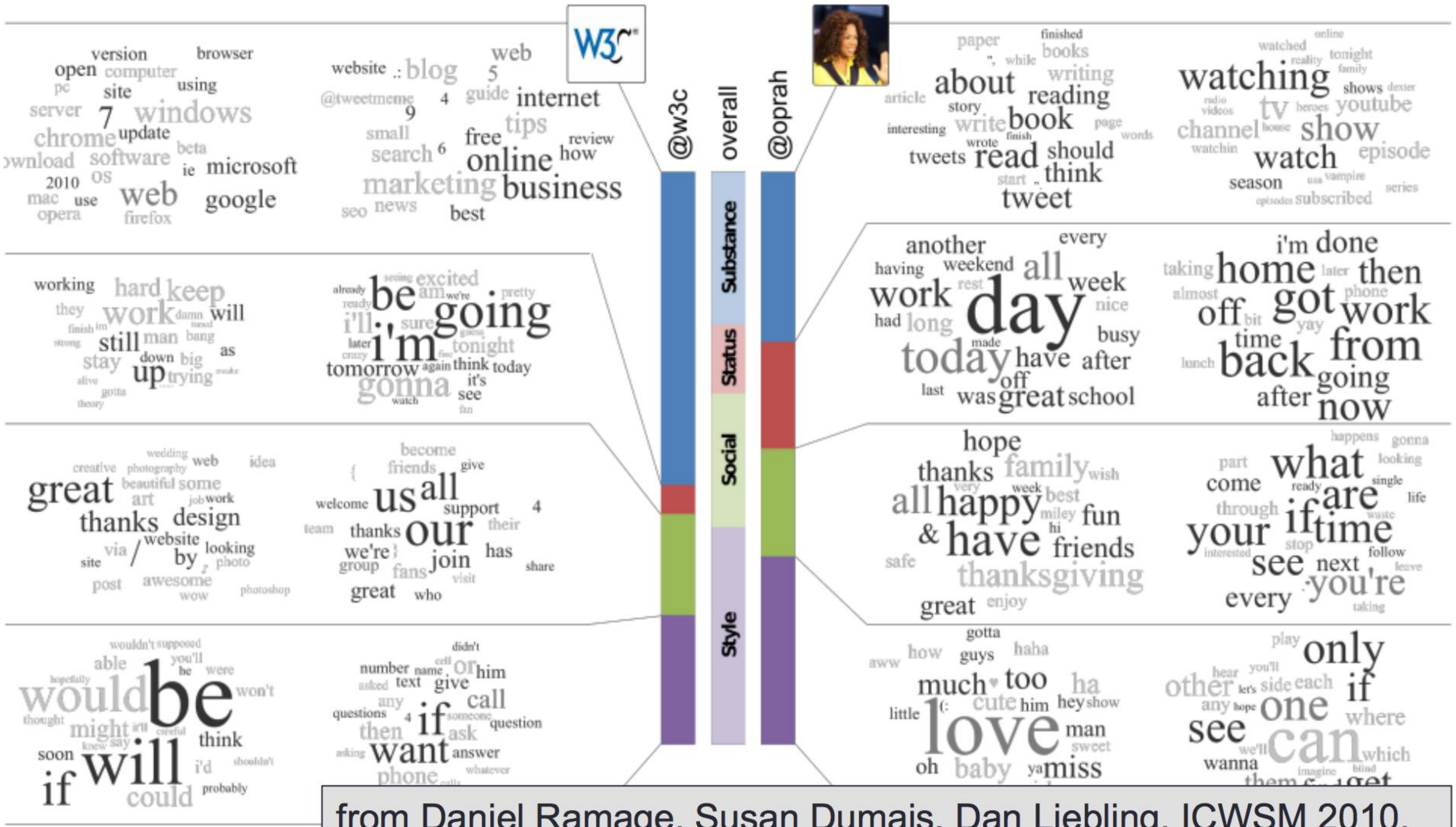
```
## Clustering Texts and Visualizing
#One method for clustering is Ward’s
#Ward’s method produces a hierarchy of clusterings
# Ward’s method requires  a set of pairwise distance measurements

linkage_matrix = ward(cosdist)
dendrogram(linkage_matrix, orientation="right", labels=names)
plt.tight_layout()
plt.show()
```



Example 2:
Twitter Mining and WordClouds

The following example will discuss and demonstrate methods for accessing Twitter data as JSON.

This data can be processed and cleaned in many ways, including methods already discussed.



from Daniel Ramage, Susan Dumais, Dan Liebling. ICWSM 2010.

Suggested Packages

```
##Gates
###Packages-----------------------

import tweepy
from tweepy import OAuthHandler
import json
from tweepy import Stream
from tweepy.streaming import StreamListener
import sys

import json
from nltk.tokenize import word_tokenize
from nltk.tokenize import TweetTokenizer
import re

from os import path
from scipy.misc import imread
import matplotlib.pyplot as plt
##install wordcloud
from wordcloud import WordCloud, STOPWORDS
```

Setting up Keys and OAuth

```
## Create all the keys and secrets that you get
## from using the Twitter API-----------------------------------

consumer_key = 'mnDxxxxxAAUcM2'
consumer_secret = 'qzwDO9o6Im6xxxxxcuXnlEBuLRYsU'
access_token = '8385586021xxxwkSdqi9xxG'
access_secret =hswxbxExxx7Aj2u5TJxxxxxxxxxXAnF'

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)

api = tweepy.API(auth)
```

The Listener Class

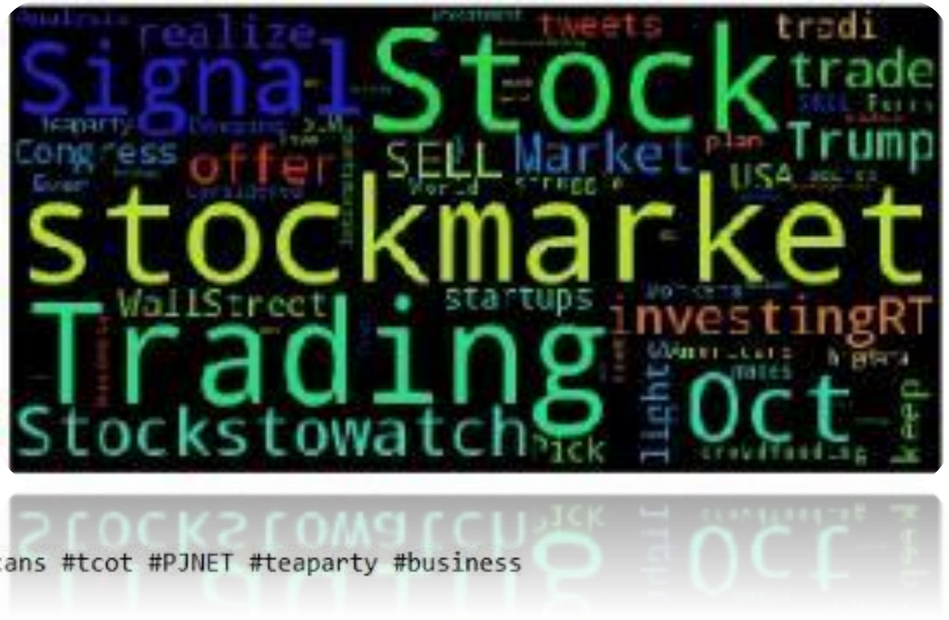
```
class Listener(StreamListener):
  print("In Listener...")
  tweet_number=0 #count the tweets
  #__init__ runs as soon as an instance of the class is created
  def __init__(self, max_tweets, hfilename, rawfile):
    self.max_tweets=max_tweets
    print(self.max_tweets)
  #on_data() is a function of StreamListener as is on_error and on_status
  def on_data(self, data):
    self.tweet_number+=1
    print("In on_data", self.tweet_number)
    try:
      print("In on_data in try")
      with open(hfilename, 'a') as f:
        with open(rawfile, 'a') as g:
          tweet=json.loads(data)
          tweet_text=tweet["text"]
          print(tweet_text,"\n")
          f.write(tweet_text) # the text from the tweet
          json.dump(tweet, g)  #write the raw tweet
    except BaseException:
      print("NOPE")
      pass
    if self.tweet_number>=self.max_tweets:
      sys.exit('Limit of '+str(self.max_tweets)+' tweets reached.') #or return False
  #method for on_error()
  def on_error(self, status):
    print("ERROR")
    if(status==420):
      print("Error ", status, "rate limited")
      return False
```

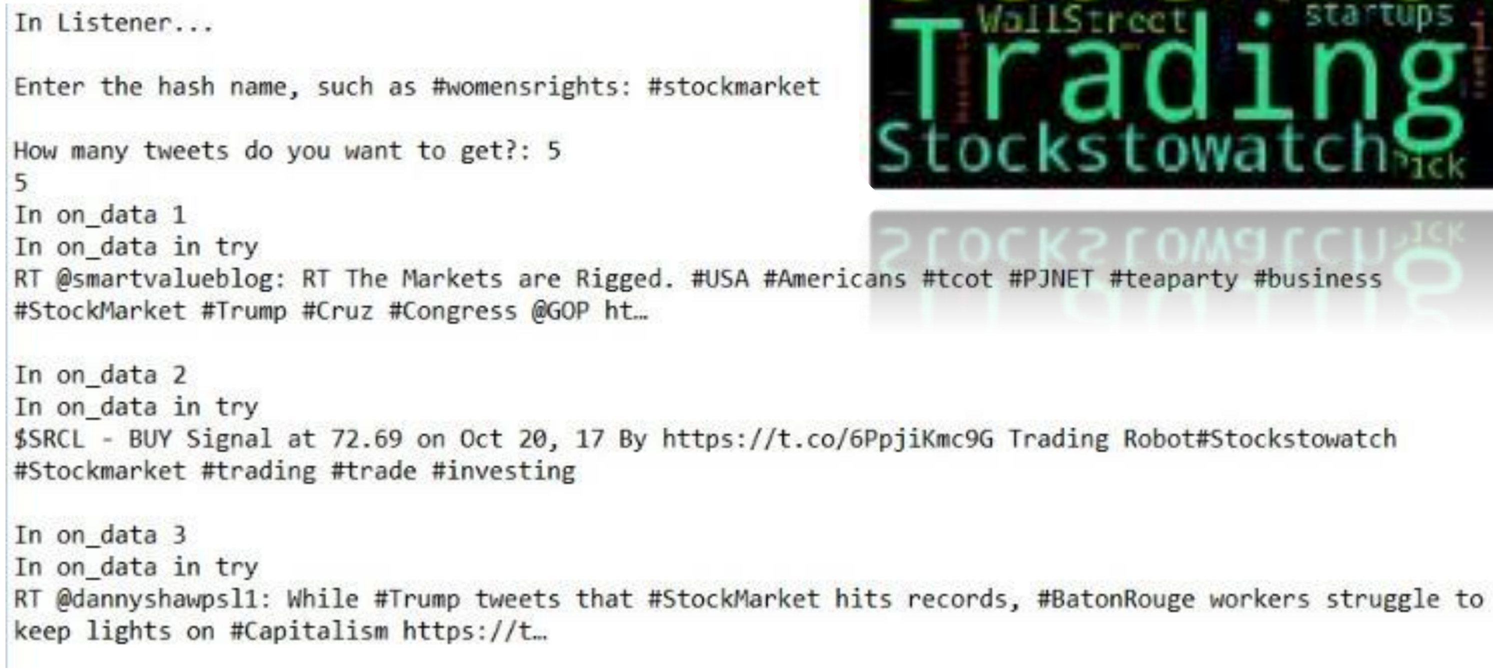
Call Listener…

```
hashname=input("Enter the hash name, such as #womensrights: ")
numtweets=eval(input("How many tweets do you want to get?: "))
if(hashname[0]=="#"):
    nohashname=hashname[1:] #remove the hash
else:
    nohashname=hashname
    hashname="#"+hashname

#Create a file for any hash name
hfilename="file_"+nohashname+".txt“   #dynamic file naming
rawfile="file_rawtweets_"+nohashname+".txt“
#from Tweepy
twitter_stream = Stream(auth, Listener(numtweets, hfilename, rawfile))
#twitter_stream.filter(track=['#womensrights'])
twitter_stream.filter(track=[hashname])
```

Word Cloud & a few Tweets for #stockmarket

- From this point, you can create files, evaluate text, compare text, evaluate sentiment, etc.





```
In Listener...

Enter the hash name, such as #womensrights: #stockmarket

How many tweets do you want to get?: 5
5
In on_data 1
In on_data in try
RT @smartvalueblog: RT The Markets are Rigged. #USA #Americans #tcot #PJNET #teaparty #business
#StockMarket #Trump #Cruz #Congress @GOP ht…

In on_data 2
In on_data in try
$SRCL - BUY Signal at 72.69 on Oct 20, 17 By https://t.co/6PpjiKmc9G Trading Robot#Stockstowatch
#Stockmarket #trading #trade #investing

In on_data 3
In on_data in try
RT @dannyshawpsl1: While #Trump tweets that #StockMarket hits records, #BatonRouge workers struggle to
keep lights on #Capitalism https://t…
```

Example 3: scikit code example

```
## Basic scikit learn tokenizer and counter
## http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-#representation
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
# See: http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-#representation
# For code on IDF and TF_IDF

vectorizer = CountVectorizer(min_df=1, ngram_range=(1,2), token_pattern=r'\b\w+\b')

corpus = ['This is the first document.',
        'This is the second document.',
        'And the third one.',
        'Is this the first document?']

X = vectorizer.fit_transform(corpus)
print(X)
W=vectorizer.get_feature_names()
print(W)
#The output for W is
#['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
print(X.toarray())
```

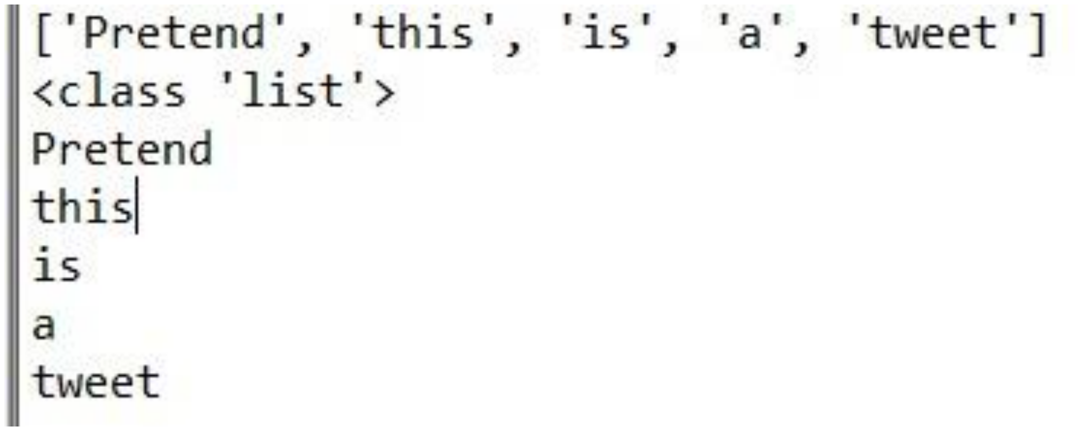
continued

```
#Column of matrix
print(vectorizer.vocabulary_.get('document'))
#Result is 1
print(vectorizer.vocabulary_.get('FruitCake'))
#Result is None
print(vectorizer.vocabulary_.get('this'))
#Result is 18
analyze = vectorizer.build_analyzer()
Y = analyze("This is a text document to analyze.")
print(Y)
#The output Y is
#['this', 'is', 'text', 'document', 'to', 'analyze']
```

http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation

```
#Example 4
from nltk.tokenize import word_tokenize
tweet2 = "Pretend this is a tweet"

BagOfWords=word_tokenize(tweet2)
print(BagOfWords)
print(type(BagOfWords))
for eachword in BagOfWords:
    print(eachword)
```

RESULT:



```
['Pretend', 'this', 'is', 'a', 'tweet']
<class 'list'>
Pretend
this
is
a
tweet
```

Sentiment Analysis

reworded from Dan Jurafsky's slides

Sentiment Analysis

The process of determining the **opinion** of a piece of text.

At this stage, this translates to determining if the writer’s attitude is **positive**, **negative**, or **neutral** in the particular piece of writing.

- Algorithms may be statistical, machine learning, or natural language processing.

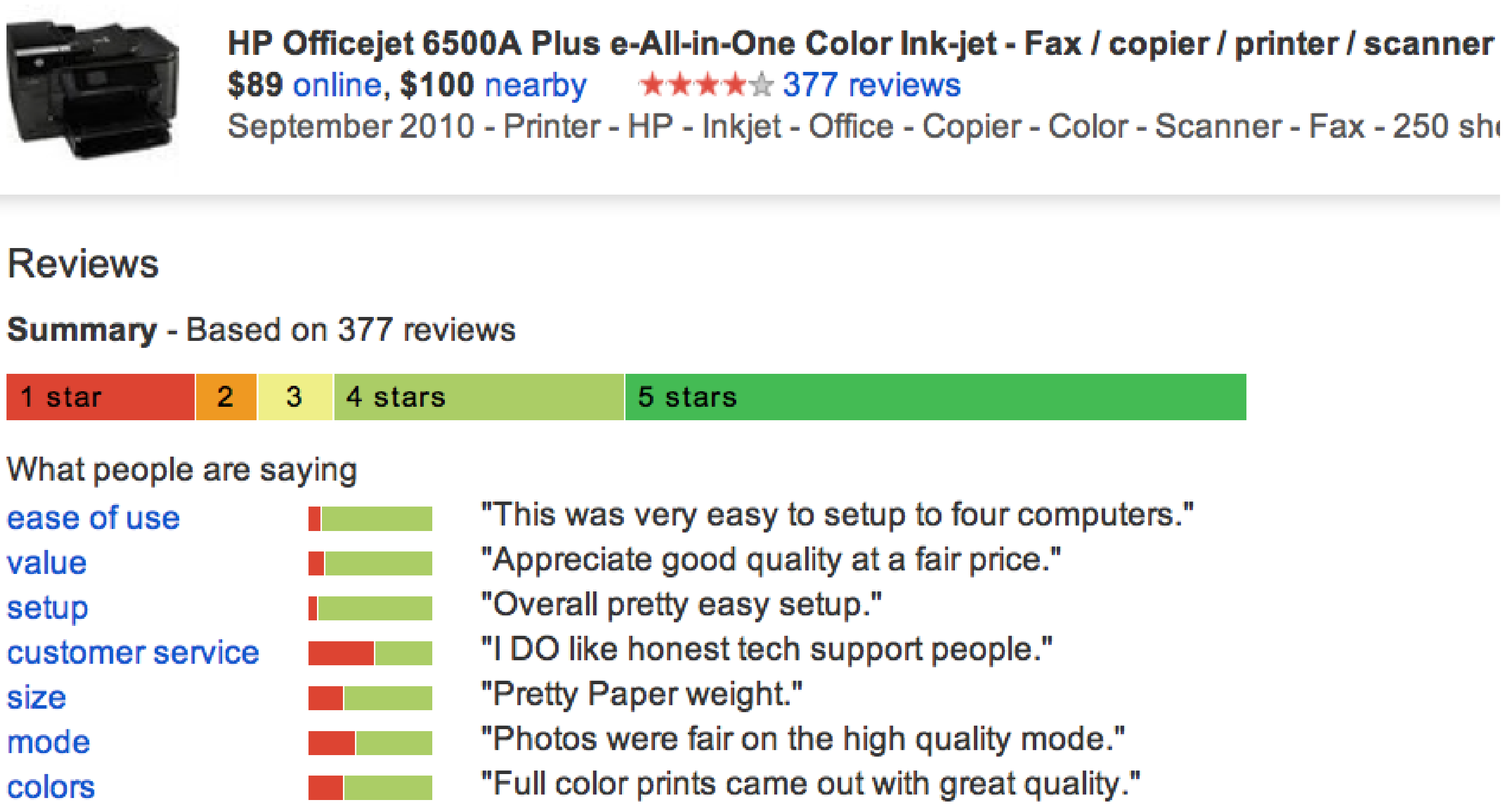
- Sentiment generally has a **strength** associated with it.

Movie review sentiment

- 👎 unbelievably disappointing
- 👍 Full of zany characters and richly applied satire, and some great plot twists
- 👍 this is the greatest screwball comedy ever filmed
- 👎 It was pathetic. The worst part about it was the boxing scenes.

Dan Jurafsky's example

Product Search – Reviews as a Proxy for Sentiment



HP Officejet 6500A Plus e-All-in-One Color Ink-jet - Fax / copier / printer / scanner
$89 online, **$100** nearby ★★★★☆ 377 reviews
September 2010 - Printer - HP - Inkjet - Office - Copier - Color - Scanner - Fax - 250 sh

Reviews

Summary - Based on 377 reviews

1 star | 2 | 3 | 4 stars | 5 stars

What people are saying

ease of use	"This was very easy to setup to four computers."
value	"Appreciate good quality at a fair price."
setup	"Overall pretty easy setup."
customer service	"I DO like honest tech support people."
size	"Pretty Paper weight."
mode	"Photos were fair on the high quality mode."
colors	"Full color prints came out with great quality."

Dan Jurafsky's example

Sentiment Algorithms

- Many different **methods** that vary in accuracy depending on **type of text data**.

Method 1

- Create a **lexicon** (vocabulary) of **positive** and **negative** words and **count the frequency** of them in each sentence or paragraph.

Method 2

- **Use the sentence structure** to augment the lexicon. For example, **a negation word indicates that the sentiment is reversed**
- Example: *I did **not** like the move*

Sentiment Algorithms cont.

- Use a standard **machine learning** algorithm and setup a **classification** task:
  - **Classes {positive, negative, neutral}**
  - This approach can incorporate linguistics features and syntactic features.
  - Can have hard classes (single label), or probabilistic value of labels.

- ***Approach****:*
  1. *Generate* a set of **training records**, $D = \{X_1, X_2, \ldots, X_n\}$ where **each record is labeled to a sentiment class**.
  2. Create a **classification model**.
  3. Then for unlabeled data (a given instance of unknown class), the **model is used to predict** a class label for it.

Sentiment Classifier

- **Naïve Bayes** classifier generally performs better than others.
- When possible **have a class label for noise**.
- Classifier for sentiment on Twitter is different from traditional sentiment detection algorithms.
  - There is noise, new vocabulary, and emojis to consider.
- **Hand-labeled training data** is particularly important for Twitter and other social media sites.

Things that make sentiment hard

- Limited training data
- Sarcasm

  Please keep talking – I always yawn when I am interested.
- Ambiguous word in the text

  Susy went to the bank

Topic Modeling

Topic Models

- **Topic models** can help you automatically discover patterns in a corpus (set of documents)
  - Unsupervised learning

- Topic models are **generative models** that assume an underlying distribution.

- Topic models automatically…
  - **Group topically-related words into "topics"**
  - Associate tokens and documents with those topics

Topic Discovery

1. I ate a banana and cereal for breakfast.
2. I like to eat chocolate and bananas.
3. Puppies and kittens are cute.
4. We adopted a puppy earlier today.
5. Look at the puppy eating cereal!

Parameters to specify: Number of topics

Sentence 1 & 2: 100% Topic A (eating_
Sentence 3 & 4: 100% Topic B (puppies)
Sentence 5: 60% Topic A and 40% Topic B

Topic A: 30% cereal, 30% banana, 10% breakfast, …
Topic B: 30% puppy, 10% kittens, 10% cute

Small Conceptual Example
k = 2 topics

CORPUS:

Doc1: **summer, winter, spring**

Doc2: winter, **flu, headache**

Doc3: **sick**, flu

Doc4: sick, winter

Doc5: flu, **cough**, sick, headache

Doc 6: summer, flu

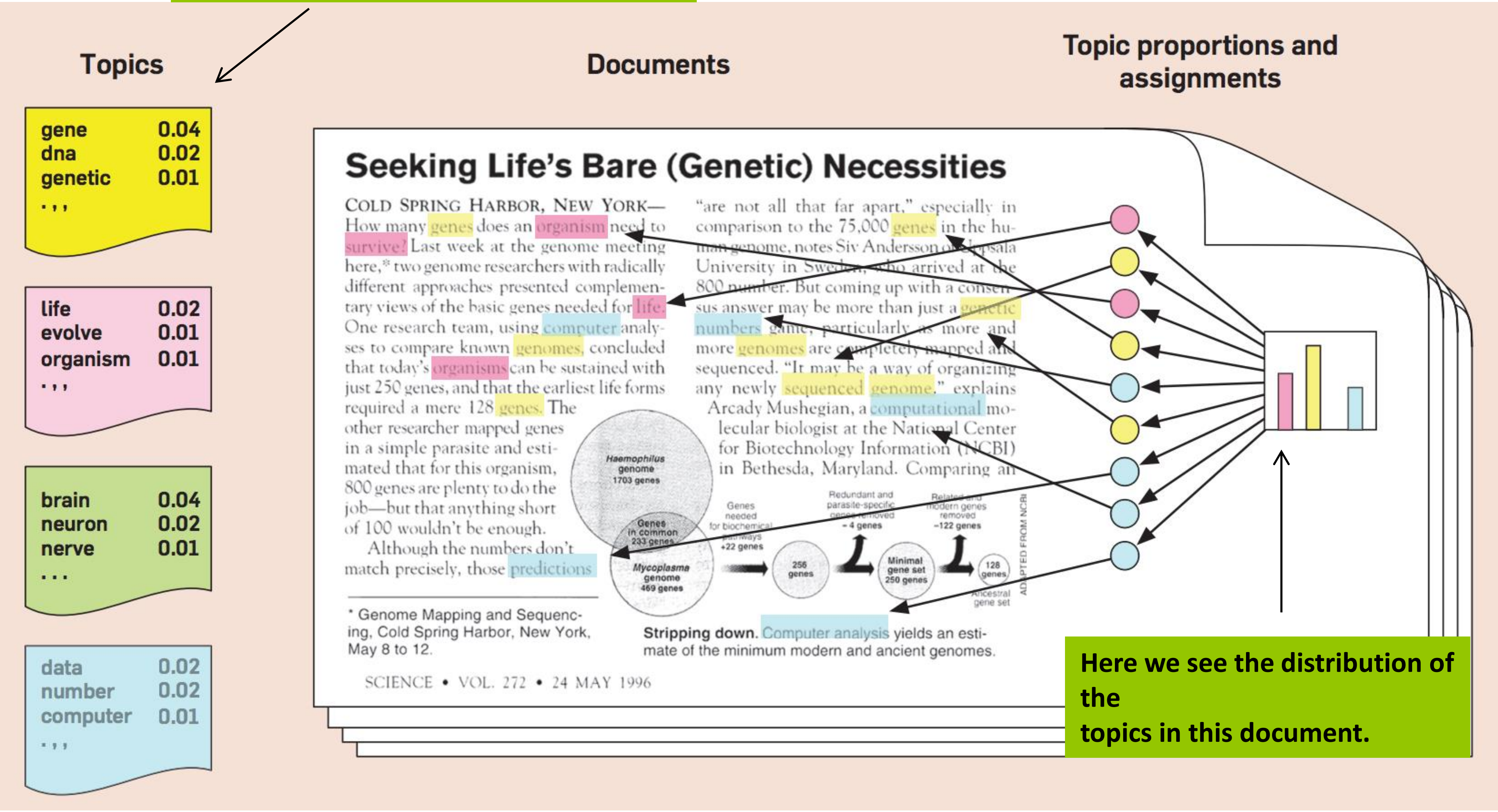
Predict:

Doc7: spring, cough

words	TOPIC 1 (weather)	TOPIC 2 (health)
summer	33%	0%
winter	33%	0%
spring	33%	0%
flu	0%	25%
sick	0%	25%
cough	0%	25%
headache	0%	25%

	TOPIC 1 (weather)	TOPIC 2 (health)
Doc1	100%	0%
Doc2	33%	66%
Doc3	0%	100%
Doc4	50%	50%
Doc5	0%	100%
Doc6	50%	50%
Doc7	50%	50%

Generated Topics – Each Topic has a set of words and the probability of importance of the word.



Topics
Documents
Topic proportions and assignments
gene 0.04
dna 0.02
genetic 0.01
life 0.02
evolve 0.01
organism 0.01
brain 0.04
neuron 0.02
nerve 0.01
data 0.02
number 0.02
computer 0.01
Haemophilus genome 1703 genes
Genes in common 233 genes
Mycoplasma genome 469 genes
Genes needed for biochemical pathways +22 genes
Redundant and parasite-specific genes removed −4 genes
Related and modern genes removed −122 genes
256 genes
Minimal gene set 250 genes
128 genes
Ancestral gene set
ADAPTED FROM NCBI

Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK— How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

Here we see the distribution of the topics in this document.

Blei, 2012

Topic Modeling Algorithm: **Latent Dirichlet Allocation (LDA)**

- **LDA represents documents as mixtures of topics** that contain **words** with certain **probabilities**. It is a **bag of words** model.

- It assumes documents are produced by:
  - Deciding on the **number of words in the document**. [Standard approach is to assume a Poisson Distribution]
  - Choosing a **topic mixture** for the document using a **Dirichlet distribution** over a fixed set of K topics.
  - Generating each word in the document by picking a topic according to the multinomial distribution you sampled and then using the topic to generate the word according to the topic distribution.

- **It iteratively uses this approach until it converges.**

RE: http://cseweb.ucsd.edu/~dhu/docs/research_exam09.pdf

LDA Learning Process

1) Choose a **fixed number of K topics to discover**,
2) Goal: Use LDA to learn the **topic representation** of each **document** and the **words associated to each topic**.
3) Go through each document, and **randomly assign each word** in the document **to one of the K topics**.

For each document (d), go through each word (w) in the document and for each topic (t) **compute:**

p(t|d) : the proportion of words in document d that are currently assigned to topic t

p(w|t): the proportion of assignments to topic t over all documents that come from this word w

4) **Reassign w a new topic**, where you **choose topic t** with probability **p(topic t | document d) * p(word w | topic t)**

5) Repeat a large number of times until you hit steady state

Suggested YouTube Video: https://www.youtube.com/watch?v=3mHy4OSyRf0

Notes About LDA

Reassign word w a new topic t with probability

p(topic t | document d) * p(word w | topic t)

- this is essentially the **probability that topic t generated word w**

After repeating a large number of times this reaches a roughly steady state where assignments are “pretty good”.

Use these assignments to estimate:

1) The **topic mixtures of each document**

- by counting the proportion of words assigned to each topic within that document.

2) The **words associated to each topic** - **by counting the proportion of words assigned to each topic overall**.

LDA Software

- **Ski-kit LDA Library**

  http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html

- **Details about the Ski-kit implementation that uses an online variational Bayes Algorithm**

  http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html

More Resources and References:

1. http://cseweb.ucsd.edu/~elkan/250B/topicmodels.pdf

2. https://rstudio-pubs-static.s3.amazonaws.com/79360_850b2a69980c4488b1db95987a24867a.html

3. https://pypi.python.org/pypi/lda

4. https://de.dariah.eu/tatom/topic_model_python.html

5. https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/

6. http://www.kdnuggets.com/2016/07/text-mining-101-topic-modeling.html (this one is only OK)
7. http://www.tfidf.com/