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

Roberts et al. (2014 AJPS)

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

Crowley and Wong (2022 working)

- Demonstrates a usage of embedding methods at the sentence level
- Uses this to examine sentiment (e.g., in Loughran and McDonald 2011) in a fine-grained manner

Technical Discussion

Python

- LDA
  - gensim is the easiest to use in general
    - Installation is not always straightforward
- 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
  - `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
  - See the `word2vec` and `rword2vec` packages

> 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

How I work [on ML-based projects]

High level overview: Process

1. Idea generation
2. Prototyping
   - Testing out different approaches to measurement or data collection
3. Data collection
   - Implement in an automated fashion
   - Note that sometimes you need to update the data in the review process.
4. Implementation
   - Run/train the desired algorithm on the collected data
   - Again, best to keep this automated
5. Data manipulation
   - Build a data set with the implementation’s output + standard measures
   - Automate this too
6. Econometrics
   - Keep automating
7. Writing

High level overview: Tools

Hardware

- Data stored in RAID 1 arrays (redundant disks) and duplicated across machines
  - Use HDDs for large amounts of data, SSDs for small amounts (<2TB) or temporary storage
- The more CPU cores, the better (so long as memory scales to match)
- Large memory amount for text analytics
  - 64-128GB is good for most tasks
  - 512GB for large matrix problems
- **Nvidia** GPU for neural network training and inference
  - 10-100x speedup for most algorithms
  - Nvidia is needed for CUDA
  - CUDA is needed for most ML libraries

Software

- Python via miniconda for data collection and processing
  - Pycharm and JupyterLab for GUIs
- R for data manipulation, econometrics, and visualization
  - RStudio for GUI
- Stata for econometrics
- sftp for data transfer
- Nomachine for remote access

> I run everything under Linux – a bit more stable for long computations and better multithreading in python

Working with python

- Preferred distribution: `miniconda`
  - Anaconda without the GUI frontend
- Why?
  - Command line simplicity of `pip`
  - Solid collection of packages when including `conda-forge`
  - Significantly easier installs of more complex software
    - E.g., Tensorflow + CUDA + cuDNN as a one-liner
  - Handles virtual environments

> Why not base python?

- `pip` + `virtualenv` is fairly flaky when you need specific installs across python versions on Linux
  - It’s a bit better on Windows
- In the past I managed `virtualenv`s with `poetry`, but since summer 2021 that is no longer reliable

Base miniconda setup

1. Install from https://docs.conda.io/en/latest/miniconda.html
2. Install to a custom folder where you have sufficient storage
   - E.g., I use `/media/Data/Anaconda/` on Linux or `D:\Anaconda` on Windows
3. Install mamba (faster package resolution): `conda install -c conda-forge mamba`
4. Install JupyterLab: `mamba install -c conda-forge jupyterlab`
5. Install the kernel module: `mamba install -c conda-forge nb_conda_kernels`
   - Lets you access all your anaconda virtual environments from the same JupyterLab instance
6. Install `pip`: mamba install pip
   - Conda + `conda-forge` doesn't have *every* package
7. Add `conda-forge` to the default channel list: `conda config --add channels conda-forge`

> Make separate projects using `mamba create -n $name python=$version mamba pip ipykernel $other_packages_here`

Miniconda for this class

```
mamba create -n MLSS python=3.9 mamba pip ipykernel ipywidgets numpy pandas statsmodels scikit-learn nltk scipy spacy textacy
  xgboost matplotlib seaborn umap-learn requests graphviz python-graphviz shap pillow tensorflow tensorflow-hub wasabi==0.9.1
  gensim pyLDAvis keras-preprocessing doubleml pydot
conda activate MLSS
conda install mkl-service
conda install -c powerai tensorflow-gan
python -m spacy download en_core_web_sm
conda deactivate
```

- Also need to install graphviz: https://graphviz.org/

Miniconda for Crowley and Wong (2022)

```
# Base environment
mamba create -n R017 python=3.9 mamba pip ipykernel spacy textacy numpy scikit-learn h5py dask dask-ml nltk cython
mamba install opencv psutil
conda activate R017
conda install mkl-service
python -m spacy download en_core_web_sm
conda deactivate
```

```
# 10-K downloader and parser
mamba create -n S001 python=3.9 mamba pip ipykernel spacy textacy numpy scikit-learn h5py dask dask-ml nltk cython
conda activate S001
conda install mkl-service
python -m spacy download en_core_web_sm
conda deactivate
```

```
# FinBERT
mamba create -n T017 python=3.9 mamba pip ipykernel cython h5py sentencepiece python_abi numpy pandas pytz
conda activate T017
mamba install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
mamba install -c huggingface transformers==4.14.1 tokenizers==0.10.3 huggingface_hub
conda install mkl-service
conda deactivate
sudo apt-get install git-lfs
```

Working with R

- R doesn’t have as robust of virtual environments as python
  - `renv` might work well enough these days though
- Instead, I tend to keep track of dependencies in the scripts themselves
  - Especially if it isn’t from CRAN
    - I.e., just leave a comment with the install procedure in the script

Coding paradigms

`tidyverse`

- Clean, readable, moderately efficient
- Based around a pipe operator `%>%`

```r
df <- df %>%
  group_by('gvkey') %>%
  mutate(obs=n()) %>%
  ungroup()
```

`data.table`

- Computationally efficient and SQL-like
- Overloads `[` with new syntax

```r
df[ , obs := .N, by='gvkey']
```

Working with R

- Tidyverse syntax is great for data manipulation
  - Leaves an easy to understand list of transformations in the code
  - E.g.: Compiling data output from python scripts and from databases into 1 file for analysis
- Data.table is great for:
  - Time consuming computations
  - Computations on large datasets
    - Lower memory usage than base R, tidyverse, or pandas (python)
  - E.g.: Computing pairwise distances in matched observations across a 20M row dataset
- R includes matrix algebra in the base install
- R supports MKL for more efficient CPU usage

Econometrics

- R
  - Strong programming tools and consistent syntax
  - All standard econometric tools are included or available through CRAN
  - Some libraries are efficiently multithreaded
  - Some implementations of algorithms are significantly faster
    - E.g., HDFE through `fixest` is 10-50x faster than Stata
  - Some newer methods are only in R (or R and python both)
- Stata
  - The language itself is lacking, so custom functions are tricky
  - All standard econometric tools are included or available through `ssc`
    - E.g., `reghdfe`, `ppmlhdfe`, `egen`, `unique`, `outreg2`
  - Many specialized econometric tools are also included
  - As of more recent versions, it supports having multiple data frames in memory
  - Efficient multithreading is paywalled

How did I do this paper?

1. Idea generation: Came out of discussion with Franco when visiting Rotman
2. Prototyping: Tried a variety of approaches based on LDA, word vectors, and dependency parsers before settling on OpenIE
   - OpenIE was the only algorithm that really captured the context of the words
3. Data collection: python script to download and parse 10-K filings
4. Implementation:
   - OpenIE in Java with ~120GB RAM allocated
   - Masking and filtering code in python, multithreaded
   - USE in python (Tensorflow) on GPU
   - MiniBatch K-means in python (scikit-learn)
   - Instruments by coding Google Apps Script from python (auto-generate forms)
5. Data manipulation: R to add everything into a giant data-frame, mostly using `data.table`
6. Econometrics: In R
   - OLS is just implemented in Base R; CV LASSO in `glmnet`; Double LASSO in `hdm`
   - Some simulation in R: Coded in a tight loop using `doParallel` and `foreach`
7. Writing

Tips on computation

1. If you are working with csv files, gzip them. Python and R both work well with them
2. Data bigger than your RAM? Use HDF5 – SAS-like functionality usable from R and python
3. Always keep documentation for your code, including the order to run everything
4. Python specific
   - Multithreading in python is worth the extra work if the process will take hours
   - Numpy arrays are nearly always faster if you are doing math on lists.
   - If you can write your problem as matrix algebra, implement it as such in numpy.
     - Matrices are extraordinarily efficient.
   - Many python libraries use C code under the hood. These are significantly faster than pure python libraries. You can compile your own C code libraries *using python code* using `Cython`
5. R specific
   - Learn both tidyverse and data.table, and use data.table for slower tasks or those that use a lot of RAM
   - Matrix algebra is, again, the most efficient approach.
     - Need to calculate CAPM-based returns? You use a linear regression. Linear regression is solvable in closed form with matrix algebra
6. Stata specific
   - You can run multiple copies of Stata side-by-side to max out your CPU :)

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 (from smallest to largest input):
  - 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 relations
    - Grammatical relations (weakly when using stemming)
    - Social relationships
    - etc.

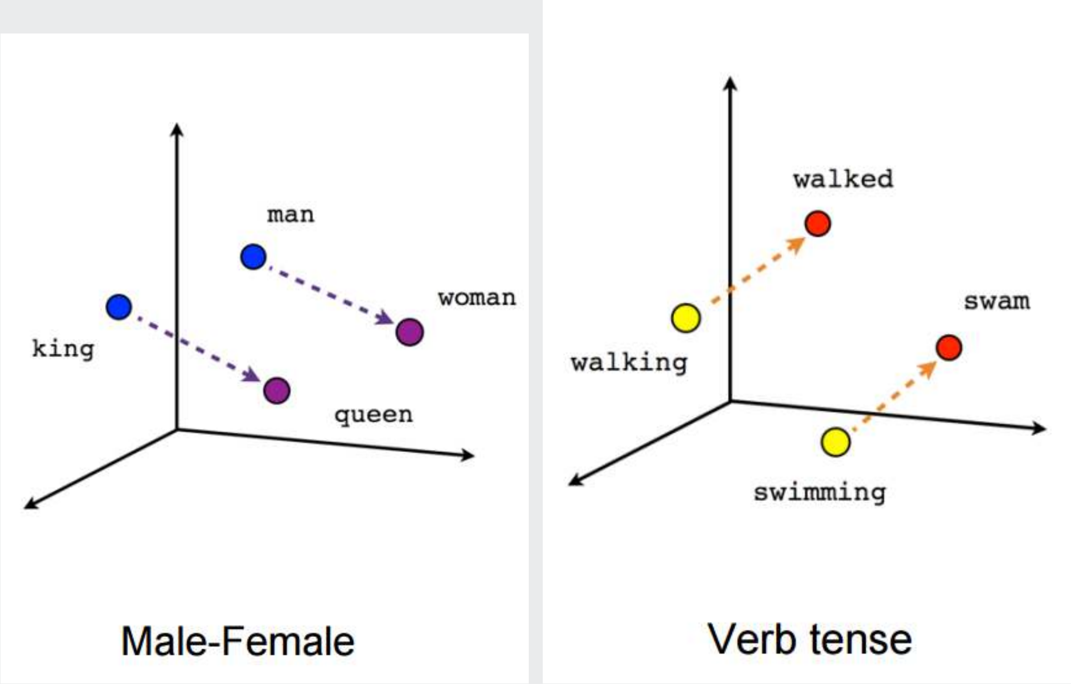
> Word vectors try to encapsulate all of the above implicitly, through by encoding words as a vector based on how features manifest themselves in text

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

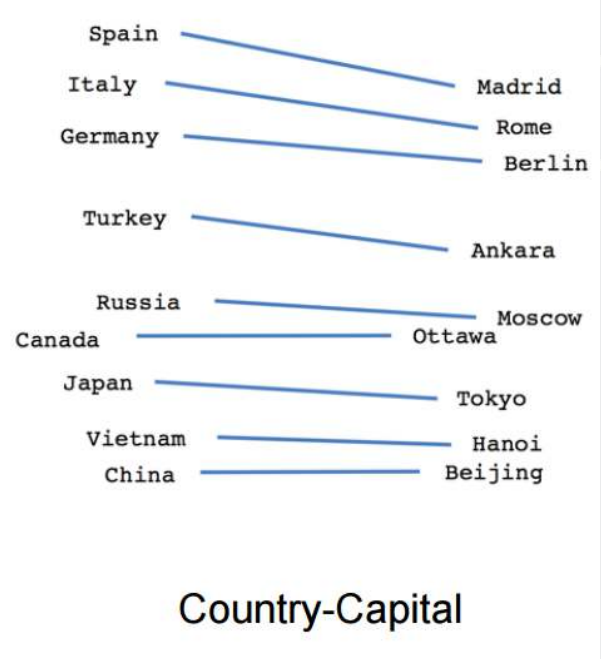
- The above is a simplified illustrative example
- Notice how we can tell apart different animals based on their relationship with people
- Notice how we can distinguish turkey (the animal) from Turkey (the country) as well

What it retains: word2vec



Male-Female

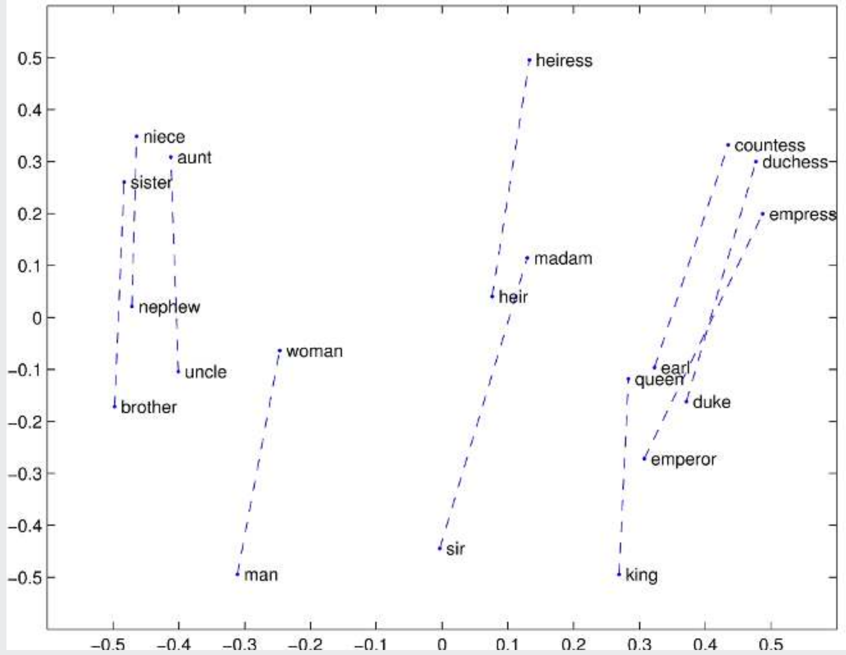
Verb tense

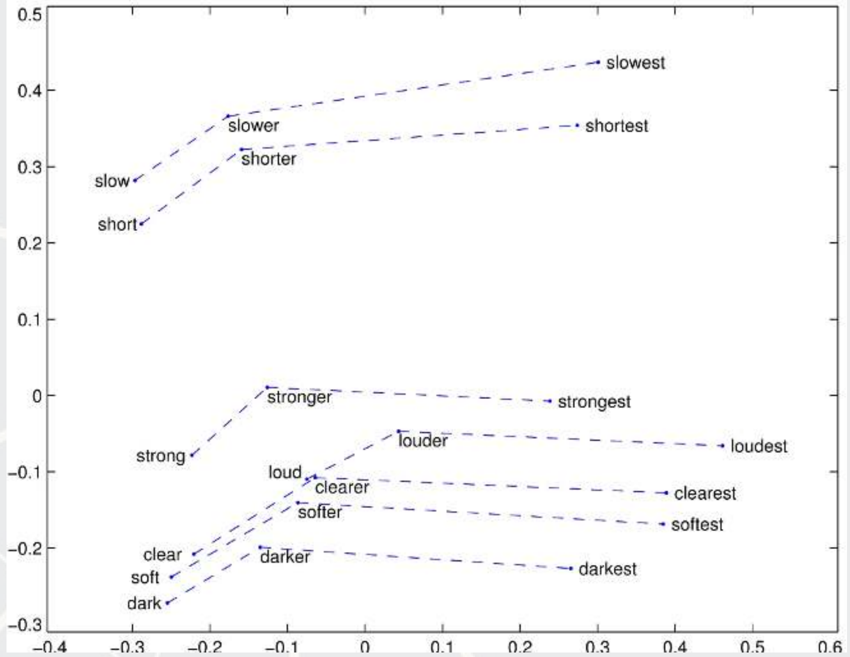


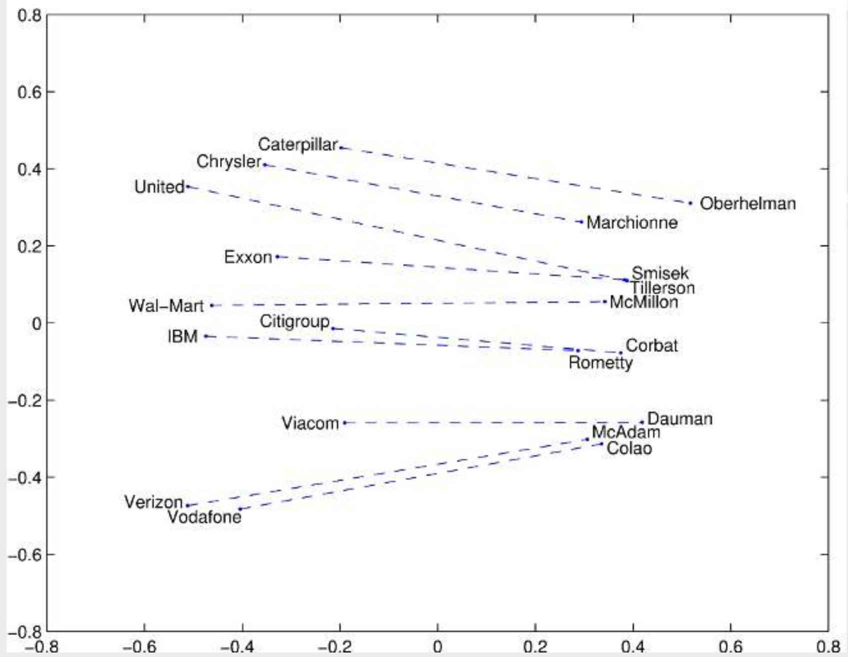
Country-Capital

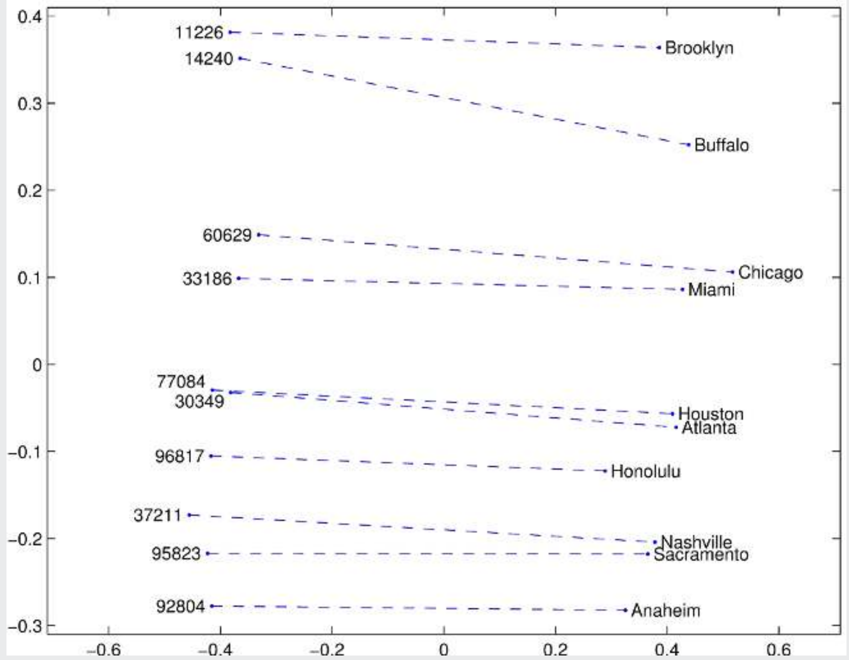
> Relations are retained as vectors between points (distance + direction)

What it retains: GloVe



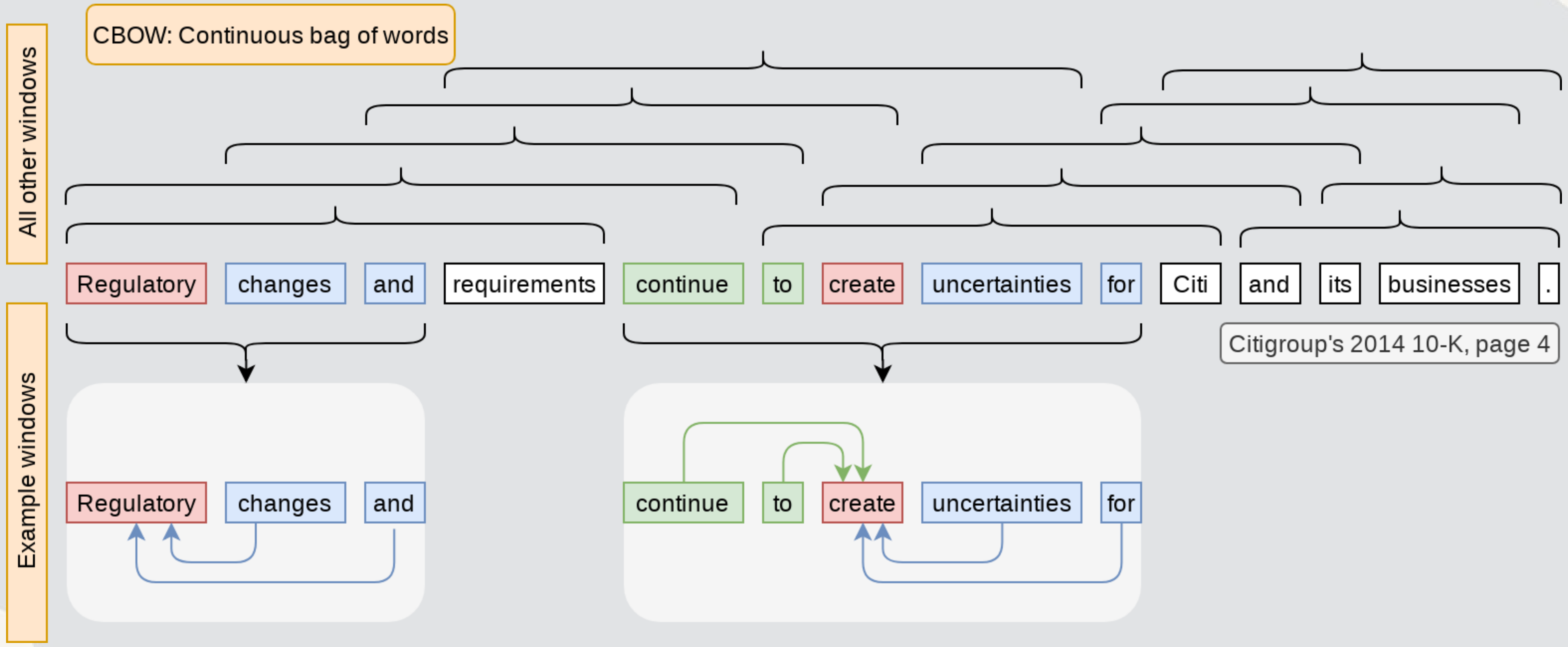






How does word order work?

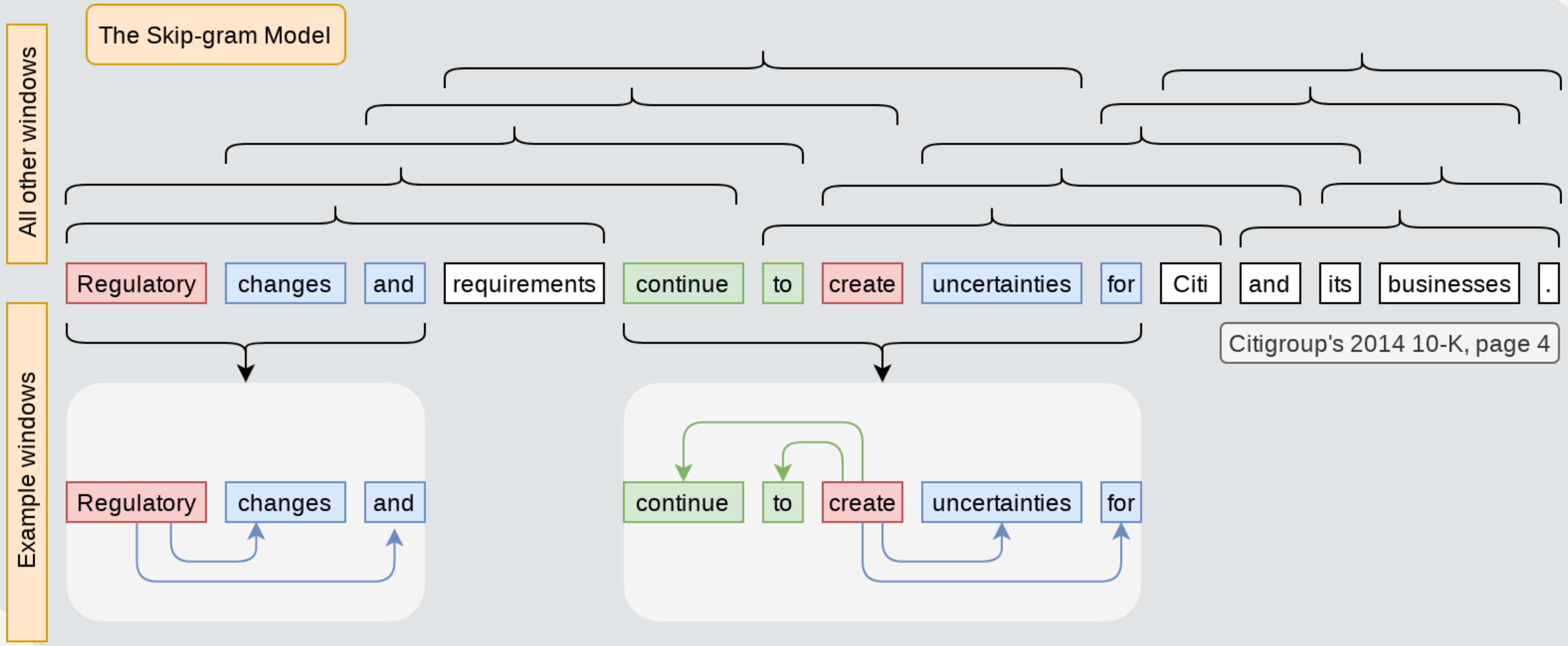
Infer a word’s meaning from the words around it



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

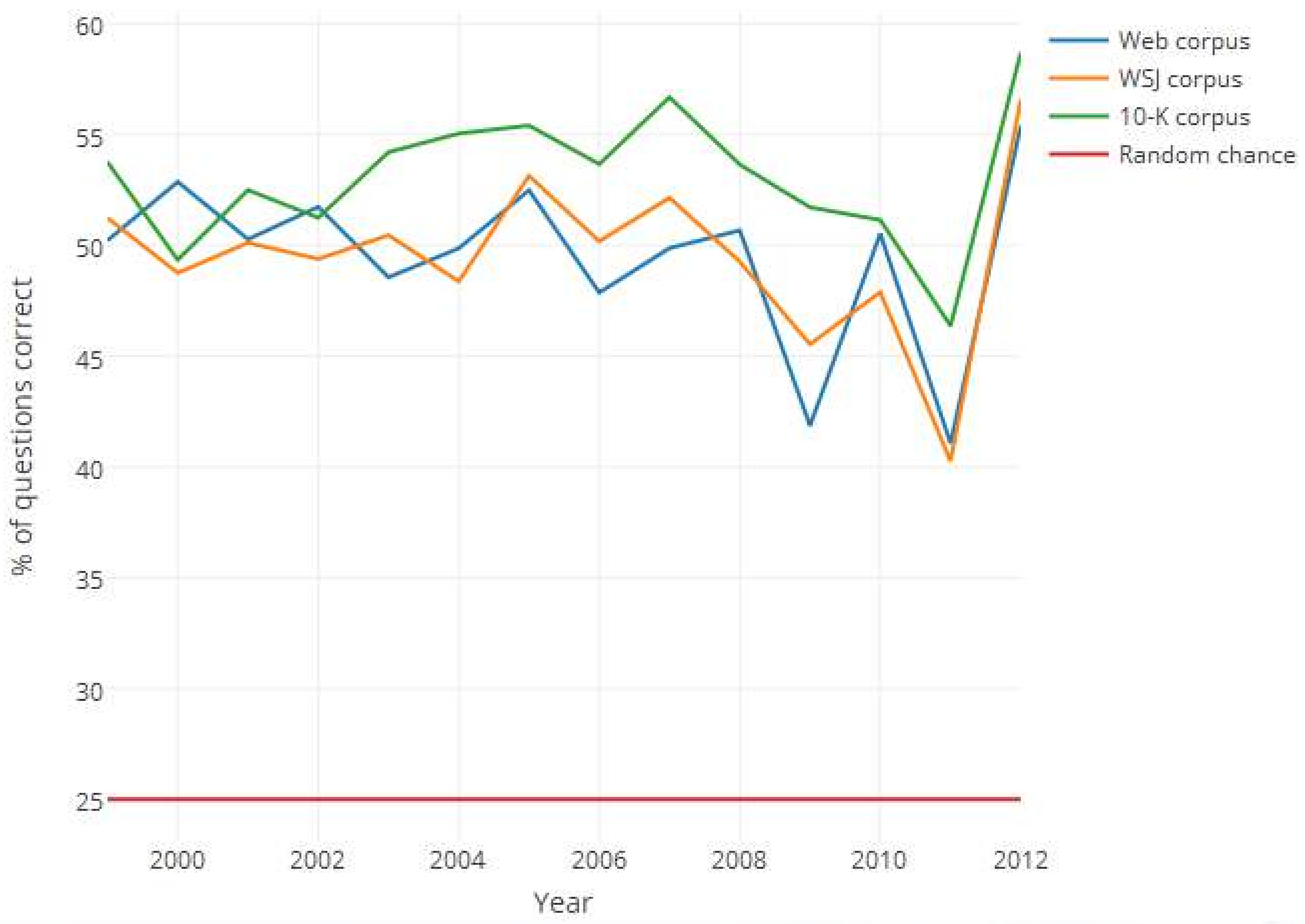
An example of using word2vec

- In 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

> 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, Louisiana, 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

```
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)
```

```
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

```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!

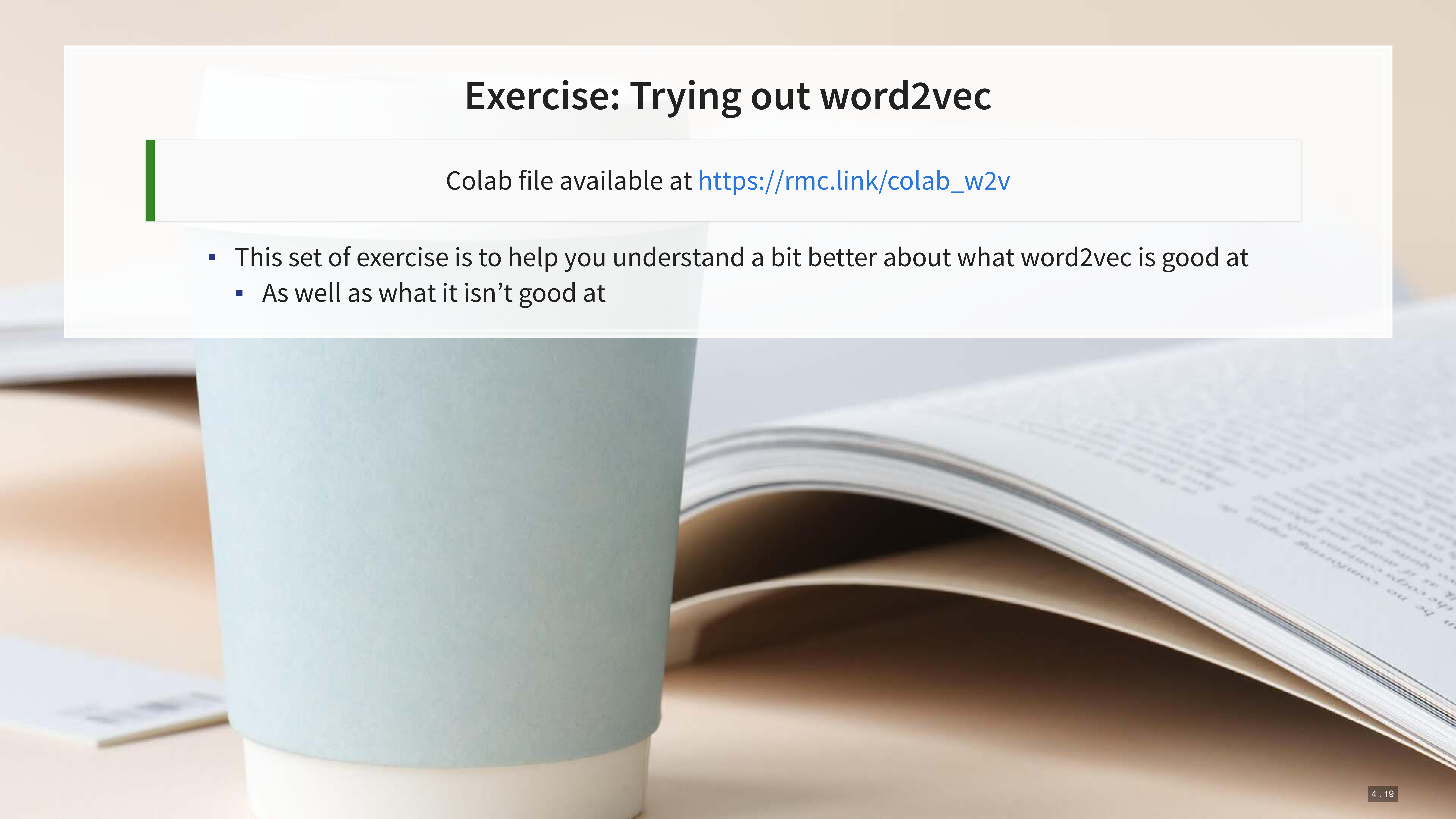
- Note that since word2vec’s original answer was `Queen`, this implies it was second best
  - If Queen is the closest word to King, then this would be mathematically uninteresting
    - It’s actually 7th though!

```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

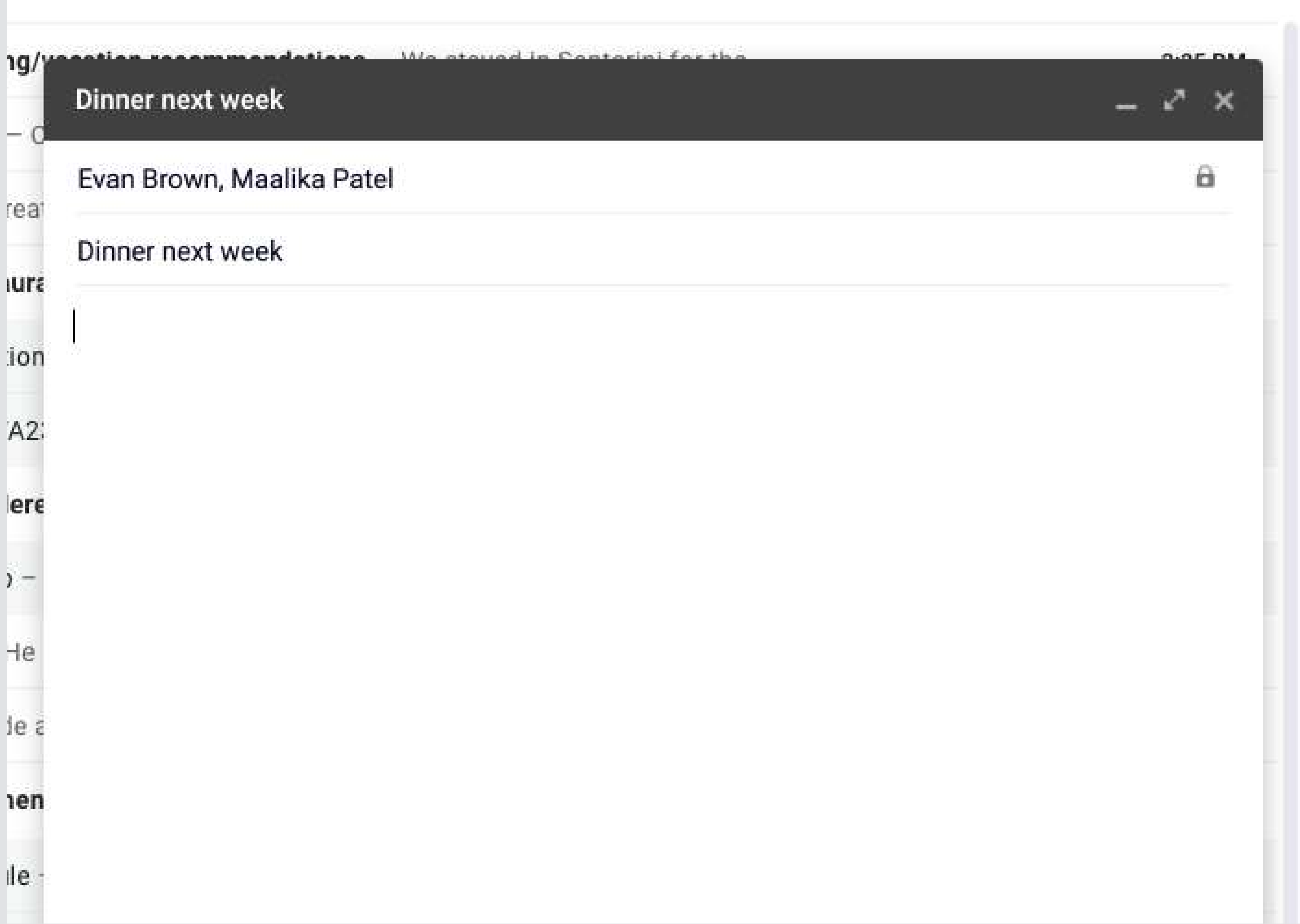
> Colab file available at https://rmc.link/colab_w2v

- 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



Dinner next week

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...

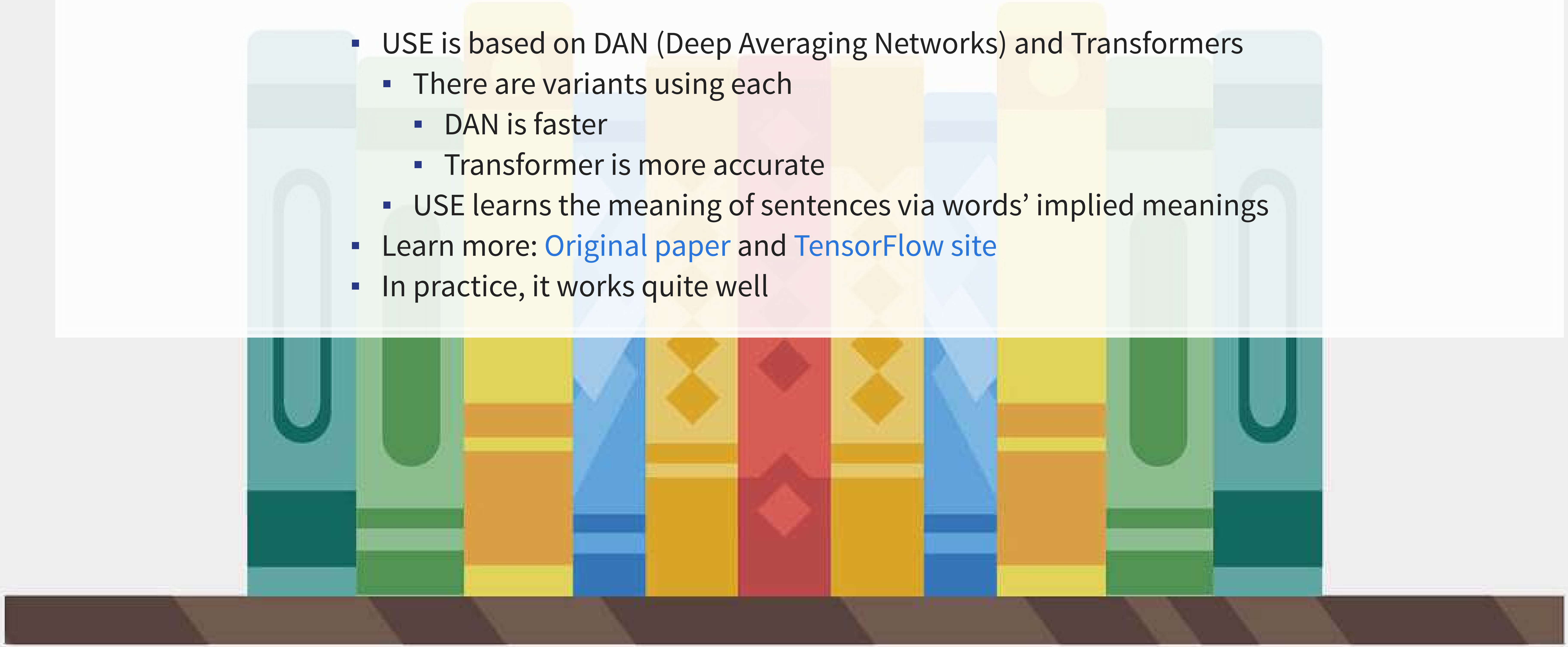
Cavaet on using USE

- 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 (Deep Averaging Networks) and Transformers
  - There are variants using each
    - DAN is faster
    - Transformer is more accurate
  - 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

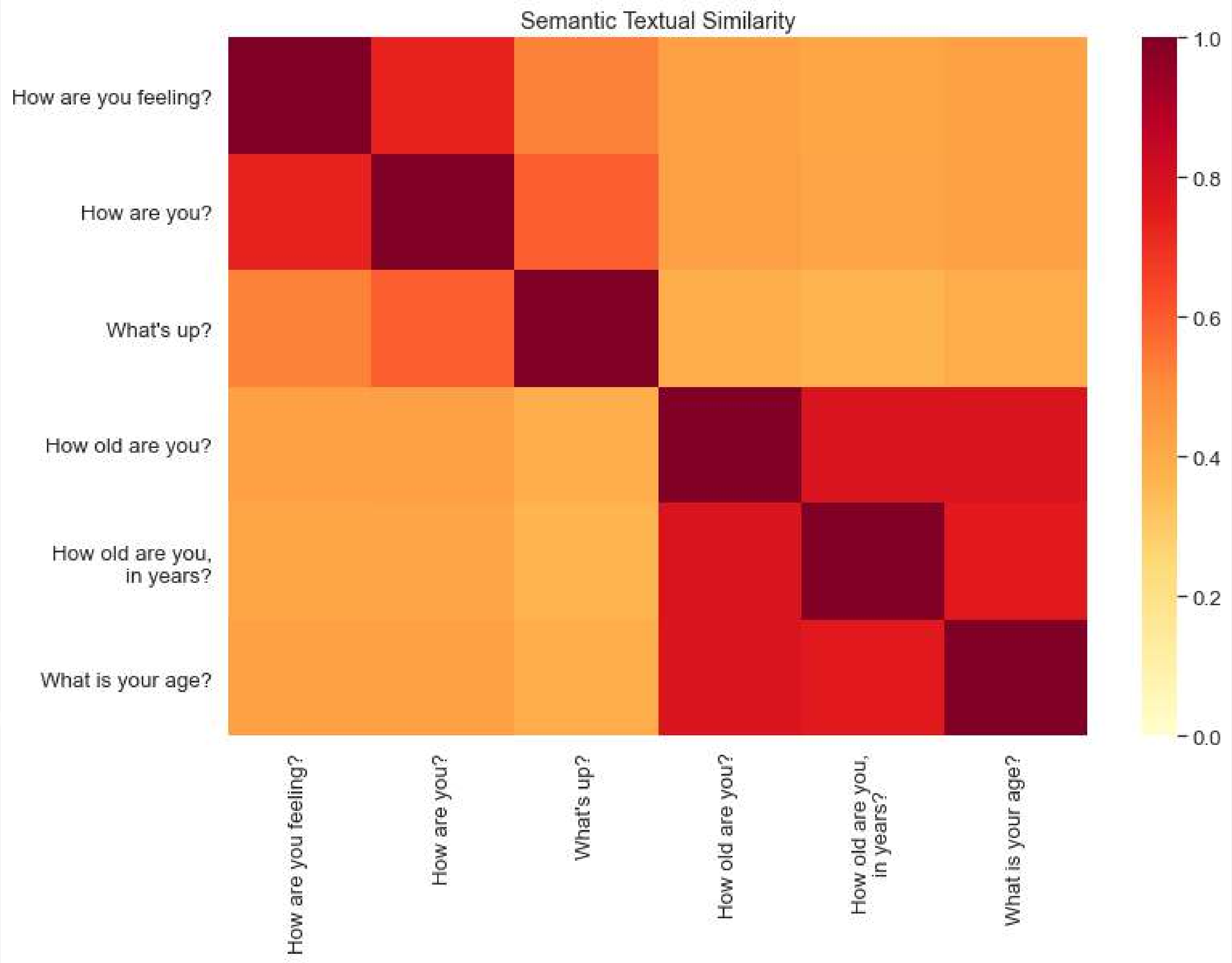
- The model we will be using is the Universal Sentence Encoder (USE) Transformer v5 by Cer et al. (2018)
- Converts text that is between phrase and paragraph length into 512-dimensional vectors

```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

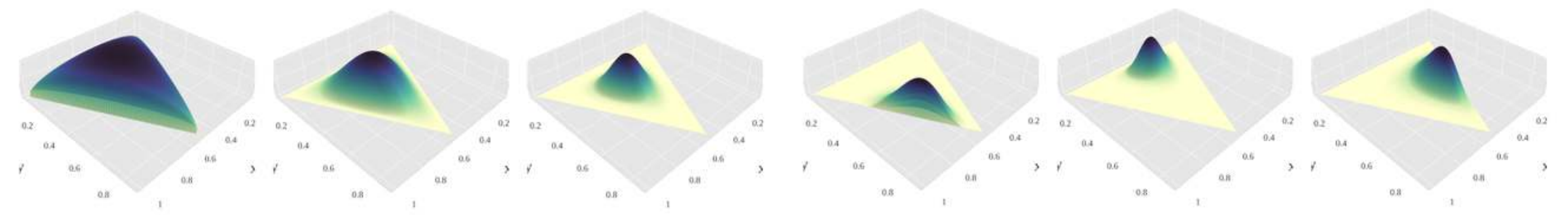
```python
messages = ["How are you feeling?","How are you?","What's up?",
    "How old are you?","How old are you, in years?","What is your age?"]
embeddings = embed(messages)
plot_similarity(messages, embeddings, 90)
```



LDA

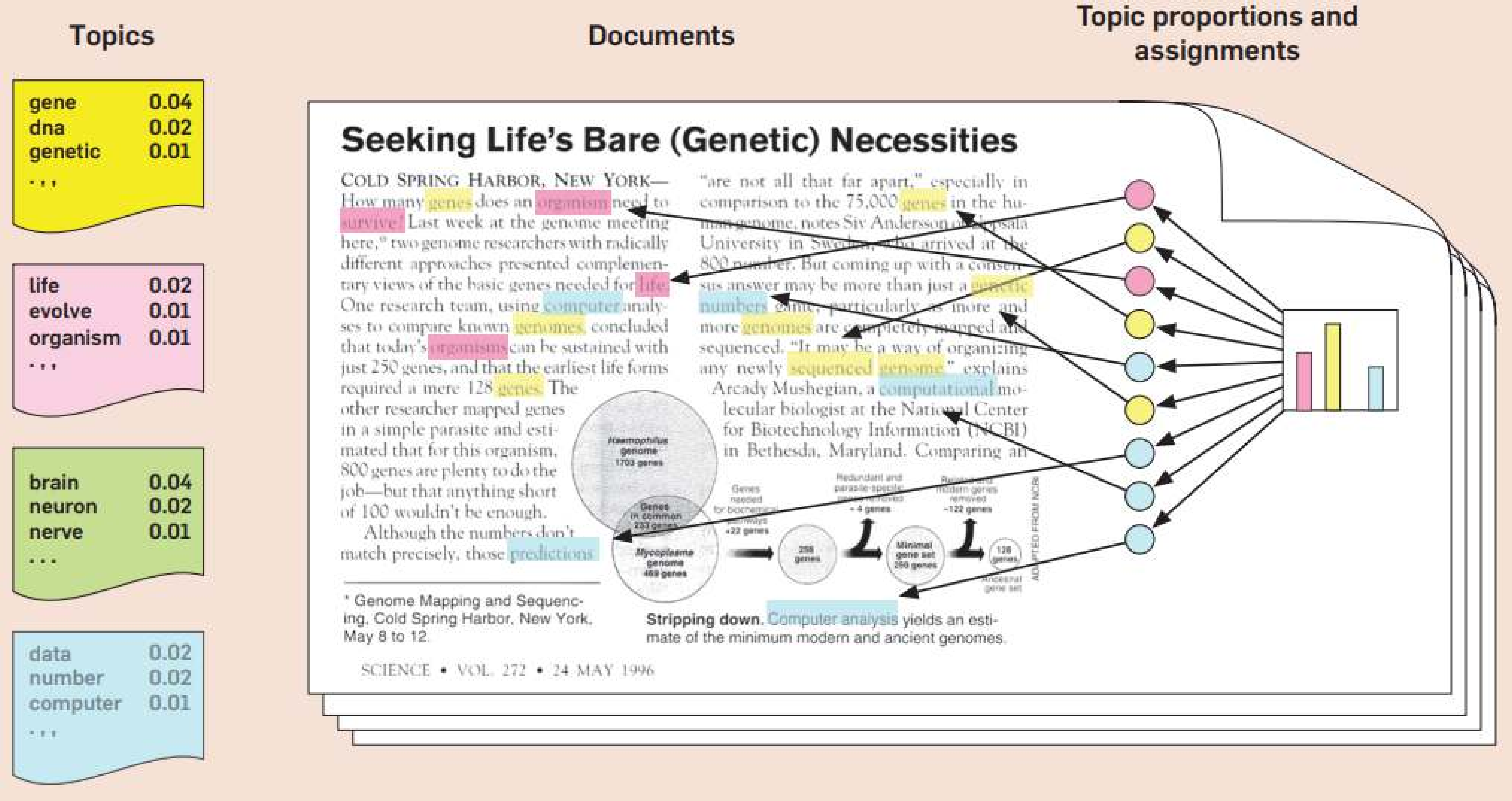
What is LDA?

- **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

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
. . .	

Documents

Topic proportions and assignments

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

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

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

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

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

```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)
```

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

2. Examine a topic

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

```
## [('abbott', 0.012434472), ('party', 0.008847061), ('government', 0.007975474), ('power', 0.007954226)]
##  [('labor', 0.007714725), ('conservative', 0.006868049), ('s&p', 0.006789061), ('political', 0.0066726357)]
##  [('rudd', 0.006448531), ('policy', 0.006251966)]
```

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: abbott party government power labor conservative s&p political rudd policy
## 1: school ms. people white district security service inc. user officer
## 2: benefit city life home trip live people de blasio york
## 3: play williams city game partner set season . good azarenka
## 4: company work price day people end start share take retirement
## 5: % fund bank fee economy government investor mortgage financial crisis
## 6: market % country china u.s. report group car buy investor
## 7: company lhota city catsimatidis retiree work health plan people york
## 8: % health blasio de voter likely city old york support
## 9: president house rule u.s. congress vote company syria obama include
```

Examining the LDA model

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

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

Click here to see the output

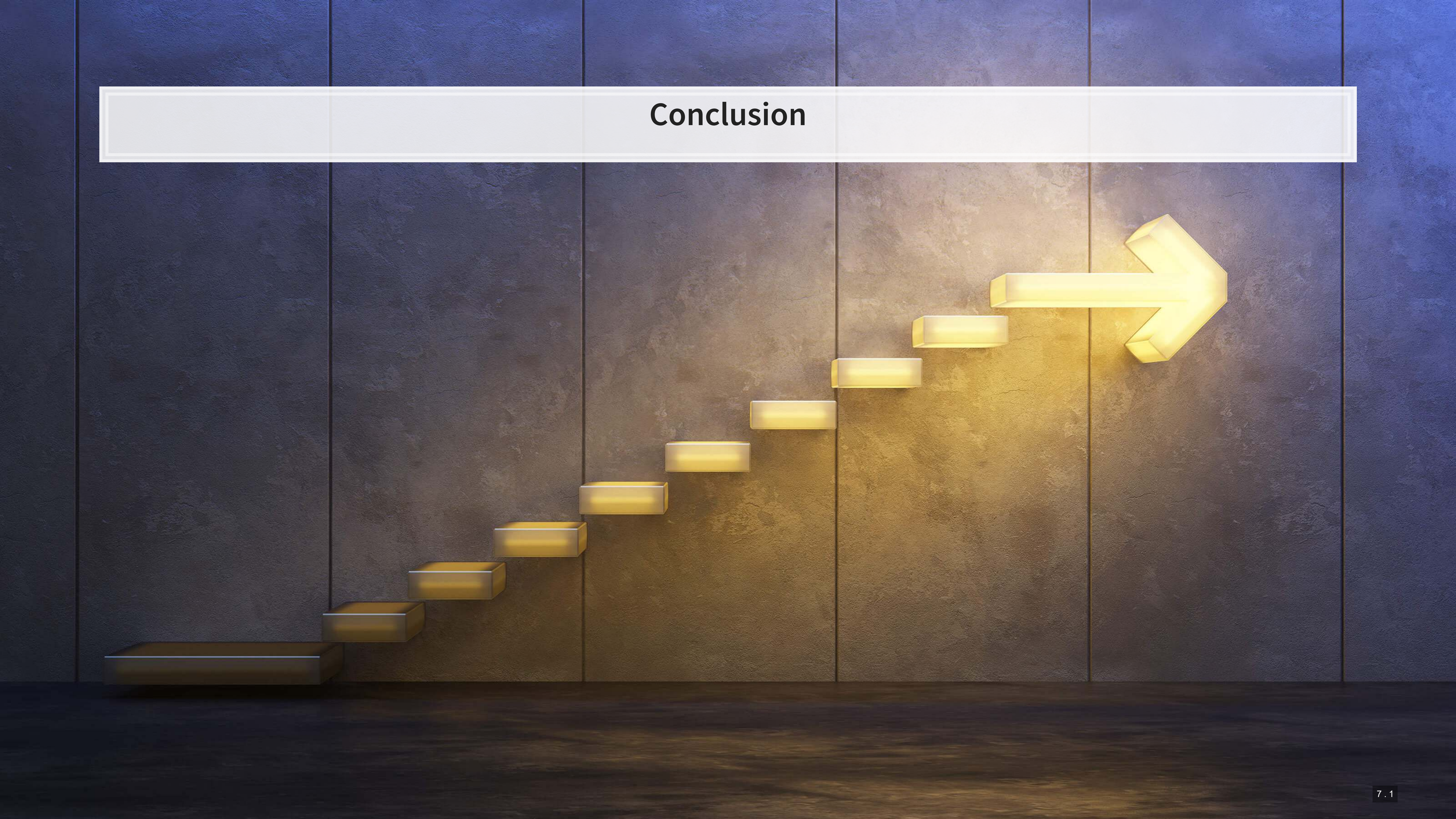
STM

- 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. and M. H. Franco Wong. “Understanding Sentiment through Context.” Working paper, 2022.
- 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.