ANLY 540 - Final Project: Text analysis and language modeling for job postings

Anubha Gupta, Gaurav Gade and Siddharth Parmar

Contents

Introduction:

Language modeling, text analysis and semantic analysis are powerful tools to analyze large chunks of textual data. Writing a proper job requirement is a complex and detailed process and keeps evolving according to the needs of the company. Most jobseekers have a hard time finding the best position that matches their interests. Especially since job requirements can vary significantly even though the job titles are the same or similar. In today's modern internet era, there are several applicants per job role. According to Glassdoor [5], each corporate job in America attracts 250 resumes. Only 4-6 are selected and 1 candidate is offered the job. Most of these resumes go through the initial step of resume parsing where the resume content is parsed and analyzed by a text mining algorithm that scores every candidate based on criteria set by the company. As part of this study, we are going to analyze a subset of 19,000 job postings to determine which keywords are most frequently occurring in the job requirement. We will investigate the relationship between these top keywords which are usually a combination of management, technical and inter-personal skills. Lastly, we will perform a job similarity analysis and perform topic modeling to view patterns between different job roles.

Why does this topic matter?

In today's digital age, job applications have increasingly become online, and data driven. Most companies use some level of computational text parsing of resumes to shortlist candidates before a recruiter physically looks at the resume. There may be significant variance between different job postings even when the job title may be the same. Hence, it is important for the job seeker to be able to identify roles that are most relevant to his or her skillset. Analyzing job postings will help us to reveal interesting patterns between the job data. We hope to detect similarities in job requirements and find out strongly correlated skills. We have used topic modeling, similarity models, word to vector and network modeling to analyze these postings.

The LDA visualizations have been used to identify common keywords across the postings for the top 5 themes. Universities can utilize the inferences from this study to better plan coursework so that students will receive most of the skills required to meet the most frequency sought after job roles. The study results may be of huge interest to professional job-related companies like Indeed.com and LinkedIn. As part of this research study, we are performing text analysis and language modeling on job profiles

What is the background knowledge that someone would need to understand the field or area that you have decided to investigate?

Anyone who has applied for jobs online as a job seeker or has advertised jobs online in the capacity of an employer should have first-hand experience of working with job postings. This research study does not require any prior knowledge. The intended audience for this study would be job seekers, recruiters, students and University professors.

Related Background:

While applying for jobs online, all three of us noticed a huge variance in job description even though the job titles were similar. Our research study aims to analyze these 2,000 postings using text modeling techniques and linguistic models like vector space, clustering and network models to determine which job descriptions are semantically similar. This is an efficient method for job search.

While we were students hunting for jobs in college, it was difficult to gauge which skills are hot in demand for a given job title. By using these linguistic models, we can determine the hottest skills that are currently required in the job market for a specific job. So, college students as well as universities can customize their curriculum accordingly.

Research Question

Our research aims to semantically compare job descriptions to determine the most relevant / similar job(s) for a specific job search. We seek to find patterns between themes using topic modeling. We aim to analyze entity-relationships using network modeling to find correlation between job requirements. Lastly, we seek to build a classifier model that can predict whether the job is an IT job or not.

Define what the problem is and what the goal of the analysis is. Then, specify at least one research question that you answer with your analysis. This section should be a full paragraph.

Method

The data we have utilized for the purpose of our analysis was sourced from Yahoo! Mailing group (Link for data: https://groups.yahoo.com/neo/groups/careercenter-am). The dataset consists of 2,000 job postings that were posted through the Armenian human resource portal CareerCenter, which was the only online human resource portal in the early 2000s. The data consists of job posts from 2004-2005. [1]

The size of the dataset is 9 MB and it consists of 24 columns. We have also created our own columns wherever necessary. The fields or variables in the dataset are as follows:

- JobID: A numeric ID used for referencing all job postings uniquely.
- Jobpost: This field captures the details of the original job post.
- Date: This field captures the date the job was posted on the job portal.
- Title: Title is the job position for the job opening.
- Announcement Code: This is an internal code, and is usually missing.
- Company: This field captures the name of the organization for which the job is advertised.
- Eligibility: Eligibility of the candidate.
- **Audience**: Potential candidates for application.
- **StartDate**: Start date of work.
- **Duration**: Duration of the employment
- **Salary**: Salary of the candidate
- **ApplicationP**: Application Procedure
- **OpeningDate**: Opening date of the job announcement
- **Deadline**: Deadline for the job announcement
- **Notes**: Additional Notes
- **Term**: This field captures whether the advertised job is part-time or full-time.
- **Location**: the geographic location where the job will be located.
- **JobDescription**: Job description
- **JobRequirement**:Job requirements and/or duties
- **RequiredQual**:Required qualification
- **AboutCompany**: Information about the company
- **Year**: Year of the announcement (derived from the field date)
- **Month**: Month of the announcement (derived from the field date)
- **Attach**: Attachments
- **IT**: This field stores TRUE or FALSE depending on whether the job posting is for an IT job or not. This variable is created by a simple search of IT job titles within column "Title".

The above variables fit into our research question since we are trying to semantically compare job descriptions to determine the most relevant jobs for a specific job search and our dataset consists of around 2,000 job postings for a varied set of job openings for multiple technical and non-technical domains. Hence, we have a robust dataset with fields that have captured the job details and descriptions, which form the backbone for our analysis.

The first step in analyzing the data in order to answer the research question of finding similar jobs. By studying the data, we realized that the Job Requirement Column gives a detailed description to a job title. For example: we are looking for a software developer job title and like a specific job description, we had to find a way to filter similar job profiles that would interest the user. There are hundreds of titles with the word "developer" and simply filtering on that would not give us the best results. So, we choose to create a corpus from Job Requirements and create an analytics model that can filter out the closest job profiles.

We also have approached the research question for additional aspects of the job application process. In the dataset we have a defined column that assigns a job profile if it is an IT job or not.

We created another model that reads the job requirement and job title, and can tell the user if a job post falls under the category of IT.

To help the job seekers on creating resumes is another important factor that has been addressed. Often users with great resumes do not even make it to the interview as they get filtered out for not using the right keywords. We ran a topic modelling algorithm that picks out most common keywords related to a job profile. We also ran a network model that allows us to look for most commonly paired words used in job descriptions. For example: Design and develop. This model also provided further insights to the skills one must have for a particular role.

Analysis

Network Modeling.

Step1: Attaching required libraries:

```
##r chunk
library(reticulate)
library(tm)

## Loading required package: NLP

library(topicmodels)
library(tidyverse)

## -- Attaching packages ---------------------------------------------------------------------------
tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.1
## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## Warning: package 'stringr' was built under R version 4.0.3

## -- Conflicts ------------------------------------------------------------------------------------
tidyverse_conflicts() --
## x ggplot2::annotate() masks NLP::annotate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()

library(tidytext)
library(slam)
library(lubridate)

library(gutenbergr)

## Warning: package 'gutenbergr' was built under R version 4.0.3
```

```
library(stringr)
library(dplyr)
library(tidyr)
library(jsonlite)

library(widyr)
library(ggplot2)
library(igraph)

library(ggraph)

job = read.csv("C:/Users/nagar/Documents/ANLY 540 Language
Modeling/Project/Job/job_final.csv")
# install.packages("lubridate",repos="http://cran.r-project.org")



job$Date <- lubridate::dmy(job$Date)
job$Term <- as.factor(job$Term)
job$IT <- as.factor(job$IT)



book = job[c(1,4,9)]
book = na.omit(book)

book = book %>% mutate(JobID = row_number())



book$JobRequirment = array(book$JobRequirment)
book$Title = array(book$Title)
book$JobID = array(book$JobID)

book = na.omit(book)
```

Clean up the data

In this section, we created a tibble/dataframe of the individual words from the job requirement. We used unnest_tokens and anti_join on the column job requirement to create a unigram list of words without stop words included.

```
#Tibble:

book_word <- book %>%
  unnest_tokens(word, JobRequirment) %>%
    anti_join(stop_words)

## Joining, by = "word"
```

Simple statistics

We used the count function to determine the most frequent words used in the job requirement column that are not stop words.

Output: These are the top 19 most occurring words in job requirement.

```
book_word %>%
  #count(word, sort = TRUE)

##                word    n
## 1              ensure 1276
## 2             develop 1069
## 3             provide 1064
## 4             project  972
## 5         development  877
## 6             support  859
## 7         responsible  853
## 8             reports  850
## 9          activities  811
## 10            prepare  802
## 11         management  744
## 12             design  721
## 13               team  702
## 14            perform  623
## 15            maintain  615
## 16          technical  591
## 17            related  585
## 18              staff  585
## 19        participate  561
book %>% count(book$Title, sort = TRUE)
```

Results, top 25 job titles.

```
book %>% count(book$Title, sort = TRUE)

                                                                  book$Title
## 1                                                              Accountant
## 2                                                           iOS Developer
## 3                                                    Senior Java Developer
## 4                                                           Web Developer
## 5                                                    Medical Representative
## 6                                                            QA Engineer
## 7                                                     Senior Android
Developer
## 8                                                          Android Developer
```

```
## 9                                                    Chief Accountant
## 10                                                          Lawyer
## 11                                              Administrative Assistant
## 12                                                   Project Manager
## 13                                          Receptionist/ Administrative
Assistant
## 14                                                    .NET Developer
## 15                                                     HR Manager
## 16                                             Senior .NET Developer
## 17                                                   Java Developer
## 18                                              Senior PHP Developer
## 19                                             Procurement Specialist
## 20                                               Software Developer
## 21                                             Accounting Assistant
## 22                                              Marketing Manager
## 23                                             Marketing Specialist
## 24                                          Customer Service
Representative
```

Collocates clean up

We create a tibble/dataframe that includes the collocate pairs from the job requirement using pairwise_count.

```
title_word_pairs <- book_word %>%
  pairwise_count(word, JobID, sort = TRUE, upper = FALSE)
```

```
## Warning: `distinct_()` is deprecated as of dplyr 0.7.0.
## Please use `distinct()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

## Warning: `tbl_df()` is deprecated as of dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
head(title_word_pairs)
```

```
## # A tibble: 6 x 3
##   item1   item2          n
##   <chr>   <chr>      <dbl>
## 1 develop development   315
## 2 reports prepare       303
## 3 provide support       273
## 4 ensure  provide       265
## 5 develop provide       261
## 6 reports ensure        257
```

Create a network plot

We created a network plot of the collocates - **n > 175 and n < 200**. This eliminates low frequency words as well as highly occurring non-qualitative words (noise)

```
set.seed(52550)
title_word_pairs %>%
 filter(n >= 175 & n <= 200) %>%
 graph_from_data_frame() %>%
 ggraph(layout = "fr") + #use ?ggraph to see all the options
 geom_edge_link(aes(edge_alpha = n, edge_width = n), edge_colour = "purple") +
 geom_node_point(size = 1) +
 geom_node_text(aes(label = name), repel = TRUE,
                point.padding = unit(0.3, "lines")) +
 theme_void()
```

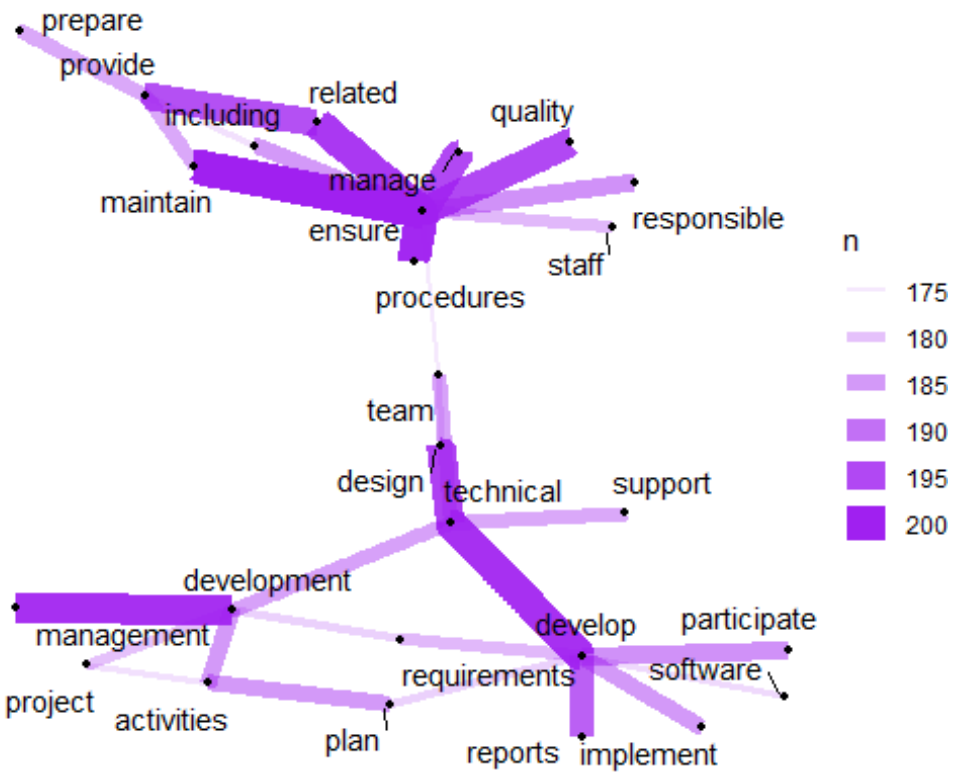


Fig1.1 – Network Plot showing relationship between key terms used in IT job postings.

Interpretation:

The network plot shows us major words such as design, technical and develop that other related branched words such as participate, software reports. This helps us understand that the technical design and develop functions in a job description also require the skills of reporting and working in teams. It also needs focus on implementation and most of these are paired with software.

Similarly, ensure, quality, procedures are the branched words mostly paired with manage. For a managerial role, using keywords such as ensure quality and maintain procedures with help boost the probability of job seekers getting filtered for a job applied.

Topic Modelling

We have used gensim topic modeling on the job posting data to evaluate which topics are most prominent along with the LDA visualization.

```
##python chunk
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
import nltk
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

import nltk
import gensim
from nltk.corpus import abc

import string
import pyLDAvis


import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt

import gensim.corpora as corpora
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()


#import data
dfhome  = pd.read_csv('C:/Users/nagar/Documents/ANLY 540 Language
Modeling/Project/Job/job_final.csv')

#cleanup for this specific example
df = dfhome[['JobRequirment', 'Title', 'Date', 'Company']]
df.dropna(inplace=True)

## C:/Users/nagar/AppData/Local/r-miniconda/envs/r-reticulate/python.exe:1:
SettingWithCopyWarning:
## A value is trying to be set on a copy of a slice from a DataFrame
##
## See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
df.info()

## <class 'pandas.core.frame.DataFrame'>
## Int64Index: 1814 entries, 0 to 1813
## Data columns (total 4 columns):
##  #   Column         Non-Null Count  Dtype
## ---  ------         --------------  -----
##  0   JobRequirment  1814 non-null   object
##  1   Title          1814 non-null   object
##  2   Date           1814 non-null   object
##  3   Company        1814 non-null   object
## dtypes: object(4)
## memory usage: 70.9+ KB
```

Gensim Modeling in Python for Topic Modelling

Transfer the df[Requirement] to Python and convert it to a list for processing.

Processed text:

```
##python chunk


speeches = list(r.job["JobRequirment"])
```

Process the text using Python.

```
##python chunk

##create a spot to save the processed text
processed_text = []

##loop through each item in the list
for speech in speeches:
  #lower case
  speech = speech.lower()
  #remove punctuation
  speech = speech.translate(str.maketrans('', '', string.punctuation))
  #create tokens
  speech = nltk.word_tokenize(speech)
  #take out stop words
  speech = [word for word in speech if word not in stopwords.words('english')]
  #stem the words
  speech = [ps.stem(word = word) for word in speech]
  #add it to our list
  processed_text.append(speech)

processed_text[0]
```

```
## ['respons', 'prepar', 'daili', 'bank', 'payment', 'order', 'account', 'record', 'monitor', 'gasolin',
'balanc', 'monthli', 'consumpt', 'base', 'report', 'mileag', 'prepar', 'calcul', 'pay', 'busi', 'trip', 'expens',
'prepar', 'maintain', 'monitor', 'staff', 'attend', 'record', 'calcul', 'relat', 'time', 'offic', 'record', 'calcul',
'actual', 'vacat', 'sick', 'day', 'use', 'prepar', 'monitor', 'servic', 'contract', 'respons', 'payrol', 'calcul',
'maintain', 'gift', 'accord', 'armenian', 'legisl', 'track', 'disburs', 'sponsor', 'children', 'enter',
'donorperfect', 'onlin', 'perform', 'random', 'audit', 'sponsor', 'children', 'direct', 'director', 'financ',
'segreg', 'administr', 'charit', 'expens', 'monthli', 'basi', 'current', 'done', 'year', 'audit', 'purpos',
'segreg', 'monthli', 'quarterli', 'basi', 'order', 'avoid', 'huge', 'task', 'yearend', 'correspond', 'ifr',
'track', 'maintain', 'tax', 'exempt', 'given', 'govern', 'ensur', 'meet', 'requir', 'stipul', 'exempt', 'avoid',
'futur', 'issu', 'prepar', 'maintain', 'job', 'descript', 'amaa', 'staff', 'maintain', 'uptod', 'job', 'process',
'amaa', 'base', 'process', 'creat', 'intern', 'audit']
```

Create the dictionary and term document matrix in Python.

```
##python chunk
#create a dictionary of the words
dictionary = corpora.Dictionary(processed_text)
```

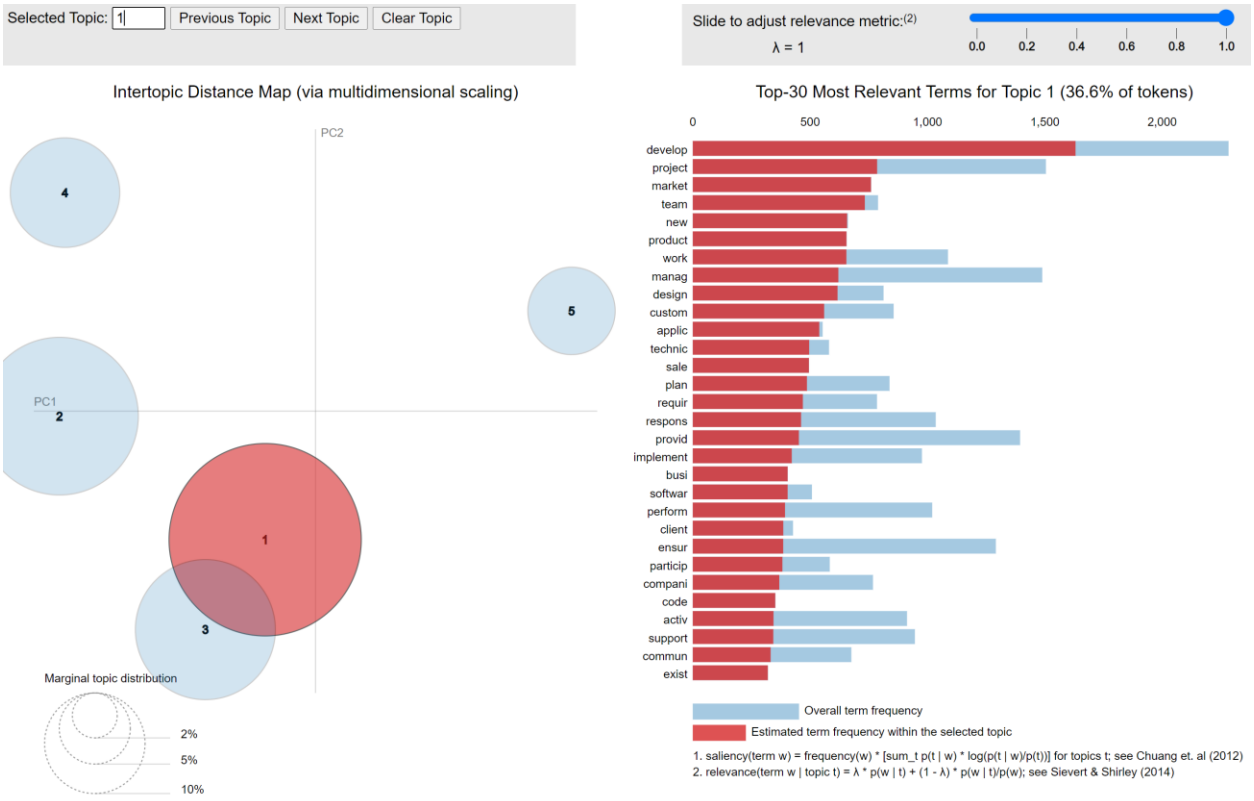
Create the LDA Topics model in Python using the same number of topics as used in the Factor Analysis assignment.

```
##python chunk



#create a TDM
doc_term_matrix = [dictionary.doc2bow(doc) for doc in processed_text]

lda_model = gensim.models.ldamodel.LdaModel(corpus = doc_term_matrix, #TDM
                                            id2word = dictionary, #Dictionary
                                            num_topics = 5,
                                            random_state = 100,
                                            update_every = 1,
                                            chunksize = 100,
                                            passes = 10,
                                            alpha = 'auto',
                                            per_word_topics = True)
print(lda_model.print_topics())
```

```
## [(0, '0.062*"test" + 0.032*"develop" + 0.020*"media" + 0.017*"websit" + 0.015*"materi" +
0.014*"design" + 0.013*"content" + 0.013*"public" + 0.013*"work" + 0.012*"use"'), (1,
'0.027*"project" + 0.017*"activ" + 0.016*"support" + 0.015*"implement" + 0.014*"includ" +
0.014*"develop" + 0.013*"program" + 0.013*"coordin" + 0.013*"organ" + 0.013*"provid"'), (2,
'0.018*"manag" + 0.017*"ensur" + 0.015*"provid" + 0.013*"control" + 0.013*"intern" +
0.013*"legal" + 0.012*"system" + 0.012*"oper" + 0.011*"secur" + 0.011*"inform"'), (3,
'0.032*"report" + 0.031*"account" + 0.026*"prepar" + 0.025*"financi" + 0.014*"custom" +
0.014*"respons" + 0.013*"perform" + 0.012*"make" + 0.012*"data" + 0.012*"tax"'), (4,
'0.035*"develop" + 0.017*"project" + 0.016*"market" + 0.016*"team" + 0.014*"new" +
0.014*"product" + 0.014*"work" + 0.013*"manag" + 0.013*"design" + 0.012*"custom"')]
```

Create the interactive graphics html file.

```
##python chunk

vis = pyLDAvis.gensim.prepare(lda_model, doc_term_matrix, dictionary, n_jobs = 1)
pyLDAvis.save_html(vis, 'LDA_Visualization.html') ##saves the file
```



Interpretation:

The LDA visualization plot shows the 5 most prominent topics that have been created based on IT job postings. The blue charts show the overall term frequency and the red graph show the estimated term frequency.

The 5 themes reveal the following:

Theme 1: test, develop, media, website, material, design, content, public, work, use. These seem to be related to designing web pages and web-site development for public work and government projects.

Theme 2: project, active, support, implement, including, develop, program, coordination, organ, provide. These words seem to indicate higher level program co-ordination and implementation,

Theme 3: management, ensure, provide, control, intern, legal, system, operations, security, information. These words seem to be related to legalities, compliance and security. May also include network security and operations.

Theme 4: report, account, preparation, financial, custom, response, perform, make, data, tax. These words seem to include tax filing, tax compliance, financial reporting and accounting.

Theme 5: develop, project, market, team, new, product, work, manag, design, custom. These words seem to include product development, product management, design projects and customizations.

Similarity between job descriptions.

Clean up the data

In this section, we created a tibble/dataframe of the individual words from the job requirement using unnest_tokens and anti_join to create a unigram list of words without stopwords included.

```
#Tibble:

book_word <- book %>%
   unnest_tokens(word, JobRequirment) %>%
      anti_join(stop_words)

## Joining, by = "word"
```

SIMILARITY

```
##python chunk

stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
   # lower case and remove special characters\whitespaces
   doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I|re.A)
   doc = doc.lower()
   doc = doc.strip()
   #remove punctuation
   doc = doc.translate(str.maketrans('', '', string.punctuation))
   # tokenize document
   tokens = nltk.word_tokenize(doc)
   # filter stopwords out of document
   filtered_tokens = [token for token in tokens if token not in stop_words]
   # re-create document from filtered tokens
   doc = ' '.join(filtered_tokens)
   return doc

normalize_corpus = np.vectorize(normalize_document)
```

```
##python chunk
norm_corpus = normalize_corpus(list(df['JobRequirment']))
len(norm_corpus)

## 1814

from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(ngram_range=(1, 2), min_df=2)
tfidf_matrix = tf.fit_transform(norm_corpus)
tfidf_matrix.shape

## (1814, 26305)

##python chunk
from sklearn.metrics.pairwise import cosine_similarity

doc_sim = cosine_similarity(tfidf_matrix)
doc_sim_df = pd.DataFrame(doc_sim)
doc_sim_df.head()

##       0        1        2    ...     1811     1812     1813
## 0  1.000000  0.028014  0.015618  ...  0.000000  0.016249  0.013300
## 1  0.028014  1.000000  0.008781  ...  0.008853  0.015593  0.000000
## 2  0.015618  0.008781  1.000000  ...  0.028164  0.013717  0.005667
## 3  0.021605  0.010784  0.164321  ...  0.000000  0.012416  0.005633
## 4  0.016182  0.010444  0.005497  ...  0.008087  0.022031  0.000000
##
## [5 rows x 1814 columns]

##python chunk
def article_recommender(Title, JobRequirment, doc_sims):
    # find article id
    article_idx = np.where(JobRequirment == Title)[0][0]
    # get article similarities
    article_similarities = doc_sims.iloc[article_idx].values
    # get top 5 similar article IDs
    similar_article_idxs = np.argsort(-article_similarities)[1:11]
    # get top 5 article
    similar_articles = JobRequirment[similar_article_idxs]
    # return the top 5 articles
    return similar_articles

article_recommender("Chief Financial Officer", #name of article must be in dataset
                    df["Title"].values, #all article names
                    doc_sim_df #pd dataframe of similarity values
                    )
```

```
## array(['Finance Controller', 'Senior Manager of Support Services',
##        'Finance Analyst', 'Chief Financial Officer (CFO)',
##        'Chief Financial Officer (CFO)', 'Senior Finance Specialist',
##        'Auditor', 'Chief Accountant', 'Financial Controller',
##        'Senior Auditor'], dtype=object)
```

```
array(['DevOps Software Engineer', 'Software Engineer',
       'Senior Java Developer', 'Java Developer', 'Java Developer',
       'Java Developer', 'Front End Developer', 'Web UI Developer',
       'Java Developer', 'UX Designer'], dtype=object)
```

Interpretation:

The job similarity module is extremely useful to quickly find out similar job roles based on the job requirement, roles and responsibility rather than just the job titles. As we know, job titles can vary significantly and by running this module, we can get the most similar roles even though the job titles may be different.

In this case, we tried to find out job titles like ‘CFO’ and ‘Software Engineer’. For CFO, we found that the job titles similar to CFO based on job requirement were:

1. Finance Analyst
2. Chief Financial Officer (CFO)
3. Auditor
4. Senior Auditor
5. Chief Financial Officer (CFO)
6. Senior Finance Specialist
7. Chief Accountant
8. Financial Controller

For ‘Software engineer’, the titles similar were:

1. DevOps Software Engineer
2. Senior Java Developer
3. Java Developer
4. Java Developer
5. Software Engineer
6. Java Developer
7. Front End Developer
8. UX Designer

9. Java Developer
10. Web UI Developer

Word2vec model:

Cleanup for Word2vec model

```
##python chunk

SoData = dfhome

REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]') #remove symbols with space
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]') #take out symbols altogether
STOPWORDS = set(stopwords.words('english')) #stopwords

def clean_text(text):
    text = BeautifulSoup(text, "lxml").text # HTML decoding
    text = text.lower() # lowercase text
    text = REPLACE_BY_SPACE_RE.sub(' ', text) # replace REPLACE_BY_SPACE_RE
symbols by space in text
    text = BAD_SYMBOLS_RE.sub('', text) # delete symbols which are in BAD_SYMBOLS_RE
from text
    text = ' '.join(word for word in text.split() if word not in STOPWORDS) # delete stopwords
from text
    return text


SoData['JobRequirment'] = SoData['JobRequirment'].apply(clean_text)
```

Split the data

Split the data into testing and training data.

```
##python chunk

X =SoData['JobRequirment']
y = SoData['IT']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 42)
```

Process the data

For word2vec, we created the tokenized vectors of the text.

```
##python chunk

tokenized_train = [nltk.tokenize.word_tokenize(text)
                   for text in X_train.to_list()]
tokenized_test = [nltk.tokenize.word_tokenize(text)
                  for text in X_test.to_list()]
```

Build Model

We used Word2Vec model.

```
# build word2vec model
w2v_model = gensim.models.Word2Vec(tokenized_train,
                                   size=100,
                                   window=6,
                                   min_count=2,
                                   iter=5,
                                   workers=4)
```

Convert the model

Converted the model into a set of features to use in our classifier.

```
##python chunk
def document_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index2word)

    def average_word_vectors(words, model, vocabulary, num_features):
        feature_vector = np.zeros((num_features,), dtype="float64")
        nwords = 0.

        for word in words:
            if word in vocabulary:
                nwords = nwords + 1.
                feature_vector = np.add(feature_vector, model.wv[word])
        if nwords:
            feature_vector = np.divide(feature_vector, nwords)

        return feature_vector

    features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features)
                for tokenized_sentence in corpus]
    return np.array(features)

##python chunk
# generate averaged word vector features from word2vec model
avg_wv_train_features = document_vectorizer(corpus=tokenized_train,
                                            model=w2v_model,
                                            num_features=100)
avg_wv_test_features = document_vectorizer(corpus=tokenized_test,
```

```
                                model=w2v_model,
                                num_features=100)
```

Build a classifier model

We used logistic regression to classify the data.

```
##python chunk

#define your outcomes
my_tags = ["TRUE","FALSE"]

#build a log model
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=10000)

#fit the data to the log model
logreg = logreg.fit(avg_wv_train_features, y_train)
```

Examine the results

Accuracy, recall, and precision of the model.

```
##python chunk
#predict new data
y_pred = logreg.predict(avg_wv_test_features)

#print out results
print('accuracy %s' % accuracy_score(y_pred, y_test))

## accuracy 0.837465564738292

print(classification_report(y_test, y_pred,target_names=my_tags))

##               precision    recall  f1-score   support
##
##        TRUE       0.84      0.98      0.90       280
##       FALSE       0.83      0.36      0.50        83
##
##    accuracy                           0.84       363
##   macro avg       0.84      0.67      0.70       363
## weighted avg       0.84      0.84      0.81        36
```

Interpretation:

The word2vec is a neural network similar to a skip gram model that allows creation of meaningful models that understand word embeddings from large bodies of texts.

In our case we have a huge list of job requirements that can been categorized as IT and NON-IT jobs. This logistic regression model allows us to predict the category. We trained 80% of the dataset we had and 20% of the dataset was tested. The output shows 86% accuracy of the model.

Discussion

The accuracy of our model is about 86% which is based on Logistic regression classifier. The data that we have concentrated on for our analysis is all 2015 data and is based on Armenian job market data.

We analyzed related literature and found some interesting research papers that discuss their approaches and findings for similar real world problems. The research paper 'Document-based Recommender System for Job Postings using Dense Representations'[2]discusses the usage of dense vector representations to enhance a large-scale job recommendation system and tries to rank German job advertisements regarding their similarity with the best results obtained by combining job titles with full-text job descriptions. The research paper 'Job Recommendation From Semantic Similarity of LinkedIn Users' Skills'[3] tries to find out relationships between jobs and people skills making use of data from LinkedIn users' public profiles. The authors have applied Latent Semantic Analysis (LSA), and hierarchical clustering of job positions to build a job recommendation system. Another paper 'An Information-Geometric Approach to Document Retrieval and Categorization' tries to develop from first information-geometric principles a general method for learning the similarity between text documents and derived a canonical similarity function - known as the Fisher kernel. [4] Another paper "Methods and Metrics for Cold-Start Recommendations" discusses how the authors developed a method for recommending items that combines content and collaborative data under a single probabilistic framework using a naive Bayes classifier on the cold-start problem, to obtain deeper understanding of the performance characteristics of recommender systems. [5]

Conclusion and Recommendations

The online job market is a good indicator of overall demand for labor in the local economy.

The linguistic models that we created can be used in the following ways:

•Understand the demand for certain professions, job titles, or industries

•Help universities with curriculum development.

•Identify skills that are most frequently required by employers, and how the distribution of necessary skills changes over time.

•Make recommendations to job seekers and employers.

•Find similar jobs based on the ones that you have previously applied for.

Credit Guidelines

Developed Research Question: Anubha, Gaurav and Siddharth

Acquired Data: Anubha, Gaurav and Siddharth

Analyzed Data: Anubha, Gaurav and Siddharth

Researched Related Work: Anubha, Gaurav and Siddharth

Wrote Introduction: Anubha, Gaurav and Siddharth

Wrote Method: Anubha, Gaurav and Siddharth

Wrote Analysis/Results: Anubha, Gaurav and Siddharth

Wrote Discussion: Anubha, Gaurav and Siddharth

Revisions/Edits: Anubha, Gaurav and Siddharth

Citations/References:

1. Hab, M. (2016). Online Job Postings. Retrieved 2020, from https://www.kaggle.com/madhab/jobposts/notebooks
2. Elsafty, A. E., Reidl, M. R., & Beimann, C. B. (2017). Document-based Recommender System for Job Postings using Dense Representations. Document-Based Recommender System for Job Postings Using Dense Representations, 216–224. https://aclweb.org/anthology/N18-3027.pdf
3. 2016
   Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, Karin Pasini, Roberto Pasolini 10.5220/0005702302700277
4. Hoffman, T. H. (n.d.). Learning the Similarity of Documents: An Information-Geometric Approach to Document Retrieval and Categorization. Https://Proceedings.Neurips.Cc/. https://proceedings.neurips.cc/paper/1999/file/9d2682367c3935defcb1f9e247a97c0d-Paper.pdf
5. Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. 2002. Methods and metrics for cold-start recommendations. In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’02, pages 253–260, Tampere, Finland