ML for SS: Embeddings and topic modeling

Session 6

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

Overview

Papers

Huang et al. (2018 MS)

- This is a nicely motivated paper in terms of its usage of LDA
  - Needed to answer the research question

Crowley, Huang, and Lu (2020 working)

- Demonstrates a usage of embedding methods for
- Also showcases a variant of LDA for social media classification

Roberts et al. (2014 AJPS)

- Demonstrates an interesting variant of LDA that can help with identifying differences in information across groups or conditions

Technical Discussion

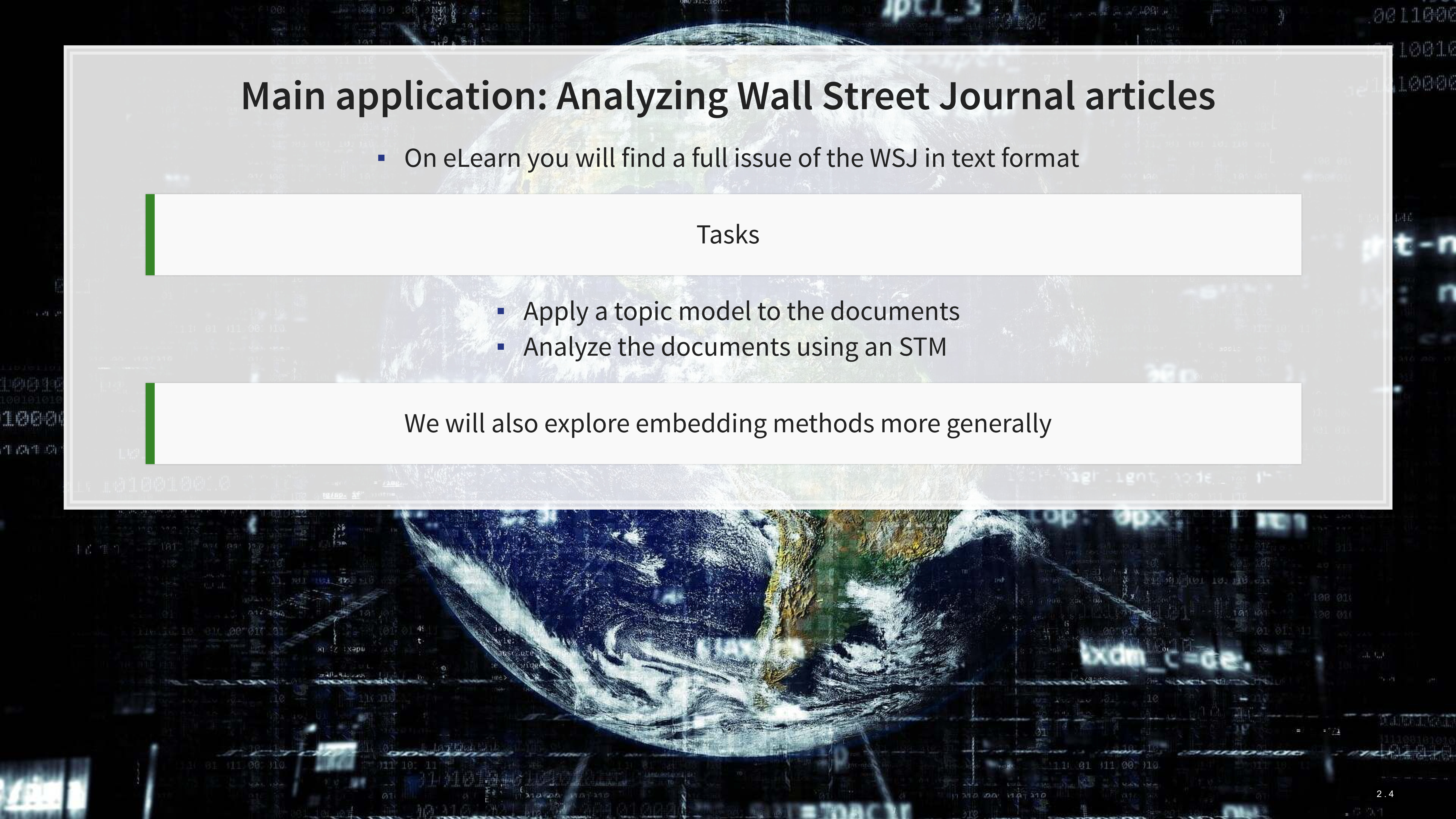
Python

- LDA
  - gensim is the easiest to use in general
    - Some incompatibilities with python 3.9 however
- Word2Vec
  - gensim is again quite easy to use
  - fastText is another good option
  - Tensorflow is also an option
- USE
  - Tensorflow is the best choice

R

- LDA
  - `stm` can do a lot more than just standard LDA
    - STM is only available in R
  - `lda` and `topicmodels` both play nicely with `quanteda`
  - `mallet` gives an interface to the venerable MALLET Java package, capable of more advanced topic modeling
- Word2vec
  - The `word2vec` and `rword2vec` packages may be useful

> Both R and python are good for LDA. Python is better for embedding methods. R is the only option for STM.



Main application: Analyzing Wall Street Journal articles

- On eLearn you will find a full issue of the WSJ in text format

Tasks

- Apply a topic model to the documents
- Analyze the documents using an STM

We will also explore embedding methods more generally

Embeddings

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.

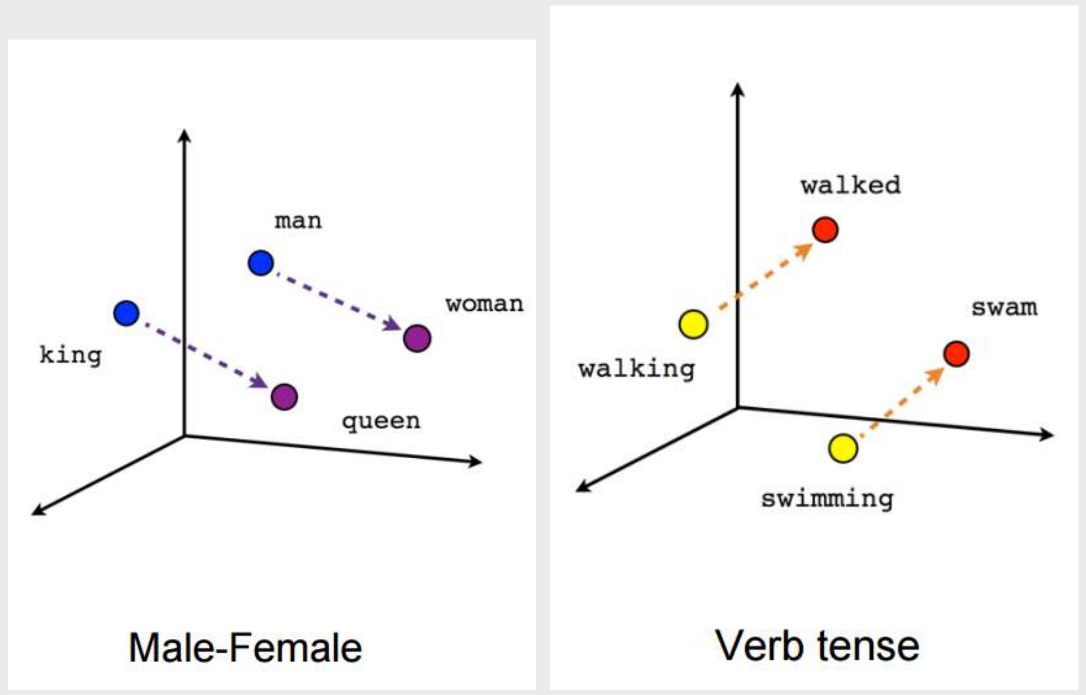
> Word vectors try to encapsulate all of the above implicitly, through by encoding words as a vector based on how features manifest themselves in text

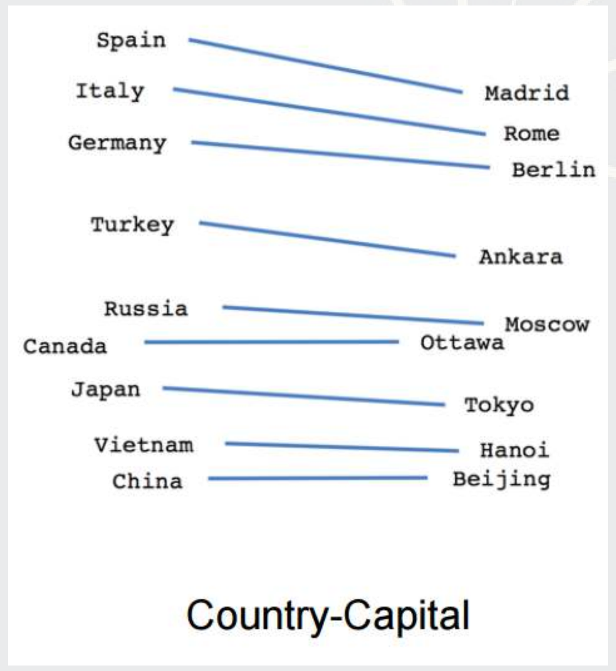
Word vectors: Simple example

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

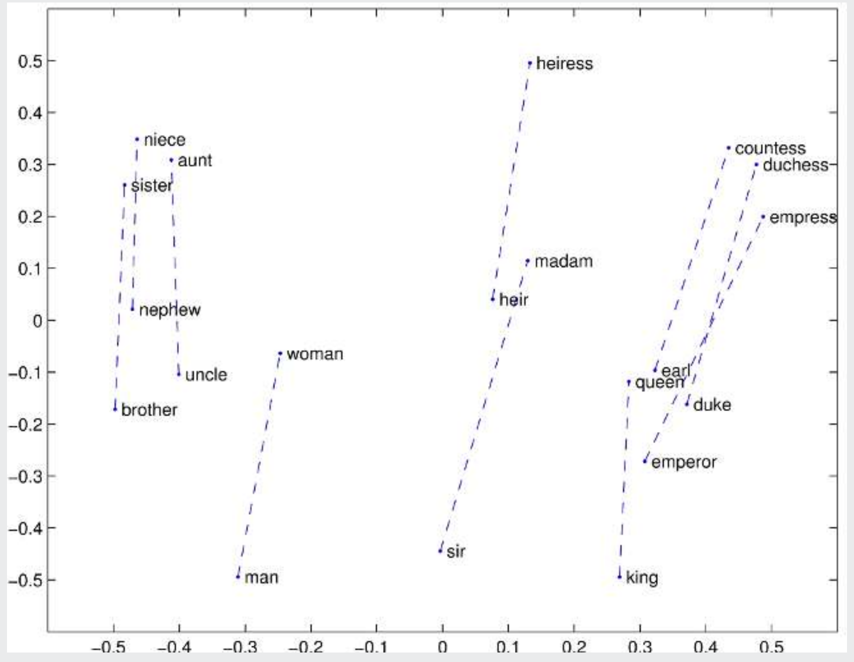
- The above is a simplified illustrative example
- Notice how we can tell apart different animals based on their relationship with people
- Notice how we can distinguish turkey (the animal) from Turkey (the country) as well

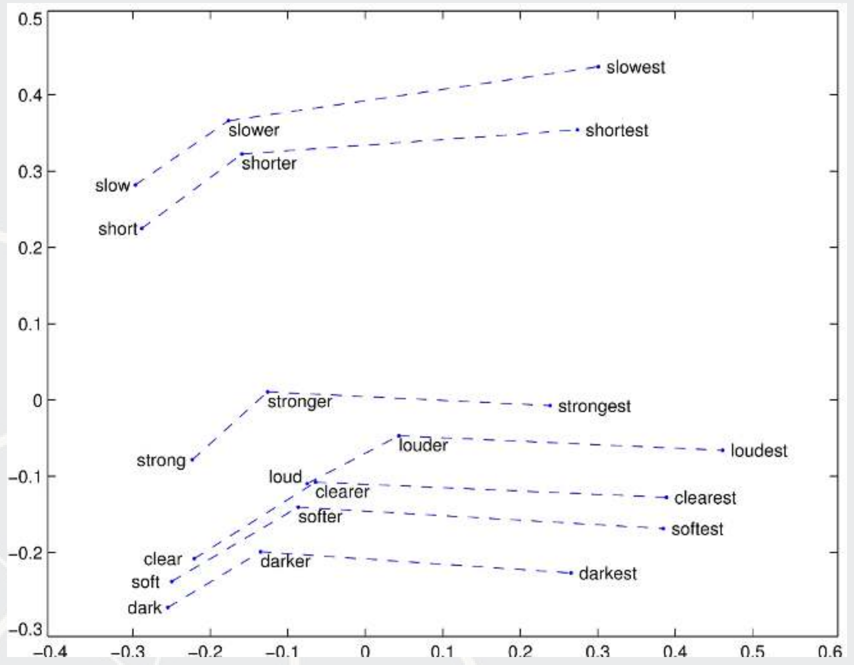
What it retains: word2vec

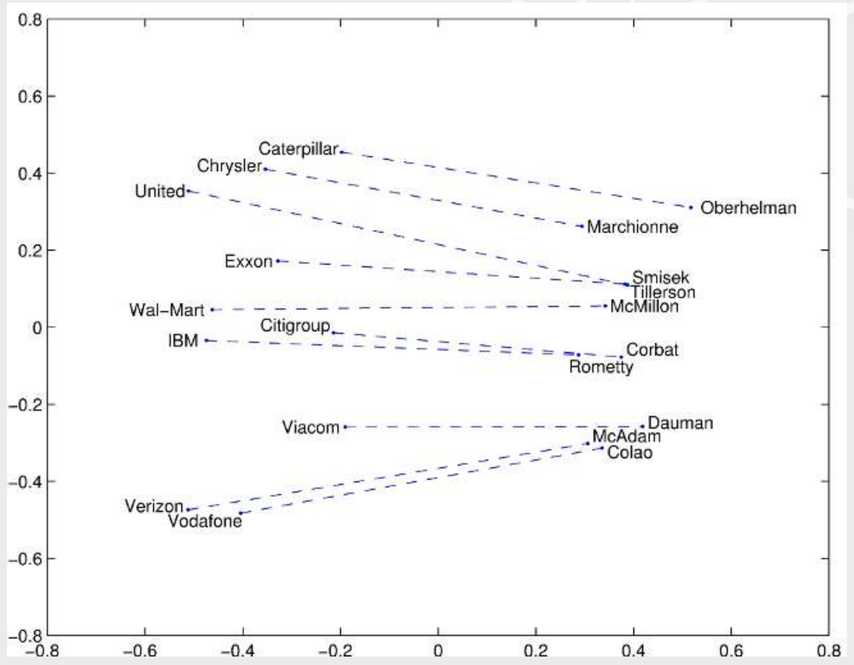


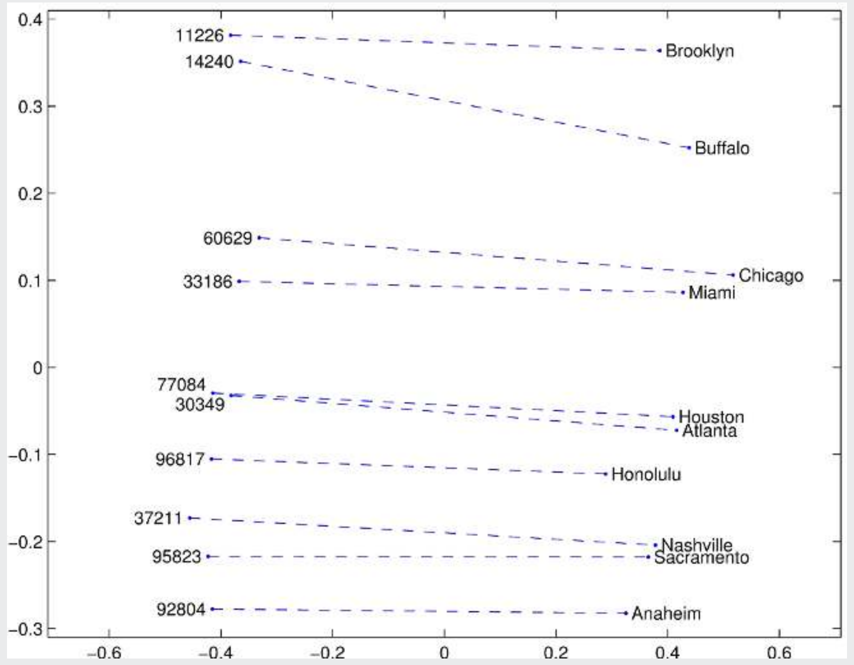


What it retains: GloVe



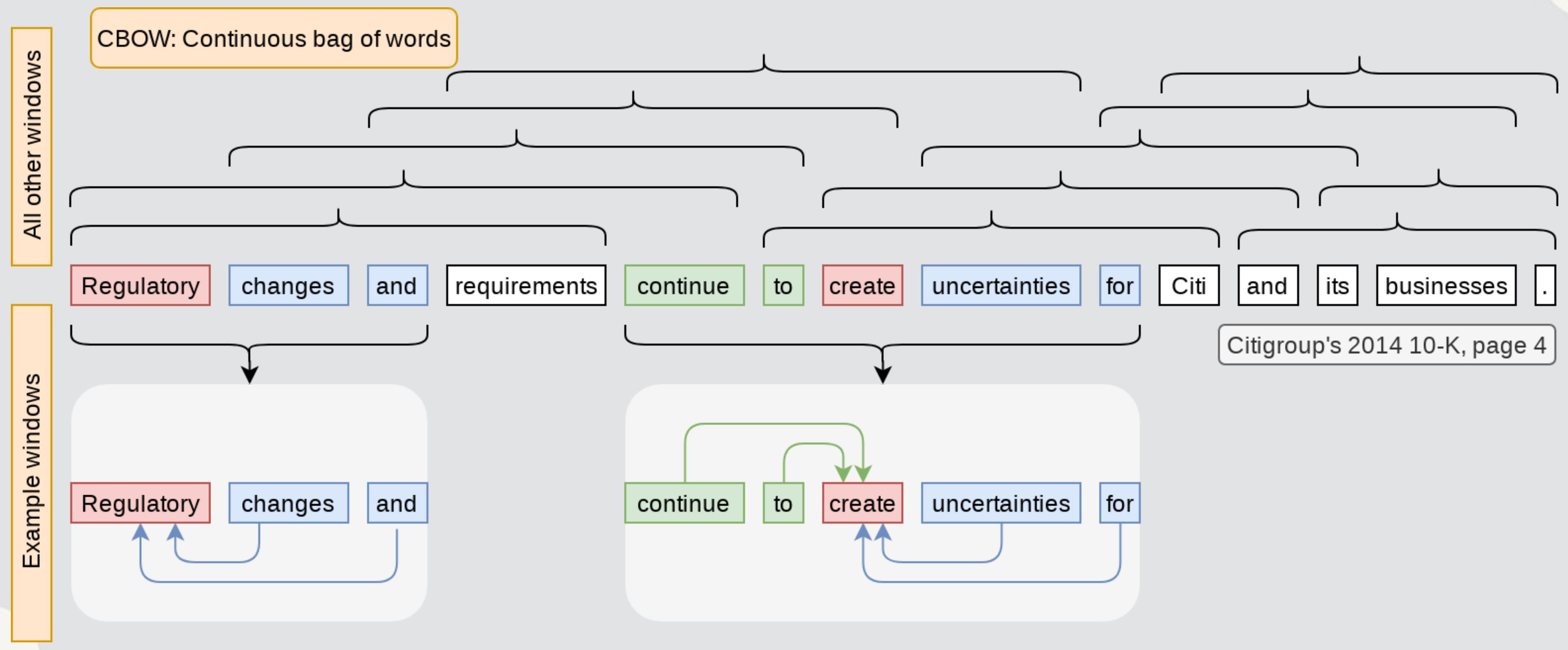






How does word order work?

Infer a word’s meaning from the words around it

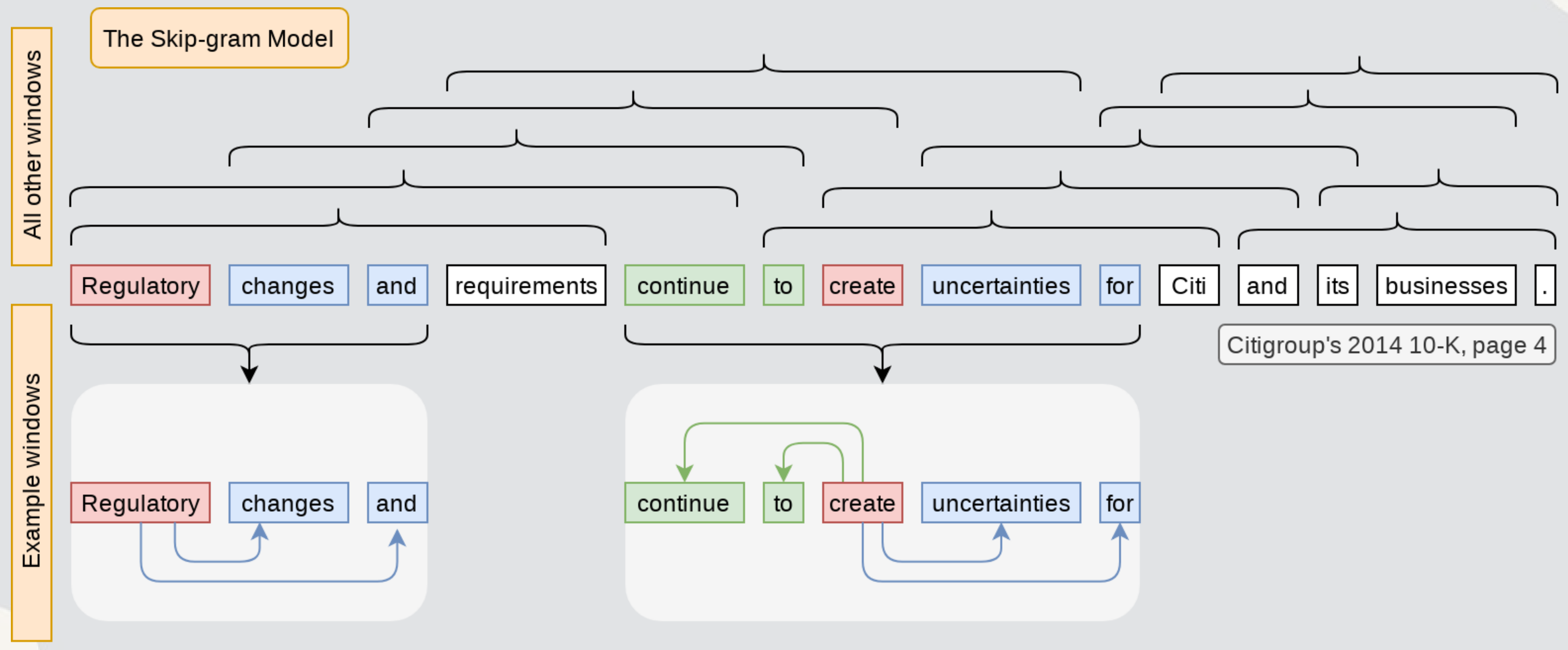


CBOW: Continuous bag of words
All other windows
Example windows
Regulatory changes and requirements continue to create uncertainties for Citi and its businesses .
Citigroup's 2014 10-K, page 4
Regulatory changes and
continue to create uncertainties for

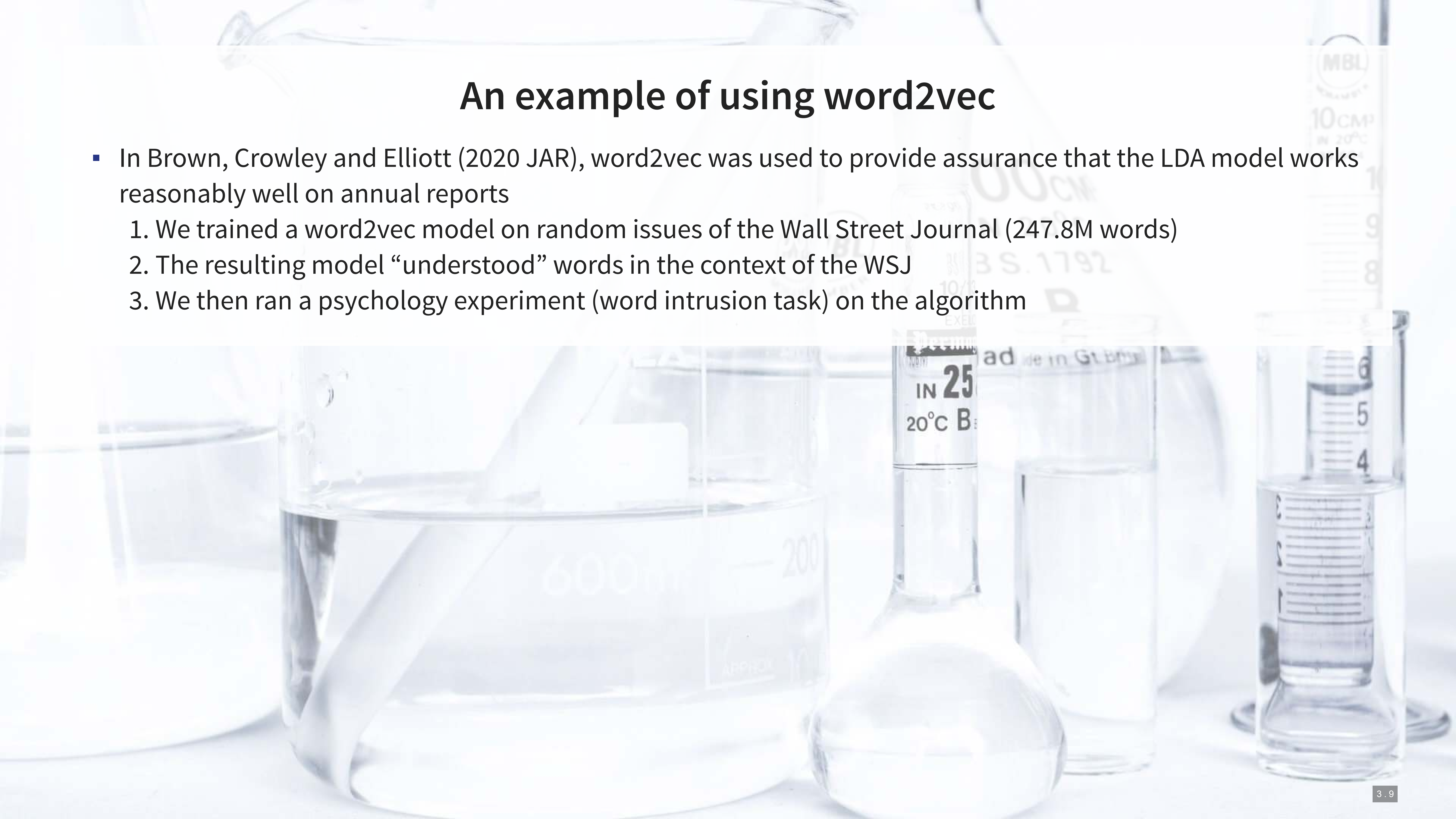
Refered to as CBOW (continuous bag of words)

How else can word order work?

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

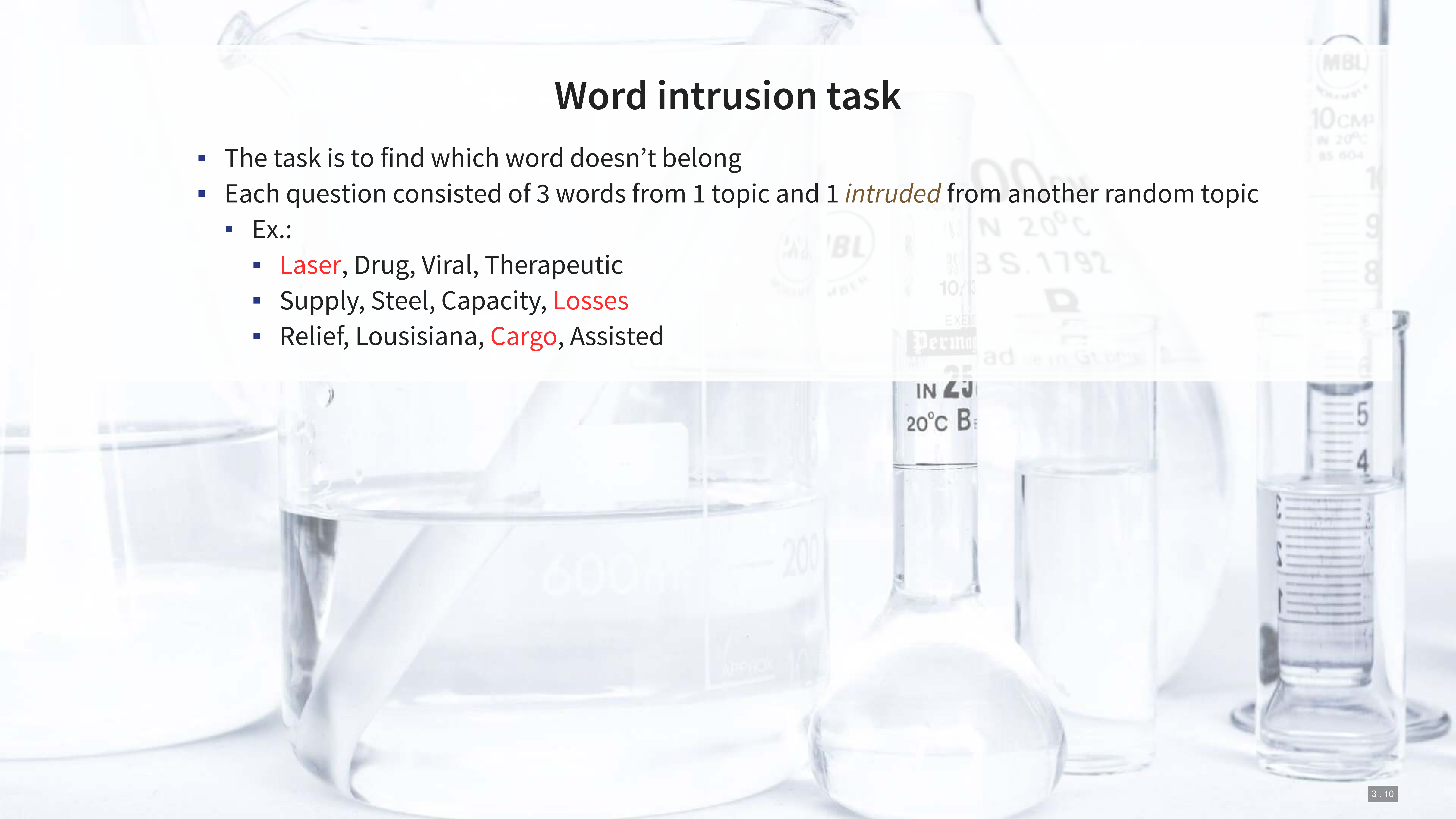


Refered to as the Skip-gram model



An example of using word2vec

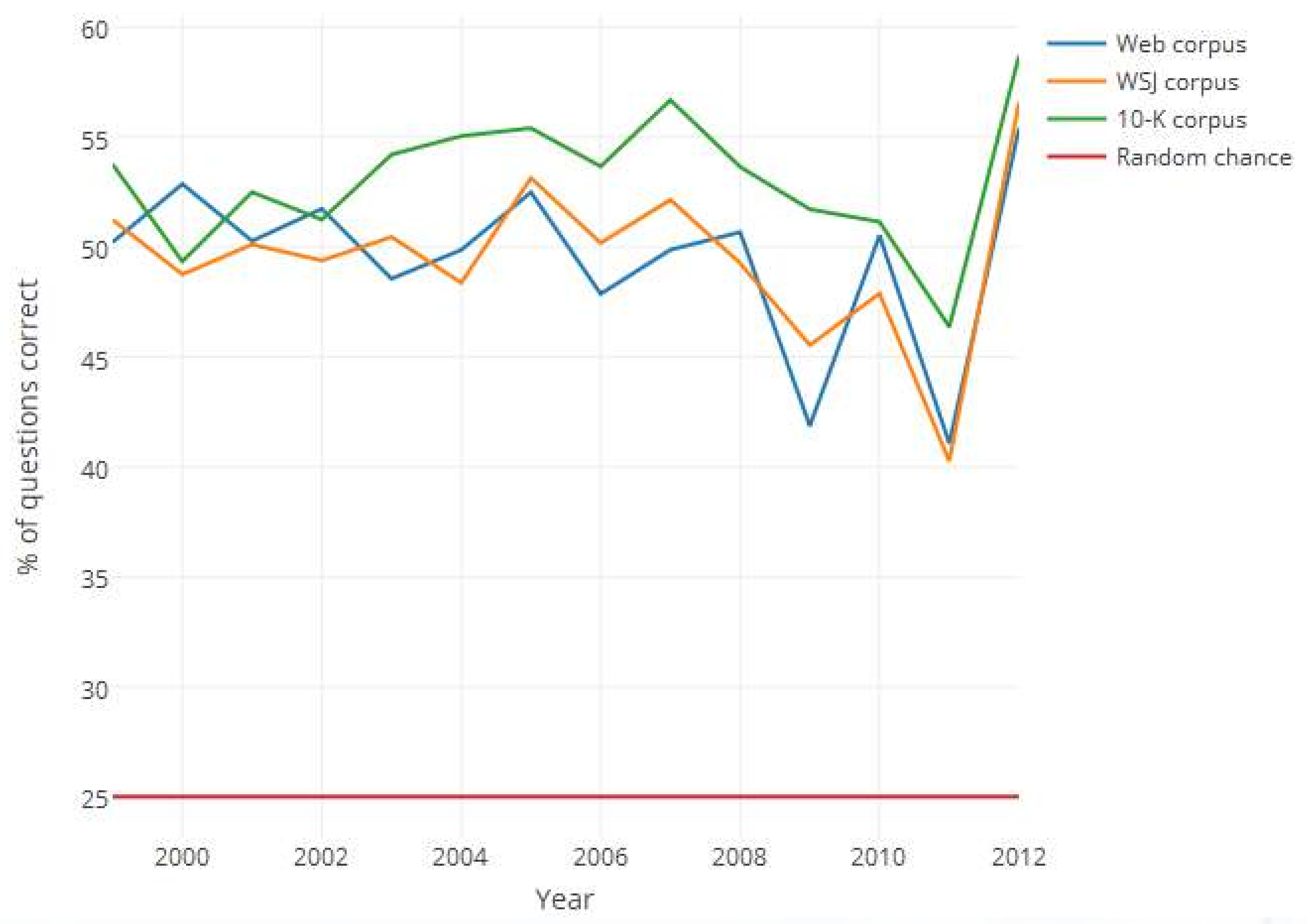
- In Brown, Crowley and Elliott (2020 JAR), word2vec was used to provide assurance that the LDA model works reasonably well on annual reports
  1. We trained a word2vec model on random issues of the Wall Street Journal (247.8M words)
  2. The resulting model “understood” words in the context of the WSJ
  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
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', 0.45336747169494463)
##  ('Yvonne_Stickney', 0.4507041573524475) ('L._Bonauto', 0.4422135353088379)
##  ('gal_pal_Gayle', 0.4408389925956726) ('Alveda_C.', 0.4402790665626526)
##  ('Tupou_V.', 0.4373864233493805) ('K._Letourneau', 0.43510314822196696)
```

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

```python
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
```

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

```
## Queen 2.7364814
```

It's still pretty good though!

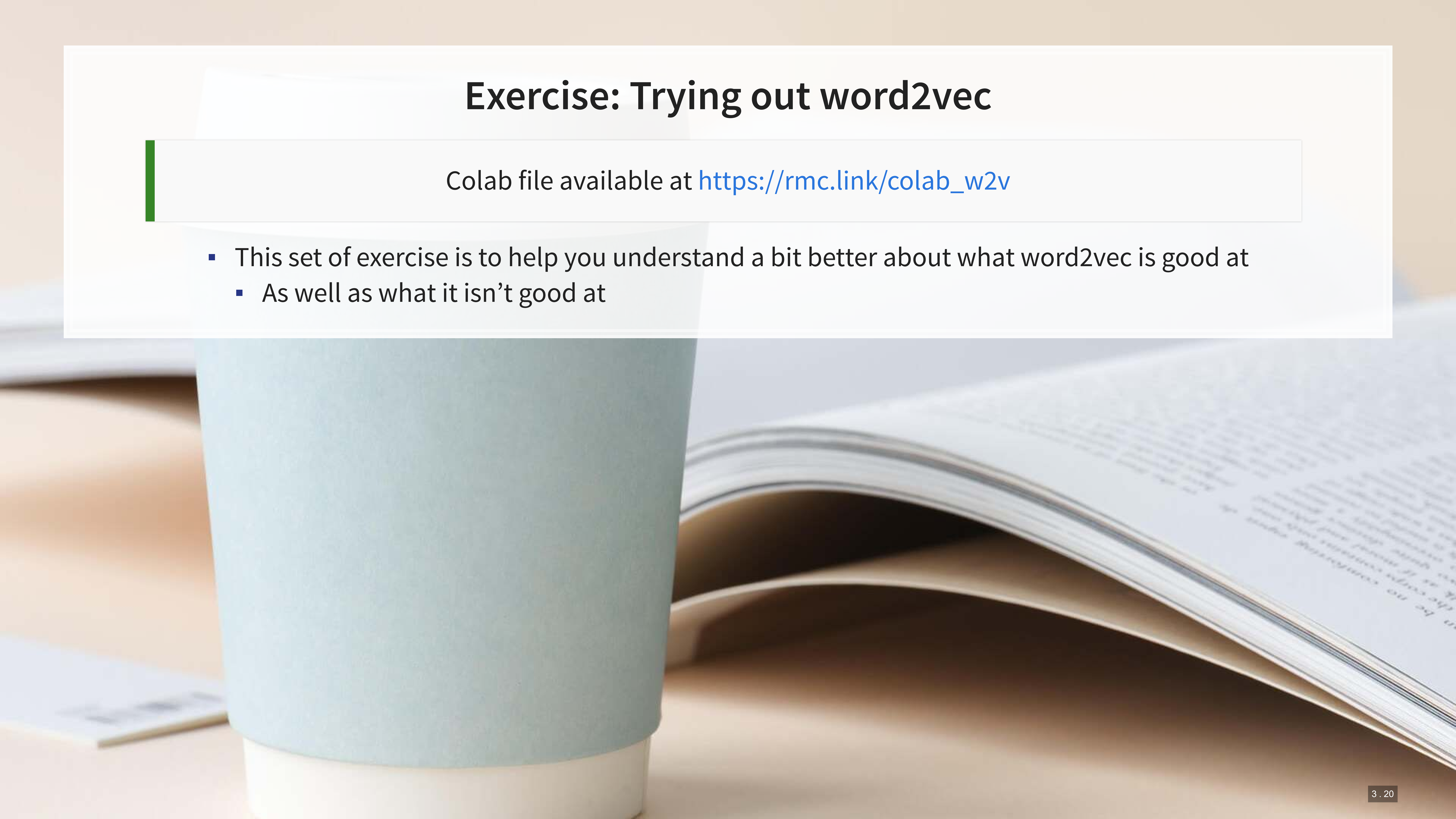
- Note that since word2vec's original answer was `Queen`, this implies it was second best
  - If Queen is the closest word to King, then this would be mathematically uninteresting
    - It's actually 7th though!

```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

Colab file available at https://rmc.link/colab_w2v

- 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

Universal Sentence Encoder (USE)

Universal Sentence Encoder (USE)

Focuses on representing sentence-length chunks of text

Evan Brown, Maalika Patel

Dinner next week

A fun example of with USE

- Predict Shakespeare with Cloud TPUs and Keras

Hamlet

Starting text

SCENE I. Elsinore. A platform before the Castle.

Enter Francisco and Barnardo, two sentinels.

BARNARDO.
Who’s there?

FRANCISCO.
Nay, answer me. Stand and unfold yourself.

BARNARDO.
Long live the King!

Original text

FRANCISCO.
Barnardo?

BARNARDO.
He.

FRANCISCO.
You come most carefully upon your hour.

BARNARDO.
’Tis now struck twelve. Get thee to bed, Francisco.

FRANCISCO.
For this relief much thanks. ’Tis bitter cold,
And I am sick at heart.

BARNARDO.
Have you had quiet gua...

Machine generated (USE)

MARLAND.
So count what? Make haste.

THESEUS.
Where her an unchaste Marcius may wonder such sad,
When dues?

LANCASTER.
I thank you. Ler.

HELEN.
And honest Prince, faith, sweet and Exeter.

LUCENTIO.
It wrose rest your measure made at...

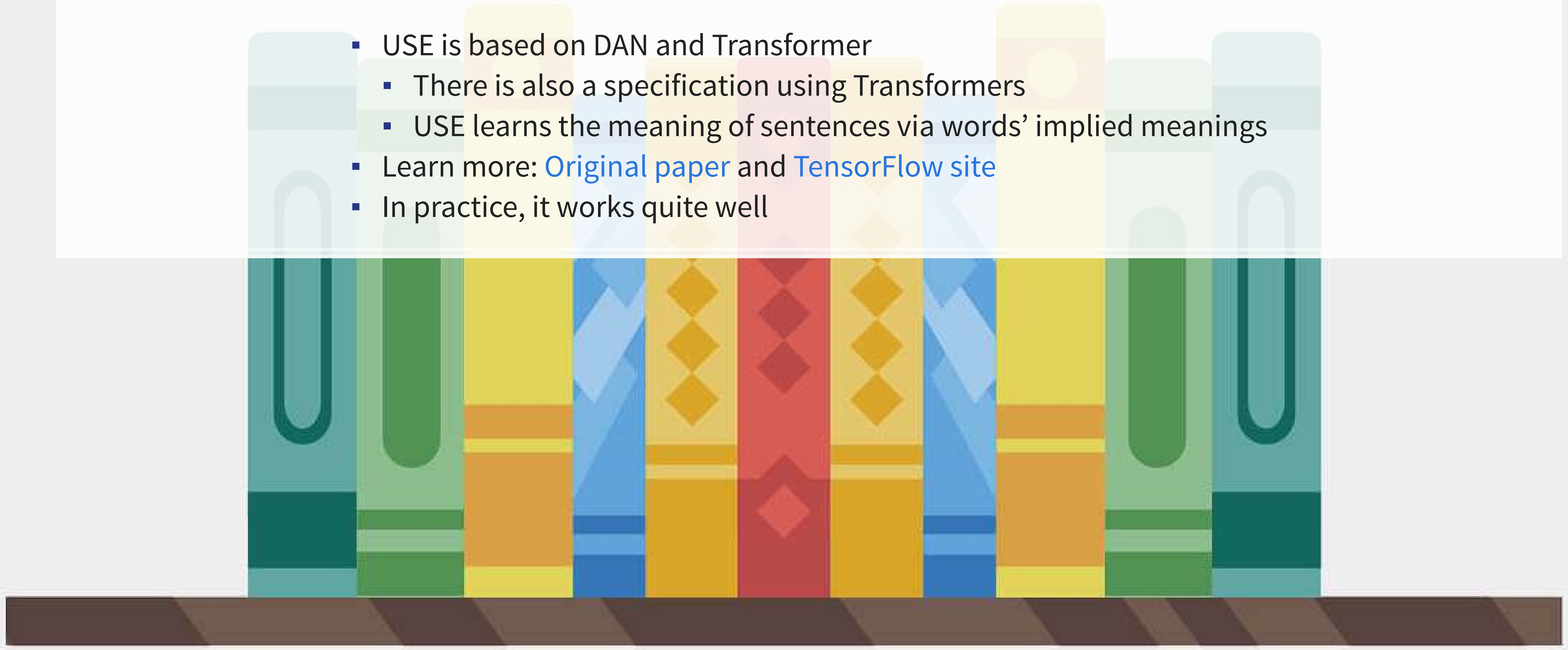
Cavaet on using USE

- One big caveat: USE only knows what it's trained on
  - Ex.: Feeding the same USE algorithm WSJ text

> Samsung Electronics Co., suffering a handset sales slide, revealed a foldable-screen smartphone that folds like a book and opens up to tablet size. Ah, horror? I play Thee to her alone;
> And when we have withdrom him, good all.
> Come, go with no less through.
>
> Enter Don Pedres. A flourish and my money. I will tarry. Well, you do!
>
> LADY CAPULET.
> Farewell; and you are

How does USE work?

- USE is based on DAN and Transformer
  - There is also a specification using Transformers
  - USE learns the meaning of sentences via words' implied meanings
- Learn more: Original paper and TensorFlow site
- In practice, it works quite well



Using USE

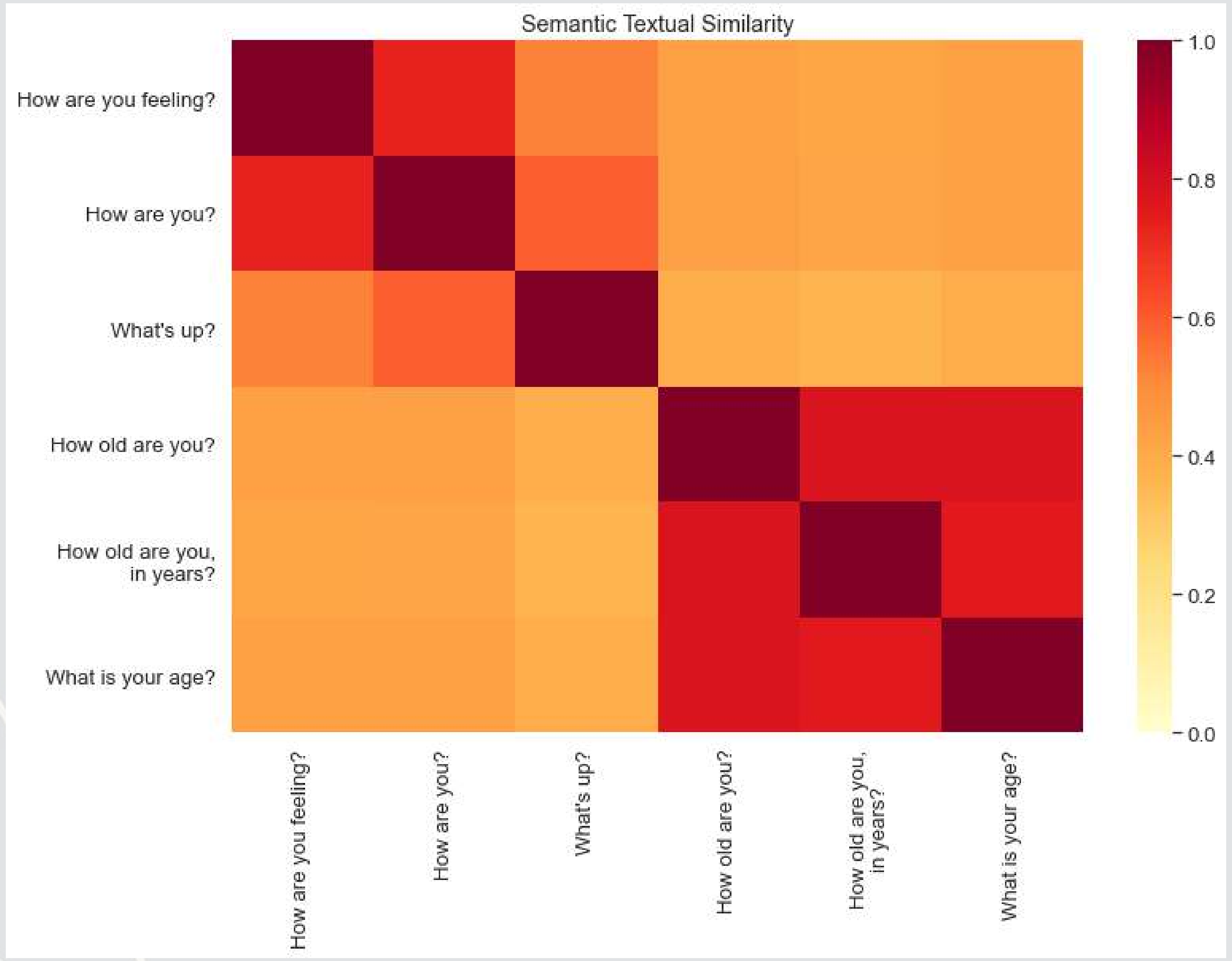
- The model we will be using is the Universal Sentence Encoder (USE) Transformer v5 by Cer et al. (2018)
- Converts text that is between phrase and paragraph length into 512-dimensional vectors

```python
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder-large/5")

messages = ['Two words',
            'This is a sentence.',
            'This is a few sentences.  They are strung together.  They are in one string'
           ]

embeddings = embed(messages)
embeddings
```

```
## <tf.Tensor: shape=(3, 512), dtype=float32, numpy=
## array([[-1.0184747e-02, -3.1019164e-02, -4.2781506e-02, ...,
##          1.0805108e-01,  7.7099161e-05, -6.1001875e-03],
##        [-1.2058644e-02, -3.8627390e-02,  1.5427187e-03, ...,
##          3.3353332e-02, -7.0963770e-02, -1.7223844e-03],
##        [ 3.6280617e-02,  1.7835487e-03, -7.6090815e-03, ...,
##          5.9779502e-02, -1.0792013e-01, -6.0476218e-03]], dtype=float32)>
```

Compare sentences with USE

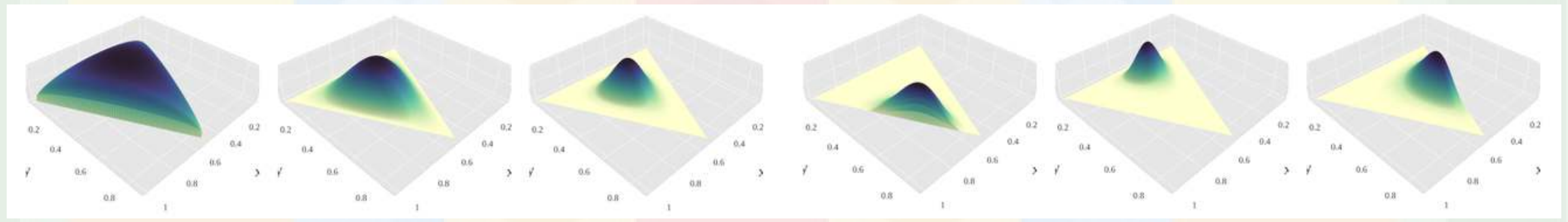
```python
messages = ["How are you feeling?","How are you?","What's up?",
    "How old are you?","How old are you, in years?","What is your age?"]
embeddings = embed(messages)
plot_similarity(messages, embeddings, 90)
```



LDA

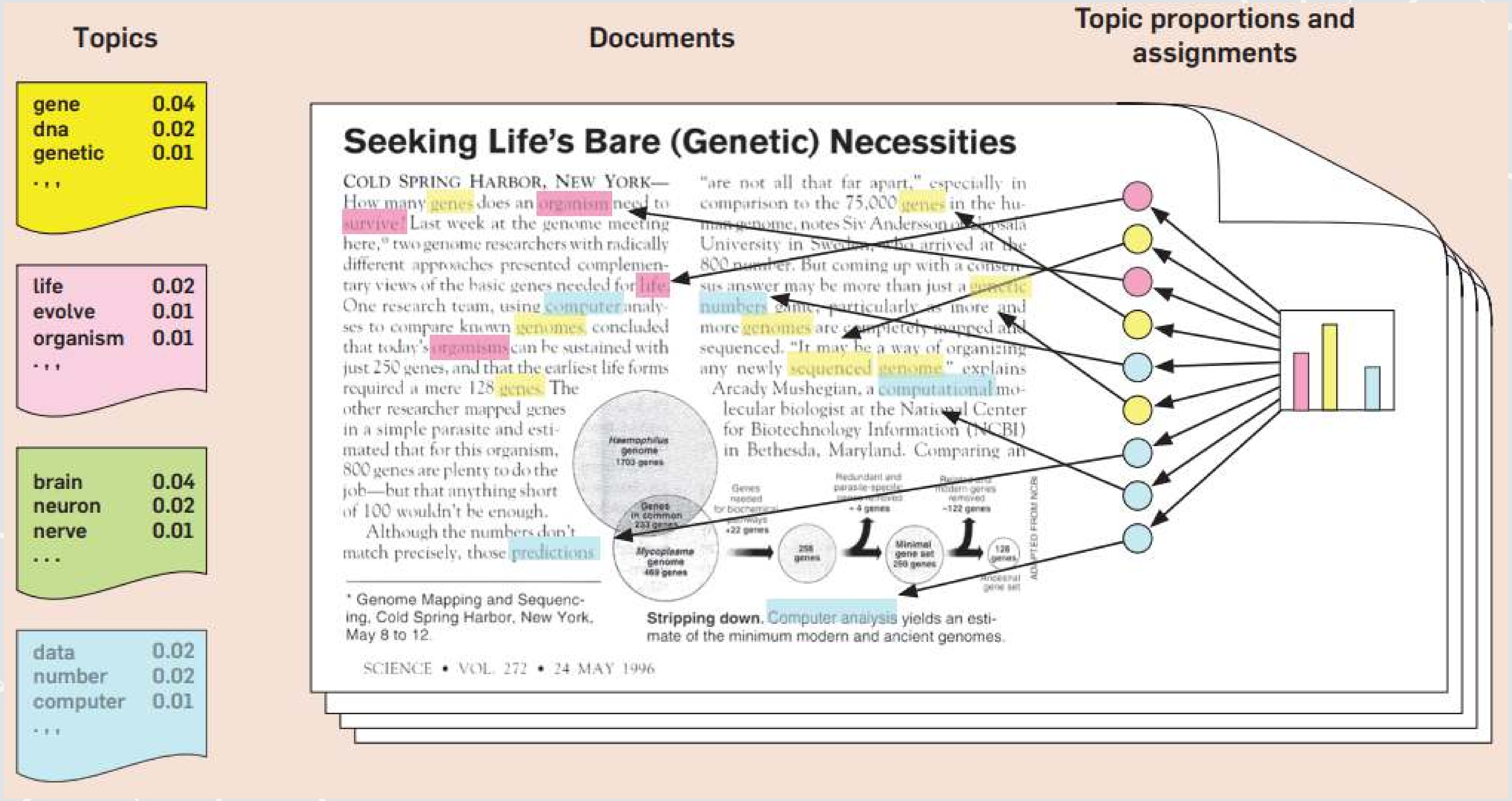
What is LDA?

- **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



From Blei, Ng, and Jordan (2003). More details from the creator

An example of LDA



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
256 genes
Redundant and parasite-specific genes removed −4 genes
Minimal gene set 250 genes
Related and modern genes removed −122 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

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

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!
- 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(articles)
words.filter_extremes(no_below=3, 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 articles]

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

# Save the output
lda.save('../../Data/lda_WSJ')
```

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

```
## M:\Python_environments\Teaching_ML_v1\lib\site-packages\gensim\similarities\__init__.py:15: UserWarning: The gensim.similar
##   warnings.warn(msg)
```

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

2. Examine a topic

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

```
## [('economy', 0.011805763), ('china', 0.010748677), ('%', 0.009966104), ('bank', 0.009680315)]
##  [('growth', 0.006861799), ('official', 0.0065729623), ('sunday', 0.0061146426), ('debt', 0.005799608)]
##  [('u.s.', 0.005709917), ('market', 0.005480105)]
```

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: economy china % bank growth official sunday debt u.s. market
## 1: s&p school home public rating credit downgrade go firm store
## 2: city de blasio president york % campaign candidate tax support
## 3: company price car williams % u.s. end retiree plan open
## 4: benefit security school obama district officer house president claim support
## 5: % fund fee investor stock survey management york hedge u.s.
## 6: work day city life people old plan retirement live take
## 7: government fund play bank crisis president house financial people federal
## 8: company service go ms. rule catsimatidis market mobile u.s. like
## 9: market emerge group yard smith report jet investor bond %
```

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

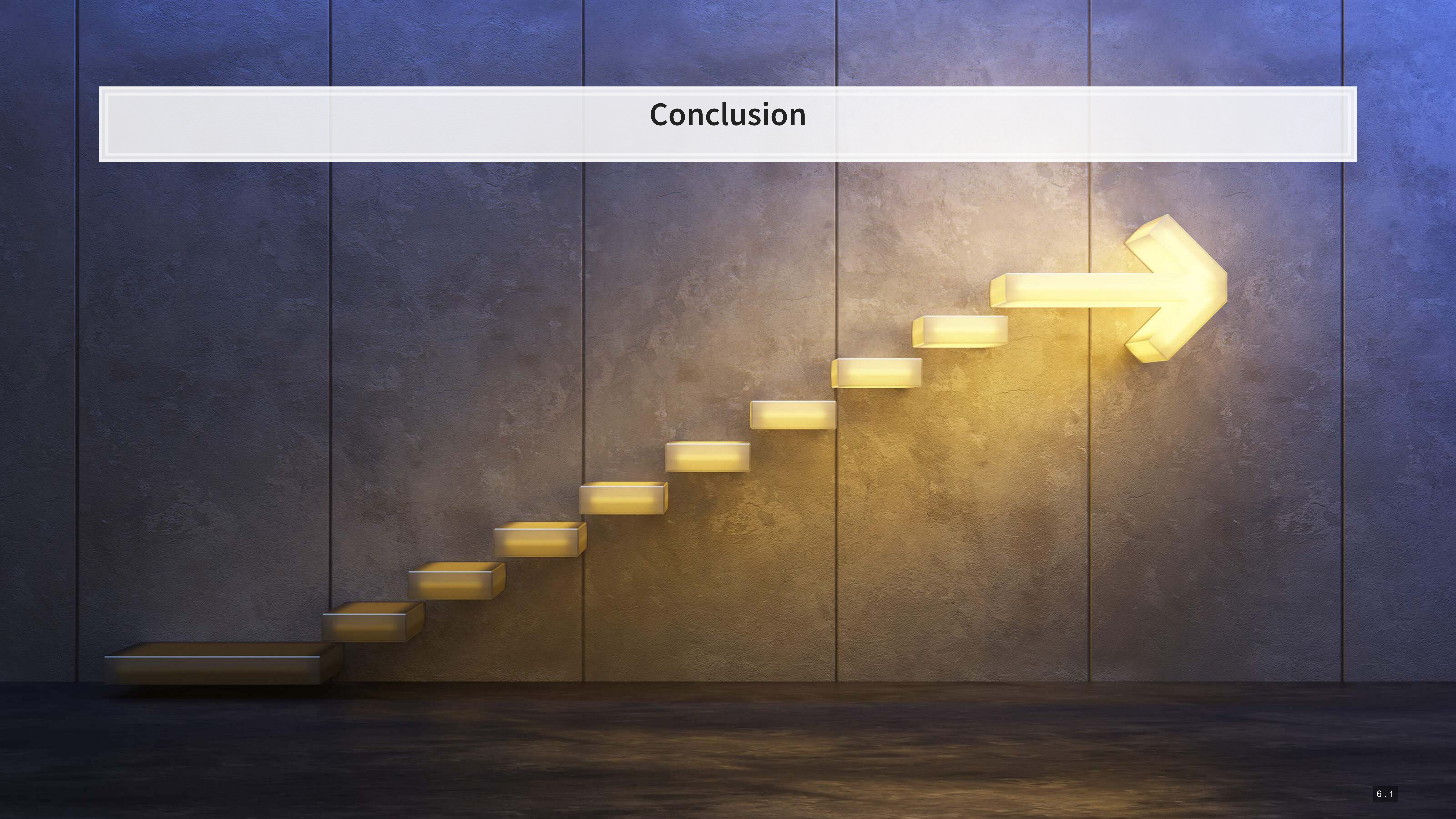
STM

- STM (Structural Topic Modeling) adds two elements to the standard LDA approach:
  1. Covariates can be included in determining the distribution of topics overall ("prevalence")
  2. Covariates can be included in determining the weights of words within topics ("content")

This allows us to better examine the impact of characteristics on textual content

A worked out example is in the R code file

Conclusion



Wrap-up

Embeddings are useful in many contexts, but usually not as the final measure

- Can use them to more accurately compare textual similarity
- Can use them as inputs into a model

LDA models work well as measures and can capture meaningful variation in text

- Provides document-level insight into content distribution

STM provides more power for analyses interested in if textual content differs across groups or treatments

Packages used for these slides

Python

- gensim
- numpy
- pandas
- pyLDAvis
- seaborn
- spacy
- tensorflow
- tensorflow_hub

R

- gender
- knitr
- reticulate
- revealjs
- quanteda
- readtext
- stm
- stmBrowser
- tidyverse

References

- Blei, David M., Andrew Y. Ng, and Michael I. Jordan. “Latent dirichlet allocation.” the Journal of machine Learning research 3 (2003): 993-1022.
- 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.
- Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant et al. “Universal sentence encoder.” arXiv preprint arXiv:1803.11175 (2018).
- Crowley, Richard M., Wenli Huang, and Hai Lu. “Executive Tweets.” Rotman School of Management Working Paper (2020).
- 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.
- Roberts, M.E., Stewart, B.M., Tingley, D., Lucas, C., Leder-Luis, J., Gadarian, S.K., Albertson, B. and Rand, D.G., 2014. Structural topic models for open-ended survey responses. American Journal of Political Science, 58(4), pp.1064-1082.