# CS168 Spring Assignment 5

SUNet ID(s): 05794739
Name(s): Luis A. Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

## Part 1

(a) See Appendix for code.

(b) See Figure 1 for the plot. From the plot we can see that the singular values decay relatively quickly. This implies that there exists a fairly low-rank approximation of $M$.

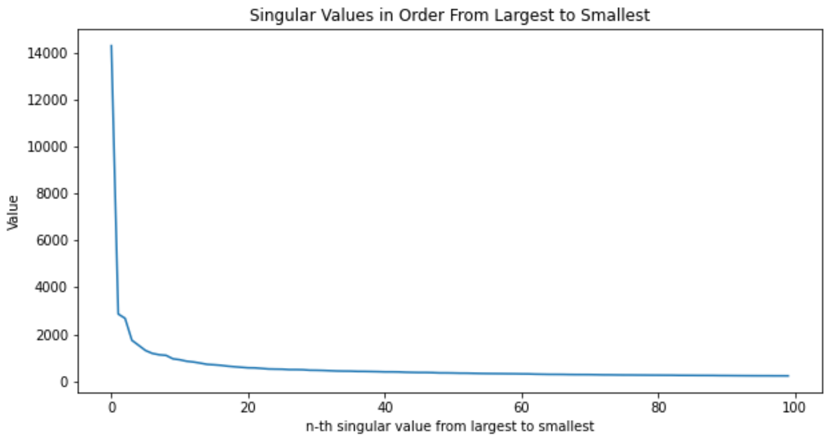

Figure 1: Singular values of $\tilde{M}$ from largest to smallest

(c) A few interesting vectors. We have $v_3$ which gives the following:

```
Top 10 Words
['michael', 'thomas', 'george', 'jr', 'william', 'robert', 'david', 'james', 'john'
Bottom 10 Words
['specific', 'any', 'data', 'provide', 'these', 'certain', 'different', 'systems',
```

This appears to be a vector encoding the concept of a 'name' or 'title' (maybe also person)?

Also $v_6$

```
Top 10 Words
['electronic', 'web', 'online', 'research', 'computer', 'engineering', 'technology'
Bottom 10 Words
['troops', 'him', 'killed', 'soldiers', 'they', 'them', 'were', 'had', 'emperor', '
```

This appears to encode the concept of 'technology'.

Also $v_8$:

```
Top 10 Words
['entered', 'became', 'came', 'moved', 'served', 'joined', 'led', 'brought', 'gave'
Bottom 10 Words
['him', 'them', 'it', 'himself', 'her', 'students', 'who', 'government', 'people',
```

Which appears to encode the concept of 'movement'.

Also $v_{15}$:

```
Top 10 Words
['subsequent', 'scenes', 'combat', 'completed', 'training', 'artillery', 'battle',
Bottom 10 Words
['brazil', 'products', 'food', 'income', 'municipality', 'born', 'population', 'nam
```

This appears to encode the concept of military units.

And lastly, $v_{22}$:

```
Top 10 Words
['institute', 'band', 'records', 'remix', 'and', 'university', 'vol', 'cd', 'album'
Bottom 10 Words
['characters', 'television', 'location', 'character', 'actor', 'channel', 'bbc', 'p
```

Which appears to encode musical storage devices.

The reason why not all vectors have easy to intepret semantics is because they're capturing the directions of highest variability, and as such might capture fairly complex semantic meaning that are difficult to interpret.

(d) i The plot is presented in Figure 2. We observe that the more feminise a word is, the closer the projection gets to 1.0.

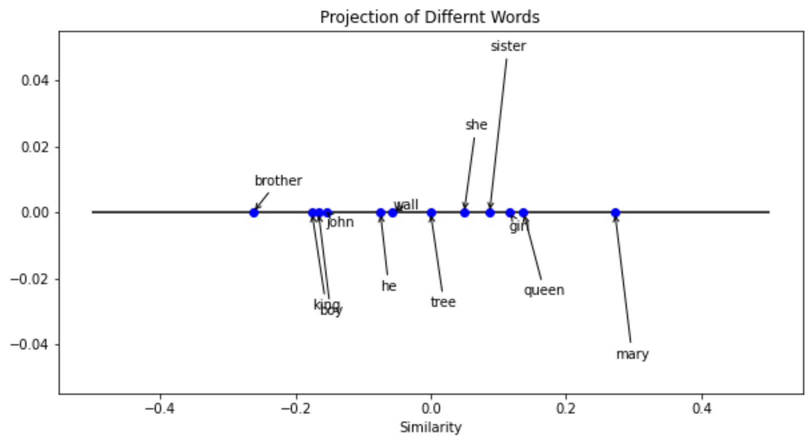

Figure 2: Projection of multiple words onto $v_2 - v_1$ where $v_2$ is the embedding vector corresponding to woman and $v_1$ is the one corresponding to man.

ii The plot is presented in Figure 3. We observe that certain words (which should have little association with femininity) are, and other words are counter associated.

This is likely due to bias in our training data – that is to say, female words appeared frequently with words such as 'nurse' and 'teacher' while far less frequently with words such as 'engineer' and 'science'. This has led to the model capturing these biased relation in its embeddings, which is bad since it is prejudiced against women in certain fields.

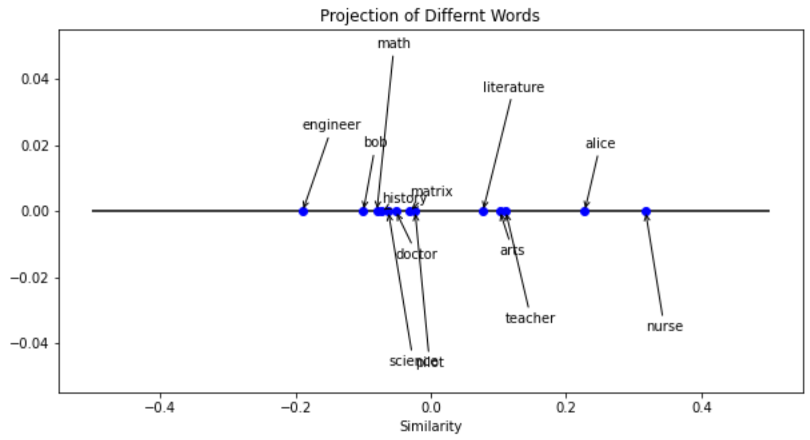

Figure 3: Projection of multiple words onto $v_2 - v_1$ where $v_2$ is the embedding vector corresponding to woman and $v_1$ is the one corresponding to man.

iii The model learns what's in the data – this is no fault of the model. As such, one approach would be to modify the input data so as to remove any gender-related words, or make sure that their co-occurrences with other words are always equal.

(e) i The closest word to 'stanford' is 'harvard'.

ii The accuracy on the analogy task is 31.15487914055506%.

The method got any analogies related to places basically wrong (eg, if it requires factual information, the model is not able to accurately incorporate). It also struggled with analogies related to familial relationships. Lastly, it also appeared to struggle with analogies related to parts of speech (grammar).

## Part 2

(a) The rank-1 approximation of the picture of the moon should be a grey-ish square (darker on the outside, closer to the moon color near the center) surrounded by complete darkness.

The reason for this is we can imagine taking the average of each row of pixels. This becomes are vector which will simply be scaled (either by 0 where it is black) or some non-zero value when crossing the moon area.

(b) The 150-rank approximation is seen in Figure 4.

(c) 1170 would recover the original image in its entirety.

(d) The approximation can be stored in at most 150x(1600 + 1170) = 415,500 bits of information. The original image (even if we don't store the 0 bits) requires 1,649,901 memory units. As such, in the worst case, we're saving 4x the amount of memory.

(e) Removing the haze in its entirety will requires having eigenvectors whose components are precisely 0 (so they aren't scaled up and down). Having these zero values essentially destroys significant amount of information. As such, these vectors aren't utilized until the very end when we compress the image less.

## Appendix

# HW5

May 7, 2020

## 1 CS 168 Spring Assignment 5

SUNet ID(s): 05794739

Name(s): Luis A. Perez

Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

## 2 Imports

[5]:
```
import collections
import matplotlib.pyplot as plt
import scipy

import numpy as np
from PIL import Image
from sklearn import decomposition
import pandas as pd
import seaborn as sns
import os
import warnings

from typing import Dict, List, Text, Tuple

# Make figure larger
plt.rcParams['figure.figsize'] = [10, 5]

# Set numpy seed for consistent results.
np.random.seed(1)
```

## 3 Part 1

### 3.1 Part 1a

[11]:
```
class Globals:
    DATA_PATH = 'data/'

def load_data(normalize=True, sparse=True):
    """Loads M (co-occurrence symmetric matrix) and and mapping from row/column␣
↪idx to word."""
    with open(os.path.join(Globals.DATA_PATH, "co_occur.csv")) as f:
        lines = f.readlines()
    M = np.array([[float(count) for count in line.split(",")] for line in lines␣
↪])
    with open(os.path.join(Globals.DATA_PATH, "dictionary.txt")) as f:
        words = f.readlines()
    idx_to_words = dict(enumerate([word.strip() for word in words]))
    normM = np.log(1 + M) if normalize else M
    return scipy.sparse.csr_matrix(normM), idx_to_words
```

[309]:
```
M, idx_to_words = load_data()
```

### 3.2 Part 1b

[310]:
```
U, S, VT = scipy.sparse.linalg.svds(M, k=100)
```

[311]:
```
def problem1b(M, S):
    plt.title("Singular Values in Order From Largest to Smallest")
    plt.xlabel("n-th singular value from largest to smallest")
    plt.ylabel("Value")
    plt.plot(range(len(S)), sorted(S, reverse=True))
    plt.savefig("figures/singular_values.png", format='png')
    plt.close()
```

[312]:
```
problem1b(M, S)
```

### 3.3 Part 1c

Repeat with different values if $i$.

[313]:
```
def get_most_and_least_frequent_words(i):
    """Returns list of 10 most/least frequent words in eigenvector v_i."""
    vi = U[:, U.shape[1] - i]
    sorted_idx = np.argsort(vi)
    bottom_10 = [idx_to_words[i] for i in sorted_idx[:10]]
    top_10 = [idx_to_words[i] for i in sorted_idx[-10:]]
    print("Top 10 Words")
    print(top_10)
    print("Bottom 10 Words")
    print(bottom_10)
```

[314]:
```
get_most_and_least_frequent_words(3)
```

```
Top 10 Words
['michael', 'thomas', 'george', 'jr', 'william', 'robert', 'david', 'james',
'john', 'born']
Bottom 10 Words
['specific', 'any', 'data', 'provide', 'these', 'certain', 'different',
'systems', 'its', 'use']
```

[218]:
```
correction = np.zeros(x.shape)
correction[3985] = 1
```

[315]:
```
get_most_and_least_frequent_words(6)
```

```
Top 10 Words
['electronic', 'web', 'online', 'research', 'computer', 'engineering',
'technology', 'software', 'science', 'digital']
Bottom 10 Words
['troops', 'him', 'killed', 'soldiers', 'they', 'them', 'were', 'had',
'emperor', 'attacked']
```

[316]:
```
get_most_and_least_frequent_words(8)
```

```
Top 10 Words
['entered', 'became', 'came', 'moved', 'served', 'joined', 'led', 'brought',
'gave', 'took']
Bottom 10 Words
['him', 'them', 'it', 'himself', 'her', 'students', 'who', 'government',
'people', 'players']
```

[317]:
```
get_most_and_least_frequent_words(15)
```

```
Top 10 Words
['subsequent', 'scenes', 'combat', 'completed', 'training', 'artillery',
'battle', 'brief', 'naval', 'squadron']
Bottom 10 Words
['brazil', 'products', 'food', 'income', 'municipality', 'born', 'population',
'name', 'milk', 'province']
```

[318]:
```
get_most_and_least_frequent_words(22)
```

```
Top 10 Words
['institute', 'band', 'records', 'remix', 'and', 'university', 'vol', 'cd',
'album', 'lp']
```

```
Bottom 10 Words
['characters', 'television', 'location', 'character', 'actor', 'channel', 'bbc',
'plot', 'journalist', 'actress']
```

## 3.4 Problem 1d

[319]:
```python
def problem1d(U):
    word_to_idx = {word : idx for idx, word in idx_to_words.items() }
    normU = U / np.linalg.norm(U, axis=1, keepdims=True)
    v1 = normU[word_to_idx["woman"], :]
    v2 = normU[word_to_idx["man"], :]
    v = v1 - v2

    def project(to_project, title):
        # See https://stackoverflow.com/questions/23186804/
↪graph-point-on-straight-line-number-line-in-python
        xmin = -0.5
        xmax = 0.5
        y = 0
        height = 1

        plt.hlines(y, xmin, xmax)

        for i, word in enumerate(to_project):
            x = np.dot(v, normU[word_to_idx[word]])

            plt.plot(x, y, 'bo')

            plt.annotate(word, (x,y), xytext = (x, y + np.random.uniform(low=-0.
↪05, high=0.05)),
                         arrowprops={'arrowstyle': '->'})

        plt.title("Projection of Differnt Words")
        plt.xlabel("Similarity")
        plt.savefig(f'figures/{title}.png', format="png")
        plt.close()

    project(['math', 'matrix', 'history', 'nurse', 'doctor', 'pilot',
             'teacher', 'engineer', 'science', 'arts', 'literature',
             'bob', 'alice'],
           title='projection_ton_femaleness_2')
    project(['boy', 'girl', 'brother', 'sister', 'king', 'queen',
                  'he', 'she', 'john', 'mary', 'wall', 'tree'],
            title='projection_onto_femaleness')
```

[320]:
```python
problem1d(U)
```

## 3.5 Problem 1e

[321]:
```python
def closest_word(query, idx_to_ignore=None):
    query_idx = None
    word_to_idx = {word : idx for idx, word in idx_to_words.items() }
    if type(query) == str:
        query_idx = word_to_idx[query]
        query = U[word_to_idx[query], :]
    sims = np.dot(U, query.T)
    correction = np.zeros(sims.shape)
    if idx_to_ignore:
        for idx in idx_to_ignore:
            correction[idx] = 1
    if query_idx:
        correction[query_idx] = 1
    idx = np.argmax(sims - 10 * np.max(sims) * correction)
    return idx_to_words[idx]

def load_analogy_examples():
    with open(os.path.join(Globals.DATA_PATH, 'analogy_task.txt')) as f:
        tasks = f.readlines()
    return [tuple(word.strip() for word in task.split(' ')) for task in tasks]

def accuracy_on_analogy_task(tasks):
    # Task is 'a' is to 'b' as 'c' is to 'd'
    word_to_idx = {word : idx for idx, word in idx_to_words.items() }
    def _v(x):
        return U[word_to_idx[x], :]
    wrong = []
    for (a, b, c, d) in tasks:
        v1, v2, v3 = _v(a), _v(b), _v(c)
        guess = closest_word(v2 - v1 + v3, idx_to_ignore=[
            word_to_idx[a],
            word_to_idx[b],
            word_to_idx[c]
        ])
        if guess != d:
            wrong.append((a, b, c, d, guess))
    return 1 - len(wrong) / len(tasks), wrong
```

[322]:
```python
def problem1e():
    # Part 1.
    print(f"The closest word to 'stanford' is '{closest_word('stanford')}'.")

    # Part 2.
    acc, wrong = accuracy_on_analogy_task(load_analogy_examples())
    print(f"The accuracy on the analogy task is {100*acc}%.")
    return wrong
```

[323]:
```
wrong_analogies = problem1e()
```

```
The closest word to 'stanford' is 'harvard'.
The accuracy on the analogy task is 31.15487914055506%.
```

## 4 Problem 2

### 4.1 Problem 2b

[70]:
```
def recover_low_rank(image_array, ks):
    max_k = max(ks)
    U, D, VT = scipy.sparse.linalg.svds(image_array, k=max_k)
    D = np.diag(D)
    approx = {}
    for k in ks:
        approx[k] = np.dot(np.dot(U[:, :k], D[:k, :k]), VT[:k, :])
    return approx

def load_image():
    alice = Image.open(os.path.join(Globals.DATA_PATH, 'p5_image.gif'))
    bits = scipy.sparse.csr_matrix(np.asarray(alice).astype(np.float32))
    return bits
```

[71]:
```
def get_image(bits):
    bits = bits - np.min(bits)
    bits = bits / np.max(bits)
    data = (bits * 255).astype(np.uint8)
    image = Image.fromarray(data)
    return image
```

[85]:
```
def problem2bd():
    img = load_image()
    ks = [1,3,10,20,50,100,150 ,200,400, 800, 1700]
    recovered = recover_low_rank(img, ks)
    compressed = get_image(recovered[150])
    compressed.save("figures/low_rank_alice.png", format='PNG')
```

[86]:
```
problem2bd()
```

[ ]:
```

```

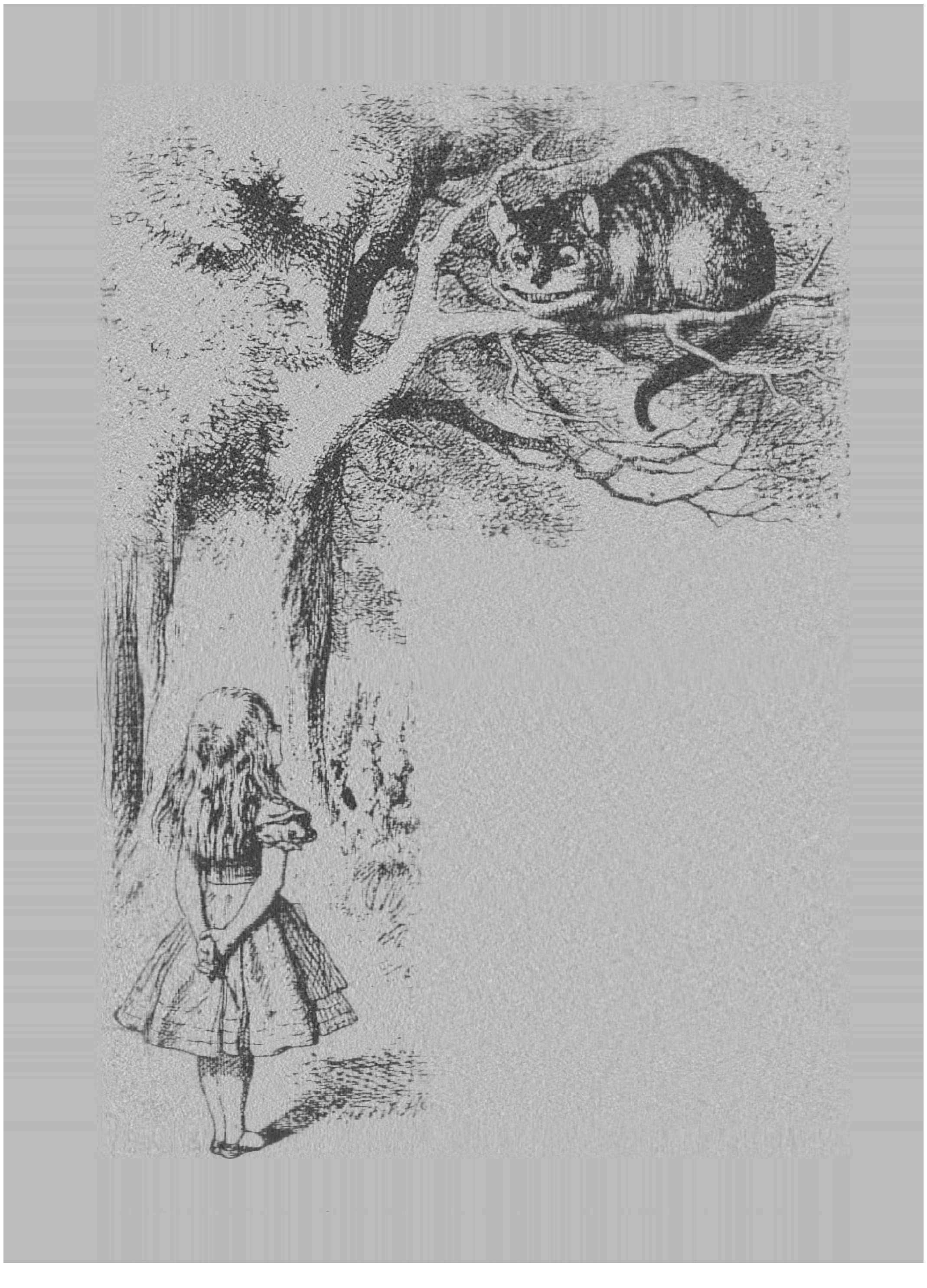

Figure 4: Compressed Image.