**June 2018 UCLA Computational Sociology Working Group Workshop:**

**Intro to Word2Vec Language Modeling for Social Science Applications**

Alina Arseniev-Koehler

This 2h workshop will cover how to do **text mining with Word2Vec**, a state-of-the-art method to model text data. We'll look at how to train, explore and visualize models of GoogleNews, and samples of social science applications of Word2Vec.

This workshop is for those with all levels of Python experience. We'll use sections from this [code](https://github.com/arsena-k/Word2Vec-bias-extraction/blob/master/Part_A_W2V_training_performance_exploring.ipynb) and this [code](https://github.com/arsena-k/Word2Vec-bias-extraction/blob/master/Part_B_Bolukbasi_W2V_Dimension_Extraction.ipynb), which we aim to make as streamlined as possible.

Sample empirical questions we'll explore:

1. How are words gendered in Google News? For example, is "politician" masculine or feminine in Google News? What about "genius"? What about "slender"?
2. What words are the *most* feminine or masculine words in GoogleNews?
3. Which words connote low-class, and which words connote high class?
4. Which illnesses, and occupations, are the most "low-class"? Which are the most "high-class"?
5. What is portrayed as "healthy" in GoogleNews? How can we visualize this?
6. How do we check the robustness of our findings?

**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.
* If you have a chance, try opening a Jupyter Notebook and playing with Python ahead of time. Jupyter Notebooks are a visually appealing way to play with Python code. For example, try out Part 1 of the UCLA CompSoc Python workshop Jupyter Notebook [here](https://github.com/UCLACompSoc/Text-Mining-in-Python-Tutorial/blob/master/Text_Mining_Presentation_2-21-18.ipynb).
* **Download** the attached file ["examiner-date-tokens.csv."](https://ucla.box.com/s/byb7heow7ej73gonmkifqn11qmpxq2tm)
* **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** this [github repository](https://github.com/arsena-k/Word2Vec-bias-extraction) (click the green button "clone or download")

**OUTLINE:**

1. 5-15 min intro and motivation
2. ~45 min working through Part\_A code
   1. Train your own model
   2. Use and explore your model and/or a pre-trained model
   3. Visualize results
3. Break
4. ~45 min working through sections of Part\_B code
   1. Motivation/explanation for extracting “biases” or “dimensions” and a simple example with a pre-trained model
   2. Visualize Results
   3. Doing this kind of text mining more robustly (as time permits)

**INSTRUCTIONS TO GET STARTED TODAY:**

After downloading the [github repository](https://github.com/arsena-k/Word2Vec-bias-extraction) folder, and add any other files related to this workshop (like the CSV, and the pre-trained Word2Vec model which you should have downloaded above) into this folder. For now, don’t use subfolders.

* Since this folder has the Jupyter notebooks you’ll use today, this folder is called your **“working directory**.” That means that when you are using a Jupyter Notebook 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[[1]](#footnote-1):**

* Install 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 black window open up. That is our “terminal.”
  + In the terminal window, type “conda”
    - \*if you have more than one environment (e.g. py 2 and py 3, type “source activate [name of your environment with py 3])
  + Type “sudo pip install genism” to install the genism library, *or* replace “genism” with the library of your choice. You might be able to just type “pip install genism.”
  + To open Jupyter notebook from this window, type “jupyter notebook”
* Install new library via terminal, PC
  + To get to terminal, start in lower left corner, search “cmd” and then you should see a small black window open up. That is our “terminal.”
  + In the terminal window, type “conda”
    - \*if you have more than one environment (e.g. py 2 and py 3, type “activate [name of your environment with py 3])
  + Now type “pip install genism” to install the genism library, *or* replace “genism” with the library of your choice.
  + To open Jupyter notebook from this window, type “jupyter notebook”
* *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 make sure you’ve installed 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 black window open up.
      * PC: Start in lower left corner, search “cmd” and you should see a small black window open up.
  + Type “anaconda”
  + Type “jupyter notebook”
* *OR,* open Anaconda, and click “Home” on the sidebar, then click on the orange icon for “Jupyter Notebook.”
* Now, 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 extension).

***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?**

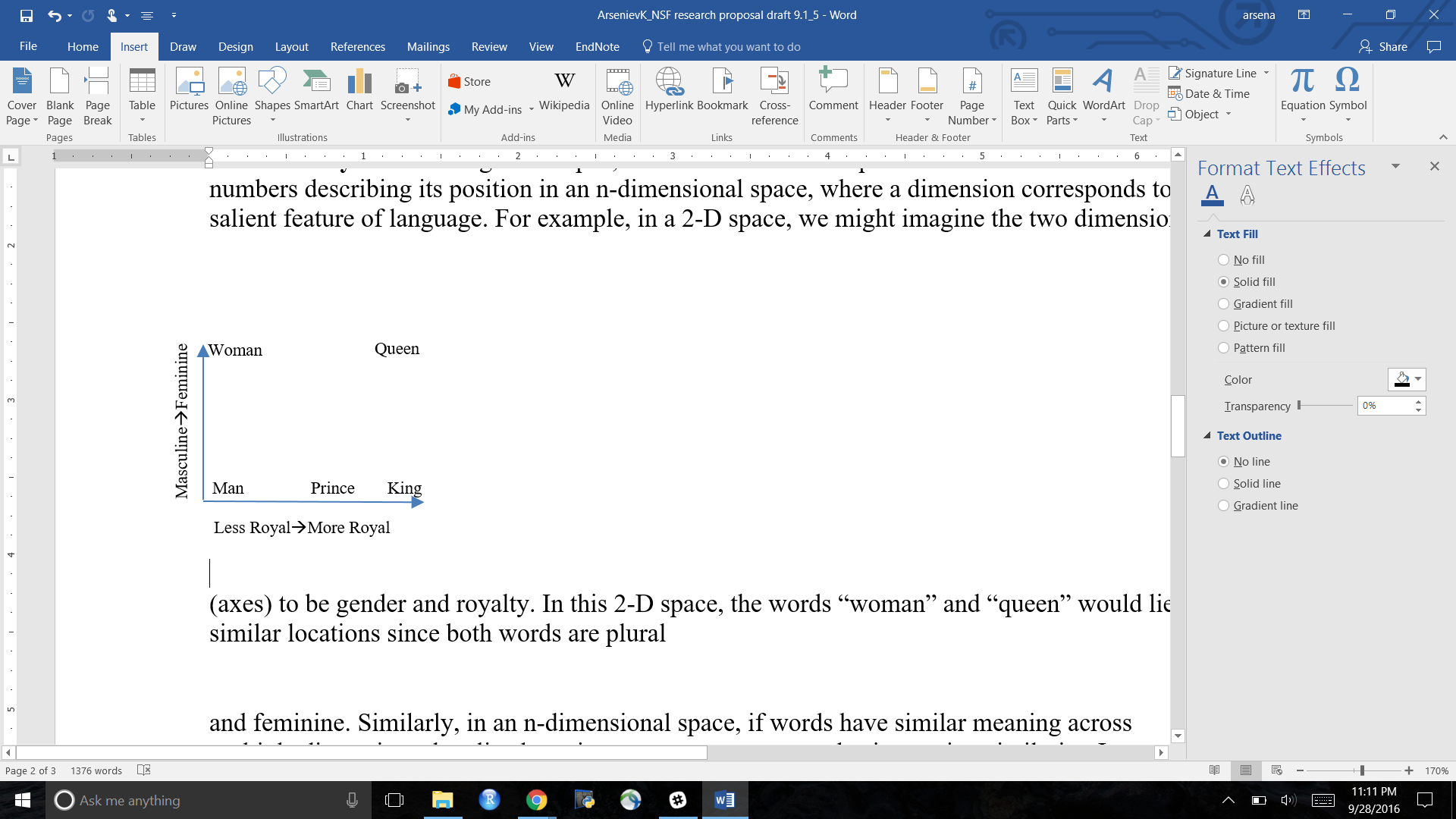
Given a large corpus of text data, Word2Vec learns to represent each word as a **vector (list of numbers)** describing its position in an n-dimensional space. A dimension corresponds to a salient, latent feature of language. In practice, Word2Vec *independently* decides on the dimensions: in other words, 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[[2]](#footnote-2) 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. A Python library is a set of functions, which we use so you don’t have to write out a formula for all the standard things we want to do, like finding an average or median. For example, the Python library “statistics” already has functions to take averages, find medians, and standard deviations. You only need to install a library once, but then each time you want to *use* the library or a function from the library, you need to tell Python in your script, as we’ll do in the Jupyter Notebooks with “import statistics” to tell it we are now using the statistics library. [↑](#footnote-ref-1)
2. 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-2)