GENERAL ASSEMBLY

LATENT VARIABLES AND NLP

Jonathan Balaban

DAT2

LEARNING OBJECTIVES

- Understand *latent* variables
- Understand the uses of *latent variables*
- Use the *word2vec* and *LDA* algorithms of `gensim`

PRE-WORK REVIEW

- Install `gensim` with `pip install gensim`
- Define probability distributions, specifically discrete multinomial distributions
- Recall NLP essentials, including experience with `spacy`

LATENT VARIABLE MODELS

- This lesson will continue on natural language processing with an emphasis on *latent variables models*.

- Mining and Refining data is a key part of the data science workflow.

- In our last class, we saw many techniques for mining the data, including preprocessing, building linguistic rules to uncover patterns, and creating classifiers from unstructured data.

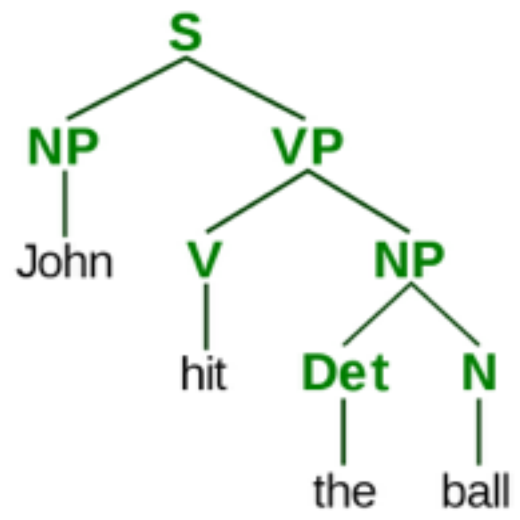
- In this class, we'll continue with methods to Refine our understanding of the text by attempting to uncover structure or organization in the text.

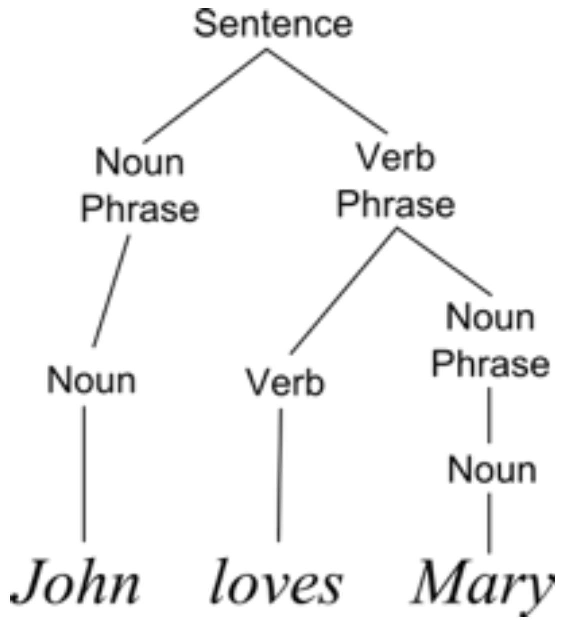
LATENT VARIABLE MODELS

- Many advances in NLP are based on using data to learn rules of grammar and language.
    - Tokenization
    - Stemming or lemmatization
    - Parsing and tagging
- Each of these are based on a classical or theoretical understanding of language.

LATENT VARIABLE MODELS

- Tokenization:
  - John hit the ball → [John, hit, the, ball]
  - Where did you go → [Where, did, you, go]
- Stemming or lemmatization: shouted → shout, better → good
- Parsing and tagging:





LATENT VARIABLE MODELS

- *Latent variable models* are different in that they try to understand language based on **how** the words are used.
- For example, 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* techniques (discovering patterns or structure) to extract the information.

LATENT VARIABLE MODELS

Traditional NLP Models	Latent Variable Models
Focused on theoretical understanding of language	Focused on how the language is actually used in practice
Tries to learn the rules of a particular language	Infers meaning from how words are used together
Preprogrammed set of rules	Uses unsupervised learning to discover patterns or structure

LATENT VARIABLE MODELS

Traditional NLP Models

‘bad’ and ‘badly’ are related because they share a common root.

‘Python’ and ‘C++’ are both programming languages because they are often a noun preceded by the verb ‘program’ or ‘code’.

Latent Variable Models

‘bad’ and ‘badly’ are related because they are used the same way or near the same words.

‘Python’ and ‘C++’ are both programming languages because they are often used in the same context.

LATENT VARIABLE MODELS

LATENT VARIABLE MODELS

- *Latent variable models* are models in which we assume the data we are observing has some **hidden, underlying structure** that we can't see, and 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.

LATENT VARIABLE MODELS

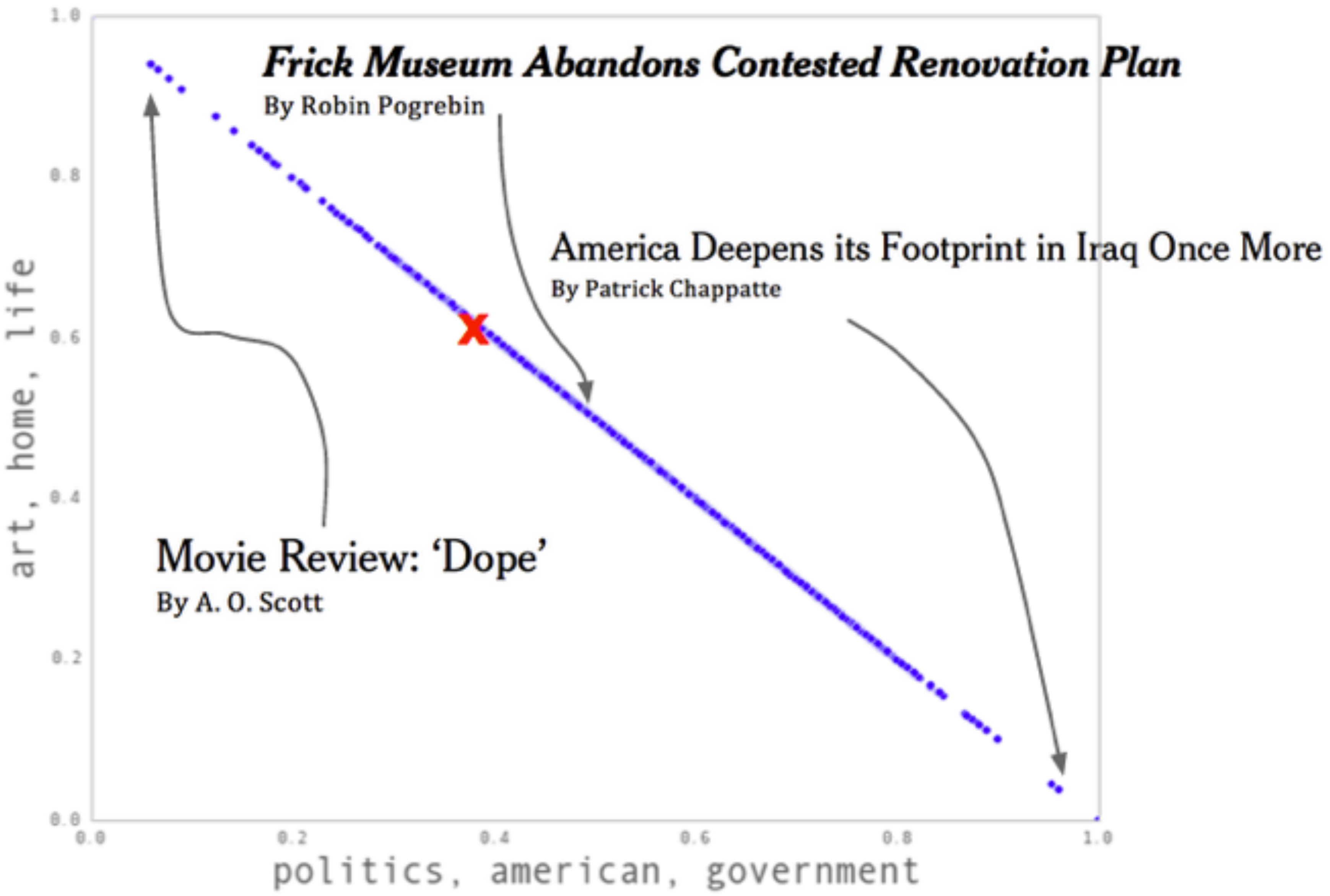
- While language (in the classical sense) is defined by a set of pre-structured grammar rules and vocabulary, we often break those rules and create new words (e.g. selfie).

- Instead of attempting to train our model on the rules of proper grammar, we'll ignore grammar and seek to uncover alternate hidden structures.

LATENT VARIABLE MODELS

- Latent variable techniques are often used for recommending news articles or mining large troves of data to find commonalities.
- Topic modeling, a method we'll cover today, is used in the NY times recommendation engine.
- The New York Times attempts to map their articles to a latent space of topics using the content of the article.

LATENT VARIABLE MODELS



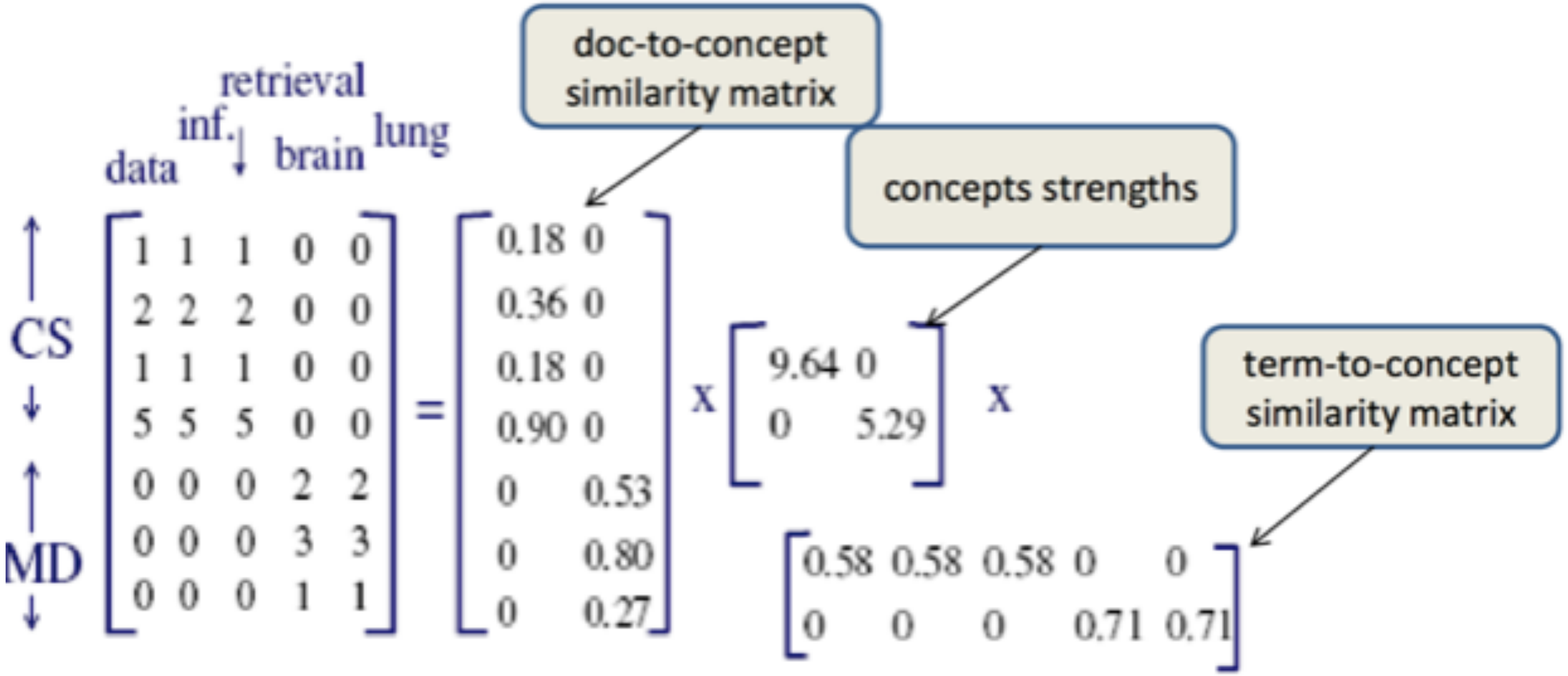
LATENT VARIABLE MODELS

- Lyst, an online fashion retailer, uses latent representations of clothing descriptions to find similar clothing.

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

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).



DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- 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.

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- For example, an article that contains the word ‘IPO’ is likely to contain the word ‘stock’ or ‘NASDAQ’.
- Therefore, those columns are repetitive and likely to represent the same concept or idea.
- For classification, we may only care that there are finance-related words.

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- One way to deal with this is through regularization - L1/Lasso regularization tends to remove repetitive features by bringing their learned coefficients to 0.

- Another is to perform *dimensionality reduction* ,where we first identify the correlated columns and the replace them with a column that represents the concept they have in common.

- For instance, we could replace ‘IPO’, ‘stocks’, and ‘NASDAQ’ with a single column - ‘HasFinancialWords’ column.

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- There are many techniques to do this automatically and most follow a very similar approach.

    a. Identify correlated columns.

    a. Replace them with a new column that encapsulates the others.

Doc #	Car	Truck	Van	Dog
6344	1	1	1	0
6345	0	1	1	1
6346	1	1	1	0

Doc #	Vehicle	Dog
6344	1	0
6345	1	1
6346	1	0

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- The techniques vary in how they define correlation and how much of the relationship between the original and new columns you need to save.

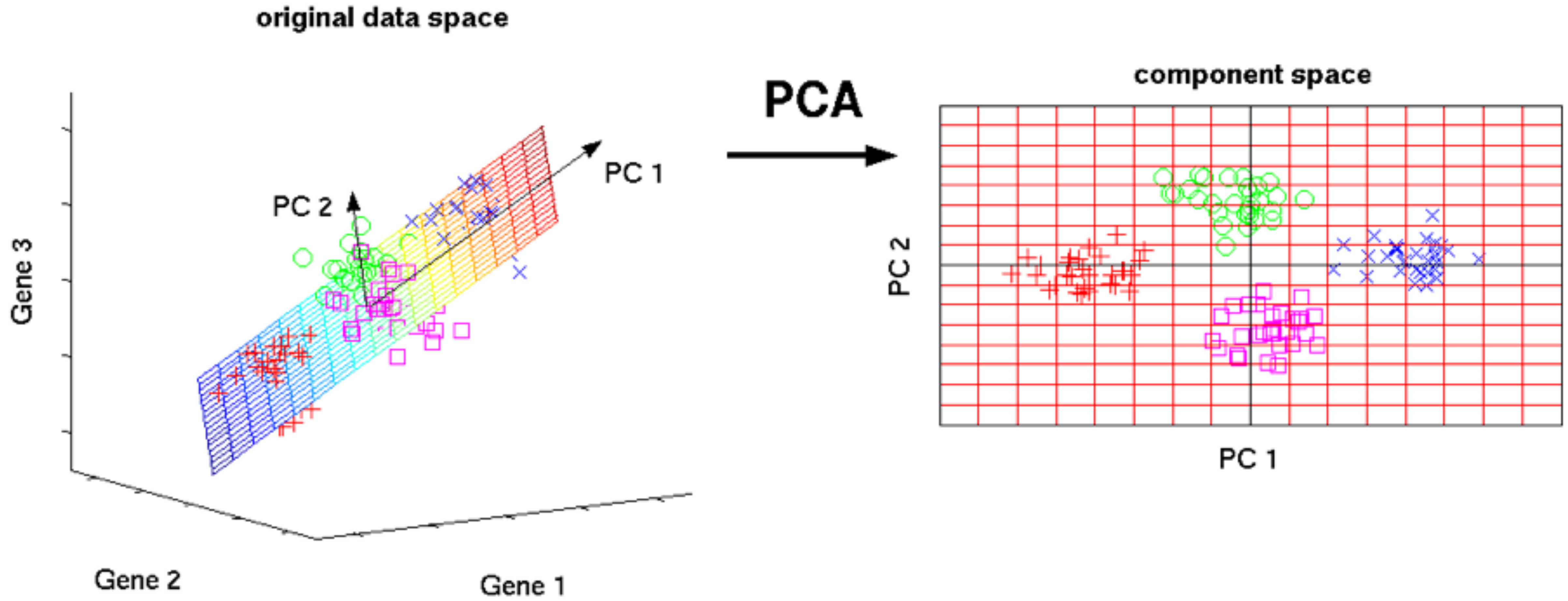
- Dimensionality techniques can vary between *linear* and *non-linear*.

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- There are many techniques build into scikit-learn.
- One of the most common is **Principal Component Analysis** (**PCA**).
- PCA, when applied to text data, is sometimes known as **Latent Semantic Indexing** (**LSI**).

DIMENSIONALITY REDUCTION IN TEXT REPRESENTATION

- PCA helps reduce the feature space into fewer dimensions.



MIXTURE MODELS AND LANGUAGE PROCESSING

- Mixture models (specifically **LDA** or **Latent Dirichlet Allocation**) take this concept further and generate more structure around the documents.

- Instead of just replacing correlated columns, we create clusters of common words and generate probability distributions to explicitly state how related words are.

MIXTURE MODELS AND LANGUAGE PROCESSING

- To understand this better, let’s imagine a new way to generate text:

    a. Start writing a document

        i. Choose a topic (sports, news, science).

        ii. Choose a random word from that topic.

        iii. Repeat.

    b. Repeat for the next document.

MIXTURE MODELS AND LANGUAGE PROCESSING

- This ‘model’ of text is assuming that each document is some *mixture* of topics.

- It 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.

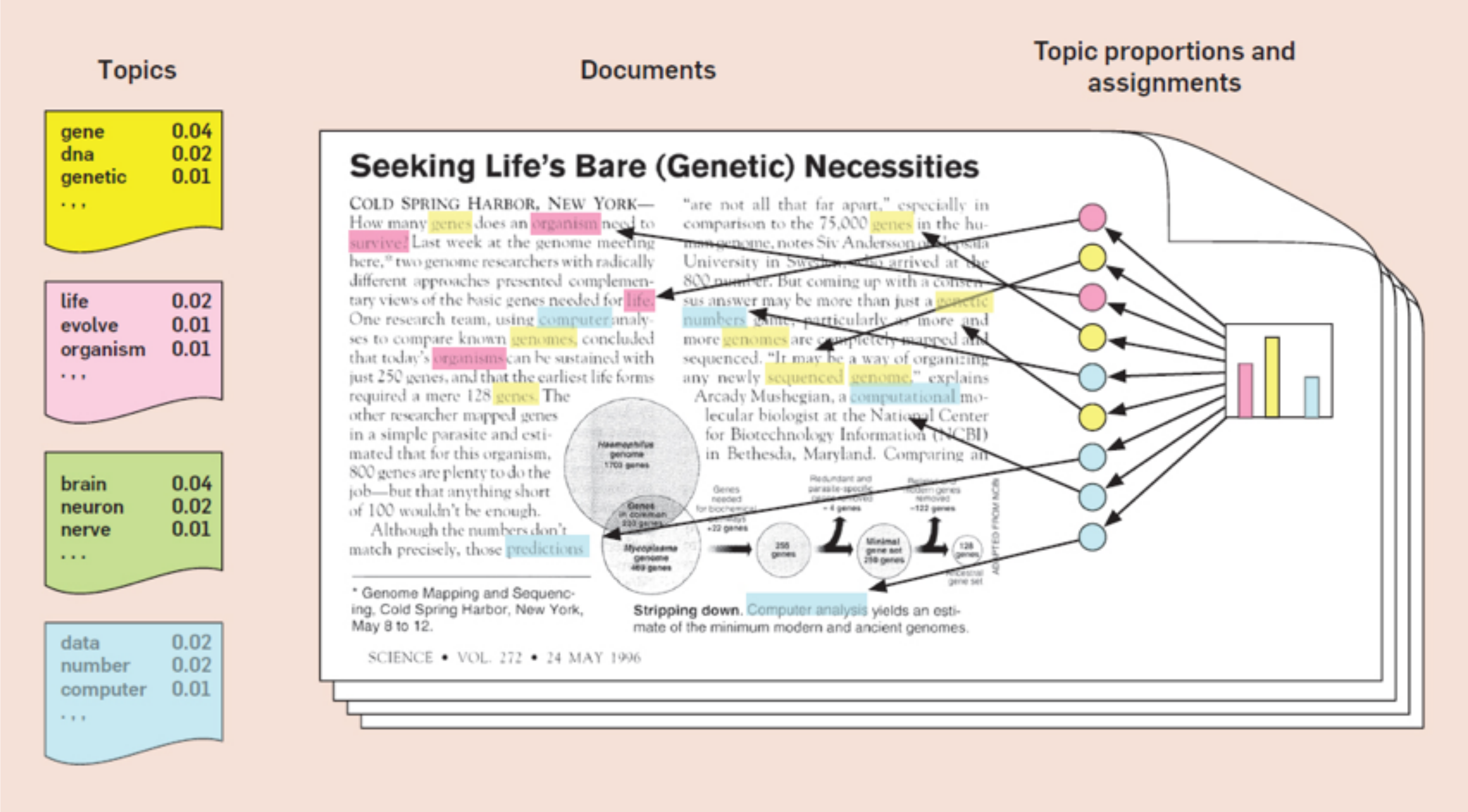
MIXTURE MODELS AND LANGUAGE PROCESSING

- *Latent Dirichlet Allocation* is a model that assumes this is the way text is generated and then attempts to learn two things:

    a. The word distribution of each topic

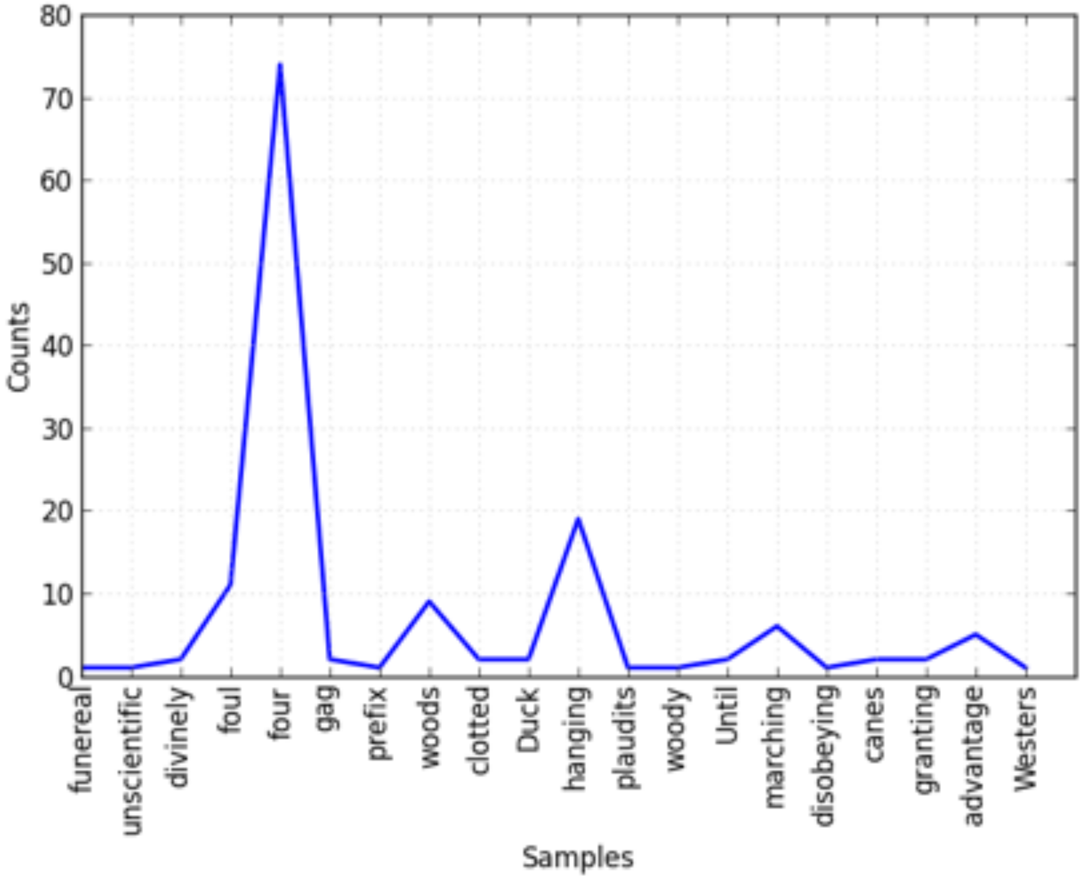
    a. The topic distribution of each document.

MIXTURE MODELS AND LANGUAGE PROCESSING



MIXTURE MODELS AND LANGUAGE PROCESSING

- The *word distribution* is a multinomial distribution of each topic representing what words are most likely from that topic.



MIXTURE MODELS AND LANGUAGE PROCESSING

- For example, let's say we have three topics: sports, business, and science.

- For each topic, we uncover the most likely words to come from them:

```
sports: [football: 0.3, basketball: 0.2, baseball: 0.2, touchdown: 0.02 ... genetics: 0.0001]
science: [genetics: 0.2, drug: 0.2, ... baseball: 0.0001]
business: [stocks: 0.1, ipo: 0.08,  ... baseball: 0.0001]
```

- For each word and topic pair, we learn some probability: `P(word|topic)`.

MIXTURE MODELS AND LANGUAGE PROCESSING

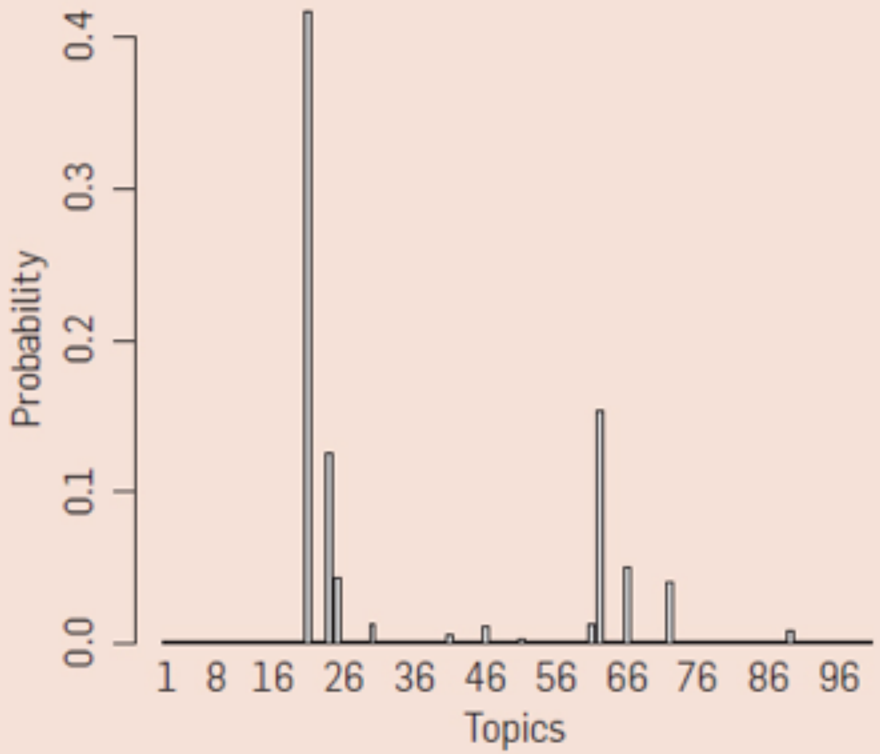
- The *topic distribution* is a multinomial distribution for each document representing what topics are most likely to appear in that document.

- For all our of sample documents, we have a distribution over {sports, science, business}.

```
ESPN article: [sports: 0.8, business: 0.2, science: 0.0]
Bloomberg article: [business: 0.7, science: 0.2, sports: 0.1]
```

- For each topic and document pair, we learn some probability, `P(topic|document)`.

MIXTURE MODELS AND LANGUAGE PROCESSING



"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

MIXTURE MODELS AND LANGUAGE PROCESSING

- 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 to the ‘model’:

    1. Supervised topic models guide the process with pre-decided topics.
    2. Position-dependent topic models ignore which words occur in which document and instead focus on *where* they occur.
    3. Variable number topic models test different numbers of topics to find the best model.

DEMO

LDA IN GENSIM

LDA IN GENSIM

- `gensim` is a library of language processing tools focused on latent variable models of text.
- It was originally developed by grad students dissatisfied with current implementations of latent models.
- Documentation and tutorials are available on the package's website.

LDA IN GENSIM

- Let’s first translate a set of documents (articles) into a matrix representation with a row per document and a column per feature (word or n-gram).

LDA IN GENSIM

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(binary=False,
                     stop_words='english',
                     min_df=3)

docs = cv.fit_transform(data.body.dropna())

# Build a mapping of numerical ID to word
id2word = dict(enumerate(cv.get_feature_names()))
```

LDA IN GENSIM

- We want to learn which columns are correlated (i.e. likely to come from the same topic).
- This is the *word distribution*.
- We can also determine what topics are in each document, the *topic distribution*.

LDA IN GENSIM

```python
from gensim.models.ldamodel import LdaModel
from gensim.matutils import Sparse2Corpus

# First we convert our word-matrix into gensim's format
corpus = Sparse2Corpus(docs, documents_columns = False)

# Then we fit an LDA model
lda_model = LdaModel(corpus=corpus, id2word=id2word, num_topics=15)
```

LDA IN GENSIM

- In this model, we need to explicitly specify the number of topic we want the model to uncover.

- This is a critical parameter, but there isn't much guidance on how to choose it.
  - Try to use domain expertise where possible.

LDA IN GENSIM

- Now we need to assess the *goodness of fit* for our model.
- Like other unsupervised learning techniques, our validation techniques are mostly about interpretation.
- Use the following questions to guide you:
  - Did we learn reasonable topics?
  - Do the words that make up a topic make sense?
  - Is this topic helpful towards our goal?

LDA IN GENSIM

- We can evaluate fit by viewing the top words in each topic.
- `gensim` has a `show_topics` function for this:

```
num_topics = 25
num_words_per_topic = 5
for ti, topic in enumerate(lda.show_topics(num_topics = num_topics,
num_words_per_topic = n_words_per_topic)):
    print("Topic: %d" % (ti))
    print (topic)
    print()
```

LDA IN GENSIM

- Some topics will be clearer than others. The following topics represent clear concepts:

```
0.009*cup + 0.009*recipe + 0.007*make + 0.007*food + 0.006*sugar
→ Cooking and Recipes
```

```
0.013*butter + 0.010*baking + 0.010*dough + 0.009*cup + 0.009*sugar
→ Cooking and recipes
```

```
0.013*fashion + 0.006*like + 0.006*dress + 0.005*style
→ Fashion and Style
```

INTRODUCTION

WORD2VEC

WORD2VEC

- *Word2Vec* is another unsupervised model for latent variable NLP.
- It was originally released by Google and further refined at Stanford.
- This model creates *word vectors*, multidimensional representations of words.

```
assembly → [0.12315, 0.23425, 0.89745324, 0.235234, 0.234234, …]
```

- This is similar to having a distribution of concepts or topics that the word may come from.

WORD2VEC

- If we take our usual document-word matrix and take its transpose, 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.

WORD2VEC

- 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' ... and millions more]
```

- We could make a feature or column for each of these contexts.

- We could represent ‘Paris’ in a sparse feature with all possible contexts.

WORD2VEC

- Additionally, 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 want to use **dimensionality reduction** to find a *few* concepts per word instead of *all* possible contexts.

WORD2VEC

- 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 can replace the [‘_ is a city’, ‘_ is a capital’, ‘I flew into _ today’] columns by a single column, ‘IsACity’.

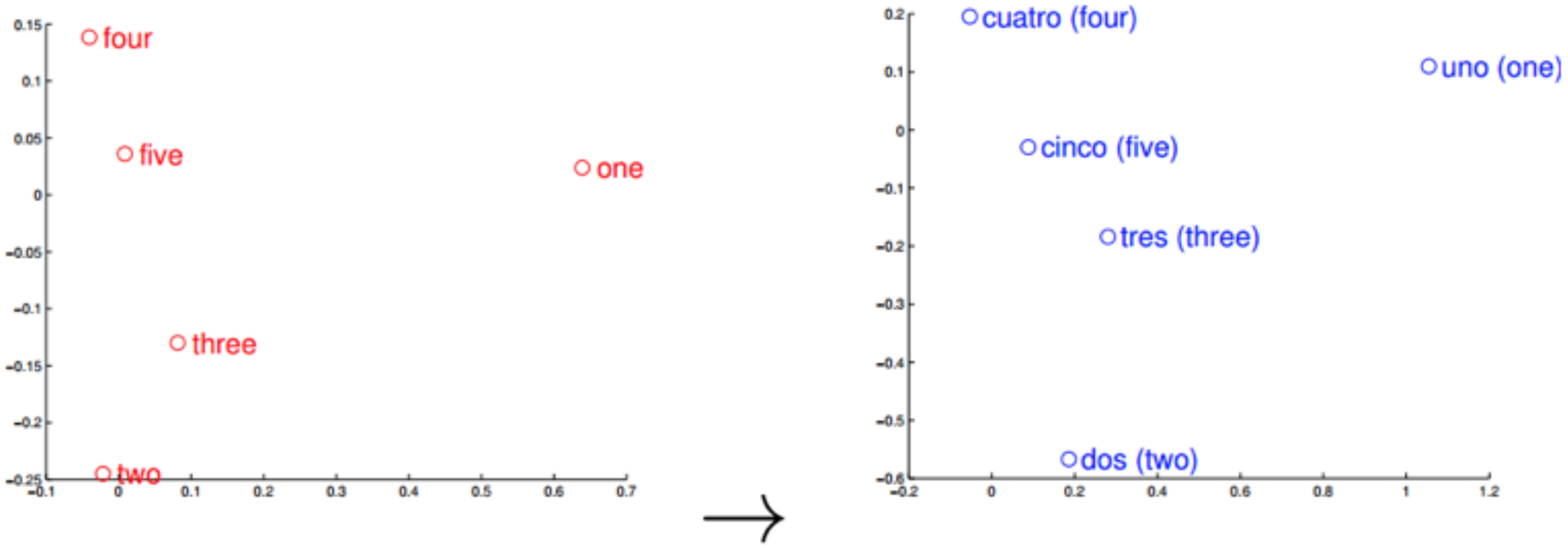
WORD2VEC

- 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
```

WORD2VEC

- If we have data for other languages, Word2Vec could also be used for translation.



WORD2VEC IN GENSIM

- We will build a Word2Vec model using the body text of the articles available in the StumbleUpon dataset.

```
from gensim.models.word2vec import Word2Vec


# Setup the body text
text = data.body.dropna().map(lambda x: x.split())


from gensim.models import Word2Vec
model = Word2Vec(text, size=100, window=5, min_count=5, workers=4)
```

WORD2VEC IN GENSIM

- The `Word2Vec` class has many arguments.
    - `size` represents how many concepts or topics we should use.
    - `window` represents how many words surrounding a sentence we should use as our original feature.
    - `min_count` is the number of times that context or word must appear.
    - `workers` is the number of CPU cores to use to speed up model training.

WORD2VEC IN GENSIM

- The model has a most_similar function that helps find the words *most similar* to the one you queried.

- This will return words that are most often used in the same context:

```
model.most_similar(positive=['cookie', 'brownie'])
```

- It can easily identify words related to those from this dataset.

INDEPENDENT PRACTICE

TWITTER LAB

ACTIVITY: TWITTER LAB

EXERCISE

DIRECTIONS (45 minutes)

In this exercise, we will compare some of the classical NLP tools from the last class with these more modern latent variable techniques. We will do this by comparing information extraction on Twitter using two different methods.

NOTE: There is a pre-existing file of captured tweets you can use. It is located in the class repo for lesson-14. However, you can also *collect your own tweets* following the instructions in twitter-instructions.md.

ACTIVITY: TWITTER LAB

EXERCISE

STARTER CODE

Refer to the starter code provided in the class repository for lesson-14.

LOADING THE DATA

```
tweets = [tweet for tweet in open('../../assets/dataset/captured-tweets.txt', 'r')]
```

SETTING UP SPACY

```
from spacy.en import English
nlp_toolkit = English()
```

ACTIVITY: TWITTER LAB

EXERCISE

TASKS AND QUESTIONS

1. Use `spacy` to write a function to filter tweets down to those where Google is announcing a product. How might we do this? One way might be to identify verbs, where 'Google' is the noun and there is some action like 'announcing'
   a. Write a function that can take a sentence parsed by spacy and identify if it mentions a company named 'Google'. Remember,spacy can find entities and code them as ORG if they are a company.
   b. BONUS: Make this function work for any company.
   c. Write a function that can take a sentence parsed by spacy and return the verbs of the sentence (preferably lemmatized).
   d. For each tweet, parse it using spacy and print it out if the tweet has 'release' or 'announce' as a verb.
   e. Write a function that identifies countries. **HINT**: the entity label for countries is GPE (or "GeoPolitical Entity").
   f. Re-run (d) to find country tweets that discuss 'Iran' announcing or releasing.

ACTIVITY: TWITTER LAB

EXERCISE

TASKS AND QUESTIONS

1. Build a word2vec model of the tweets we have collected using gensim.
    a. First take the collection of tweets and tokenize them using spacy.
        i. Think about how this should be done.
        ii. Should you only use upper-case or lower-case?
        iii. Should you remove punctuations or symbols?

    b. Build a word2vec model.
        i. Test the window size as well - this is how many surrounding words need to be used to model a word. What do you think is appropriate for Twitter?

    c. Test your word2vec model with a few similarity functions.
        i. Find words similar to 'Syria'.
        ii. Find words similar to 'war'.
        iii. Find words similar to "Iran".
        iv. Find words similar to 'Verizon'.

    d. Adjust the choices in (b) and (c) as necessary.

ACTIVITY: TWITTER LAB

EXERCISE

TASKS AND QUESTIONS

1. Filter tweets to those that mention 'Iran' or similar entities and 'war' or similar entities.
    a. Do this using just spacy.
    b. Do this using word2vec similarity scores.

CONCEPT REVIEW

- Latent variable models uncover structure and meaning 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 word representation from its original use.
- 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.

LESSON

Q & A

LESSON

EXIT TICKET

DON'T FORGET TO FILL OUT YOUR EXIT TICKET