Session 4: Algorithms on Text Data

2021 July 19

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/

Main application

Application: Analyzing every annual report from 2014

- All 10-K filings in EDGAR in 2014
  - This keeps the data small enough to keep in memory easily, but large enough to get some power in our analyses

- Supervised classification
  - Method from Hassan et al. (2019 QJE)

- Unsupervised classification
  - Word-level: word2vec
  - Document-level: LDA

Supervised methods

The Hassan et al. (2019 QJE) approach

- Just like how we can used data about a phenomenon to supervise algorithm construction with numeric data (i.e., regression), Hassan et al. (2019 QJE) suggests a similar idea based on using text to supervise text.
- The methodology requires 3 sets of textual information:
  1. Data that you want to analyze
  2. Data that represents the information you want to quantify the extent of
  3. Data that represents the rest of the information, e.g., what you **don't** want to quantify

> There is a simple requirement here: what you want and what the baseline text in your file is must be sufficiently different

- The method is mentioned in the computer science literature in Song and Wu (2008) and Schütze et al. (2008)

The study

> Goal: measure political risk

- Data:
  1. Conference call transcripts from 2002 to 2016
  2. Political text: American Politics Today (Bianco and Canon); articles from NYT, USA Today, WSJ, Washington Post on “domestic politics”
  3. Nonpolitical text: Financial Accounting (Libby, Libby and Short); articles from NYT, USA Today, WSJ, Washington Post on “performance,” “ownership changes,” and “corporate actions;” the Santa Barbara Corpus of Spoken American English (excluding politics-related episodes)

> A lot of baseline data is needed! But why?

Other work needed

1. Cleaning up the data
   - Removing a lot of bi-grams based on part-of-speech tags that are unlikely to be relevant
   - Removing Bi-grams with: i, ve, youve, weve, im, youre, were, id, youd, wed, thats
   - Removing “princeton university”
2. Removing 3 synonyms for risk due to contextual differences: questions, question, venture

What do they do with the data?

- They construct a list of *bi-grams* (2 word phrases) such that
  - Each bi-gram appears in the political baseline
  - Each bi-gram *never* appears in the nonpolitical baseline
- They will weight words accordingly
- They will measure risk by using these weights paired with phrases where a synonym for risk is nearby.

Top political bi-grams (weight)

1. the constitution
2. the states
3. public opinion
4. interest groups
5. of government

Top risk words (frequency)

1. risk
2. risks
3. uncertainty
4. variable
5. chance

Building the measure

- Let $\mathbb{N}$ be the set of bi-grams that are non-political
- Let $\mathbb{P}$ be the set of bi-grams that are political
- b is a bi-gram in the set of bi-grams in the set $\mathbb{B}$
- Let the set $\mathbb{B}$ ($\mathbb{P}$) have B (P) elements
- Let r_b be the closest risk word to b
- Let $f_{b,\mathbb{P}}$ be the number of times that b appears in $\mathbb{P}$

Total discussion of politics

$$Politics_{i,t} = \frac{\sum_{b \in \mathbb{B}_{i,t}} \mathbf{I}\left[b \in \mathbb{P} \backslash \mathbb{N}\right] \times \frac{f_{b,\mathbb{P}}}{P}}{B_{i,t}}$$

Building the measure

- Let $\mathbb{N}$ be the set of bi-grams that are non-political
- Let $\mathbb{P}$ be the set of bi-grams that are political
- b is a bi-gram in the set of bi-grams in the set $\mathbb{B}$, valued by its position in the document
- Let the set $\mathbb{B}$ ($\mathbb{P}$) have B (P) elements
- Let r_b be the closest risk word to b, valued by its position in the document
- Let $f_{b,\mathbb{P}}$ be the number of times that b appears in $\mathbb{P}$

Total discussion of political risk

$$PRisk_{i,t} = \frac{\sum_{b\in\mathbb{B}_{i,t}|\{r_b\in\mathbb{B}_{i,t}\}} \mathbf{I}\left[b \in \mathbb{P}\backslash\mathbb{N}\right] \times \mathbf{I}\left[|b - r_b| < 10\right] \times \frac{f_{b,\mathbb{P}}}{P}}{B_{i,t}}$$

Benefits of the method

1. More complete than a dictionary approach
2. Very clean approach given that political discussion should be fairly different from other discussion in annual reports
3. Generally applicable for any easy to pick out discussion
   - So long as you can find training data

What do we need to know to implement it?

1. How to chunk text into bi-grams
2. How to tokenize text
   - Done in Session 3 ✓
3. How to count words or phrases
   - Use a `Counter()` ✓

> Optional advanced stuff: You can vectorize most of the calculation and just use matrix algebra with `numpy`

Workflow

- Set up blacklists

```python
word_blacklist = "i i've you've we've i'm you're we're i'd you'd we'd that's".split(' ')
pattern_blacklist = ["PRP|PRP", "IN|IN", "RB|RB", "WRB|RB", "IN|RB", "RB|IN",
                     "IN|WRB", "WRB|IN", "DT|IN", "IN|DT", "RB|WRB", "RB|DT",
                     "DT|RB", "WRB|DT", "DT|WRB", "SYM|SYM"]
gram_blacklist = 'princeton|university'
```

- Define the main function for cleaning

```python
def grammer(doc, n, processed_patterns, word_blacklist, gram_blacklist, lower=True, stopwor
    if not stopword:
        grams = textacy.extract.ngrams(doc, n=n, filter_stops=False, filter_nums=True)
    else:
        grams = textacy.extract.ngrams(doc, n=n, filter_stops=True, filter_nums=True)
    ngrams = Counter()
    for gram in grams:
        pos = '|'.join([word.tag_ for word in gram])
        if not lower:
            text = '|'.join([word.text for word in gram])
        else:
            text = '|'.join([word.text for word in gram]).lower()
```

Process a document

- We'll use the same data as Session 3

```python
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])
nlp.max_length = 10000000

with open('../../Data/0001104659-14-015152.txt', 'rt') as f:
    text = f.read()

document = nlp(text)
grams = grammer(document, n=2, processed_patterns=pattern_blacklist,
                word_blacklist=word_blacklist,
                gram_blacklist=gram_blacklist)

# Intermediary measures
gram_count = sum(grams.values())
gram_set = set(grams)
```

What is this `set()`?

- Sets in python are an interesting and rather useful structure
- Like lists, they contain a bunch of objects, such as text in our case
- Unlike lists, they do not have an order and cannot contain duplicates
- Also unlike lists, they are very fast to query
  - E.g., if you ask if something is in a very large set, the response is quick
- We can apply set functions to them!
  - `set1 & set2` represents the intersection of the two sets
    - Much faster than `[i for i in list1 if i in list2]`
  - `set1 | set2` represents the union

Applying a hypothetical dictionary

- The hypothetical weighted dictionary:

```
weights = {'earnings|foreign':0.5, 'currency|foreign':0.4, 'foreign|currencies':0.35, 'fore
           'foreign|currency':0.25, 'foreign|investment':0.2, 'foreign|holdings':0.2}
weight_set = set(weights)
```

- Use set intersection to quickly get the overlap:

```
shared_keys = list(gram_set & weight_set)
```

- Determine the aggregate weight of the overlapping text

```
ns = len(shared_keys)
v_weights = np.empty(ns)
v_counts = np.empty(ns)
c = 0
for key in shared_keys:
    v_weights[c] = weights[key]
    v_counts[c] = grams[key]
```

Finalize the measure

```python
measure = spec_weight / gram_count if gram_count > 0 else 0
measure
```

```
## 0.0004479123801082587
```

- Note that this exercise shows you how to calculate a simple disclosure score, not the risk score from Hassan et al. (2019 QJE)
  - For the risk score, you need to replace the counts by a count of times the bi-gram was within 10 words of a risk word

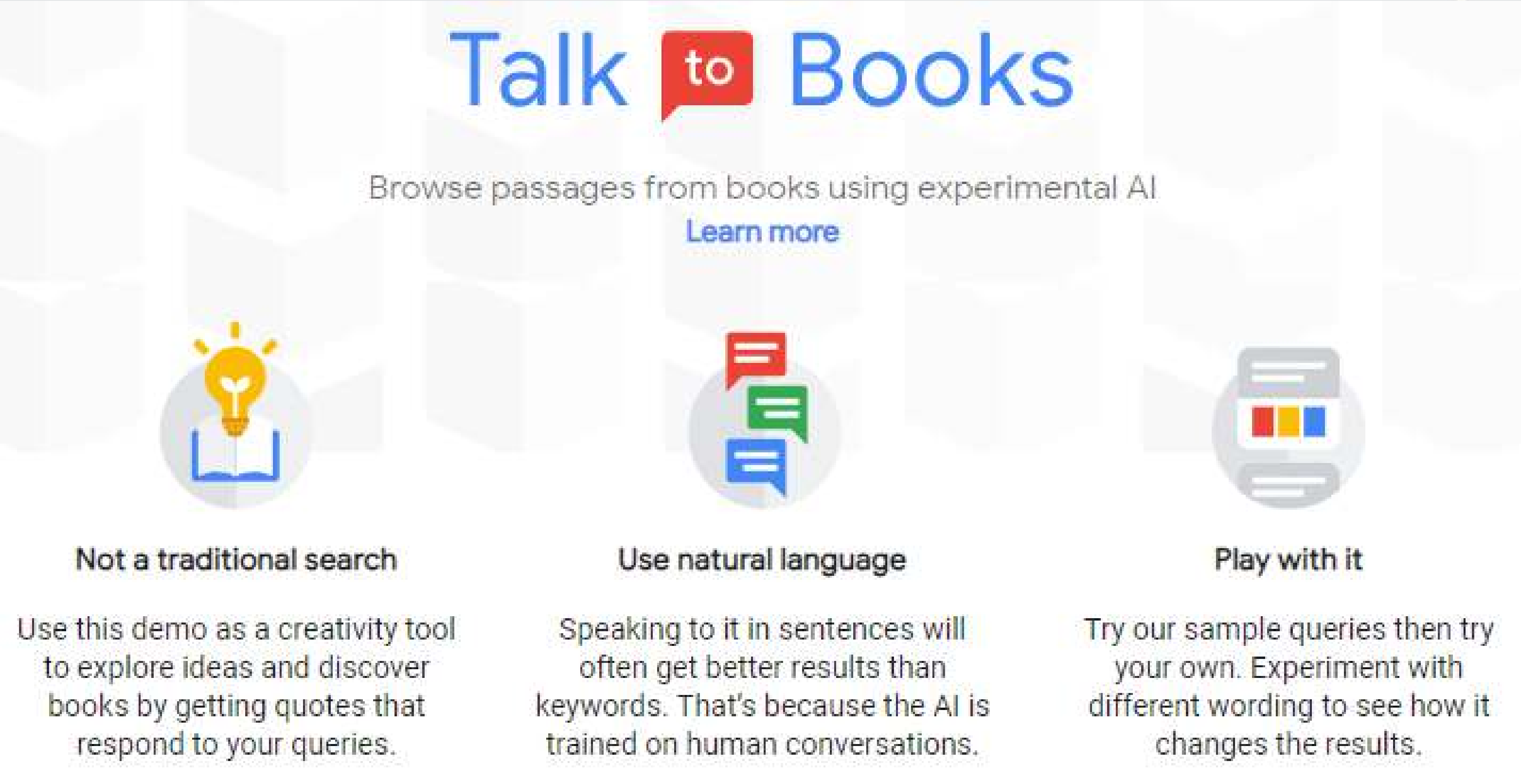
Exercise

> Do Set 1 in the `Session 4-Exercises` file

- This goes through a simplified version of the same calculation
- In fact, the exercise is pretty much just cosine-similarity under an L_1 distance metric

Unsupervised methods

A motivating example



What are unsupervised methods?

> Unsupervised methods try to find some patterns in data without being told what exactly it is that they should find

- Examples of what can be accomplished:
  1. Word meaning
     - Calculate similarity of words
       - Useful for finding synonyms or comparing text snippets
  2. Sentence or phrase meaning
     - Useful for directly comparing sentence or paragraph similarity
  3. Document classification
     - Useful for clustering documents
  4. Content grouping (topic analysis)
     - Useful for simplistic text summarization in a quantified manner

What are “vector space models”

- Different ways of converting some abstract information into numeric information
  - Focus on maintaining some of the underlying structure of the abstract information
- Examples (in chronological order):
  - Word vectors:
    - Word2vec
    - GloVe
  - Paragraph/document vectors:
    - Doc2Vec
  - Sentence vectors:
    - Universal Sentence Encoder
  - Topic vectors:
    - Latent Dirichlet Allocation (LDA)

Word vectors

- Instead of coding individual words, encode word meaning
- The idea:
  - Our old way (encode words as IDs from 1 to N) doesn't understand relationships such as:
    - Spatial
    - Categorical
    - Grammatical (weakly when using stemming)
    - Social
    - etc.

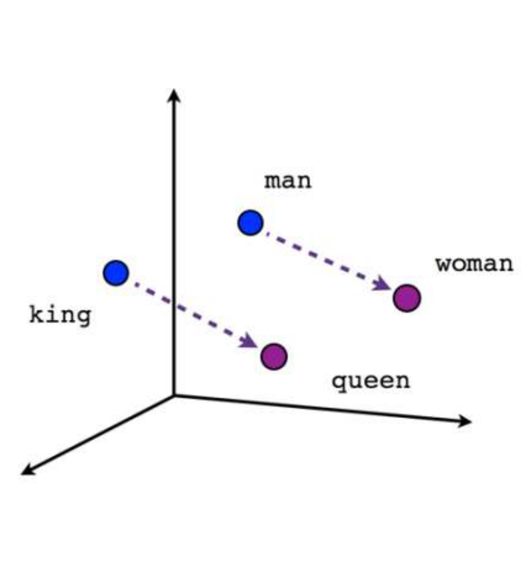
> Word vectors try to encapsulate all of the above implicitly, through by encoding words as a vector based on how features manifest themselves in text

Word vectors: Simple example

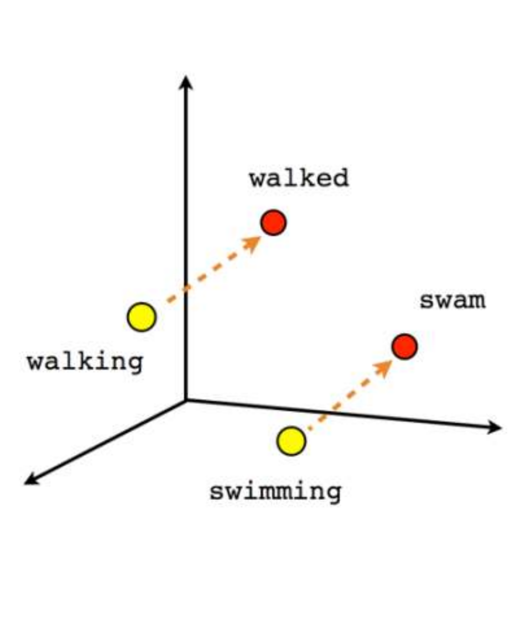
words	f_animal	f_people	f_location
dog	0.5	0.3	-0.3
cat	0.5	0.1	-0.3
Bill	0.1	0.9	-0.4
turkey	0.5	-0.2	-0.3
Turkey	-0.5	0.1	0.7
Singapore	-0.5	0.1	0.8

- The above is a simplified illustrative example
- Notice how we can tell apart different animals based on their relationship with people
- Notice how we can distinguish turkey (the animal) from Turkey (the country) as well

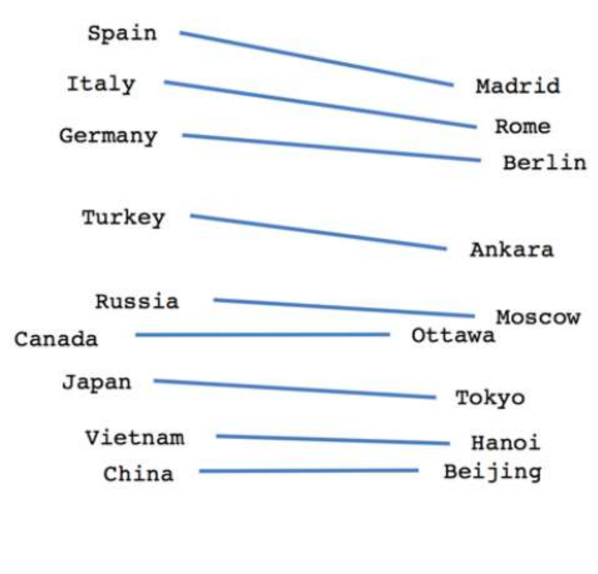
What it retains: word2vec



Male-Female

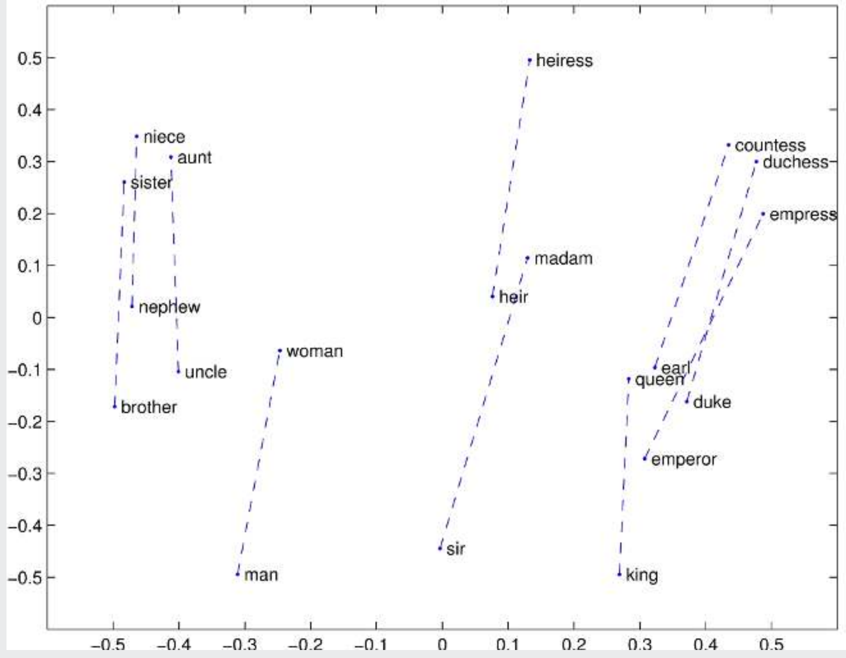


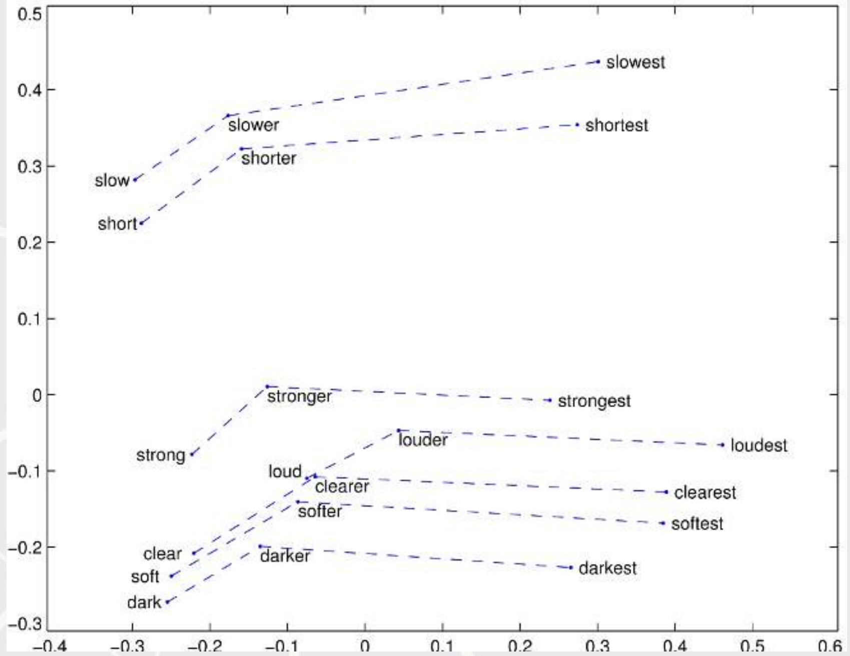
Verb tense

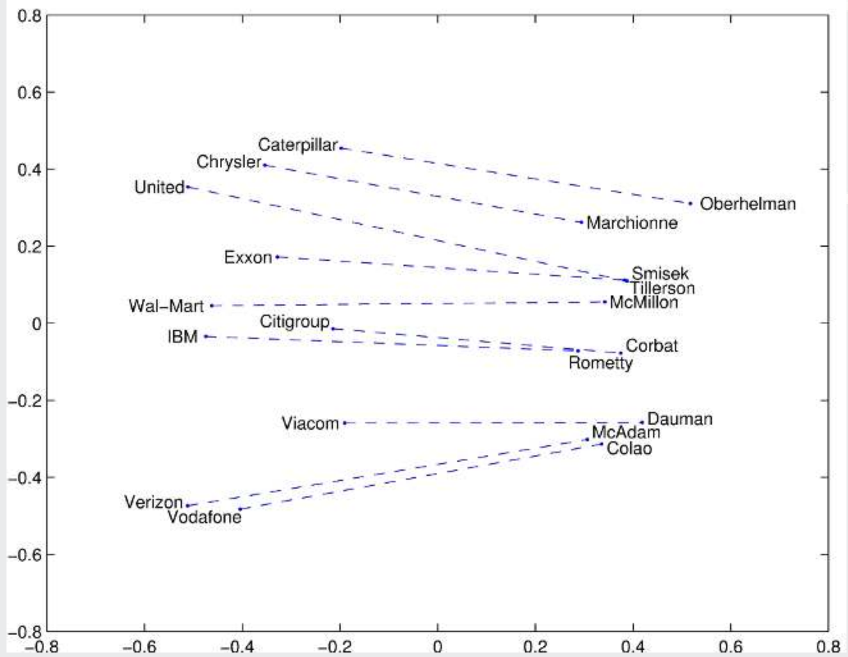


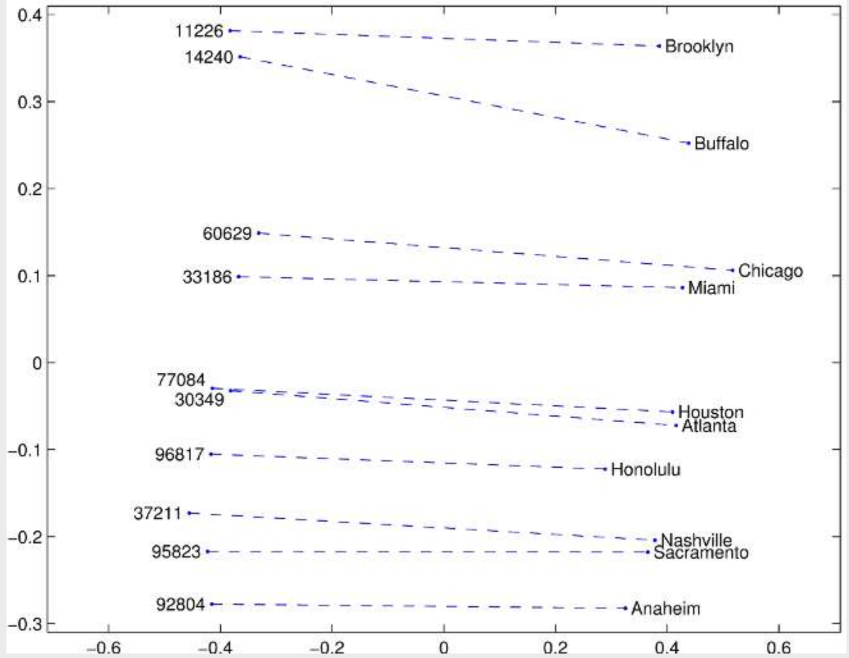
Country-Capital

What it retains: GloVe







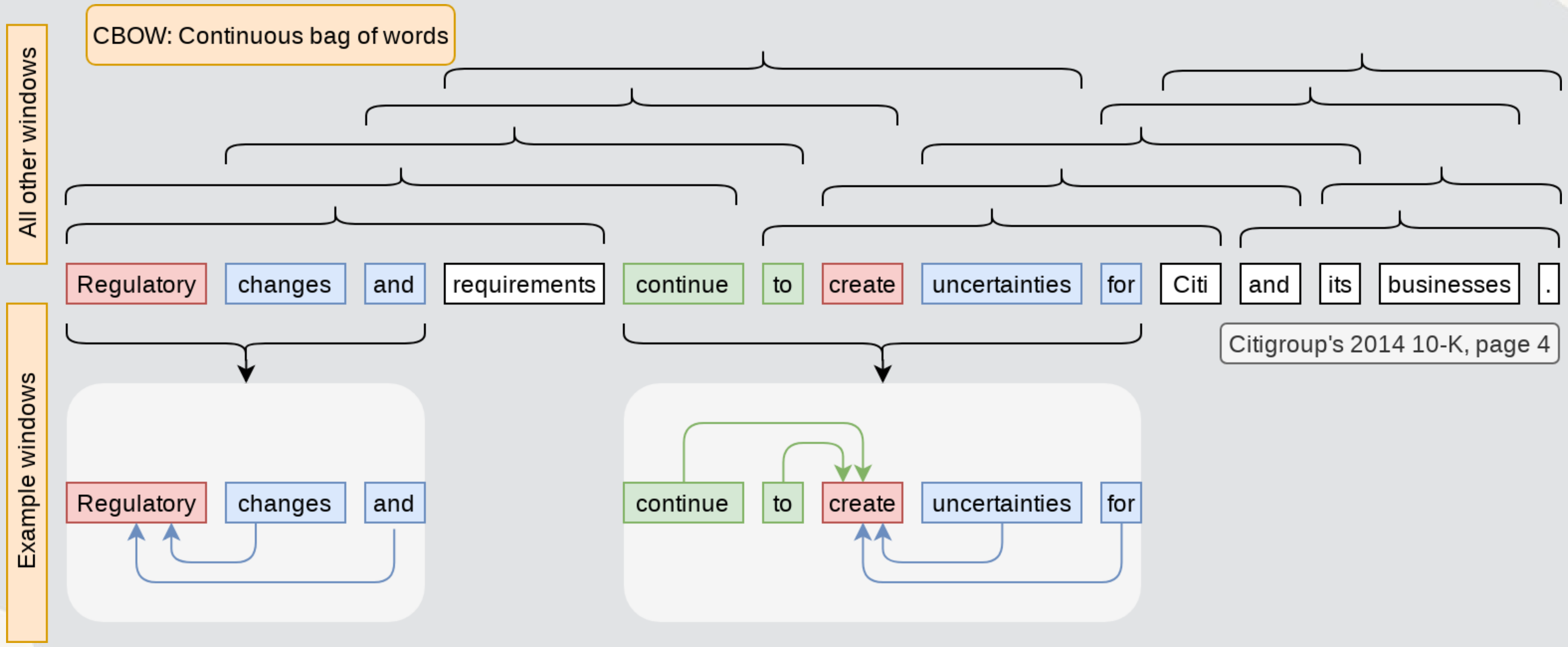


How to build word vectors

- In python:
  - `gensim` allows you to import pre-trained models
    - It also allows you to train your own
  - `tensorflow` allows you to use pre-trained models or train your own
- In R:
  1. Word co-occurrence-based like GloVe
     - Available from the `text2vec` package
  2. Word order based (using a neural network)
     - Available from the `rword2vec` package

How does word order work?

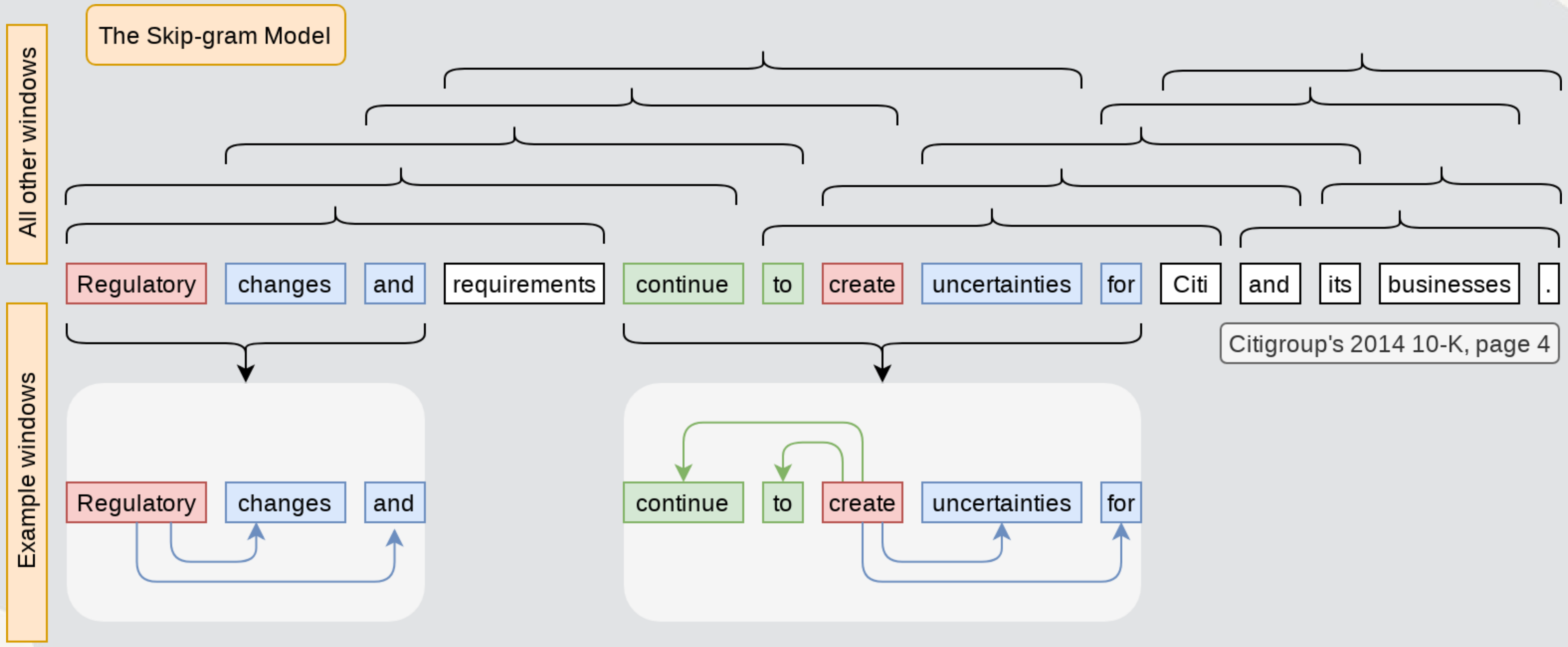
Infer a word's meaning from the words around it



Refered to as CBOW (continuous bag of words)

How else can word order work?

Infer a word’s meaning by *generating* words around it



Refered to as the Skip-gram model

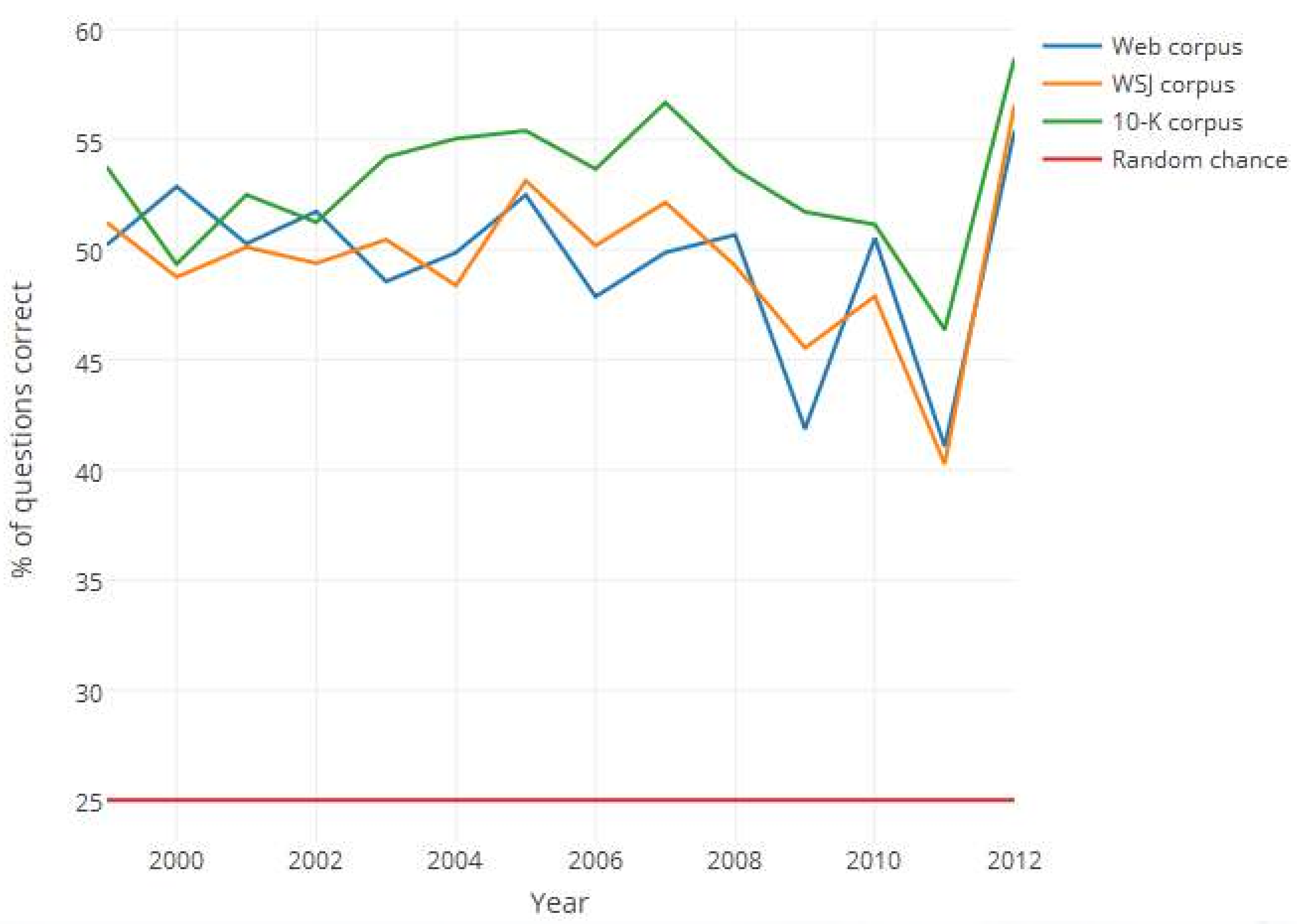
An example of using word2vec

- In the BCE paper from Session 6, word2vec was used to provide assurance that the LDA model works reasonably well on annual reports
  1. We trained a word2vec model on random issues of the Wall Street Journal (247.8M words)
  2. The resulting model “understood” words in the context of the WSJ
  3. We then ran a psychology experiment (word intrusion task) on the algorithm

Word intrusion task

- The task is to find which word doesn’t belong
- Each question consisted of 3 words from 1 topic and 1 *intruded* from another random topic
  - Ex.:
    - Laser, Drug, Viral, Therapeutic
    - Supply, Steel, Capacity, Losses
    - Relief, Lousisiana, Cargo, Assisted

Results



Web corpus
WSJ corpus
10-K corpus
Random chance
% of questions correct
60
55
50
45
40
35
30
25
2000
2002
2004
2006
2008
2010
2012
Year

Loading in word2vec with Gensim

- The `gensim` package comes with the ability to download word2vec and GloVe vectors from a repository
- The code below would allow you to download a model trained on Google News
  - In this model, each word is represented as a 300-dimensional vector

```python
import gensim
import gensim.downloader

base_w2v = gensim.downloader.load('word2vec-google-news-300')
```

Note: The model it downloads is 1.7GB

- The model will be stored in `~/gensim_models/`
  - ~ represents your user directory
  - You can safely delete this directory after you are done using it

Examining word2vec: Odd one out

```python
base_w2v.doesnt_match(['Queen', 'King', 'Prince', 'Peasant'])
```

```
## 'Peasant'
```

```python
base_w2v.doesnt_match(['Singapore', 'Malyasia', 'Indonesia', 'Germany'])
```

```
## 'Germany'
```

```python
base_w2v.doesnt_match(['Euro', 'USD', 'RMB', 'computer'])
```

```
## 'computer'
```

```python
base_w2v.doesnt_match(['mee goreng', 'char kway teoh', 'laksa', 'hamburger'])
```

```
## 'hamburger'
```

Examining word2vec: Closest words

```python
base_w2v.most_similar(['Earnings'])
```

```
## ('Pro_Forma_EPS', 0.6441532373428345) ('Diluted_EPS', 0.636042058467865)
##  ('Goodwill_Impairment', 0.6357625126838684) ('Tax_Expense', 0.6289322376251221)
##  ('Reconciling_Items', 0.6285154819488525) ('Restructuring_Charges', 0.6268271207809448)
##  ('Backs_FY##', 0.6254147291183472) ('Raises_FY##_EPS', 0.6230234503746033)
##  ('Restructuring_Charge', 0.6216667294502258) ('FFO_Per_Share', 0.6207219958305359)
```

```python
base_w2v.most_similar('IASB')
```

```
## ('Accounting_Standards_Board', 0.7211726307868958) ('FASB', 0.6697319149971008)
##  ('IAASB', 0.6319378614425659) ('IAS##', 0.6150702834129333)
##  ('FASB_IASB', 0.593984842300415) ('Exposure_Draft', 0.5892050266265869)
##  ('Board_IASB', 0.5818656086921692) ('IFRS', 0.5813880562782288)
##  ('GNAIE', 0.5802473425865173) ('Solvency_II', 0.574397087097168)
```

Examining word2vec: Closest words

```python
base_w2v.most_similar(['KPMG'])
```

```
## ('PwC', 0.8044512867927551) ('PricewaterhouseCoopers', 0.8032213449478149)
##  ('Deloitte', 0.7856791019439697) ('Grant_Thornton', 0.7815379500389099)
##  ('PriceWaterhouseCoopers', 0.7609084248542786) ('KMPG', 0.7575340270996094)
##  ('PricewaterhouseCoopers_PwC', 0.7438496351242065) ('Pricewaterhouse_Coopers', 0.7163813710212708)
##  ('Delloitte', 0.7009097337722778) ('KPMG_LLP', 0.7008424401283264)
```

```python
base_w2v.most_similar(['Arthur_Andersen'])
```

```
## ('Arthur_Andersen_LLP', 0.7720072269439697) ('Peat_Marwick', 0.6542829275131226)
##  ('Price_Waterhouse', 0.6524070501327515) ('KPMG_Peat_Marwick', 0.6093755960464478)
##  ('Peat_Marwick_Mitchell', 0.6006763577461243) ('&_Lybrand', 0.5949062705039978)
##  ('Arthur_Andersen_accounting', 0.559570848941803) ('auditor_Arthur_Andersen', 0.5569155812263489)
##  ('KPMG', 0.5496521592140198) ('Price_Waterhouse_LLP', 0.5493941903114319)
```

Examining word2vec: Analogies

> man : King :: woman : ?

- Mathematically: $King - man + woman = ?$

```python
base_w2v.most_similar(positive=['King', 'woman'], negative=['man'])
```

```
## ('Queen', 0.5515626668930054) ('Oprah_BFF_Gayle', 0.47597548365592957)
##  ('Geoffrey_Rush_Exit', 0.46460166573524475) ('Princess', 0.4533674716949463)
##  ('Yvonne_Stickney', 0.4507041573524475) ('L._Bonauto', 0.4422135353088379)
##  ('gal_pal_Gayle', 0.4408389925956726) ('Alveda_C.', 0.4402790665626526)
##  ('Tupou_V.', 0.4373864233493805) ('K._Letourneau', 0.4351031482219696)
```

The sleight of hand behind this

- Word2Vec implementations usually bar a word in the analogy from being an output
  - E.g., it will never report **man : King :: woman : King**
    - But this is actually the mathematical answer

```
analogy = base_w2v['King'] + base_w2v['woman'] + base_w2v['man']
analogy = analogy / np.linalg.norm(analogy)
print('King', np.linalg.norm(analogy - base_w2v['King']))
```

```
## King 1.9888592
```

```
print('Queen', np.linalg.norm(analogy - base_w2v['Queen']))
```

```
## Queen 2.7364814
```

It’s still pretty good though!

- Note that since word2vec’s original answer was `Queen`, this implies it was second best
  - If Queen is the closest word to King, then this would be mathematically uninteresting
    - It’s actually 7th though!

```python
base_w2v.most_similar('King')
```

```
## [('Jackson', 0.5326348543167114), ('Prince', 0.5306329727172852), ('Tupou_V.', 0.5292826294898987), ('KIng', 0.522750139236
```

What is this good for?

1. You care about the words used, by not stylistic choices
   - Abstraction
2. You want to crunch down a bunch of words into a smaller number of dimensions without running any bigger models (like LDA) on the text.
   - E.g., you can toss the 300 dimensions of the Google News model to a Lasso or Elastic Net model
     - This is a big improvement over the past method of tossing vectors of word counts at Naive Bayes
3. You want synonyms for a set of words that are selected in a less-researcher-biased fashion
   - You can even get n-gram synonyms this way
   - A popular method for augmenting small dictionaries

Exercise: Trying out word2vec

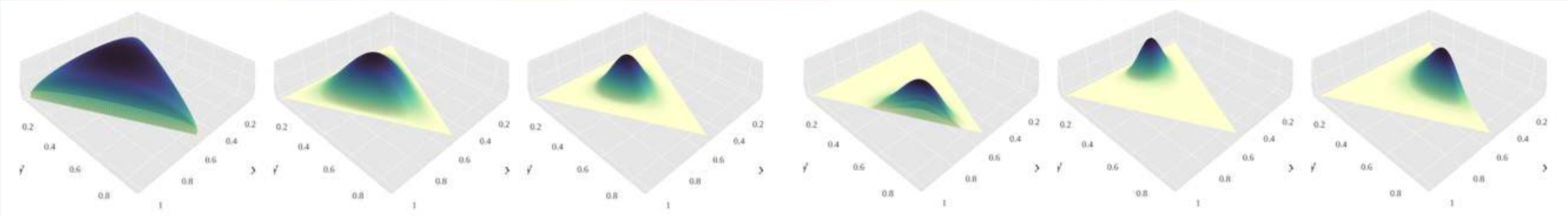
> Do Set 2 in the `Session 4-Exercises` file

- This set of exercise is to help you understand a bit better about what word2vec is good at
  - As well as what it isn't good at

Understanding documents using topic analysis

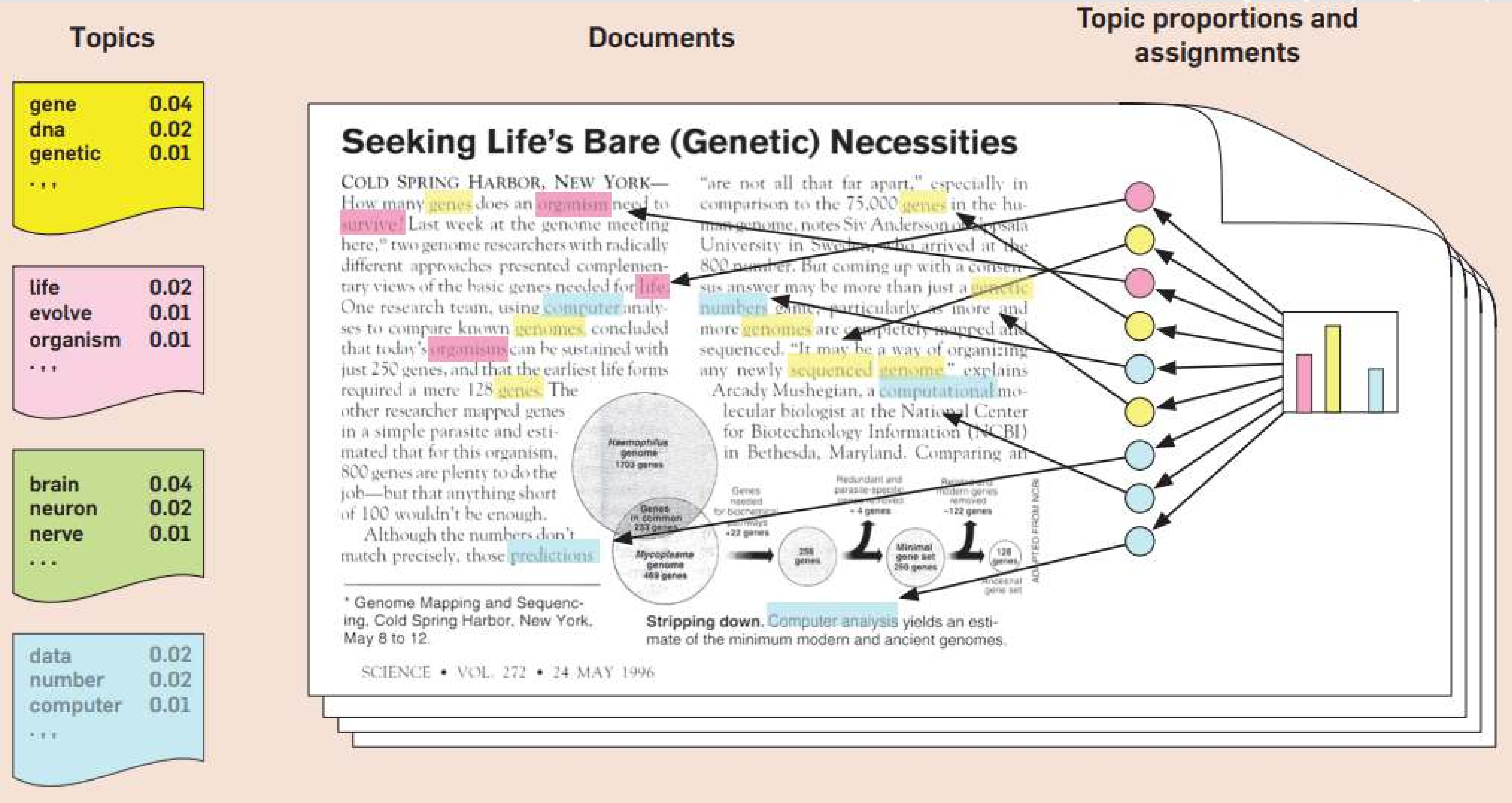
What is LDA?

- **L**atent **D**irichlet **A**llocation
- One of the most popular methods under the field of *topic modeling*
- LDA is a Bayesian method of assessing the content of a document
- LDA assumes there are a set of topics in each document, and that this set follows a *Dirichlet* prior for each document
    - Words within topics also have a *Dirichlet* prior



> More details from the creator

An example of LDA



How does it work?

1. Reads all the documents
   - Calculates counts of each word within the document, tied to a specific ID used across all documents
2. Uses variation in words within and across documents to infer topics
   - By using a Gibbs sampler to simulate the underlying distributions
     - An MCMC method

- It's quite complicated in the background, but it boils down to a system where generating a document follows a couple rules:
  1. Topics in a document follow a multinomial/categorical distribution
  2. Words in a topic follow a multinomial/categorical distribution

Motivating example: Huang et al. 2015 MS

- A goal of the paper is to see what analysts discuss in their reports, with a focus on two possibilities
  1. Discussing content from conference calls
  2. Discussing content that is beyond the scope of conference calls

> Analysts discuss both, and both are useful to investors

- As expected, there is some variation when both are useful:
  - If a conference call is more difficult to understand, analysts discussing the same content is more useful
  - If executives have incentives to withhold information, then information beyond the conference call is more useful

Why LDA in this paper: Conference calls

- There are a lot of potential things that could be discussed in conference calls
  - Some are consistently brought up:
    - Economic conditions
    - Earnings
    - Expenses
    - Product launches
  - Some can be particular to a specific time and a specific company
    - Example: comments on why a buggy video game was released (Source)
      - “I don’t expect next-gen performance on last-gen, but I would like to be able to play through the game.”
- If a researcher takes a dictionary approach, they can probably get many of the consistently discussed topics
  - But they will likely miss the less common or more variable topics

Why LDA in this paper: Analyst reports

> The goal is to have a text model that is valid for both document types

- Train LDA on both document types!
- Difficult to ensure the same words are used for a dictionary approach
  - Need to repeat your dictionary creation task for both settings (doubles workload)

Why LDA in this paper: Other considerations

1. The model is trained *per industry*
   - Since relevant topics for analyst reports and conference calls change quite a bit by industry, this allows them to get a more accurate portrayal of each industry
     - This adds essentially no extra work for the researcher, other than calling some for loops
     - Contrast with a dictionary approach, where you would now need to multiple your effort by the number of industries
2. Not all words are as relevant
   - LDA provides Bayesian-based probabilistic weights rather than 1/0 indicators

Implementing LDA

Implementing LDA in python

- The best package for this is `gensim`
  - As long as your data fits in memory comfortably, it is easy to use
  - If not, you will need to construct a generator to pass to it, which is more complex
    - The code file for this session has an example of this!
- There is also an implementation in `sklearn`
- In terms of computation time, you will likely spend more time prepping your text than running the LDA model

Prepping text

- We will take a more thorough approach using `spaCy` for preprocessing
  - Remove stopwords using `spaCy`'
  - Remove numbers, symbols, and punctuation based on a neural network dependency parser
  - Lemmatize words based on the word and its POS tags
- If accuracy is less important or your computer can't handle `spaCy`'s approach, another approach is:
  - Use a regex or NLTK to tokenize into words
  - Use the `stop-words` package or NLTK to get a list of stopwords
    - Filter them out using a list comprehension

      ```
      doc = [w for w in doc if w not in stopwords]
      ```
  - Apply a word-based lemmatizer from NLTK such as WordNet

Running the LDA model

```python
# docs contains all of our cleaned 10-K filings
# doc_names contains the filings' accession numbers

# Prepare the needed parts for gensim's LDA implementation
words = gensim.corpora.Dictionary(docs)
words.filter_extremes(no_below=50, no_above=0.5)
words.filter_tokens(bad_ids=[words.token2id['_']])  # '_' is not treated as a symbol by spaCy
corpus = [words.doc2bow(doc) for doc in docs]

# Free up some memory
del docs

# Save the intermediate data -- useful if we want to tweak model parameters and re-run later
with open('../Data/corpus.pkl', 'wb') as f:
    pickle.dump([corpus, words, doc_names], f, protocol=pickle.HIGHEST_PROTOCOL)

# Run the model
lda = gensim.models.ldamodel.LdaModel(corpus, id2word=words, num_topics=10, passes=5,
                                      update_every=5, alpha='auto', eta='auto')
```

Examining the LDA model

1. Load in the LDA model along with the `corpus` structure and the document names
   - No need to do this if the model is still in memory

```
lda = gensim.models.ldamodel.LdaModel.load('../../Data/lda')
with open('../../Data/corpus.pkl', 'rb') as f:
    corpus, words, doc_names = pickle.load(f)
```

2. Examine a topic

```
# Parameters: topic number, number of words
lda.show_topic(0, 10)
```

```
## [('vehicle', 0.012847863), ('commodity', 0.010794649), ('oil', 0.007663337), ('funds', 0.007322089)]
##  [('gas', 0.005776752), ('partnerships', 0.0057128207), ('mortgage', 0.0052304408), ('swap', 0.004704417)]
##  [('futures', 0.0043137535), ('advisor', 0.0043026144)]
```

Note the weights associated with the words – some words are more meaningful than others

Examining the LDA model

3. See the top words in each topic

```python
for i in range(0,10):
    top = lda.show_topic(i, 10)
    top_words = [w for w, _ in top ]
    print('{}: {}'.format(i, ' '.join(top_words)))
```

```
## 0: vehicle commodity oil funds gas partnerships mortgage swap futures advisor
## 1: banking restaurant hotel mortgage fdic borrower lending banks tier residential
## 2: mining exploration mineral gold manufacture silver land metal tobacco ore
## 3: gaming television station contents advisor client programming casino fuel broadcast
## 4: store brand solution mobile contents card online platform channel merchandise
## 5: mortgage reit borrower residential tenant home reinsurance rating contents banking
## 6: china client solution prc manufacture manufacturer holdings contents pension raw
## 7: clinical trial drug patient fda candidate study medical care healthcare
## 8: gas oil drilling pipeline crude water exploration unitholder drill commodity
## 9: gas coal fuel plant electric pension utility generation contents transmission
```

Examining the LDA model

- The `pyLDAvis` package produces a nice interactive map of the topics

```python
ldavis = pyLDAvis.gensim_models.prepare(lda, corpus, words, sort_topics=False)
pyLDAvis.display(ldavis)
```

Click here to see the output

Topic labels

- For the sake of exposition, I will label the topics as:
  0. Investments
  1. Loans
  2. Mining
  3. Media
  4. Stores
  5. Financial
  6. Foreign
  7. Medical
  8. Oil and gas
  9. Utilities

Applying the LDA topics

- The model parameter `gamma` contains a full matrix of the raw topic amounts per document
  - We can get this by calling `.inference(corpus)` on our model
- Best to normalize this to get percentages

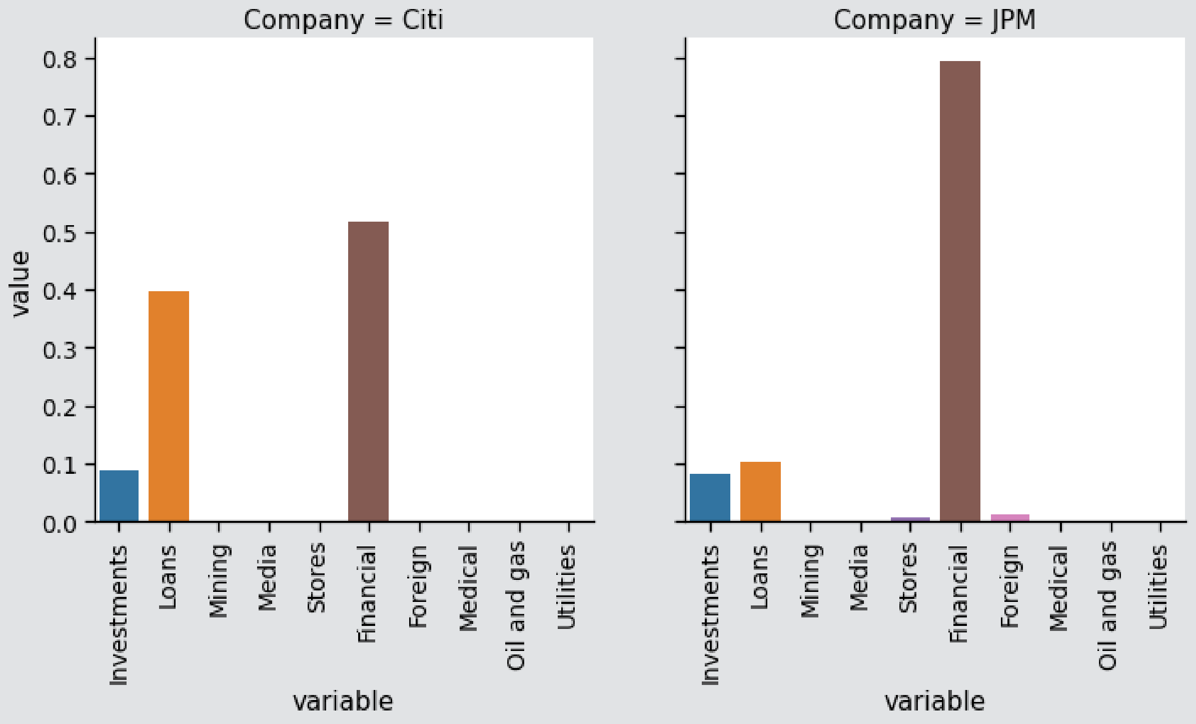
```python
gamma, _ = lda.inference(corpus)
topic_dist = topic_dist = gamma / gamma.sum(axis=1)[:,None]
topic_dist.shape
```

- Next, we can build this into a data frame and merge it with Compustat

```python
topic_names = ['Investments', 'Loans', 'Mining', 'Media', 'Stores', 'Financial',
               'Foreign', 'Medical', 'Oil and gas', 'Utilities']
df = pd.DataFrame(data=topic_dist, columns=topic_names)
df['Accession'] = doc_names
df_comp = pd.read_csv('../../Data/S4_Data.csv')
df = df.join(df_comp, how='left')
df['industry'] = sic_to_industry(df.regsic)
```

Comparing two companies' 10-Ks

```python
long = pd.melt(df[(df['Accession'] == '0001104659-14-015152') | (df['Accession'] == '0000019617-14-000289')],
               id_vars='Accession', value_vars=topic_names)
long['Company'] = np.where(
    long.Accession == '0001104659-14-015152', 'JPM', 'Citi')
with sns.plotting_context("notebook", font_scale=1.25):
    g = sns.catplot(x='variable', y='value', col='Company',
                    data=long, kind='bar')
    _ = g.set_xticklabels(rotation=90)
```

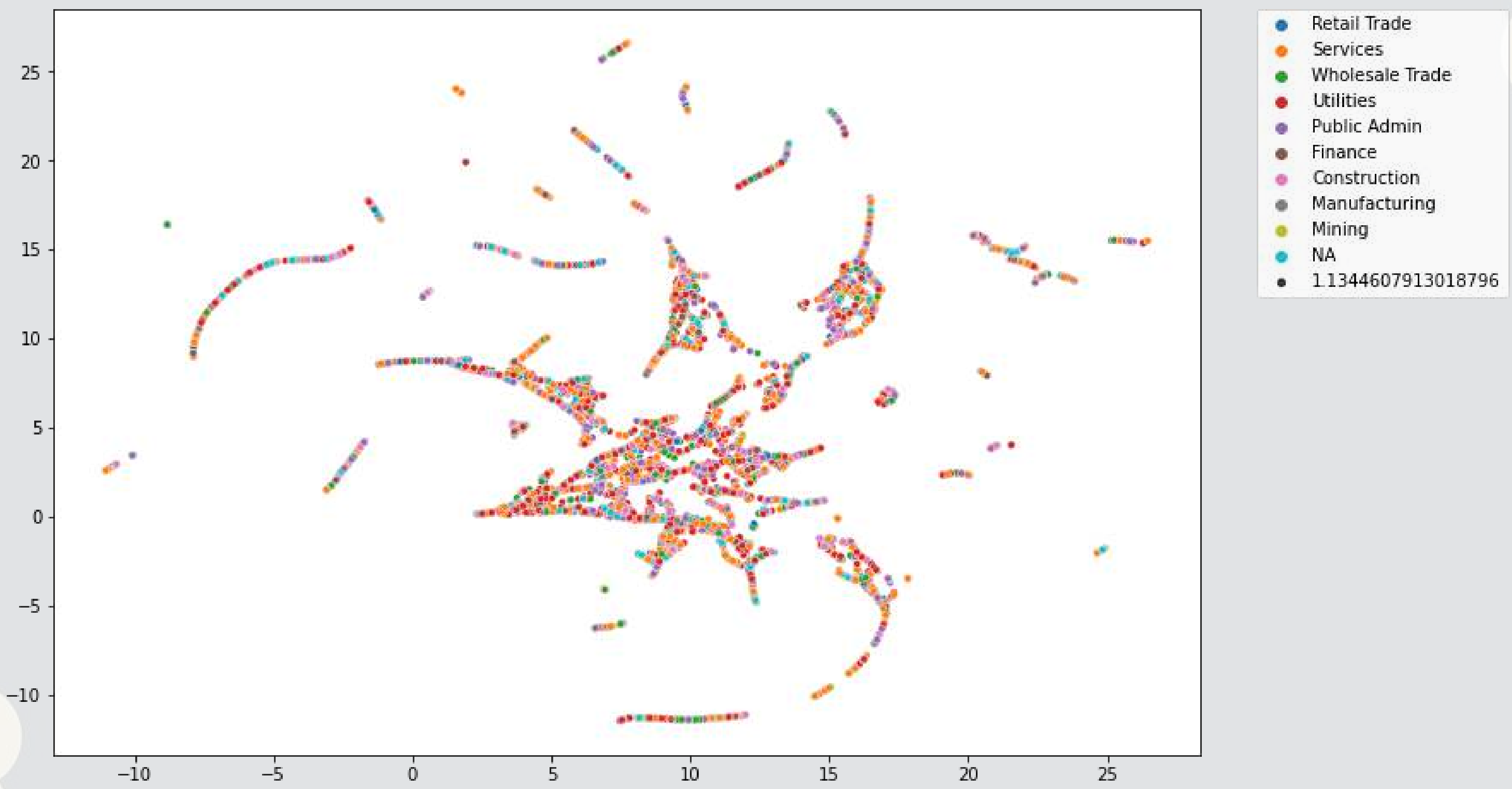


Topic weights by SIC industry

```python
long = pd.melt(df, id_vars='industry', value_vars=topic_names)
long = long.groupby(['industry', 'variable']).mean().reset_index()
with sns.plotting_context("notebook", font_scale=1.25):
    g = sns.catplot(x='variable', y='value', col='industry', col_wrap=3,
                    data=long[long.industry != 'NA'], kind='bar')
    _ = g.set_xticklabels(rotation=90)
```

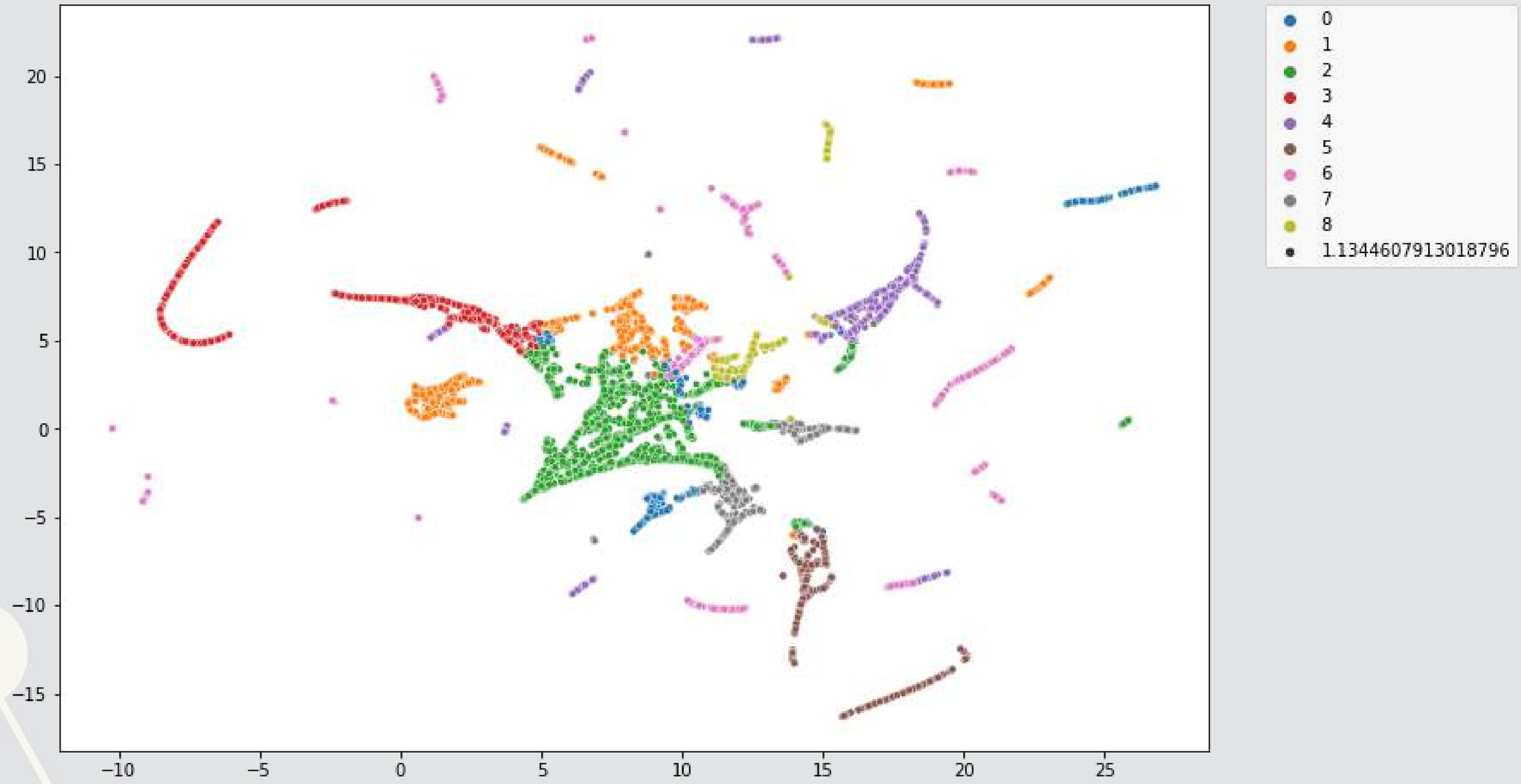
Projecting to 2D with UMAP

- Like last session, we will use UMAP to get a sense of how well topic line up with SIC industries



Projecting to 2D with UMAP

- It is also interesting to see how well the topics can be clustered
  - The below colors UMAP by a `k=9` kmeans algorithm applied to the LDA output



Things to note

- There are a number of parameters in this design to optimize
  1. The cleaning of the data removed any words in less than 50 documents or more than 50% of documents
     - This can be tweaked depending on the needs of the model
  2. LDA does have hyperparameters that can be specified: α and η
     - We used `gensim`'s auto option to learn it from the data
  3. There are also 2 hyperparameters we didn't touch: κ (`decay`) and $\boxed{\text{\tau\_}}$ (`offset`)
     - These can be tuned as well
- Most importantly, **you need to decide on the number of topics**
  - There are multiple ways to do this
    1. Iteratively conducting in-sample testing of the performance of the model on a regression of interest (see Brown, Crowley, and Elliott (2020 JAR) for details)
    2. Condition number test (see my dissertation for details)
    3. A geometric approach to orthogonalizing topics (see my dissertation for details)
    4. Based on human-reading of the output – see, e.g., Crowley, Huang, and Lu (2020)

Addendum: Using R

- There are at least four good implementations of LDA in R
  1. `stm`: A bit of a tweak on the usual LDA model that plays nicely with `quanteda` and also has an associated `stmBrowser` package for visualization (on Github)
  2. `lda`: A somewhat rigid package with difficult setup syntax, but it plays nicely with the great `LDAvis` package for visualizing models. Supported by `quanteda`.
  3. `topicmodels`: An extensible topic modeling framework that plays nicely with `quanteda`
  4. `mallet`: An R package to interface with the venerable MALLET Java package, capable of more advanced topic modeling

Optimizing K-means clustering

Overview

- Kmeans clustering is very fast to run, but suffers from the same issue as LDA:

> You need to specify the number of clusters!

- Often times the solutions to this are similar to what we discussed for LDA
  - Hand tuning
  - In sample performance
- However, there is a statistics-based, researcher-bias-free method

> The Gap Statistic

How does the Gap statistic work?

- Let…
  - k be the number of clusters,
  - B the number of simulated samples
  - W_k be the K-Means inertia score on actual data
  - $W^*_{k,r}$ be the K-Means inertia score for iteration r with synthetic data
  - $\bar{l}$ be the average of the $W^*_{k,r}$s

$$Gap(k) = \left(\frac{1}{B}\right)\sum_{r=1}^{B}\log\left(W^*_{k,r}\right) - \log\left(W_k\right) \text{ and}$$

$$s_k = sd_k\sqrt{1+\frac{1}{B}},\text{ where } sd_k = \sqrt{\left(\frac{1}{B}\right)\sum_{r=1}^{B}\left\{\log\left(W^*_{k,r} - \bar{l}\right)\right\}^2}$$

- Select the lowest k such that $Gap(k) \geq Gap(k+1) - s_{k+1}$

> I.e., select the lowest k s.t. the log-scaled error removed by clustering on real data at k is no worse than 1 SD below the log-scaled error removed at $k+1$

Implementation in python

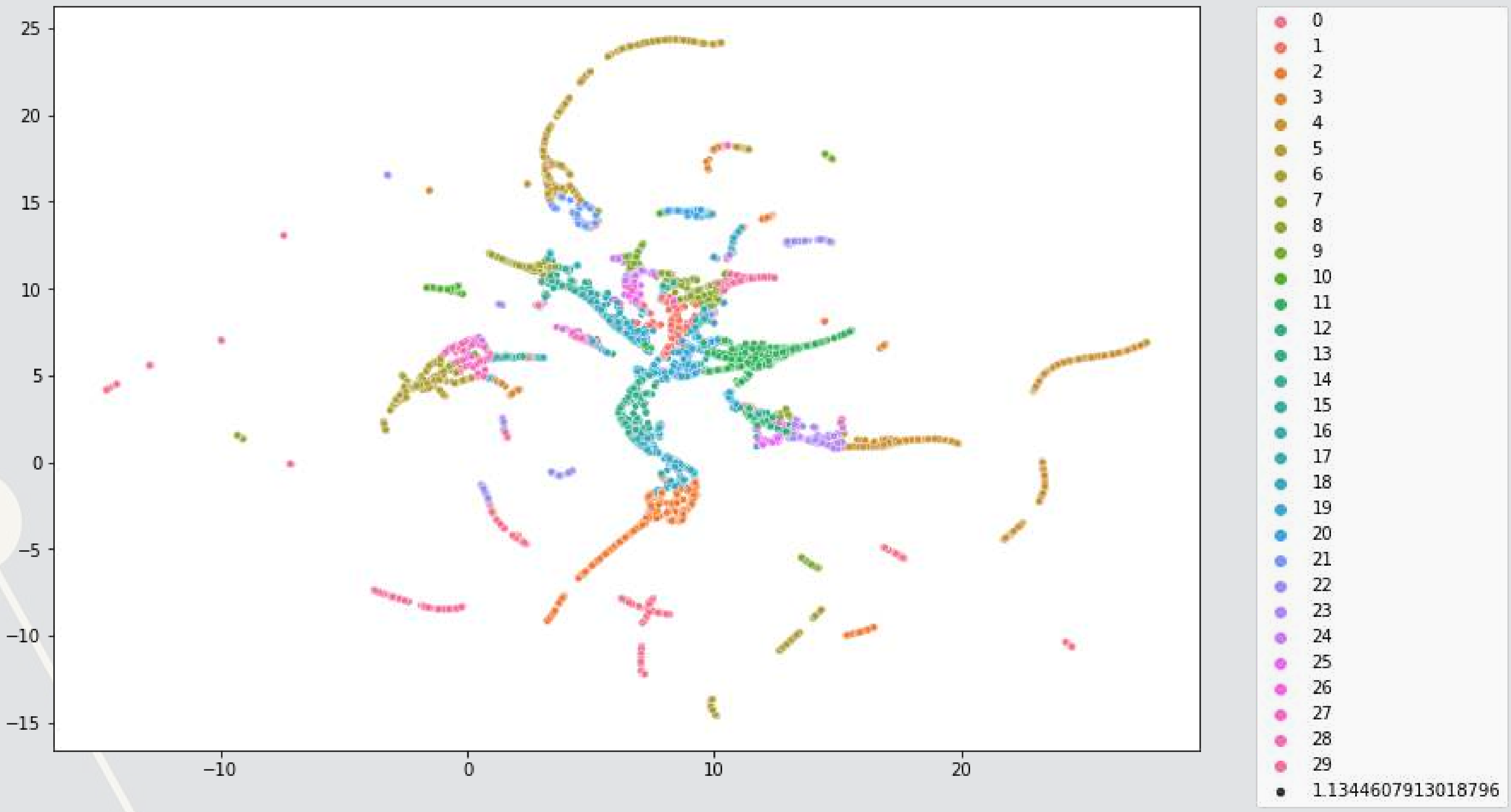
- The code is too long to put in the slides, but it is in the code file
- Sketch of the code:
  1. Iterate through k values starting at 2
  2. Determine performance (inertia) at k with real data
  3. Determine performance (inertia) at k with simulated (random) data 10 times
  4. Calculate the standard deviation of the log of performance on random data
  5. See if the 2x2 difference in log inertia between k and $k+1$ on real and random data is less than the standard deviation
     - If so, k is optimal, stop iterating
     - If not, $k = k + 1$ and start again

> $k = 30$ for the model presented here

Optimal clusting

```python
model = cluster.KMeans(n_clusters=30)
kmeans = model.fit(df[topic_names])
df['cluster_opt'] = kmeans.labels_

umap_color(df[topic_names], df.cluster_opt.astype("category"))
```



Example companies in the optimized clusters

```
df[df.cluster_opt==2][['coname', 'industry']].sample(n=10
```

```
##                                     coname        indu
## 1922                  CALLIDUS SOFTWARE INC      Public Ad
## 2690             MUNRO DEVELOPMENTS, INC.      Construc
## 5021   RYMAN HOSPITALITY PROPERTIES, INC.         Serv
## 5462                 AMPCO PITTSBURGH CORP        Utili
## 6816                 PORTLOGIC SYSTEMS INC.      Public Ad
## 5369                      ONCOTHYREON INC.      Public Ad
## 3083  FORD CREDIT AUTO OWNER TRUST 2010-B         Serv
## 5376                         GSI GROUP INC        Utili
## 3635                      GLOBALSTAR, INC.   Wholesale T
## 5977   AMERICREDIT FINANCIAL SERVICES INC         Serv
```

```
df[df.cluster_opt==3][['coname', 'industry']].sample(n=10
```

```
##                                                      coname
## 7264                AMERICAN RAILCAR INDUSTRIES, INC.
## 7307                            PHARMA-BIO SERV, INC.
## 4740                            TIME WARNER CABLE INC.
## 3165  REEF OIL & GAS INCOME & DEVELOPMENT FUND III LP
## 3215                               PNM RESOURCES INC
## 6278                          AMERICAN GREETINGS CORP
## 7615                            TIPTREE FINANCIAL INC.
## 1599                                  ARTHROCARE CORP
## 3133                              KIEWIT ROYALTY TRUST
## 1187                MS STRUCTURED TILES SERIES 2006-1
```

Conclusion

Wrap-up

Supervised text classification

- Good when you only need one class, and that class is:
  1. Easy to pick up with some other text as a prior
  2. Very different from the baseline text in your documents

Word vectors

- Easy to implement
- Useful in some context where words matter

LDA

- Good for getting a simple, quantitative summary of your data

Packages used for these slides

Python

- gensim
- matplotlib
- numpy
- pandas
- pyLDAvis
- scikit-learn
- seaborn
- spacy
- umap-learn

R

- kableExtra
- knitr
- reticulate
- revealjs

References

- Brown, Nerissa C., Richard M. Crowley, and W. Brooke Elliott. “What are you saying? Using topic to detect financial misreporting.” Journal of Accounting Research 58, no. 1 (2020): 237-291.
- Crowley, Richard Michael. “Disclosure through multiple disclosure channels.” PhD diss., University of Illinois at Urbana-Champaign, 2016.
- Crowley, Richard M., Wenli Huang, and Hai Lu. “Discretionary dissemination on Twitter.” Rotman School of Management Working Paper 3105847 (2018).
- Hassan, Tarek A., Stephan Hollander, Laurence Van Lent, and Ahmed Tahoun. “Firm-level political risk: Measurement and effects.” The Quarterly Journal of Economics 134, no. 4 (2019): 2135-2202.
- Huang, Allen H., Reuven Lehavy, Amy Y. Zang, and Rong Zheng. “Analyst information discovery and interpretation roles: A topic modeling approach.” Management Science 64, no. 6 (2018): 2833-2855.
- Schütze, Hinrich, Christopher D. Manning, and Prabhakar Raghavan. Introduction to information retrieval. Vol. 39. Cambridge: Cambridge University Press, 2008.
- Song, Min, and Yi-Fang Brook Wu, eds. Handbook of research on text and web mining technologies. IGI global, 2008.

Custom code

```python
# Convert SIC codes to their 2-digit labels

def sic_to_industry(mat):
    d = {0:'Agriculture', 1:'Mining', 2:'Construction', 3:'Manufacturing', 4:'Utilities', 5:'Wholesale Trade',
         6:'Retail Trade', 7:'Finance', 8:'Services', 9:'Public Admin', 10:'NA'}
    mat = np.where((mat >= 100) & (mat <=999), 1, mat)
    mat = np.where((mat >= 1000) & (mat <=1499), 2, mat)
    mat = np.where((mat >= 1500) & (mat <=1799), 3, mat)
    mat = np.where((mat >= 2000) & (mat <=3999), 4, mat)
    mat = np.where((mat >= 4000) & (mat <=4999), 5, mat)
    mat = np.where((mat >= 5000) & (mat <=5199), 6, mat)
    mat = np.where((mat >= 5200) & (mat <=5999), 7, mat)
    mat = np.where((mat >= 6000) & (mat <=6799), 8, mat)
    mat = np.where((mat >= 7000) & (mat <=8999), 9, mat)
    mat = np.where((mat >= 9100) & (mat <=9999), 9, mat)
    mat = np.where(np.isnan(mat), 10, mat)
    return [d[i] for i in list(mat)]
```

Custom code

```python
# From umap.plot source code on Github
def _get_embedding(umap_object):
    if hasattr(umap_object, "embedding_"):
        return umap_object.embedding_
    elif hasattr(umap_object, "embedding"):
        return umap_object.embedding
    else:
        raise ValueError("Could not find embedding attribute of umap_object")


# Cut down version of umap.plot.points to remove dependencies on  datashader, bokeh, holoviews, scikit-image, and colorcet
# Introduces a dependency on seaborn though
def umap_color(data_map, data_color, cmap='viridis', subset=None, title=None):
    reducer = umap.UMAP()
    umap_object = reducer.fit(data_map)
    embed = _get_embedding(umap_object)

    if subset is not None:
        embed_X = embed[subset,0]
        embed_Y = embed[subset,1]
        data_color = np.array(data_color[subset])
    else:
        embed_X = embed[:, 0]
        embed_Y = embed[:, 1]

    point_size = 100.0 / np.sqrt(len(embed_X))

    # color by values
    fig, ax = plt.subplots(figsize=(12,8))
    g = sns.scatterplot(ax=ax, x=embed_X, y=embed_Y, hue=data_color, size=point_size)
    _ = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    return g
```