

# HW4

April 30, 2020

## 1 CS 168 Spring Assignment 4

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

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

import numpy as np
from scipy import linalg
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

```
[34]: def pca_recover(X, Y):  
      """Returns the slope of the first component of PCA applied to [X, Y].  
  
      Args:  
      X: An (n,1) matrix of points.  
      Y: An (n,1) matrix of points.  
      """  
      data = np.concatenate((X,Y), axis=1)  
      data = data - np.mean(data, axis=0)  
      A = np.dot(data.T, data)  
      # Compute largest eigenvector.  
      _, v = linalg.eigh(A, eigvals=(1,1))  
      v = v.flatten()  
      return v[1] / v[0]  
  
[38]: def ls_recover(X, Y):  
      """Returns the slop of the best fit line as per least squares. See ↵  
      ↪pca_recover()."""  
      centeredX, centeredY = X - np.mean(X, axis=0), Y - np.mean(Y, axis=0)  
      return (np.dot(centeredX.T, centeredY) / np.sum(centeredX**2)).flatten()[0]  
  
[39]: def problem1a():  
      X = np.arange(0.001, 1.001, 0.001)  
      X = np.reshape(X, (X.shape[0], 1))  
      Y = 2*X  
      return pca_recover(X,Y), ls_recover(X, Y)  
  
[40]: problem1a()  
  
[40]: (2.0, 2.0)  
  
[105]: def problem1b():  
      pca, ls = [], []  
      for _ in range(100000):  
          X = np.random.uniform(size=(1000, 1))  
          Y = np.random.uniform(size=(1000, 1))  
          pca.append(pca_recover(X,Y))  
          ls.append(ls_recover(X, Y))  
      return np.mean(pca), np.mean(ls), np.std(pca), np.std(ls)  
  
[106]: problem1b()
```

```
[106]: (-0.2550136867924121,  
        3.252073550279325e-05,  
        142.44712135685342,  
        0.03173951330042009)
```

```
[277]: def problem1cd(title, add_x_noise):  
        X = np.arange(0.001, 1.001, 0.001)  
        X = np.reshape(X, (X.shape[0], 1))  
        cs = np.arange(0, .55, 0.05)  
        ls = []  
        pca = []  
        for _ in range(30):  
            for c in cs:  
                if add_x_noise:  
                    noisyX = X + np.random.normal(scale=np.sqrt(c), size=X.shape)  
                else:  
                    noisyX = X  
                Y = 2*X + np.random.normal(scale=np.sqrt(c), size=X.shape)  
                pca.append((c, pca_recover(noisyX, Y)))  
                ls.append((c, ls_recover(noisyX, Y)))  
  
        x, y = zip(*ls)  
        plt.scatter(x, y, label="Least Squares", color="b")  
        x, y = zip(*pca)  
        plt.scatter(x, y, label="PCA", color="r")  
        plt.title("Best Fit Line for Noisy Measurements")  
        plt.xlabel("Variance (c) of noise")  
        plt.ylabel("Least Squares/PCA Slope of Best Fit Line")  
        plt.legend()  
        plt.savefig(f"figures/{title}.png", format="png")  
        plt.close()
```

```
[278]: problem1cd("problem1c", add_x_noise=False)
```

```
[279]: problem1cd("problem1d", add_x_noise=True)
```

## 4 Problem 2

```
[352]: class Globals:  
        DATA_FOLDER = "data"  
  
        def get_genomic_matrix_data(use_mode=True):  
            """Returns the binary X matrix as defined in the problem statement."""  
            datapath = os.path.join(Globals.DATA_FOLDER, "p4dataset2020.txt")  
            with open(datapath) as f:  
                lines = f.readlines()
```

```

assert len(lines) == 995
cleaned = [[x.strip() for x in line.split(" ")]
            for line in lines]
pop = np.array([line[2] for line in cleaned])
gender = np.array([int(line[1]) for line in cleaned])
ids = np.array([line[0] for line in cleaned])

if use_mode:
    mapping = {'A': 0, 'C': 1, 'T': 2, 'G': 4}
    data = np.array([[mapping[x] for x in line[3:]]
                    for line in cleaned])

    # Find mode.
    mode = scipy.stats.mode(data, axis=0).mode
    X = (data != mode).astype(int)
else:
    mapping = {'A': [0, 0, 0, 1], 'C': [0, 0, 1, 0], 'T': [0, 1, 0, 0], 'G':
→ [1, 0, 0, 0]}
    data = np.array([[mapping[x] for x in line[3:]]
                    for line in cleaned])
    X = data.reshape((data.shape[0], -1))
return X, pop, gender, ids

```

```

[325]: def problem2b(use_mode):
    X, pop, _, _ = get_genomic_matrix_data(use_mode)
    X = X - np.mean(X, axis=0)
    pca_model = decomposition.PCA(n_components=2)
    pca_model = pca_model.fit(X)
    v1, v2 = pca_model.components_
    v1, v2 = v1 / np.linalg.norm(v1), v2 / np.linalg.norm(v2)
    xs, ys = np.dot(X, v1).flatten(), np.dot(X, v2).flatten()
    # xs, ys = projections[:,0], projections[:,1]
    label_map = {
        'ACB': 'African Caribbeans in Barbados',
        'ASW': 'Americans of African Ancestry in SW USA',
        'ESN': 'Esan in Nigeria',
        'GWD': 'Gambian in Western Divisions in the Gambia',
        'LWK': 'Luhya in Webuye, Kenya',
        'MSL': 'Mende in Sierra Leone',
        'YRI': 'Yoruba in Ibadan, Nigeria',
    }
    for label in np.unique(pop):
        idx = (pop == label).nonzero()
        plt.scatter(xs[idx], ys[idx], label=label_map[label])
    plt.legend()
    plt.title("Genomic sequence data project into 2 principal components")
    plt.xlabel("v1")
    plt.ylabel("v2")

```

```
plt.savefig(f"figures/genomic_2d_projection_use_mode={use_mode}.png",
↪format="png")
plt.close()
```

```
[394]: problem2b(use_mode=True)
```

```
[326]: def problem2d():
    X, pop, gender, _ = get_genomic_matrix_data()
    X = X - np.mean(X, axis=0)
    pca_model = decomposition.PCA(n_components=3)
    pca_model = pca_model.fit(X)
    v1, _, v3 = pca_model.components_
    v1, v3 = v1 / np.linalg.norm(v1), v3 / np.linalg.norm(v3)
    xs, ys = np.dot(X, v1).flatten(), np.dot(X, v3).flatten()
    # xs, ys = projections[:,0], projections[:,1]
    label_map = {
        1: 'Male',
        2: 'Female'
    }
    for label in np.unique(gender):
        idx = (gender == label).nonzero()
        plt.scatter(xs[idx], ys[idx], label=label_map[label])
    plt.legend()
    plt.title("Genomic sequence data project into 1st and 3d principal
↪components")
    plt.xlabel("v1")
    plt.ylabel("v3")
    plt.savefig("figures/genomic_2d_projection_2.png", format="png")
    plt.close()
```

```
[252]: problem2d()
```

```
[327]: def problem2f():
    X, pop, gender, _ = get_genomic_matrix_data()
    X = X - np.mean(X, axis=0)
    pca_model = decomposition.PCA(n_components=3)
    pca_model = pca_model.fit(X)
    _, _, v3 = pca_model.components_
    v3 = v3.flatten() / np.linalg.norm(v3)

    plt.title("Magnitude of 3rd Principal Component")
    plt.xlabel('Nucleobase Index')
    plt.ylabel('Absolute Value')
    plt.plot(range(len(v3)), np.abs(v3))
    plt.savefig("figures/x_y_chromosomes.png", format="png")
    plt.close()
```

```
[270]: problem2f()
```

```
[314]: def problem2h():  
        problem2b(use_mode=False)
```

```
[315]: problem2h()
```

```
[395]: def read_metadata():  
        """Returns a dictionary of metadata, M.  
  
        M[property][I_ID] returns the corresponding property of I_ID.  
        """  
        metadata = pd.read_csv(os.path.join(Globals.DATA_FOLDER, "20130606_g1k.  
↪ped"),  
                                header=0, sep='\t').set_index('Individual ID')  
        metadata = metadata.to_dict()  
        return metadata  
  
def problem2i(color_label):  
    meta = read_metadata()  
    X, _, _, ids = get_genomic_matrix_data(use_mode=False)  
    X = X - np.mean(X, axis=0)  
    pca_model = decomposition.PCA(n_components=4)  
    pca_model = pca_model.fit(X)  
    _, _, v3, v4 = pca_model.components_  
    xs, ys = np.dot(X, v3).flatten(), np.dot(X, v4).flatten()  
    # xs, ys = projections[:,0], projections[:,1]  
  
    labels = meta[color_label]  
    data_labels = np.array([labels[_id] for _id in ids])  
    for label in np.unique(data_labels):  
        idx = (data_labels == label).nonzero()  
        plt.scatter(xs[idx], ys[idx], label=label)  
    plt.legend()  
    plt.title("Genomic sequence data project into 1st and 3d principal_  
↪components")  
    plt.xlabel("v1")  
    plt.ylabel("v4")  
    plt.savefig("figures/genomic_2i_projection_4th.png", format="png")  
    plt.close()
```

```
[396]: problem2i(color_label='Relationship')
```