**Sept 2018 UCLA Word Embeddings Workshop:**

**Classifying Latent Constructs in Language with Word2Vec**

Alina Arseniev-Koehler

This 2-hour workshop will cover how to model language and latent constructsin language with **Word2Vec**, a state-of-the-art method to model text data.

After a brief introduction to Python, we’ll review concepts underlying modeling language with Word2Vec and latent constructs. Then we’ll switch to hands-on time with Python code to implement these concepts and, last, to classify latent constructs. We’ll focus on the gender as a latent construct today, but these methods extend to many other latent constructs, such as morality and socio-economic status.

This workshop is for those with all levels of Python experience.

**AHEAD OF TIME (takes a while):**

* **Install**[Anaconda with Python 3](https://www.anaconda.com/download/)on the laptop you'll use in the workshop. Then, open Anaconda Navigator and click "install" under the icon for Jupyter Notebook.
* **Download** this [model](https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing) (this is a pre-trained Word2Vec model from GoogleNews, and may take a while, [here's](https://code.google.com/archive/p/word2vec/) more info about the model, scroll down to "Pre-trained word-vectors and phrases").
* **Download** the attached file ["examiner-date-tokens.csv."](https://ucla.box.com/s/byb7heow7ej73gonmkifqn11qmpxq2tm)

**AHEAD OF TIME OR DAY-OF:**

* **Download** Part A.ipynb, Part B.ipynb, and helpersPartB.py from my [github](https://github.com/arsena-k/Word2Vec-bias-extraction)
* Put all your 5 downloaded files for this workshop in your downloads folder. For now, don’t use subfolders.

*What is Python, Anaconda, and Jupyter Notebook?*

* Python is a programming language. Jupyter notebook is a prettier way to look at and manage Python code. This is similar to how R studio is an interface to R.
* We can write our own functions in Python (e.g., a function that takes in numbers and spits out average) but for more complex functions that many people use (such as a function to lowercase all words in a text document) it is faster to just install a library that has that function.
* Anaconda is a bundle of things, which includes Python, Jupyter Notebooks, and many (but not all) libraries you’ll need. That way you don’t have to install as many libraries manually.

**DAY-OF SET-UP INSTRUCTIONS:**

* Since your downloads has the files you’ll use today, this folder is called your **“working directory**.” That means that when you are using a Jupyter Notebook or any .py code in here, and you try to open another file, like a CSV with your data, this is the folder that Python will automatically look in to find the CSV unless you specify it is in some other location on your computer.

**Now install libraries:**

* Install a new library via terminal, ***MAC***
  + To get to terminal, go to magnifying glass in upper right corner, type “terminal” and enter, you should see a small window open up. That is our “terminal.”
  + In the terminal window, type “conda”
  + Type “sudo pip install gensim” to install the genism library. OR replace “genism” with the library of your choice you want to download. You might be able to just type “pip install genism.”
* Install new library via terminal, ***PC***
  + To get to terminal, start in lower left corner, search “anaconda prompt” and then you should see a small black window open. That is our anaconda “terminal.”
  + In the terminal window, type “conda”
  + Now type “pip install genism” to install the genism library, *or* replace “genism” with the library of your choice.
* *OR* install new library via Anaconda interface:
  + Open Anaconda. In side bar, click “Environments”. Click on your environment with Python 3 (you may only have one environment if this is your first time using Anaconda). Now, you will see a list packages that are installed/not installed (you can toggle between these options), and you can search for a package in the search bar near the top to download a new one, like “genism.”

**Libraries to install for today** (some may already be installed when you download Anaconda):

* genism
* cython
* sklearn
* scipy
* csv
* statistics
* pandas
* string
* numpy
* random
* collections
* seaborn
* pylab

**How to open a Jupyter Notebook:**

* Go to your terminal
  + - * ***MAC:*** Go to magnifying glass in upper right corner, type “terminal” and enter, you should see a small window open up. In this terminal, type “conda.”
      * ***PC:*** Start in lower left corner, search “Anaconda Prompt” and click; you should see a small black window open up.
  + Type “jupyter notebook”
* *OR,* open Anaconda Navigator, and click “Home” on the sidebar, then click on the orange icon for “Jupyter Notebook.”
* Now, page should pop up in your browser listing files on your computer. Navigate to your working directory (the folder with your Jupyter notebooks for this workshop) and open Part A and Part B Juypter notebooks (they have a .ipynb exntension).

***BEYOND THIS WORKSHOP***

Additional *on-campus* UCLA resources for technical help that may be useful after the workshop:

* Center for Digital Humanities [computing support](https://cdh.ucla.edu/computing-support/)
* If you are affiliated with CCPR: Mike Tzen in [office hours or by appointment](https://ccpr.ucla.edu/services/statistics-and-methods-core-mission/)
* [UCLA Stats Consulting](https://stats.idre.ucla.edu/ucla/policies/)
* And, of course, I’m (Alina, arsena@g.ucla.edu) always excited to answer questions, review this code, brainstorm ideas, or do a follow-up workshop

**CONCEPTUAL INTRO TO WORD2VEC\*:**

**\***This workshop focuses on how to *use* Word2Vec, but here is some brief background on how Word2Vec and related models work.

### **1. What is Word2Vec?**

Word2Vec is a tool to graphically model language from a set of texts, such as news articles, tweets or transcribed interviews. Text is a rich source of data for social scientists but can be challenging to work with given its volume, structure and richness. It is unique from older methods to analyze and model text in that it **1) models relationships between words in the text, and 2) uses a type of machine learning architecture called “artificial neural networks” to do this modeling.** The use of machine-learning means it can handle (and requires) large quantities of data and produces high-quality representations of language.

### **2. What is a model of language?**

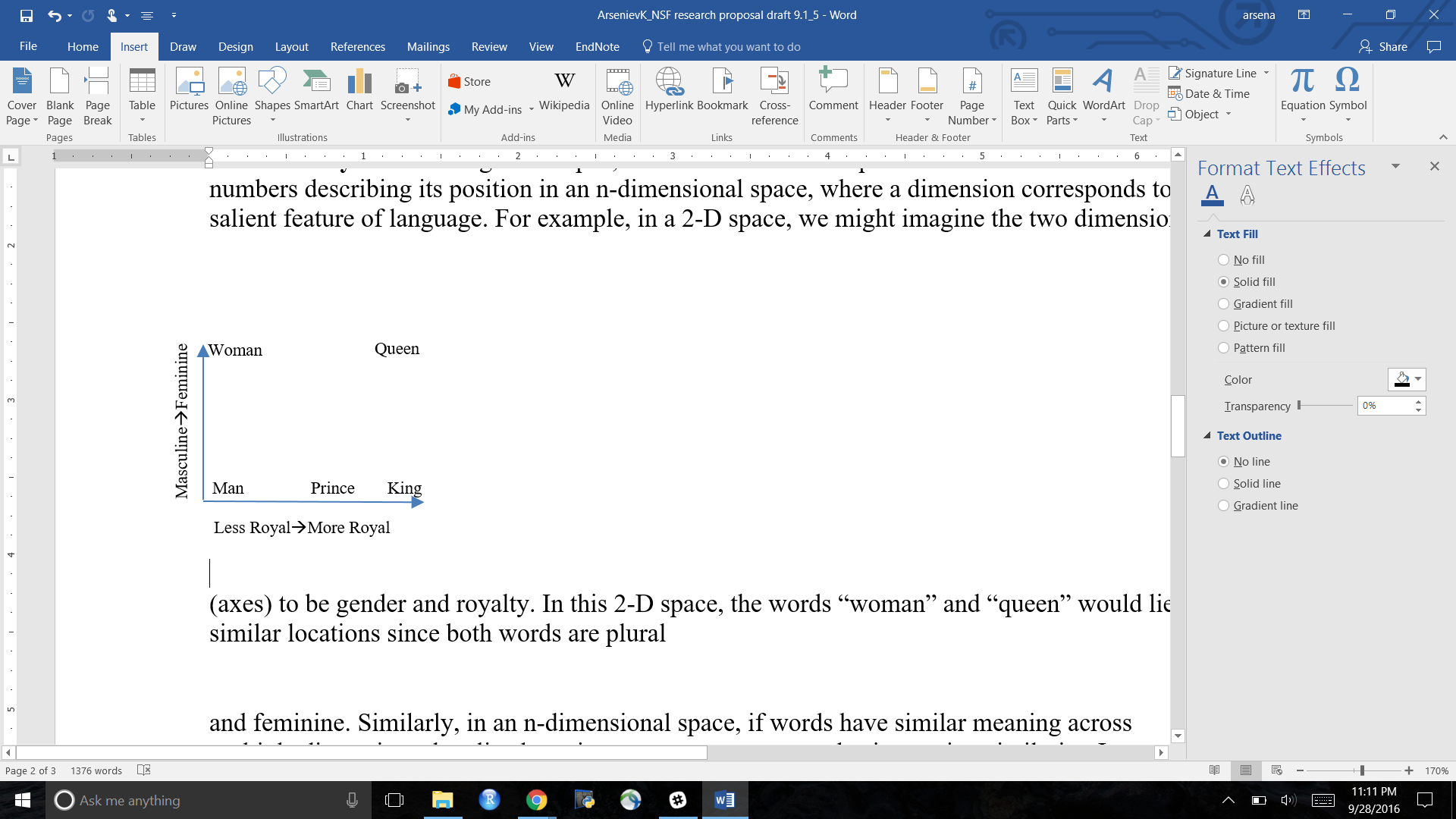
Given a large corpus of text data, Word2Vec learns to represent each word as a **vector (list of numbers)** describing its position in an n-dimensional space. A dimension corresponds to a salient, latent feature of language. In practice, Word2Vec *independently* decides on the dimensions: in other words, we don’t know what the dimensions it learns are. Each word is mapped into around 100-1000 dimensional spaces. To understand the idea of mapping words into space, it may be helpful to consider **a toy example in 2-dimensional space (a plane). Let’s assume there are just two latent learned dimensions (axes) – say, gender and royalty.** Each word in the data is then mapped to a vector describing its position on these two dimensions. As illustrated in the figure below, in this 2-D space, “queen” would have a vector of (1,1) since it is both feminine and royal, while “woman” would be (0,1) since it is feminine but not royal. In other words, the words “queen” and “woman” would share their position on the gender dimension but be differentiated by royalty. Please see Table 1 for an overview of the word-vector corresponding to words in this toy example.

Figure 1. Toy Example of language model with two dimensions to organize words, royalty and gender

|  |  |
| --- | --- |
| Table 1. Word Vectors for 2-D Toy Example of Word2Vec words Organized by Royalty and Gender | |
| “King” | [1,0] |
| “Man” | [0,0] |
| “Prince” | [.5,0] |
| “Queen” | [1,1] |
| Each vector includes the component on the x axis (masculine to feminine) and the component on the y axis (less to more royal). Each axis ranges 0-1. | |

Similarly, in an n-dimensional space, if words have similar meaning across multiple dimensions they lie closer in vector space. In linear algebra, the similarity of two vectors (i.e., spatial closeness) measured using cosine-similarity. Cosine similarity between word-vectors ranges from 0 (not at all the same) to 1 (exactly the same).

One of the neat aspects of Word2Vec is that it encodes **linguistic regularities**, such as analogies. For example, in word-vectors trained on Google News, we can perform algebra with word-vectors, such that the vector left by – + (i.e. swapping out the gender components of king), is nearly the same as the word-vector . In other words, if we swap out the gender components for “king,” we are left with the vector for “woman,” just as we intuitively would expect. We’ll see if our model correctly solves this analogy in the Jupyter Notebook for Part A, and explore the underlying ideas and possible applications more in the Jupyter notebook for Part B.

### **3. How does Word2Vec come up with word-vectors?**

Here’s a conceptual explanation:

First, imagine you are asked to guess the missing word, from a New York Times article about health and obesity:

*“Americans have grown \_\_\_\_\_ over the last generation, inviting more heart disease, diabetes and premature deaths...”*

As you might have guessed, the answer is “fatter.” Given a set of context words, it is often easy for us (and a machine!) to guess the missing word. And, the better you know language and the meaning of words, the easier this task becomes.

Word2Vec[[1]](#footnote-1) starts with assigning random word-vectors for each word in the vocabulary. Then we give the model a snippet of our text data with N context words (usually N is 2, 5, or 10 words) around a missing word, we ask it to guess the missing word.

More specifically, we ask Word2Vec, what is the word-vector with the *highest cosine similarity* to the all the other the context word-vectors?

If the model answers correctly, it will guess the word *really was* in that snippet of data. And that suggests to us that our model has learned the meaning of words well!

But if the model guesses the wrong answer -- which it probably does with random word-vectors -- we tweak the word-vectors involved in this snippet of data so that the model would have answered the task correctly.

This is a hard task, but with a large dataset, we can provide *lots* and *lots* of snippets of data to keep tweaking the word-vectors, until Word2Vec has *learned* word-vectors that are good enough to solve this task well.

And then, we can use these word-vectors as high-quality, quantitative representations of words.

🡪For more technical and detailed explanations, see [Rong 2014](https://arxiv.org/abs/1411.2738) and the [Gensim documentation](https://radimrehurek.com/gensim/models/word2vec.html), or the [original publications of Word2Vec](https://code.google.com/p/word2vec).

1. This is one possible learning task, called Context Bag of Words (CBOW). Another common learning task is Skip Gram, where Word2Vec is asked to guess the context words from a given target word. [↑](#footnote-ref-1)