Job Similarity Recommendation Engine

Introduction

With the age of the internet, the way we interact with the world fundamentally changed. Knowledge that used to be held within libraries, textbooks, and with well trained experts is spilling out into the digital commons we call the internet. Anyone with the motivation to look and an internet connection can find just about anything they are looking for. These changes are not just restricted to knowledge and consumer goods, but have expanded into services as well.

One of the more interesting services that popped up on the internet were job boards. One of the first job boards was commissioned by Jeff Taylor in April of 1994. The site stored job descriptions from the newspaper segment of human resource (https://en.wikipedia.org/wiki/Monster.com). The site grew, morphed, and changed into Monster.com. About the same time, NetStart Inc. began allowing companies to list jobs openings on their website. In 1998, NetStart changed it's name to CareerBuilder and had their Initial Public Offering (IPO) on May 12, 1999. In 2004, Indeed.com launched their job search engine which remains one of the top used job boards. As these job boards have become the norm to find new jobs, the the number of opportunities have grown exponentially. No longer people looking for jobs limited to their geographical area, local newspapers, or job search firms. Similarly, employers seeking potential employees can now reach out to the world with their postings. As the postings have grown the data has grown and so has the need for simple methods to filter through listings. In June of 2017, Google threw its hat into the ring of job search (https://techcrunch.com/2017/06/20/google-launches-its-ai-powered-jobs-search-engine/). Google has a unique interaction with data, search, and filtering developing many of the tools and platforms we use daily. Google has a lot of interesting tools at hand for mining text which has not yet been applied to the world of work. In fact, there is a wealth of theory and data already surrounding the world of work. In 1958, John Holland created a theory that broke the world of work into different categories which has been used by the U.S. government and guidance counselors since. Similarly, the U.S. Government has been categorizing and tracking jobs since 1938. In 1998, it created a publically available database outlining the characteristics of various jobs including John Holland's RIASEC code. Given the spread of online job boards, machine learning, and textual analysis, can we predict jobs which should be similar to one another just based on text? How do those recommendations compare to the theory of current job coding systems? The purpose of this study was to use publically available data to build a job similarity recommendation engine using text. Once the recommendations were made, we could examine the associated theory-based recommendations and how they related to the outcomes. If the system were to work, one could begin classifying jobs using a machine learning model rather than expert judgement.

Occupational Information Network (O*NET)

Beginning in roughly 1938 through roughly 1998, the primary source of vocational interests and job matching for the US government came from the printed book called the Dictionary of Occupational Titles (DOT). The DOT emphasized blue collar jobs in the industrial world and was updated periodically. Over time, the U.S. economy shifted towards fewer industrial jobs and more informational jobs and thus the applicability of the DOT dwindled. During the mid 1990s, through funding from the U.S. Department of Labor's Employment and Training Administration, a joint collaboration created 0*NET to help modernize the government's ability to track jobs.

Although O*NET was similar to DOT, it had a few key differences differences. First, it utilized a database and online platform rather than a printed book. Second, by using an online and database platform, it created a flexible system that was easy to update and distribute. Finally, O*NET created a formalized content model used to quantify and describe jobs and the world shifted. See Figure 1 for a visual representation of the various components measured and tracked by O*NET.

Figure 1 O*NET Content Model

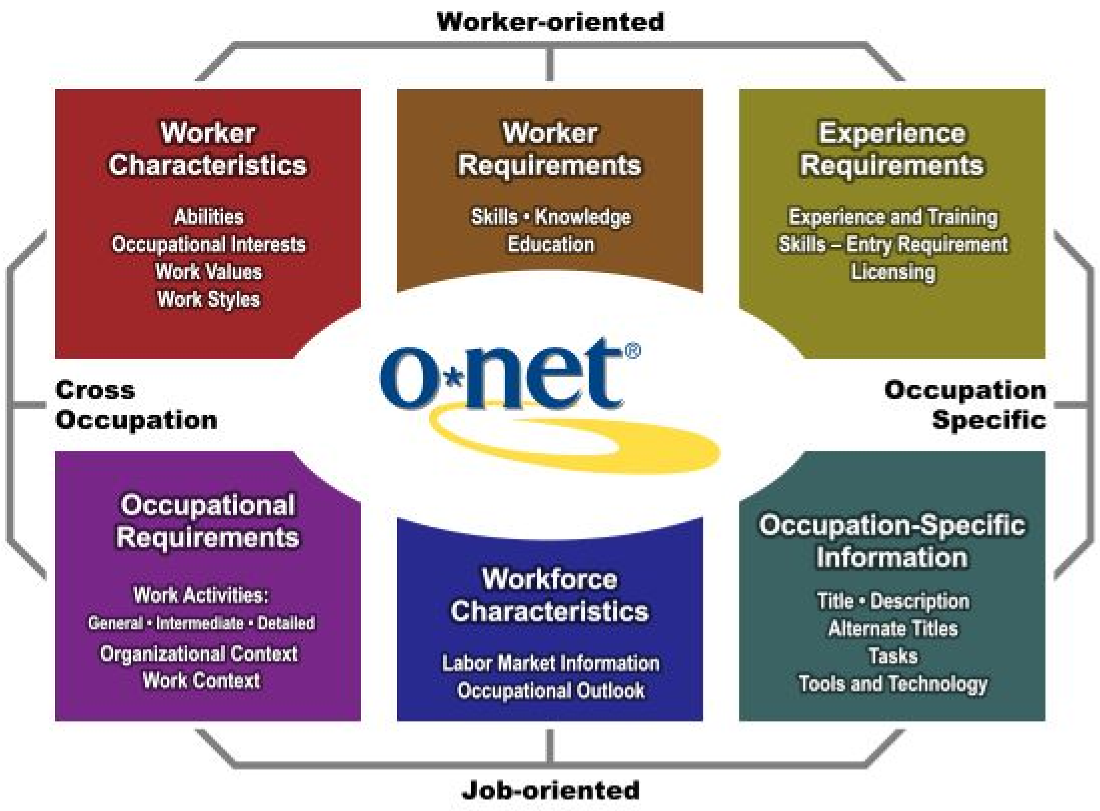
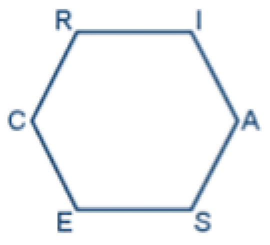


Image available at: https://www.onetcenter.org/content.html

The design of O*NET has created unique research and data science opportunities because of the standardization of data collection, storage of the data in a relational database, and it's open access nature.

Holland Codes

Figure 2. Holland Code Diagram



Note. Image linked from Wikipedia

One of the things that you may notice as you look through the various pieces of code and my analyses is the inclusion of what are called Holland or RIASEC codes. The inclusion of these codes was done as a sanity check and to set up the dataset for future analyses. Briefly, Holland codes come out of research by Dr. John Holland. His theory states jobs can be broken into six major categories: Realistic (R), Investigative (I), Artistic (A), Social (S), Enterprising (E), and Conventional (C). The face validity of these scales is high, meaning they measure what you would intuitively think. In-depth descriptions of these categories can be found here. Holland's theory states that the order and display of these jobs matters (see Figure 2). Codes which are closer to one another are more similar while those are further are apart are more different. For example, jobs that are Artistic are more closely related to jobs that are Social than jobs that are Conventional (often described as Organizers). Due to the complexity of the jobs, one to three codes are used to describe them. For example, Biomass Plant Technicians are categorized as RI (Realistic and Investigative). As part of the O*NET job analysis process Holland codes are generated for each job. These Holland codes are used by the US government and educational professionals to help try and identify students with similar interests. As a sanity check, I used these codes to make sure that the similarity of recommendations were similar to what would be expected. Since Holland codes are based on expert judgement they may not align with all recommendations. More detailed information on the creation of the Holland code for each job within the dataset is outlined in within the Data Wrangling Jupyter Notebook.

O*NET Database

The O*NET database is available for download and use on the O*NET Resource Center (https://www.onetcenter.org/database.html). On this site, users can download the database in several formats including Microsoft Excel, Tab-delimited files, SQL files for use with MySQL, PostgresSQL or compatible databases, SQL files for Microsoft SQL Server, and SQL files for Oracle. The O*NET Resource Center also provides a very comprehensive data dictionary outlining the database structure, variable definitions, and possible values. Although often overlooked, good data documentation outlines which data to use, what to be wary of, and how to join things together. Each of these components is key to good data science.

Although it's outside of the scope of this project to explain the inner workings of O*NET construction and maintenance, two key points are worth reviewing. First, potentially blindingly obvious, O*NET uses a relational database which requires a degree of standardization and structure. This standardization makes for tidy data to work with as opposed to, say, using a web scraper to create a set of documents for analysis. Perhaps not as obvious, O*NET uses standardized data collection strategies and is used as a benchmark of the government. Because O*NET is used as a benchmark, the data contained within are clean and reliable. Put simply, the data within the database is more clean, tidy, and reliable to use than trying to create my own.

Second, O*NET labels using the the 2010 Standard Occupational Classification (SOC) system. In total, O*NET organizes jobs over 1,100 occupational categories and is labeled with a hierarchical code with the format XX-XXXX.XX. Each code includes a major group, minor group, broad group, and detailed occupation. The outermost two digits represent 23 major groups (https://www.bls.gov/soc/major_groups.htm) such as Management, Legal, or Sales. The remaining digits "drill down" in specificity of the jobs, with jobs closer in numerical value theoretically being more similar. For more information on this hierarchical structure, please see the Bureau of Labor Statistics (https://www.bls.gov/soc/). For this project, I identify and search for jobs using the O*NET SOC code rather than text-based job titles.

Using the O*NET database for this project required creating a local copy I could directly query. I created an isolated development environment using Oracle's Virtualbox Manager (version 5.1.30), spinning up an Ubuntu 17.04 instance. Within the instance, I installed a MySQL server (version 5.7.20) and MySQL Workbench (version 6.3). Once configured, I downloaded the full SQL O*NET database version 22.1. I extracted the database which included roughly 36 SQL files, one for each table. Populating the database requires connecting to the instance and running each of the SQL files. Running each SQL file would then create and populate the table with the O*NET data. To speed up the process of running 36 SQL files, I concatenated the 36 SQL files into a single SQL file using the linux command line (e.g. "`cat *.sql > all_files.sql")`. I then ran mysql code at the command line inserting all the O*NET data into a MySQL server. Once completed, I had a fully functioning version of the O*NET database on my local machine that I could query and explore using MySQL Workbench and python.

Dataset Creation

Note. All steps for data set creation are available in the Data Wrangling jupyter notebook.

My desired end state for this portion of the project was to create a dataset that included the O*NET SOC code, job title, Holland code, and a single string of all the appropriate text for the job. This required creating 11 different queries using various basic SQL commands. I ran each of these queries within python to create Pandas dataframes, which I then massaged and merged together. A commented Jupyter Notebook which outlines all of these steps is available within the github repository (Data Wrangling.ipynb). I outline the general steps to create the single string in the steps below.

<u>Step 1</u>: I reviewed the O*NET Content Model (see Figure 1) to determine the different components gathered by O*NET. Based on this review I expected to find six major sections to include in my analyses: Worker Characteristics, Worker Requirements, Experience Requirements, Occupational Requirements, Workforce Characteristics, and Occupation-Specific Information.

<u>Step 2</u>: I reviewed a job on O*NET to verify the six sections from the content model were displayed. A review of the site showed that the actual display was much more granular, including 19 rather than six different sections (see table 1 for a list of all listed sections). Within each section, there were also a limited number of options displayed. For example, 5 of 24 Tasks may be displayed, leaving the user to wonder why they were not all displayed. Were they deemed unimportant or was this simply an effort to save space and make the size consumable?

Table 1. Summary sections on O*NET

Tasks*	Technology Skills*	Tools used*	Knowledge*
Skills*	Abilities*	Work Activities*	Detailed Work Activities
Work Context*	Job Zone*	Education	Credentials
Interests	Work Styles*	Work values*	Related Occupations
Wages & Employment	Job Openings	Additional information	

Note. * Sections used for analyses

<u>Step 3</u>: The next step was to choose which of the possible variables I should use. Of all the steps within this process, the selection of variables and which cases to keep is perhaps most fuzzy. Variables were chosen after a review of the data and its potential for impact. For example, some sections, such as *Wages* and *Employment,* were numerical and not appropriate for textual analysis. Others included text which didn't seem like it would be helpful. Admittedly, each decision was a judgment call which could be reviewed in a more empirical fashion in future studies. In the end, I chose 11 of the total 19 possibilities and these variables are indicated with an asterisk in Table 1.

<u>Step 4</u>: For each variable chosen, I reviewed the data dictionary to determine which values were most appropriate to use. For example, Knowledge, Skills, and Abilities are rated on a one to five scale from *Not Important* to *Extremely Important*. After reviewing the variables, it made most sense to keep variables that were marked as at least *Important*. Again, the choices here were

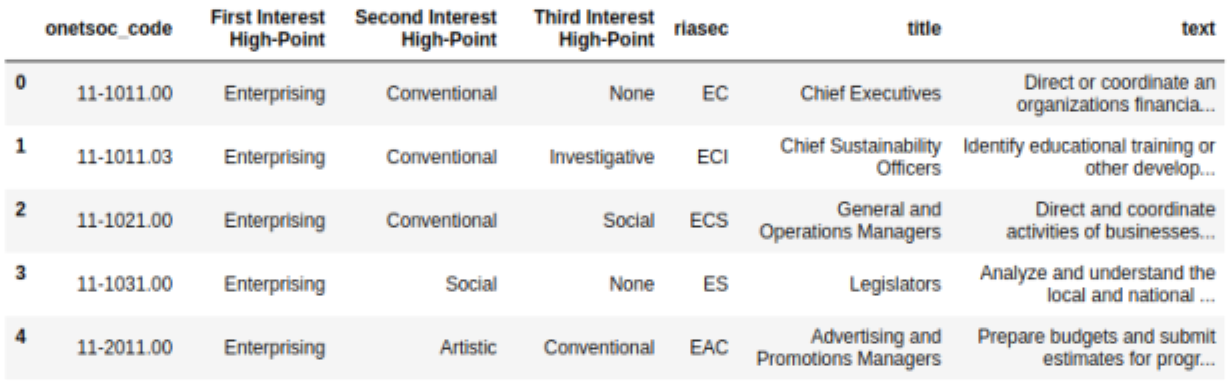
judgment-based and may be suited for fine tuning in other instances. Table 2 presents case exclusionary criteria. All queries and exclusionary criteria are available in the github repository (../functions/queries.py).

Table 2: Query exclusionary criteria

Table	Case exclusion criteria
RIASEC	Scale labeled as interest high point
Task	None
Tools & Technology	None
Knowledge, Skills, Abilities, Work Activities, Work Styles	Includes cases marked as “Important” , “Very Important” or “Extremely Important”. Drops cases marked as “Somewhat Important” or “Not important”
Work Context	None
Job zone	None
WorkValues	None

Step 5. Once the data was queried, a loop was written concatenating all the text together into a single long string and generating a data frame. The frame was then merged together with a data frame containing the onetsoc_code, Holland code, and job title. A screenshot of the final dataset is shown below.

Figure 3. Final Dataset



	onetsoc_code	First Interest High-Point	Second Interest High-Point	Third Interest High-Point	riasec	title	text
0	11-1011.00	Enterprising	Conventional	None	EC	Chief Executives	Direct or coordinate an organizations financia...
1	11-1011.03	Enterprising	Conventional	Investigative	ECI	Chief Sustainability Officers	Identify educational training or other develop...
2	11-1021.00	Enterprising	Conventional	Social	ECS	General and Operations Managers	Direct and coordinate activities of businesses...
3	11-1031.00	Enterprising	Social	None	ES	Legislators	Analyze and understand the local and national ...
4	11-2011.00	Enterprising	Artistic	Conventional	EAC	Advertising and Promotions Managers	Prepare budgets and submit estimates for progr...

Analysis

Note. All steps for data set creation are available in the word2vec jupyter notebook.

The purpose of this project was to use text to help identify jobs which are similar, thereby creating a simple recommendation system. Text is inherently difficult to work with and constitutes an entire field of study. When you dive into text analysis, you'll quickly find different models to work with. You can use things like TF-IDF (Term Frequency "TF"; Inverse Document Frequency "IDF") to determine how frequently? words occur in a document and across documents. Latent Dirichlet Allocation (LDA) is a method of topic model helping to identify common groupings of words representing general constructs. Naive Bayes classifiers are models that can classify text into various categories and are used for things like email spam and sentiment detection. Each of these modeling methods may have been used in different fashions to help address the question at hand. However, for this project, I choose to use a modeling technique called word2vec.

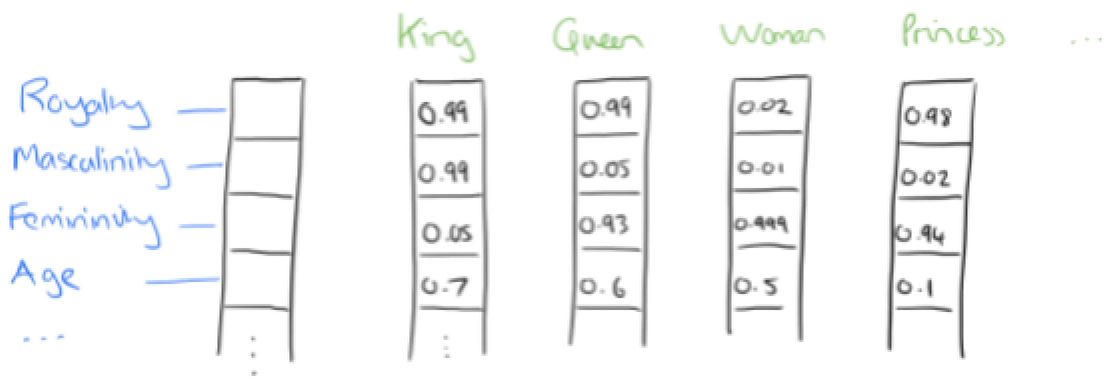
Word2Vec comes out of research conducted at Google and is in my opinion rather complicated and definitely difficult to explain. Although I provide rough overview based on my understanding, I highly recommend the following resources if you are interested in word2vec.

- Google Word2Vec
- Efficient Estimation of Word Representations in Vector Space
- Distributed Representations of Words and Phrases and their Compositionality
- Word2Vec Tutorial - The Skip-Gram Model
- An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec
- The amazing power of word vectors

Word2vec begins by training a model on a corpus of data. During training, word2vec is creating a large vocabulary. It takes your corpus of text and tokenizes it into individual words. Using the corpus and a "window," the model then determines the probability of words being seen near to one another. Behind the scenes, the model uses a dimension parameter, which corresponds to the number of neurons in a shallow neural net. The model then creates N dimensions. These dimensions are then used to determine similarity.

This form of analysis is inherently ambiguous. In fact, neural networks are often described as a "black box." I have had conversations with other data scientists who avoid using them because of the difficulty explaining the model. Adrian Colyer's blog post The amazing power of word vectors provides one of the best visual representations I've seen. As seen below, Adrian outlines four words, King, Queen, Woman, and Princess. Each word is represented by a vector of their dimension scores. In blue, he has created a representation of different dimensions such as Royalty, Masculinity, Femininity, and Age.   The the vector 'King' includes high values for royalty and masculinity while femininity is low. The meaning of the word is then determined by the combination of the dimensions. For example, Kings are royal, masculine, and have some component of age. A princess is Royal and Feminine. Unfortunately, the example below is a bit over simplified. In practice, we may not fully understand what constitutes the dimensions used in the model, making models predictive but again a bit of a "black box."

Figure 4. Vector examples



Note image linked from
https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Models

NOTE: All code for these models is available within the word2vec juptyer notebook.

For this study, I used two different sources of word and phrase vectors. First, I used pre-trained vectors published by Google (GoogleNews-vectors-negative300). This model was trained on a News dataset of about 100 billion words. The available trained model has 300 dimensions with 3 million words. Using this model is relatively simple, only requiring you to import the vectors into python.

The second model I used was trained from scratch using the O*NET database. To train the model, I followed a few steps. First, I used gensims simple preprocessing utility to put all text into lowercase and tokenize it into words. Then I used NLTK's English stopwords to remove text such as 'the', 'is' or 'are'. The purpose here was to try and clean up the text a little to speed up computations and eliminate some potential noise. It is important to note that there is debate as to whether the removal of stopwords is necessary. Once the text was processed, I used gensim 3.0's Word2Vec model to train the model. I used a size of 300 to mirror the size of Google's vector, a window width of 5, indicating the number of words on either side of the target word to analyze, and iterations set to 15. I saved the final model.

Average Word Embeddings

Now that I had two different word vector models trained, the next step was scaling the use of the model to large chunks of text. If you remember our previous example of 'King,' you realize that this is a single word. What do you do when you have more than one word? The answer is calculating the average word embeddings. In our case, that means looking at the text for each O*NET job. Calculating the average word embedding is relatively straightforward. For each job, I first took the preprocessed text. I then looped through each word and ran it through the trained model, which returned a 300 dimensional array. I saved this array in a list. Once I had an array for all the words in the text, I calculated the average value for each domain. For example, I took the first element in array one, the first element in array 2, etc. and summed them together. I then divided by the total number of arrays (words) to get the average. I then moved on to the second item in the arrays and calculated the average. I did this until I had gone through all 300 dimensions. I then returned a single array which had all 300 averages. I saved this array back into my dataset.

I calculated the average word embeddings for the Google vectors, resulting in one data frame which I saved. I then calculated the average word embeddings using my O*NET-trained word2vec model and saved the resulting dataset.

Recap

Let's take a moment to take stock of what I have described one thus far. First, I pulled the data together from O*NET, resulting in a single, long string of all the text associated with a job. Next, I created two models, which calculated word vectors. One model came from a pretrained Google model that included 300 dimensions. The second model was trained from scratch using O*NET's text as the corpus. Now that I had the models, I could start calculating the vectors for each job in the O-NET database. The problem was that there was more than one word associated with each job. To account for this, I calculated the average word embedding for all the words in the job text. I did this for each of the jobs within the database once using the Google model and a second time using the model created from scratch. The result of all of this was a list of weights for each of the dimensions in the model. The figure below presents a snapshot of what this looks like. As you can see, each row represents a single job title, while the following variables represent average word embedding values for each dimension. Note that while this screenshot only shows 9, there are in fact 300 dimensions. The next challenge was how to compare the similarity of a job such as 'Music Director' to a job such as

‘Choreographer.’ That’s where cosine similarity comes in.

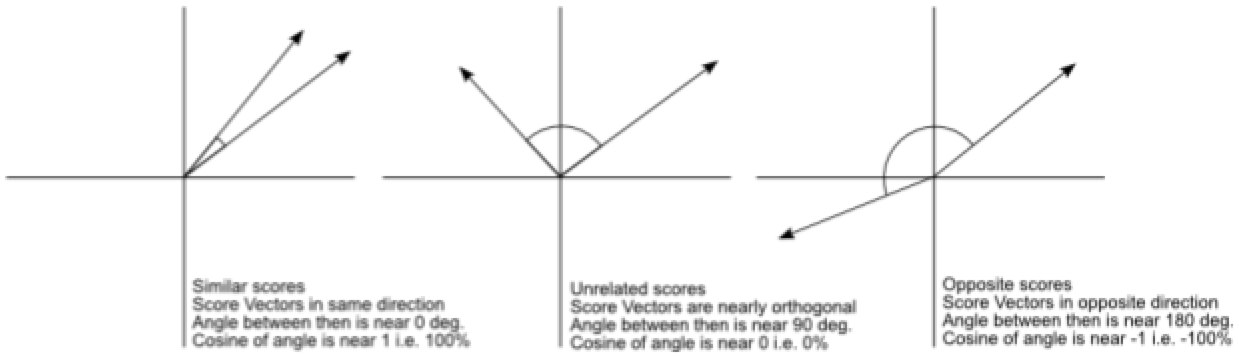
Figure 5. Example word vectors for jobs

	title	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
379	Music Directors	0.026201	0.024999	-0.017454	-0.012312	-0.047884	-0.003302	0.020082	0.011222	-0.035705	0.032271
378	Choreographers	0.027697	0.012169	-0.002602	-0.010429	-0.045809	-0.000181	0.009114	0.016606	-0.034702	0.033165
384	Public Address System and Other Announcers	0.012593	0.017254	-0.017846	-0.014706	-0.033975	0.007945	0.010145	0.011816	-0.027836	0.025319
368	Actors	0.041632	0.021967	0.001624	-0.013410	-0.054547	0.024520	-0.013511	0.023597	-0.052994	0.040552
372	Talent Directors	0.008218	0.023569	-0.009766	-0.013399	-0.055430	0.006866	0.004151	0.021393	-0.053137	0.012393

Cosine Similarity

First, remember that word2vec deals with vectors of numbers. What we need to do is determine how similar those two vectors are. This is where the Cosine Similarity comes in. The cosine similarity calculates the cosine of the angle between the two vectors. Two vectors that are exactly the same have a similarity of 1, while those that are 90 degrees apart have a similarity of zero. Thus, when doing comparisons, values closer to one are considered better than those closer to zero. Figure 6 is a graphical representation of two vectors plotted against one another and the cosine similarity between the two vectors.

Figure 6. Cosine Similarity


Similar scores
Score Vectors in same direction
Angle between then is near 0 deg.
Cosine of angle is near 1 i.e. 100%
Unrelated scores
Score Vectors are nearly orthogonal
Angle between then is near 90 deg.
Cosine of angle is near 0 i.e. 0%
Opposite scores
Score Vectors in opposite direction
Angle between then is near 180 deg.
Cosine of angle is near -1 i.e. -100%

Note. Linked from http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

The equation for the cosine similarity is displayed below. My implementation is available in the github repository in /functions/word2vec.

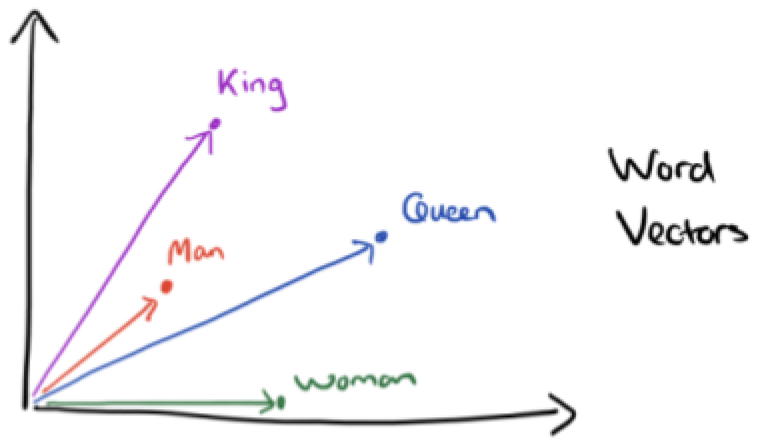
Figure 7. Cosine Similarity

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Note. Image linked from http://cs.carleton.edu/cs_comps/0910/netflixprize/final_results/knn/index.html

To clarify further, let's use another example from Adrian's blog. The graphic below shows a visual representation of how our word vectors may be plotted. The angle between the vectors determines their similarity. The closer the line, the more similar they are.

Figure 8.



Note. Image linked from https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

In this project, I wrote a small function to help calculate the cosine similarity and determine which jobs were most similar. To do this, the function extracts the job of interest into a separate dataset. I then calculated the cosine similarity for each of the remaining jobs and the target job, saving the result to a new variable in the dataset. I then sorted the similarity values in descending order. The highest values represent jobs that were most similar, while those with values at the tail end of the file represent jobs that are least similar.

Results

Note. All results from this section can be reproduced using the JobExplorer jupyter notebook.

The final step of the puzzle was to create a simple function where you can enter an O*NET job title and have the most and least similar jobs output. I did this for both the Google and the O*NET models. As the data was output, I included the O*NET SOC code as well as the Holland code. As a reminder, this was a validation check; we would expect O*NET SOC codes of similar jobs to have similar hierarchical codes. Similarly, we would also expect similar jobs to have similar Holland codes. The word2vec jupyter notebook includes an interactive widget allowing you to select a job title and look at the output. Afew concrete examples are described below.

Example 1: Chefs and Head Cooks

As we see in the data below,both models do a pretty good job. Both models identify jobs that we would expect to be similar, such as cooks, short order cooks, private household cooks, etc. When we look at the cosine, we see they are expected to be fairly close to r one another. A review of the SOC codes indicates that the major grouping (35) is similar for all of the jobs. The RIASEC codes for most of these jobs are similar to what we would expect. Whilethere are differences, these differences are probablyreflective of poor ratings from O*NET, which defined the RIASEC code.

```
***************Chefs and Head Cooks***************


ONET:35-1011.00
Holland Code:ERA


***************
Google data
***************
The most similar jobs are...
        Cooks, Institution and Cafeteria; cosine:0.99; O*NET:35-2012.00; Holland code:RC
        Cooks, Short Order; cosine:0.99; O*NET:35-2015.00; Holland code:RC
        Combined Food Preparation and Serving Workers, Including Fast Food; cosine:0.99;
O*NET:35-3021.00; Holland code:CRE
        Cooks, Restaurant; cosine:0.99; O*NET:35-2014.00; Holland code:RE
        Cooks, Private Household; cosine:0.99; O*NET:35-2013.00; Holland code:ARC
        Cooks, Fast Food; cosine:0.99; O*NET:35-2011.00; Holland code:RC
        Baristas; cosine:0.98; O*NET:35-3022.01; Holland code:ECR
```

Counter Attendants, Cafeteria, Food Concession, and Coffee Shop; cosine:0.98; O*NET:35-3022.00; Holland code:RSE
Food Servers, Nonrestaurant; cosine:0.98; O*NET:35-3041.00; Holland code:SRE
Food Preparation Workers; cosine:0.98; O*NET:35-2021.00; Holland code:RC
The least similar jobs are...
Software Developers, Applications; cosine:0.84; O*NET:15-1132.00; Holland code:IRC
Investment Underwriters; cosine:0.84; O*NET:13-2099.03; Holland code:CE
Green Marketers; cosine:0.83; O*NET:11-2011.01; Holland code:EAI
Fuel Cell Technicians; cosine:0.82; O*NET:17-3029.10; Holland code:RCI
Data Warehousing Specialists; cosine:0.75; O*NET:15-1199.07; Holland code:IC

****
O*NET data
****
The most similar jobs are...
Cooks, Institution and Cafeteria; cosine:0.98; O*NET:35-2012.00; Holland code:RC
Cooks, Private Household; cosine:0.96; O*NET:35-2013.00; Holland code:ARC
Cooks, Restaurant; cosine:0.96; O*NET:35-2014.00; Holland code:RE
Baristas; cosine:0.95; O*NET:35-3022.01; Holland code:ECR
Food Service Managers; cosine:0.95; O*NET:11-9051.00; Holland code:ECR
First-Line Supervisors of Aquacultural Workers; cosine:0.95; O*NET:45-1011.06; Holland code:ERC
Cooks, Fast Food; cosine:0.95; O*NET:35-2011.00; Holland code:RC
Dietetic Technicians; cosine:0.95; O*NET:29-2051.00; Holland code:SIR
First-Line Supervisors of Housekeeping and Janitorial Workers; cosine:0.95; O*NET:37-1011.00; Holland code:ECR
First-Line Supervisors of Landscaping, Lawn Service, and Groundskeeping Workers; cosine:0.95; O*NET:37-1012.00; Holland code:ERC
The least similar jobs are...
Methane/Landfill Gas Collection System Operators; cosine:0.44; O*NET:11-3051.05; Holland code:CER
Green Marketers; cosine:0.43; O*NET:11-2011.01; Holland code:EAI
Methane/Landfill Gas Generation System Technicians; cosine:0.34; O*NET:51-8099.02; Holland code:RCI
Data Warehousing Specialists; cosine:0.31; O*NET:15-1199.07; Holland code:IC
Fuel Cell Technicians; cosine:0.20; O*NET:17-3029.10; Holland code:RCI

Example 2: Construction Carpenters
Again, consistent with the previous example, the results returned from both models seem reasonable. As we dive into the numbers, we again see fairly good consistency between SOC codes and RIASEC values.

***************Construction Carpenters***************

ONET:47-2031.01
Holland Code:RCI

****
Google data
****
The most similar jobs are...
Rough Carpenters; cosine:1.00; O*NET:47-2031.02; Holland code:RCI
Helpers--Carpenters; cosine:0.99; O*NET:47-3012.00; Holland code:RC
Brickmasons and Blockmasons; cosine:0.99; O*NET:47-2021.00; Holland code:RCI
Cabinetmakers and Bench Carpenters; cosine:0.99; O*NET:51-7011.00; Holland code:RC
Structural Metal Fabricators and Fitters; cosine:0.99; O*NET:51-2041.00; Holland code:RC
Helpers--Brickmasons, Blockmasons, Stonemasons, and Tile and Marble Setters; cosine:0.99; O*NET:47-3011.00; Holland code:R
Mechanical Door Repairers; cosine:0.98; O*NET:49-9011.00; Holland code:R
Drywall and Ceiling Tile Installers; cosine:0.98; O*NET:47-2081.00; Holland code:RC
Sawing Machine Setters, Operators, and Tenders, Wood; cosine:0.98; O*NET:51-7041.00; Holland code:RCI
Model Makers, Wood; cosine:0.98; O*NET:51-7031.00; Holland code:RAC
The least similar jobs are...
Fuel Cell Technicians; cosine:0.83; O*NET:17-3029.10; Holland code:RCI
Methane/Landfill Gas Collection System Operators; cosine:0.83; O*NET:11-3051.05; Holland code:CER
Investment Underwriters; cosine:0.80; O*NET:13-2099.03; Holland code:CE
Green Marketers; cosine:0.79; O*NET:11-2011.01; Holland code:EAI
Data Warehousing Specialists; cosine:0.75; O*NET:15-1199.07; Holland code:IC

****
O*NET data
****
The most similar jobs are...
Rough Carpenters; cosine:0.98; O*NET:47-2031.02; Holland code:RCI
Brickmasons and Blockmasons; cosine:0.97; O*NET:47-2021.00; Holland code:RCI
Cabinetmakers and Bench Carpenters; cosine:0.96; O*NET:51-7011.00; Holland code:RC
Helpers--Roofers; cosine:0.96; O*NET:47-3016.00; Holland code:RC
Sheet Metal Workers; cosine:0.96; O*NET:47-2211.00; Holland code:R
Roofers; cosine:0.95; O*NET:47-2181.00; Holland code:RC
Explosives Workers, Ordnance Handling Experts, and Blasters; cosine:0.95; O*NET:47-5031.00; Holland code:RIC

```
        Electromechanical Equipment Assemblers; cosine:0.95; O*NET:51-2023.00; Holland
code:RCI
        Painters, Construction and Maintenance; cosine:0.95; O*NET:47-2141.00; Holland code:RC
        Drywall and Ceiling Tile Installers; cosine:0.95; O*NET:47-2081.00; Holland code:RC
The least similar jobs are...
        Special Education Teachers, Preschool; cosine:0.23; O*NET:25-2051.00; Holland code:SA
        Data Warehousing Specialists; cosine:0.22; O*NET:15-1199.07; Holland code:IC
        Green Marketers; cosine:0.20; O*NET:11-2011.01; Holland code:EAI
        Legislators; cosine:0.19; O*NET:11-1031.00; Holland code:ES
        Investment Underwriters; cosine:0.16; O*NET:13-2099.03; Holland code:CE
```

In [ ]

Example 3: Computer Programmers
Consistent with the previous results, both models do a good job identifying jobs which are going to be pretty similar. The SOC codes and RIASEC codes again show results as we would anticipate.

```
***************Computer Programmers***************


ONET:15-1131.00
Holland Code:IC


***************
Google data
***************
The most similar jobs are...
        Software Developers, Applications; cosine:1.00; O*NET:15-1132.00; Holland code:IRC
        Software Developers, Systems Software; cosine:0.99; O*NET:15-1133.00; Holland
code:ICR
        Software Quality Assurance Engineers and Testers; cosine:0.99; O*NET:15-1199.01;
Holland code:ICR
        Computer Systems Engineers/Architects; cosine:0.99; O*NET:15-1199.02; Holland
code:IRC
        Computer Systems Analysts; cosine:0.99; O*NET:15-1121.00; Holland code:ICR
        Database Administrators; cosine:0.99; O*NET:15-1141.00; Holland code:CI
        Computer and Information Systems Managers; cosine:0.99; O*NET:11-3021.00; Holland
code:ECI
        Computer User Support Specialists; cosine:0.99; O*NET:15-1151.00; Holland code:RIC
        Web Developers; cosine:0.99; O*NET:15-1134.00; Holland code:CIR
        Information Technology Project Managers; cosine:0.99; O*NET:15-1199.09; Holland
code:EC
The least similar jobs are...
```

```
	Bakers; cosine:0.77; O*NET:51-3011.00; Holland code:RC
	Landscaping and Groundskeeping Workers; cosine:0.76; O*NET:37-3011.00; Holland
code:RC
	Helpers--Pipelayers, Plumbers, Pipefitters, and Steamfitters; cosine:0.76;
O*NET:47-3015.00; Holland code:R
	Surgeons; cosine:0.75; O*NET:29-1067.00; Holland code:IRS
	Agricultural Equipment Operators; cosine:0.74; O*NET:45-2091.00; Holland code:R

***************
O*NET data
***************
The most similiar jobs are...
	Software Developers, Applications; cosine:0.99; O*NET:15-1132.00; Holland code:IRC
	Computer Systems Engineers/Architects; cosine:0.98; O*NET:15-1199.02; Holland
code:IRC
	Computer Systems Analysts; cosine:0.98; O*NET:15-1121.00; Holland code:ICR
	Software Quality Assurance Engineers and Testers; cosine:0.98; O*NET:15-1199.01;
Holland code:ICR
	Database Administrators; cosine:0.98; O*NET:15-1141.00; Holland code:CI
	Software Developers, Systems Software; cosine:0.98; O*NET:15-1133.00; Holland
code:ICR
	Computer User Support Specialists; cosine:0.98; O*NET:15-1151.00; Holland code:RIC
	Database Architects; cosine:0.98; O*NET:15-1199.06; Holland code:ICE
	Computer and Information Systems Managers; cosine:0.97; O*NET:11-3021.00; Holland
code:ECI
	Information Technology Project Managers; cosine:0.97; O*NET:15-1199.09; Holland
code:EC
The least similar jobs are...
	Welders, Cutters, and Welder Fitters; cosine:0.23; O*NET:51-4121.06; Holland code:RC
	Glaziers; cosine:0.21; O*NET:47-2121.00; Holland code:RC
	Solderers and Brazers; cosine:0.19; O*NET:51-4121.07; Holland code:R
	Musical Instrument Repairers and Tuners; cosine:0.16; O*NET:49-9063.00; Holland
code:RAI
	Helpers--Pipelayers, Plumbers, Pipefitters, and Steamfitters; cosine:0.14;
O*NET:47-3015.00; Holland code:R
```

An exploration of the outputted results revealed several things. First, the jobs recommended by the pretrained vectors of Google and the O*NET trained models performed similarly. In fact, in the examples above, as well as other examples I explored,recommended the same jobs more often than notalthough the order within the list sometimes differedt. Given the pure scale of documents, this resultwas surprising. Second, with respect to e the O*NETSOC codes, the top jobs generally followed the same sort of SOC code hierarchy, particularly at the "major job category level." This seems to indicate that the word2vec models pick up the same sorts of characteristics outlined by subject matter experts and survey methods used by O*NET. Third, when we evaluate the RIASEC codes for the most similar jobs, we again see general congruence amongst the recommended jobs. Again, these results are consistent with the type of results we would expect from Holland's theory and indicate that word2vec is actually mimicking the classifications of experts.

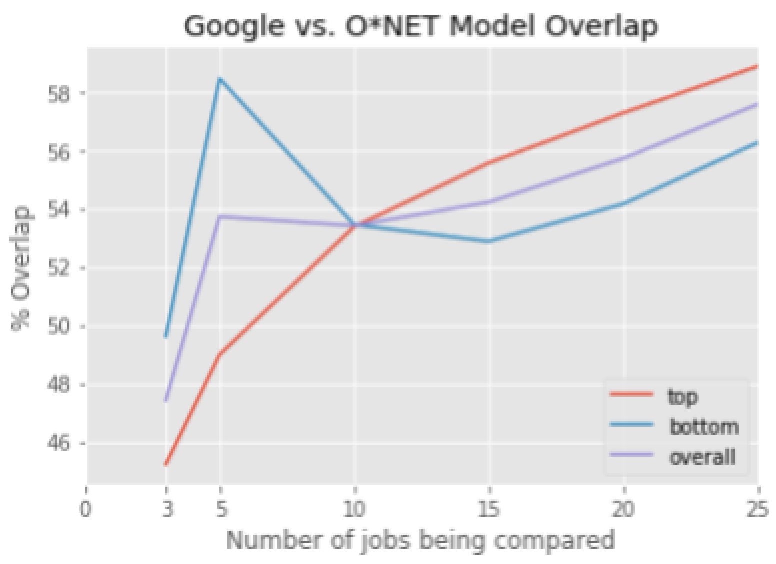
It's important to note that there instances where discrepancies pop up. Jobs with different RIASEC codes or SOC codes can appear. It's unclear exactly what these discrepancies mean. It could be related to poor quality data within O*NET, inaccurate models, or perhaps there is better way to classify the jobs using text.

Exploratory Cosine Similarity Analyses

Note. All analysis for this section are available in the word2vec_exploration jupyter notebook.

Thus far, we have looked at individual jobs and the similarity between the Google trained and O*NET trained models. I conducted some exploratory analyses to determine the percentage of overlap between the recommendations using the Google and O*NET models. To accomplish this, I calculated the cosine similarity for every job in both models. I then ran comparisons looking at the top and bottom 3, 5, 10, 15, 20, and 25 jobs recommended by the two engines. To calculate the overlap, I took the total number of common elements multiplied by two, divided by the total number of elements, and multiplied by 100 (see list comparison in the ../functions/word2vec.py file). Figure 9 presents the results of these analyses. As you can see in Figure 9, the average percent agreement between the Google and O*NET mode is lower than 46% when we just look at the top three jobs. Interestingly, there is more agreement with the bottom jobs when the number of jobs being compared is low. As the number of jobs compared increases, the agreement increases, indicating that there may be slight differences in the cosine similarity calculated by both models, leading to a shuffling list order for the jobs that are recommended as most similar to the target job. The model shows some convergence at 10 jobs, with just under 54% matching up for all. What's interesting here is that the overlap within the overall O*NET database is lower than I would have expected based on visual exploration. My hunch is that there may be lower agreement for jobs that have lower overall numbers of words or shorter descriptions within O*NET, which does occur.

Figure 9. Google vs. O*NET Model Overlap



Recommendations and Future Directions

The purpose of this study was to determine if I could use word2vec to generate recommendations for jobs that are similar to a target job using two models. Next, I wanted to examine the similarity between the recommended jobs and the the target job and determine whether the job recommendations were plausible, based on O*NET SOC codes and Holland codes. Finally, I explored how much overlap there was between word2vec models derived from the 'Google' and from O-NET engines.

First, my results showed that the jobs recommended by either engine closely mirror the theory behind Holland's codes and the O*NET SOC hierarchy, particularly at the major job category level. These results indicate that word2vec is mirroring the types of categorizations made by experts simply by exploring the words. However, this is not the case for all job as there were unexpected results. These results may be due to factors such as small amounts of text for the job, text which is not very separating, or it could be related to the O*NET database structure as a whole. The hierarchy of the ONET database is such that general categories may have little text and may be weakening the results. Future studies should examine if additional cleaning of O*NET codes need to occur to tighten up results. It's also possible that there are differences between different Holland code areas or specific O*NET SOC code groupings. The best way forward would be to filter to cases that show a low degree of overlap or inconsistency between the O*NET SOC or Holland codes to determine the root cause.

Taken as a whole, in my opinion these results are quite encouraging and exciting as an initial first step. Using word2vec, I was able to empirically identify jobs that are similar based purely on text. These validity of these results is supported by evidence that they mirror theory (Holland codes) and empirical data (O*NET SOC codes). Both of the former techniques are human dependent and require extensive resources. In contrast, using word2vec, I was able to build a machine learning classifier based only on the job text of descriptions. The results from this model   could potentially be mapped to the existing O*NET database, allowing for updates of the database from contemporary job postings.