

# Learning Objectives

After this lesson, you should be able to:

- Understand what *latent variables* are
- Understand the uses of *latent variables* in language processing
- Use the *word2vec* and LDA algorithms of *gensim*

# End-of-Course/Projects Countdown

Unit Project 4	June 21; due today
Final Project, Part 4	June 30; due in 1.5 weeks
Final Project, Part 5	July 8; due in 2.5 weeks



DS

# Announcements and Exit Tickets

A black circle containing the white text "DS".

DS

Q & A

A black circle containing the white text "DS".

DS

# Review



**DS**

# Today

# Here's what's happening today:

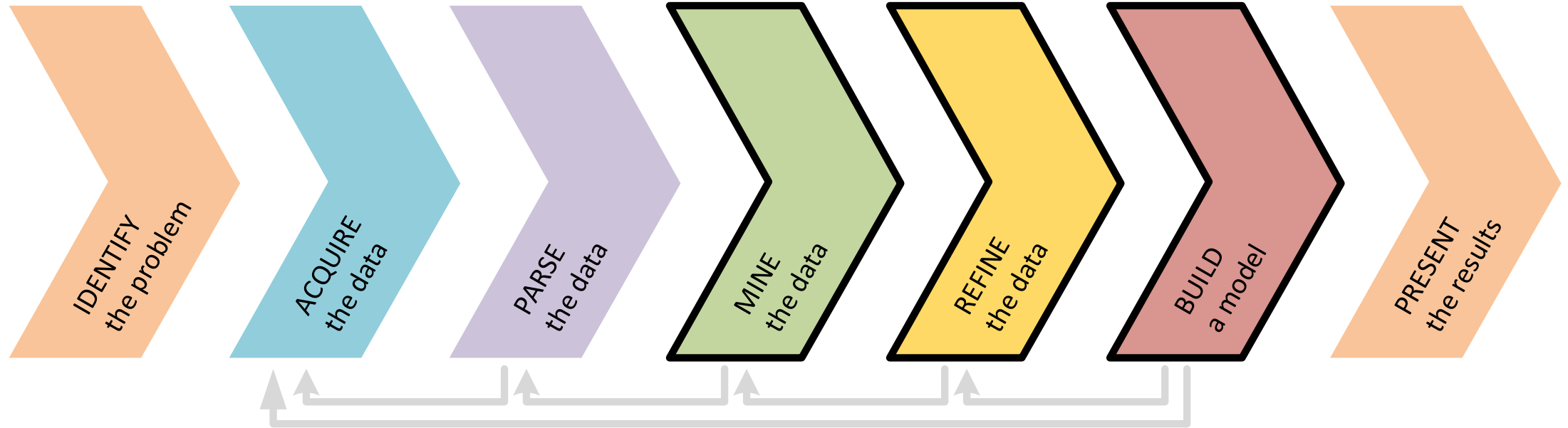
- Unit Project 4 (due today)
- Announcements and Exit Tickets
- Review
- Latent Variable Models
- Dimensionality Reduction
- Dimensionality Reduction in Text Representation
  - Principal Component Analysis (PCA) and Latent Semantic Indexing (LSI)
- Mixture Models and Language Processing
  - Latent Dirichlet Allocation (LDA)
  - Word2Vec
- Independent Practice
  - Tweets and Word2Vec
  - Bonus – Capture your own tweets and redo your work on these
- Office hours in class for final projects
- Review

# Where does Natural Language Processing fits in the course?

<b>Research Design and Data Analysis</b>	Research Design	Data Visualization in <i>pandas</i>	Statistics	Exploratory Data Analysis in <i>pandas</i>
<b>Foundations of Modeling</b>	Linear Regression	Classification Models	Evaluating Model Fit	Presenting Insights from Data Models
<b>Data Science in the Real World</b>	Decision Trees and Random Forests	Time Series Data	Natural Language Processing	Databases



Therefore, today we are refocusing on the **MINE** the data, **REFINE** the data, and **BUILD** a model steps but with a focus on natural language processing





**DS**

# Pre-Work

# Pre-Work

Before this lesson, you should already be able to:

- Install *gensim* with `conda install gensim`
- Recall and apply *unsupervised learning* techniques
- Recall probability distributions, specifically discrete multinomial distributions
- Recall NLP essentials, including experience with *spacy*

DS

# Latent Variable Models

# Latent Variable Models

- *Latent variable models* are different from the *traditional NLP models* in that they try to understand language based on **how** the words are used
  - E.g., instead of learning that ‘bad’ and ‘badly’ are related because they share the same root, we’ll determine that they are related because they are often used in the same way often or near the same words
- We’ll use unsupervised learning techniques to discover patterns or structure in the text to extract information

# Latent Variable Models (cont.)

Traditional NLP Models	Latent Variable Models
<ul style="list-style-type: none"><li>❑ Focused on theoretical understanding of language</li></ul>	<ul style="list-style-type: none"><li>❑ Focused on how the language is actually used in practice</li></ul>
<ul style="list-style-type: none"><li>❑ Tries to learn the rules of a particular language</li></ul>	<ul style="list-style-type: none"><li>❑ Infers meaning from how words are used together</li></ul>
<ul style="list-style-type: none"><li>❑ Preprogrammed set of rules</li></ul>	<ul style="list-style-type: none"><li>❑ Uses unsupervised learning to discover patterns or structure</li></ul>

# Latent Variable Models (cont.)

Traditional NLP Models	Latent Variable Models
<ul style="list-style-type: none"><li>❑ 'bad' and 'badly' are related because they share a common root</li></ul>	<ul style="list-style-type: none"><li>❑ 'bad' and 'badly' are related because they are used the same way or near the same words</li></ul>
<ul style="list-style-type: none"><li>❑ 'Python' and 'C++' are both programming languages because they are often a noun preceded by the verb 'program' or 'code'</li></ul>	<ul style="list-style-type: none"><li>❑ 'Python' and 'C++' are both programming languages because they are often used in the same context</li></ul>

# Latent Variable Models (cont.)

- *Latent variable models* are models in which we assume the data we are observing has some **hidden, underlying structure** that we can't see, but which we'd like to learn
- These hidden, underlying structures are the *latent* (i.e., hidden) variables we want our model to understand



# Text processing is a common application of latent variables

- While language (in the classical sense) is defined by a set of pre-structured grammar rules and vocab, we often break those rules and create new words (e.g., ‘selfie’)
- Instead of attempting to train our model on the rules of proper grammar, we’ll ignore grammar and seek to uncover alternate hidden structures

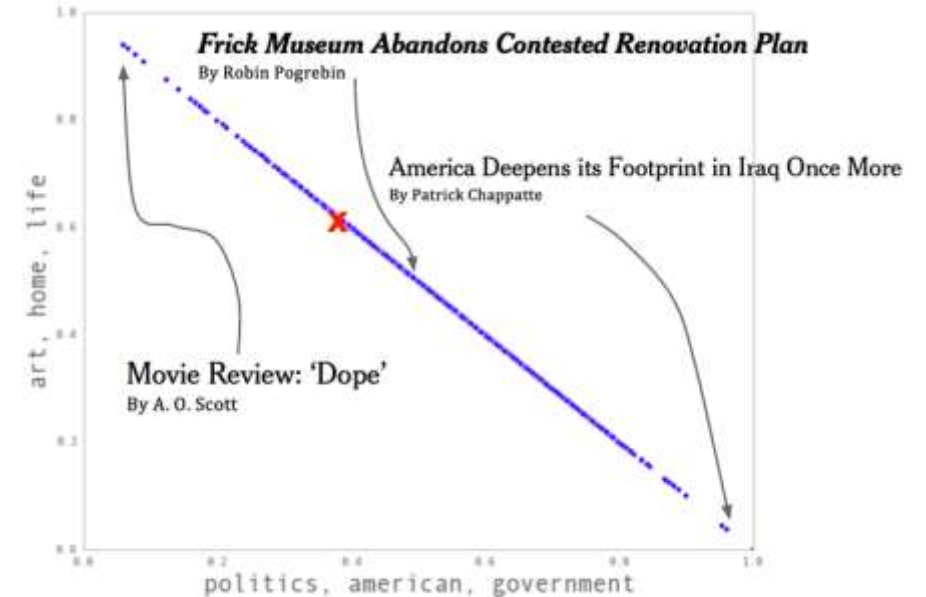
# Topic modeling is another common application of latent variables

- I.e., recommending news articles or mining large troves of data to find commonalities

The New York Times (<http://www.nytimes.com/#recommendations>) uses topic modeling to map their articles to a latent space of topics using the articles content

## EXAMPLE

- 2 latent topics, “Art” and “Politics”
- Politics–Art space
  - “America Deepens its Footprint in Iraq Once More”, as 100% Politics
  - Film review by A.O. Scott as 100% Art
  - “Frick Museum Abandons Contested Renovation Plan” as 50% Politics, 50% Art
- Reader (x) prefers to read about Art 60% of the time and Politics 40% of the time
  - The New York Times will recommend articles that are closest to them in the space








Source: "Building the Next New York Times Recommendation Engine"  
(<http://open.blogs.nytimes.com/2015/08/11/building-the-next-new-york-times-recommendation-engine/>)

Lyst (<https://www.lyst.com/>), an online fashion retailer, uses latent representations of clothing descriptions to find similar clothing

EXAMPLE

- ▶ Looking for items most similar to “boot” mostly return types of boots, as well as some close synonyms to “boot”, like “boots”, “bootie”, and “booties”

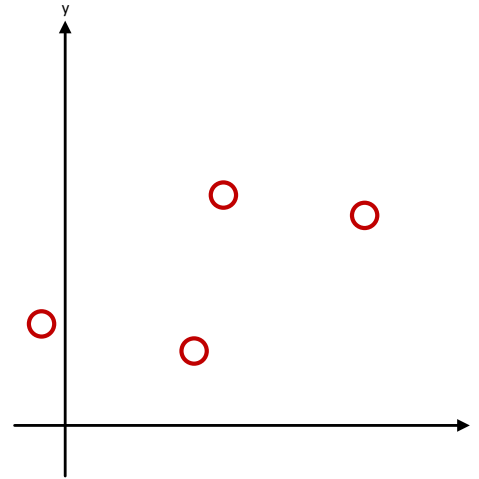
	Term	Similarity
	"riding"	0.962305
	"ankle"	0.888326
	"chelsea"	0.885039
	"gloss"	0.862084
	"combat"	0.861708

Source: “Word Embeddings For Fashion” (<http://developers.lyst.com/2014/11/11/word-embeddings-for-fashion/>)

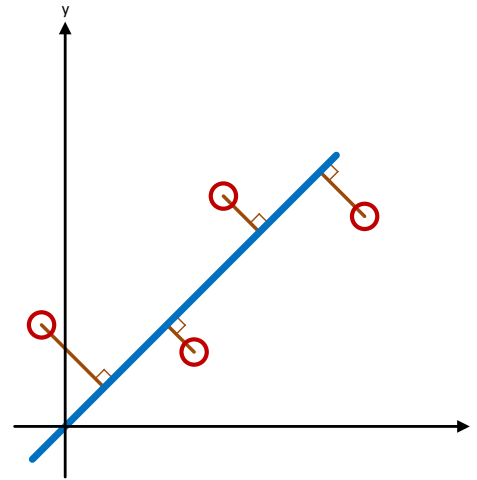
DS

# Dimensionality Reduction

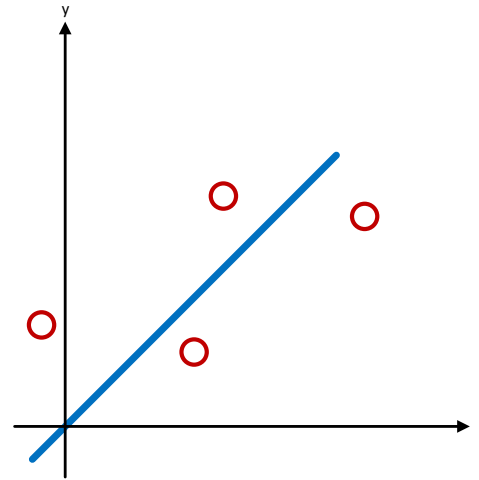
How to reduce dimensionality (2D down to 1D)  
while keeping as much information as possible?



Let's fit a "best" line

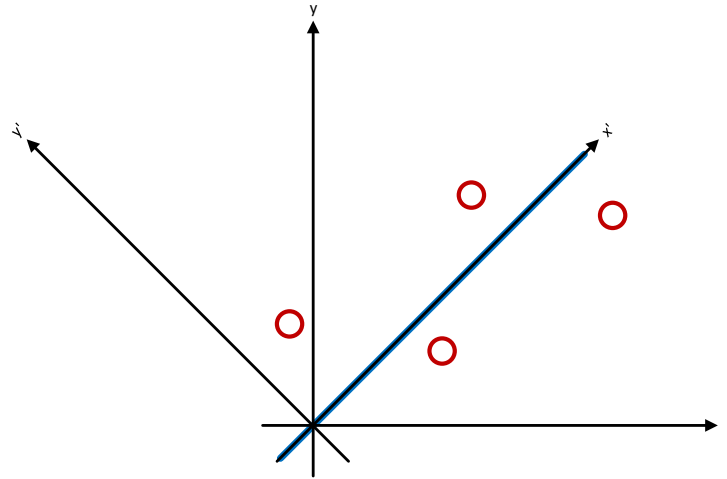


Let's fit a "best" line (cont.)

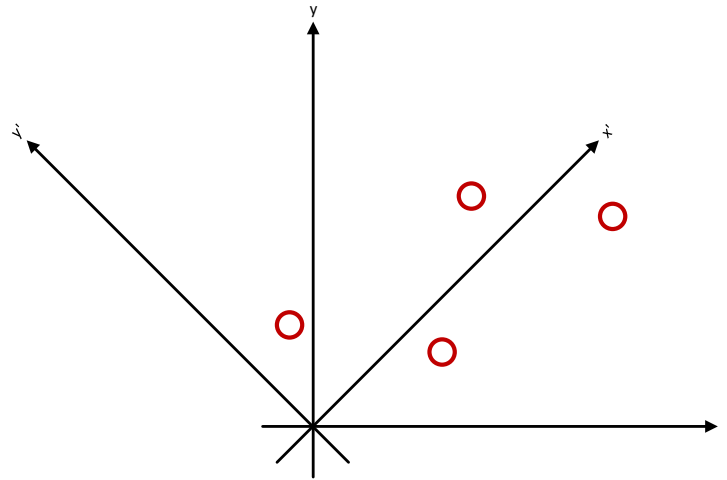




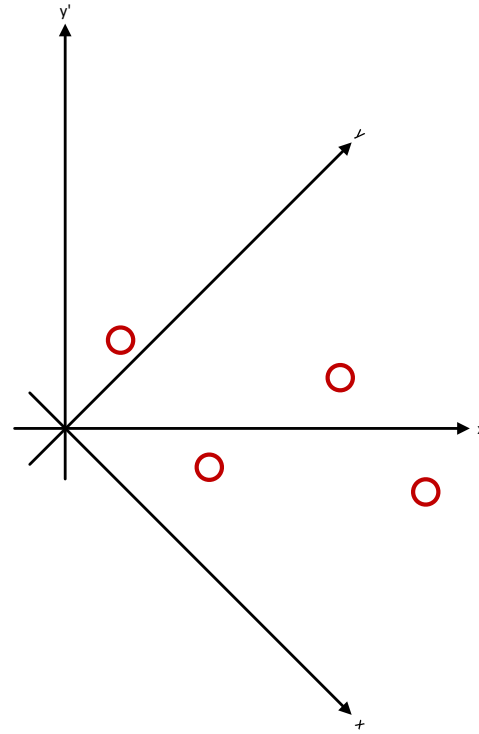
This “best” line define a new coordinate system ( $x'$  along the line and  $y'$  orthogonal to  $x'$ )



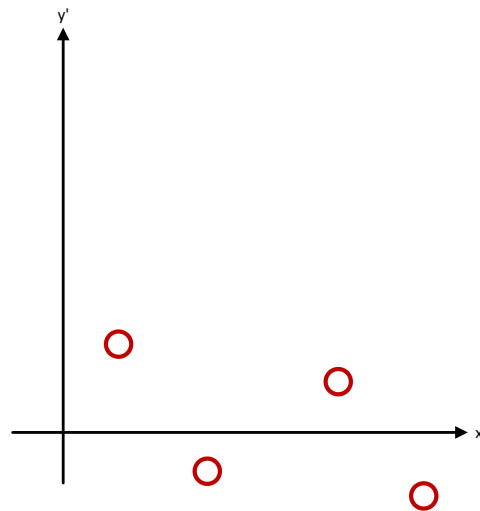
This “best” line define a new coordinate system ( $x'$  along the line and  $y'$  orthogonal to  $x'$ ) (cont.)



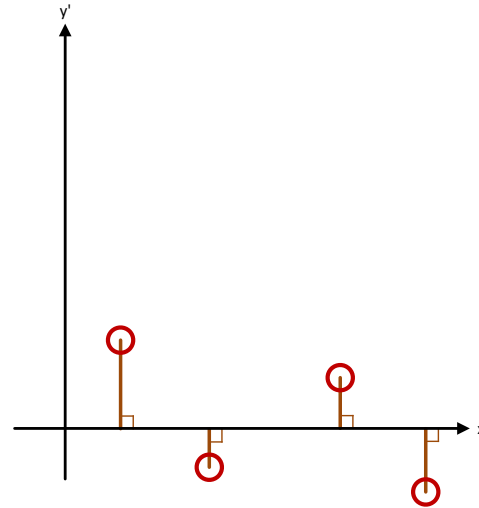
Let's consider the new coordinate system  
 $(x', y')$



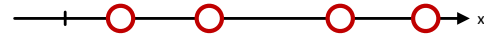
And let's remove the  $(x, y)$  coordinate system



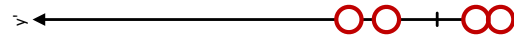
Let's now project the data points into  $x'$



Let's now project the data points into  $x'$   
(cont.)



Let's do the same work for  $y'$

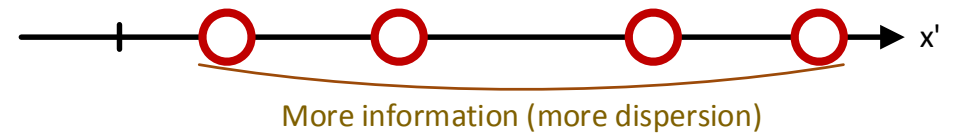
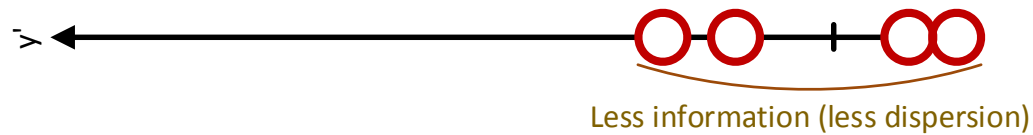


How much information have we “lost” when projecting the data points to  $(x', y')$ ?

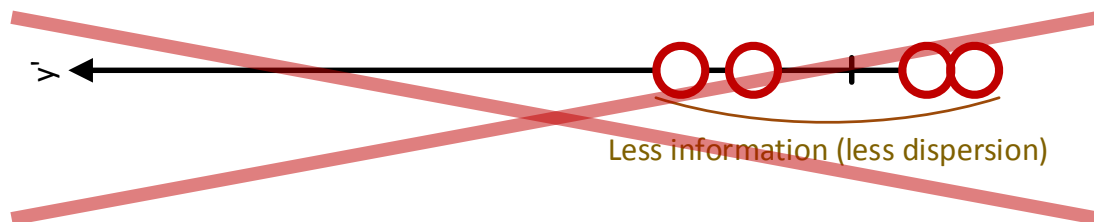




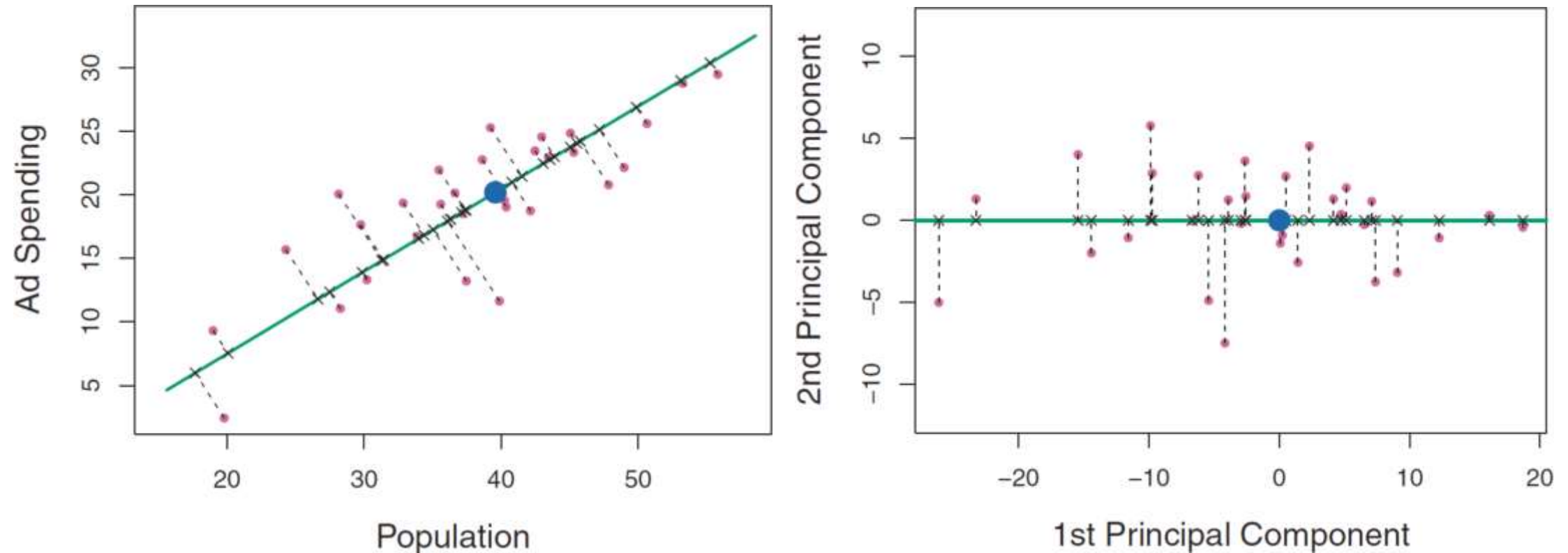
# Which axis should we take if we need to forego the other?



# The one with the “most” information...

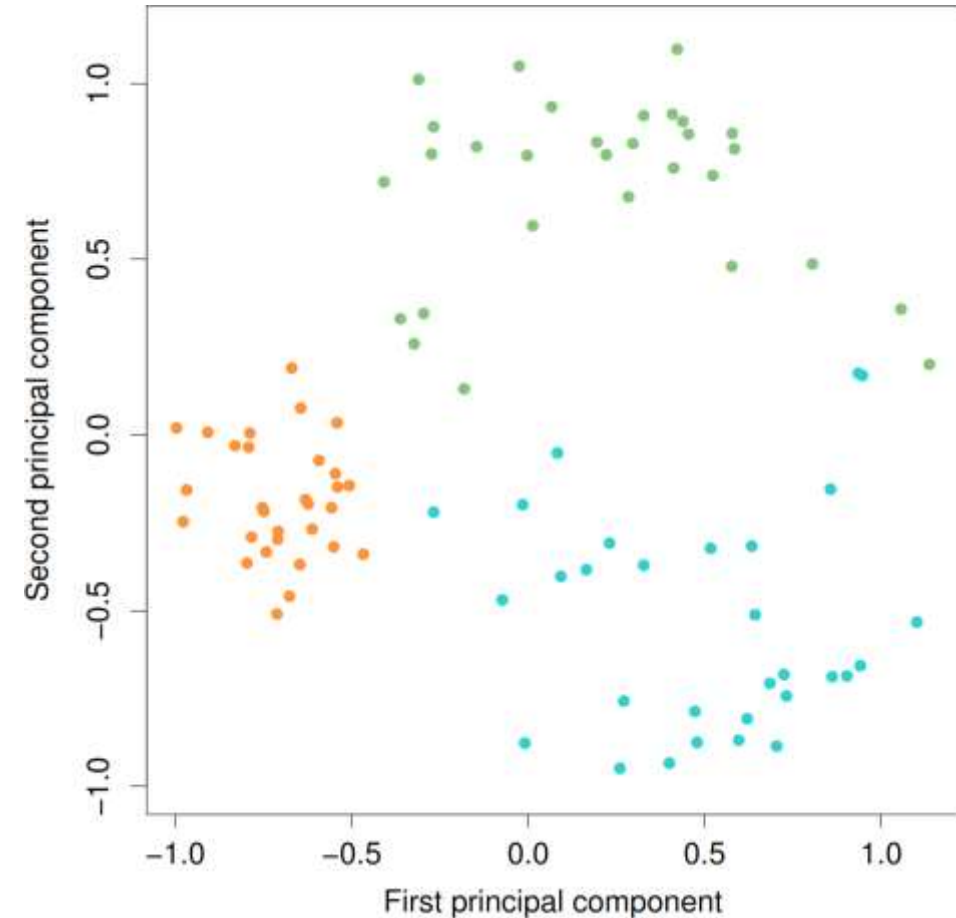
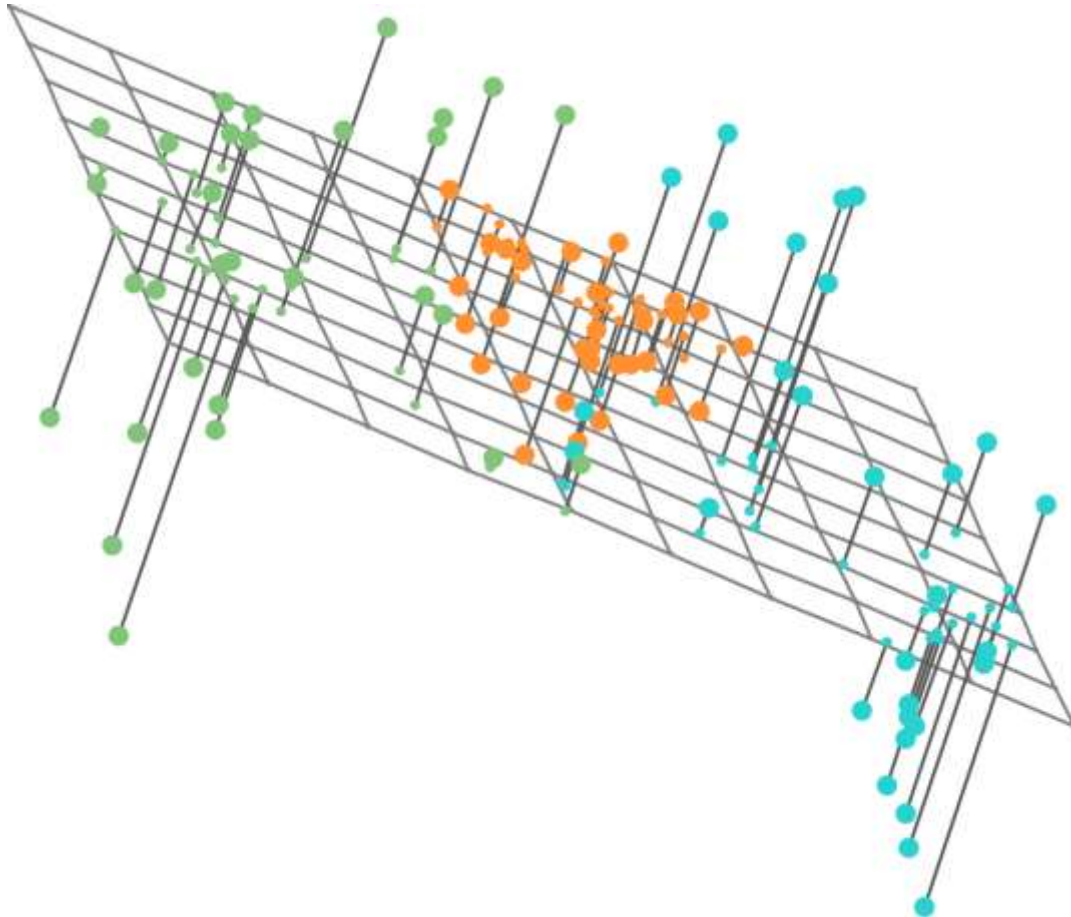


# More examples of dimensionality reduction (cont.)



Source: An Introduction to Statistical Learning with Applications in R

# More examples of dimensionality reduction (cont.)



Source: The Elements of Statistical Learning; Data Mining, Inference, and Prediction



DS

# Dimensionality Reduction in Text Representation

# Dimensionality Reduction in Text Representation

- Our previous representation of a set of text documents (articles) for classification was a matrix with one row per document and one column per word (or n-gram)
- While this sums up most of the information, it does drop a few things, mostly structure and order

# Additionally, many of the columns may be correlated

- E.g., an article that contains the word “IPO” is likely to contain the word “stock” or “NASDAQ”
  - These columns are repetitive and likely to represent the same concept or idea
  - For classification, we may only care that there are finance-related words
- One way to deal with this is to perform dimensionality reduction, where we first identify the correlated columns and then replace them with a column that represents the concept they have in common
  - E.g., we could replace the ‘IPO’, ‘stock’, and ‘NASDAQ’ columns with a single column ‘HasFinancialWords’ column

DS

# Dimensionality Reduction in Text Representation

*Principal Component Analysis (PCA) and Latent Semantic Indexing (LSI)*



# Principal Component Analysis (PCA)

- The techniques vary in how they define correlation and how much of the relationship between the original and new columns you need to save
- One of the most common is Principal Component Analysis (PCA)
- PCA helps reduce the feature space into fewer dimensions
- PCA, when applied to text data, is sometimes known as Latent Semantic Indexing (LSI)



DS

# Mixture Models and Language Processing

Mixture models take the dimensionality reduction concept further to generate more structure around document

- Mixture models assume that each document is some mixture of topics
  - E.g., an article may be mostly science but may contain some business information
- The latent structure we want to uncover are the topics (or concepts) that generate that text
- Mixture models create clusters of common words and generate probability distributions to explicitly state how related words are

DS

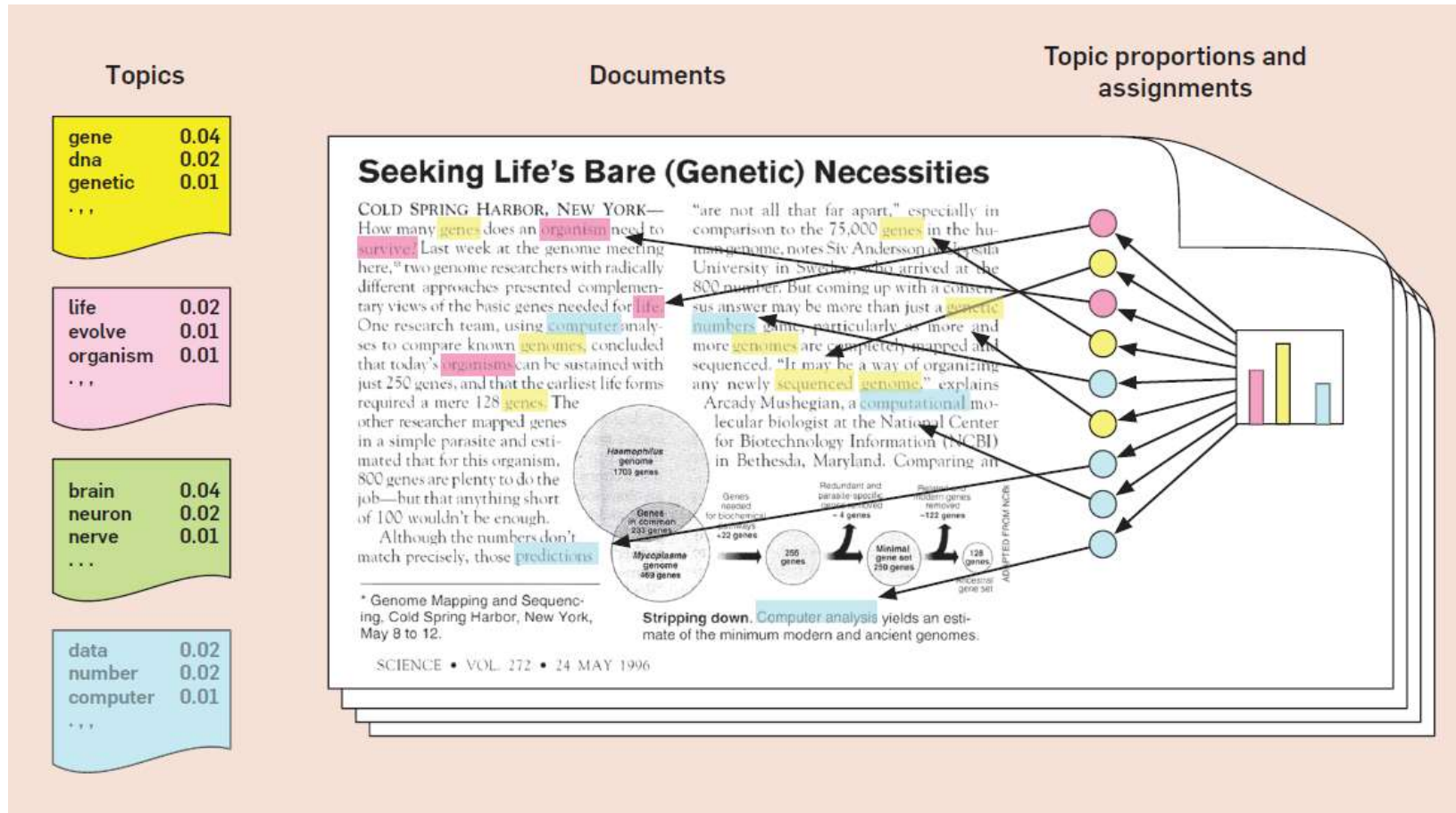
# Mixture Models and Language Processing

*Latent Dirichlet Allocation (LDA)*

Latent Dirichlet Allocation (LDA) is an unsupervised mixture model for latent variable NLP

- LDA attempts to learn two things:
  - The word distribution of each topic
  - The topic distribution of each document

# Mixture Models (cont.)



# The word distribution of each topic

- The word distribution is a multinomial distribution of each topic representing what words are most likely from that topic
- For each word and topic pair, we learn some probability:  $P(\text{word} \mid \text{topic})$
- E.g., we have three topics: sports, business, and science
- For each topic, we uncover the most likely words to come from them
  - sports→[football: .3, basketball: .2, baseball: .2, touchdown: .02, ..., genetics: .0001]
  - science→[genetics: .2, drug: .2, ..., baseball: .0001]
  - business→[stocks: .1, ipo: .08, ..., baseball: .0001]

# The topic distribution of each document

- The topic distribution is a multinomial distribution for each document representing what topics are most likely to appear in that document
- For each topic and document pair, we learn some probability:  
 $P(\text{topic} \mid \text{document})$

- E.g., for all our of sample documents, we have a distribution over sports, science, and business
  - ESPN→[sports: .8, business: .2, science: .0]
  - Bloomberg→[business: .7, science: .2, sports: .1]



# Mixture Models (cont.)

- Topic models are useful for organizing a collection of documents and uncovering the main underlying concepts
- There are many variants that attempt to add even more structure:
  - Supervised topic models guide the process with pre-decided topics
  - Position-dependent topic models ignore which words occur in which document and instead focus on where they occur
  - Variable number topic models test different numbers of topics to find the best model

A black circle containing the white text 'DS' in a bold, sans-serif font.

DS

# Mixture Models and Language Processing

*Demo / Latent Dirichlet Allocation (LDA) with gensim*

DS

# Mixture Models and Language Processing

*Word2Vec*

# Word2Vec is another unsupervised model for latent variable NLP

- Word2Vec model creates word vectors, multidimensional representations of words
- This is similar to having a distribution of concepts or topics that the word may come from

▸ E.g., `assembly` → [ .12315,  
.23425, .89745324,  
.235234, .234234, ...]

# Documents as features of a specific word

- If we transpose our usual document-word matrix, instead of talking about words as being features of a document, we can talk about documents as being features of a specific word
- In other words, how do we define or characterize a single word?
  - We can do so by defining its dictionary definition
  - Or we can enumerate all of the ways we might use it

# E.g., Paris

- Given the word 'Paris', we have many contexts or uses we may find it in:
  - ['\_ is the capital of', '\_ , France', 'the capital city \_', 'the restaurant in \_', ...]
- There are also a bunch of contexts we don't expect to find it in:
  - ['can I have a \_', 'there's too much \_ on this', ...]
- We could make a feature or column for each of these contexts and even represent 'Paris' in a sparse feature with all possible contexts

## E.g., Paris (cont.)

- In fact, the first few examples represent the same concept:
  - Paris is a city like thing, so it contains shops and restaurants
  - Paris is a capital city
- We would want to use dimensionality reduction to find a *few* concepts per word instead of *all* possible contexts

# Word2Vec (cont.)

- With LDA, we could do this by identifying the topics a word was most likely to come from
- With Word2Vec, we will replace the overlapping contexts by some concept that represents them
- Like other techniques, our goal is to identify correlated columns and replace them with a new column that represents those replaced columns
- We would replace the [ '\_ is a city', '\_ is a capital', 'I flew into \_ today' ] columns by a single column, 'IsACity'



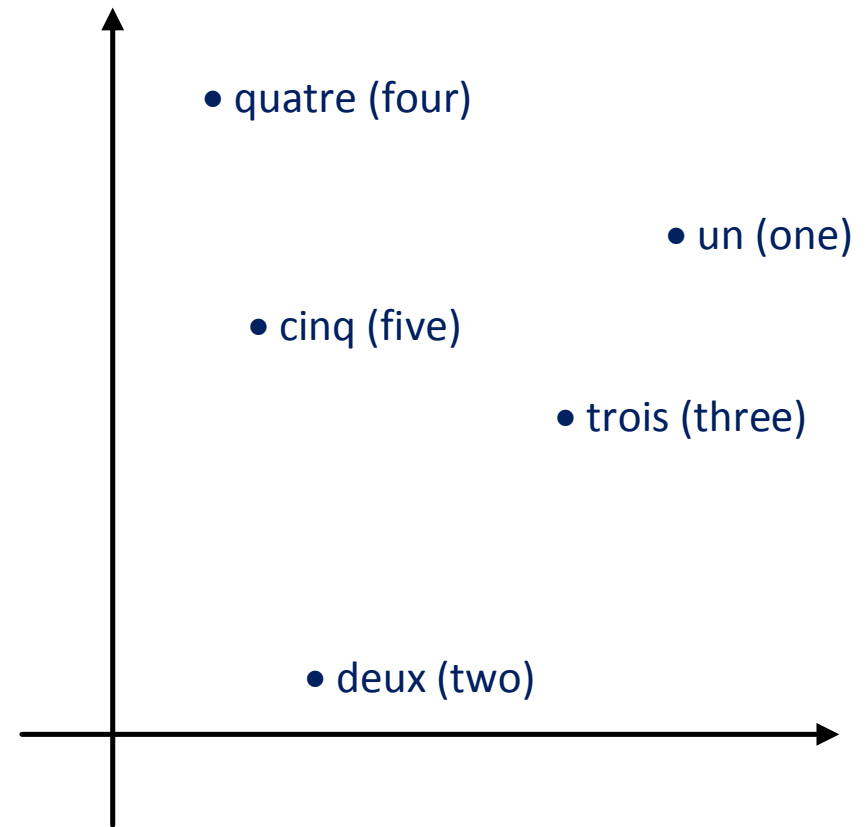
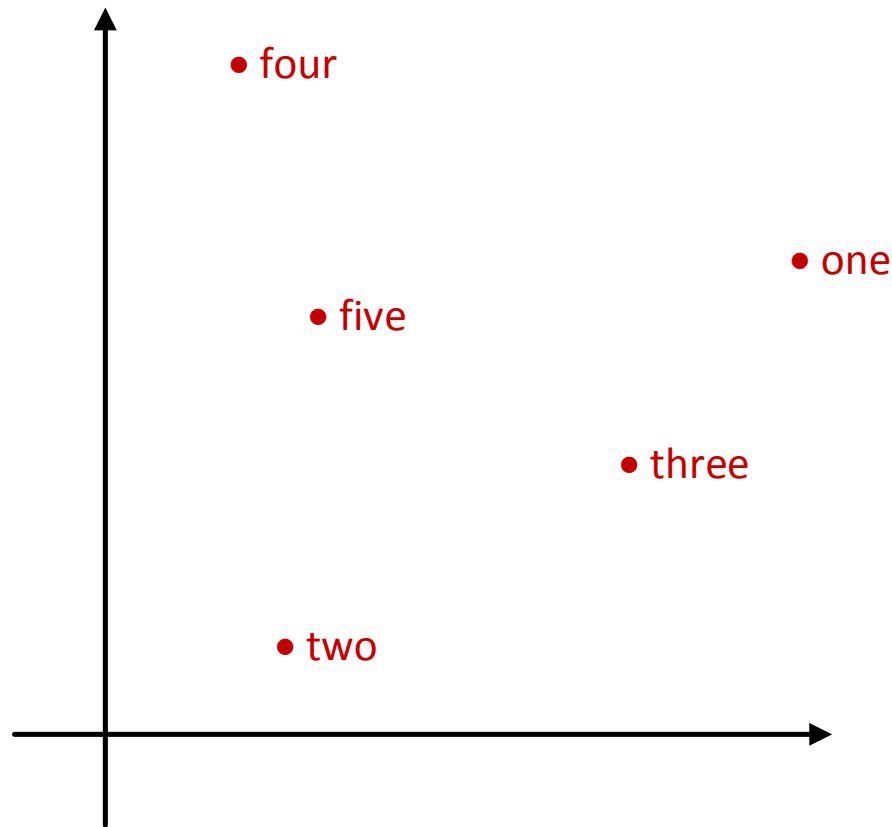
# Word2Vec (cont.)

- ▶ With a trained model, Word2Vec can be used for many tasks
- ▶ A commonly used feature of Word2Vec is being able to ask what words are similar to each other

- ▶ For example, if you ask for words similar to 'france', you would get:

```
spain 0.678515  
belgium 0.665923  
netherlands 0.652428  
italy 0.633130  
switzerland 0.622323  
luxembourg 0.610033  
portugal 0.577154
```

# Using data for other languages, Word2Vec could also be used for translation



DS

# Mixture Models and Language Processing

*Demo / Word2Vec with gensim*



**DS**

# Independent Practice

# Independent Practice | Tweets and Word2Vec



## EXERCISE

### DIRECTIONS

1. Build a *Word2Vec* model of the tweets (tweets.txt under datasets) we have collected using *gensim*
  - a) First take the collection of tweets and tokenize them
    - Think about how this should be done
    - Should you only use upper-case or lower-case?
    - Should you remove punctuations or symbols?
  - b) Then, build a *Word2Vec* model

# Independent Practice | Tweets and Word2Vec (cont.)



## EXERCISE

### DIRECTIONS (cont.)

- c) Finally, test your *Word2Vec* model with a few similarity functions
- Find words similar to 'Syria'
  - Find words similar to 'war'
  - Find words similar to 'Iran'
  - Find words similar to 'Verizon'
  - Adjust the choices in (b) and (c) as necessary

### DELIVERABLE

Answers to the above questions

# Independent Practice | Bonus – Capture your own tweets and redo your work on these



## EXERCISE

### SETTING UP TWITTER API CREDENTIALS

1. Go to <https://apps.twitter.com/>
2. Sign in, and follow the instructions below:
3. Press “Create New App”
4. Fill in form (you can enter your website or any website in the 'website' field, e.g. <http://google.com>)
5. Press “Create application”
6. Press “Manage keys”
7. Press “Create access tokens”
8. Find and save the following:
  - a) Consumer Key
  - b) Consumer Secret
  - c) Access Token Key
  - d) Access Token Secret

# Independent Practice | Bonus – Capture your own tweets and redo your work on these (cont.)



## EXERCISE

### RUNNING THE CODE

1. Install TwitterAPI: `conda install TwitterAPI`
2. Substitute the four values saved from 'Setting up Twitter API credentials' in `twitter.py` in  
  

```
access_token_key = "<ENTER ACCESS TOKEN KEY>"  
access_token_secret = "<ENTER ACCESS TOKEN SECRET>"  
api_key = "<ENTER CONSUMER KEY>"  
api_secret = "<ENTER CONSUMER SECRET>"
```
3. Run `python capture-tweets.py <topic>` to save tweets to file called `captured-tweets.txt` related to `<topic>`. E.g.: `python capture-tweets.py Google` or `python capture-tweets.py Iran`



**DS**

# Review

# Review

- Latent variable models attempt to uncover structure from text
- Dimensionality reduction is focused on replacing correlated columns
- Topic modeling (or LDA) uncovers the topics that are most common to each document and then the words most common to those topics
- Word2Vec builds a representation of a word from the way it was used originally
- Both techniques avoid learning grammar rules and instead rely on large datasets. They learn based on how the words are used, making them very flexible

**DS**

Q & A



**DS**

# Before Next Class

# Before Next Class

Before the next lesson, you should already be able to:

- Load data with *pandas*
- Plotting data with *seaborn*
- Understand correlation

# Next Class

*Time Series*

# Learning Objectives

After the next lesson, you should be able to:

- Understand what time series data is and what is unique about it
- Perform time series analysis in *pandas* including rolling mean/median and autocorrelation



DS

# Exit Ticket

*Don't forget to fill out your exit ticket [here](#)*