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()
```

[ ]:
```
```