1. (a)

I did this homework with Luning Zhao.

We work on our homework seperately, but we discuss when meeting problems.

Homework is fine.

(b) I certify that all solutions are entirely in my words. and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Siyao Jia

2.(a) $A = U \Sigma V^T$

$$= [\vec{u}_1 \cdots \vec{u}_n] \begin{bmatrix} \sigma_1 & & \\ & \ddots \sigma_d & \\ & 0 & \end{bmatrix} \begin{bmatrix} \vec{v}_1^T \\ \vdots \\ \vec{v}_d^T \end{bmatrix}$$

$$= [\sigma_1 \vec{u}_1 \quad \sigma_2 \vec{u}_2 \quad \cdots \quad \sigma_d \vec{u}_d] \begin{bmatrix} \vec{v}_1^T \\ \vdots \\ \vec{v}_d^T \end{bmatrix}$$

$$= \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_d \vec{u}_d \vec{v}_d^T$$

$$= \sum_{i=1}^{d} \sigma_i \vec{u}_i \vec{v}_i^T$$

(b) i) $A^T A = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T = \sum_i^d \sigma_i^2 \vec{v}_i \vec{v}_i^T$

$A^T A \vec{v}_j = \sum_{i=1}^{d} \sigma_i^2 \vec{v}_i \vec{v}_i^T \vec{v}_j = \sigma_j^2 \vec{v}_j$

So $\vec{v}_i$ is an eigenvector for $A^T A$, with corresponding eigenvalue $\sigma_i^2$.

ii) $A A^T = U \Sigma V^T V \Sigma U^T = U \Sigma^2 U^T$

Similarly, $\vec{u}_i$ is an eigenvector for $A A^T$, with corresponding eigenvalue $\sigma_i^2$.

(c) $\vec{u}$ can be written as $\vec{u} = \sum_{i=1}^{n} a_i \vec{u}_i$, $\vec{v}$ can be written as $\vec{v} = \sum_{i=1}^{d} b_i \vec{v}_i$

$u^T A v = \sum_{i=1}^{d} \sigma_i a_i b_i \vec{u}_i^T \vec{u}_i \vec{v}_i^T \vec{v}_i = \sum_{i=1}^{d} \sigma_i a_i b_i \leq \sigma_1 \sum_{i=1}^{d} a_i b_i$

$\|\vec{u}\| = \vec{u}^T \vec{u} = \sum_{i=1}^{n} a_i^2 = 1 \qquad \|\vec{v}\| = \sum_{i=1}^{d} b_i^2 = 1$

Let $\vec{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_d \end{bmatrix} \quad \vec{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_d \end{bmatrix}$ $\sum_{i=1}^{d} a_i b_i = \vec{a}^T \vec{b} \leq \|\vec{a}\| \|\vec{b}\| \leq \|\vec{u}\| \|\vec{v}\| = 1$

so $u^T A v \leq \sigma_1$, when $\vec{u} = \vec{u}_1$ $\vec{v} = \vec{v}_1$, $u^T A v = \sigma_1$, $(u^T A v)_{max} = \sigma_1(A)$

(d) $\rho(a^Tx, b^Ty) = \dfrac{Cov(a^Tx, b^Ty)}{\sqrt{Var(a^Tx)\,Var(b^Ty)}}$

$$Cov(a^Tx, b^Ty) = E[(a^Tx - E[a^Tx])(b^Ty - E[b^Ty])]$$
$$= E[a^T(x - E[x])(y - E[y])^Tb]$$
$$= a^T E[xy^T] b$$
$$= a^T \Sigma_{xy} b$$

$$Var(a^Tx) = E[(a^Tx - E[a^Tx])(a^Tx - E[a^Tx])]$$
$$= a^T \Sigma_{xx} a$$

$$Var(b^Ty) = b^T \Sigma_{yy} b$$

So $\rho = \max_{a,b} \dfrac{a^T \Sigma_{xy} b}{(a^T \Sigma_{xx} a)^{\frac{1}{2}} (b^T \Sigma_{yy} b)^{\frac{1}{2}}}$

$$\rho(\alpha a^*, \beta b^*) = \frac{\alpha a^{*T} \Sigma_{xy} b^* \beta}{(\alpha a^{*T} \Sigma_{xx} a^* \alpha)^{\frac{1}{2}} (\beta b^{*T} \Sigma_{yy} b \beta)^{\frac{1}{2}}} = \rho(a^*, \beta^*)$$

So if $(a^*, b^*)$ is a maximizer, then $(\alpha a^*, \beta b^*)$ is also a maximizer.

(e) $\rho = \max_{a,b} \dfrac{a^T E[XY^T] b}{\sqrt{a^T E(XX^T) a \;\; b^T E(YY^T) b}} = \max_{a,b} \dfrac{E[a^Tx, (b^Ty)^T]}{\sqrt{E[a^Tx(a^Tx)^T]\; E[b^TY(b^TY)^T]}}$

Let $a^Tx = c^T\tilde{x} = c^T \Sigma_{xx}^{-\frac{1}{2}} x \qquad a = \Sigma_{xx}^{-\frac{1}{2}} c$

$b^Ty = d^T\tilde{y} = d^T \Sigma_{yy}^{-\frac{1}{2}} y \qquad b = \Sigma_{yy}^{-\frac{1}{2}} d$

$$\rho = \max_{c,d} \frac{c^T \Sigma_{xx}^{-\frac{1}{2}} E[XY^T] \Sigma_{yy}^{-\frac{1}{2}} d}{\sqrt{E[c^T\tilde{x}\tilde{x}^Tc]}\sqrt{E[d^T\tilde{y}\tilde{y}^Td]}}$$

$$= \max_{c,d} \frac{c^T \Sigma_{xx}^{-\frac{1}{2}} E[XY]^T \Sigma_{yy}^{-\frac{1}{2}} d}{\|c\| \cdot \|d\|}$$

$$= \max_{\substack{\|c\|^2=1 \\ \|d\|^2=1}} c^T \Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}} d$$

(f) from (c), we know $\rho = \sigma_1(\underbrace{\Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}}}_{A})$ $\sigma_1$ represents maximum singular value

from (b), we know $\rho^2 = \sigma_1^2(A)$ = maximum eigenvalue of $AA^T$

$AA^T = \Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}} \Sigma_{yy}^{-\frac{1}{2}} \Sigma_{xy}^T \Sigma_{xx}^{-\frac{1}{2}} = \Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T \Sigma_{xx}^{-\frac{1}{2}}$

So $\rho^2$ is the maximum eval of $\Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T \Sigma_{xx}^{-\frac{1}{2}}$

(g) from (c), we know $c^*, d^*$ that maximize $c^T \Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}} d$ is the

$\|c\|^2=1$, $\|d\|^2=1$

first left and right singular vector of $\Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}}$

from (b) we know $c^*$ is also an eigenvector of $AA^T$ with the max eval.

So we prove that $c^*$ is an evec toresponding to the max eval of $\Sigma_{xx}^{-\frac{1}{2}} \Sigma_{xy} \Sigma_{yy}^{-1} \Sigma_{xy}^T \Sigma_{xx}^{-\frac{1}{2}}$.

(h) Similar as (g), $d^*$ is an evec of $A^TA$, so $d^*$ is an evec corresponding to the max eval of $\Sigma_{yy}^{-\frac{1}{2}} \Sigma_{xy}^T \Sigma_{xx}^{-1} \Sigma_{xy} \Sigma_{yy}^{-\frac{1}{2}}$.

(i) because whe $\Sigma_{xy} = 0$, $A=0$, so $\rho=0$ whatever c and d.

(j) First we should calculate $y^2$, then use CCA to find linear relationship between X and $y^2$. Then when evaluating X-test, we use same projection to find $y\text{-test}^2$, The square root of $y^2_{test}$ will be y-test

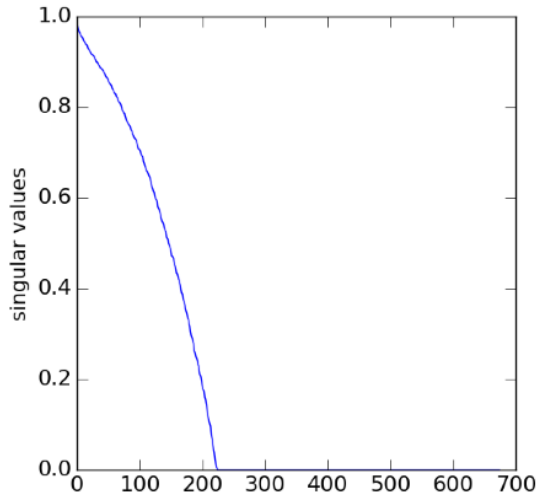
(k) Because CCA will help us find the most effective feature space that correlates X and y. This process of finding most effective feature space is a machine learning process.

3. (a) $\Sigma_{xx} = E[XX^T] = \frac{1}{n}\sum_i (x_i - \bar{x})(x_i - \bar{x})^T$

$\Sigma_{yy} = E[YY^T] = \frac{1}{n}\sum_i (y_i - \bar{y})(y_i - \bar{y})^T$

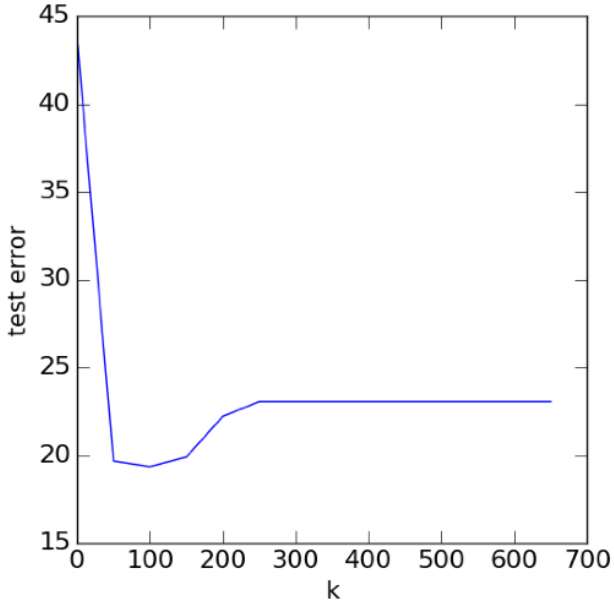
$\Sigma_{xy} = E[XY^T] = \frac{1}{n}\sum_i (x_i - \bar{x})(y_i - \bar{y})^T$

2. (b)

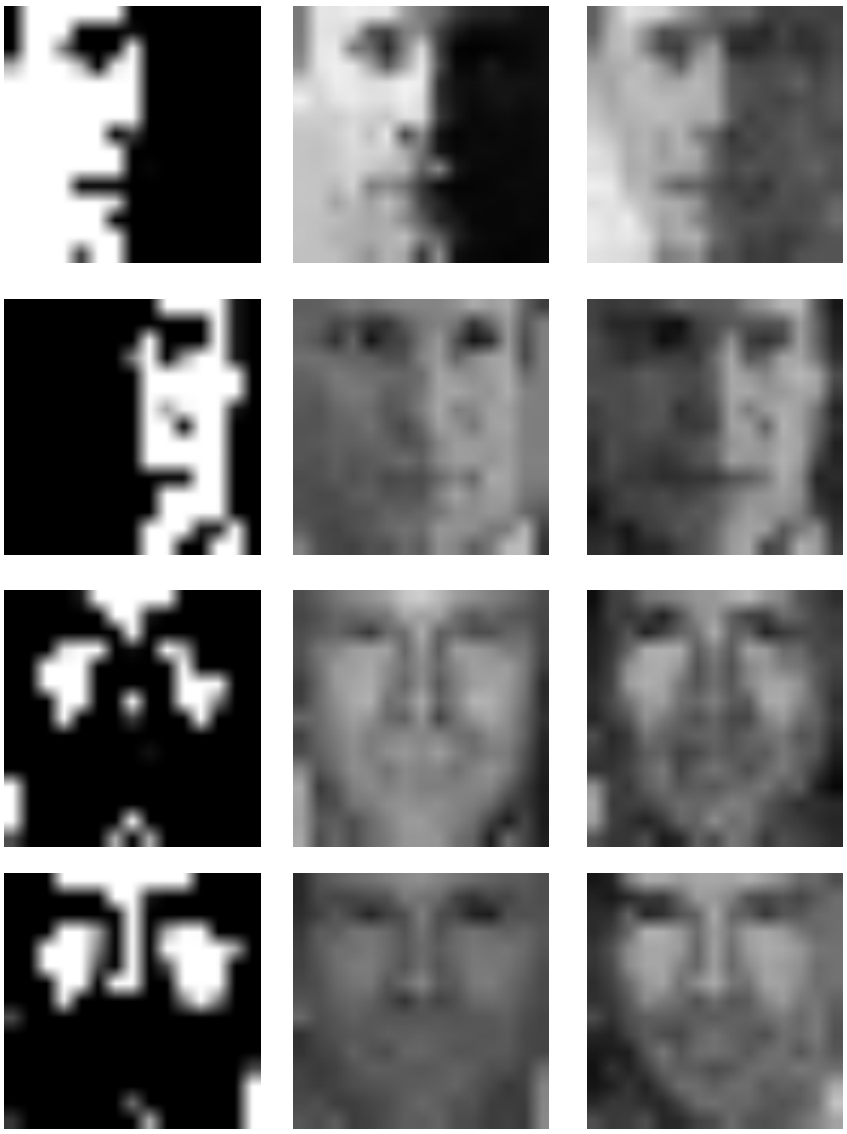



(c)

(d)




(e)



```
import os
import numpy as np
import cv2
import copy
import glob

import sys

from numpy.random import uniform

import pickle
from scipy.linalg import eig
from scipy.linalg import sqrtm
from numpy.linalg import inv
from numpy.linalg import svd
import numpy.linalg as LA
import matplotlib.pyplot as plt
import IPython
from sklearn.preprocessing import StandardScaler
import pdb


def standardized(v):
    return (v/255.0) * 2.0 - 1.0

def flatten_and_standardize(data):
    result = []
    for d in data:
      d = d.flatten()
      d = standardized(d)
      result.append(d)
    return result

class Mooney(object):

    def __init__(self):
        self.lmbda = 1e-5

    def load_data(self):
        self.x_train = pickle.load(open('x_train.p','rb'))
        self.y_train = pickle.load(open('y_train.p','rb'))
        self.x_test = pickle.load(open('x_test.p','rb'))
        self.y_test = pickle.load(open('y_test.p','rb'))

    def compute_covariance_matrices(self):
        # USE STANDARD SCALAR TO DO MEAN SUBTRACTION
        ss_x = StandardScaler(with_std = False)
        ss_y = StandardScaler(with_std = False)

        num_data = len(self.x_train)

        x = self.x_train[0]
        y = self.y_train[0]

        x_f = x.flatten()
        y_f = y.flatten()

        x_f_dim = x_f.shape[0]
        y_f_dim = y_f.shape[0]

        self.x_dim = x_f_dim
        self.y_dim = y_f_dim

        self.C_xx = np.zeros([x_f_dim,x_f_dim])
        self.C_yy = np.zeros([y_f_dim,y_f_dim])
        self.C_xy = np.zeros([x_f_dim,y_f_dim])

        x_data = []
        y_data = []

        for i in range(num_data):
            x_image = self.x_train[i]
            y_image = self.y_train[i]
```

```
            # FLATTEN DATA
            x_f = x_image.flatten()
            y_f = y_image.flatten()

            # STANDARDIZE DATA
            x_f = standardized(x_f)
            y_f = standardized(y_f)

            x_data.append(x_f)
            y_data.append(y_f)

        # SUBTRACT MEAN
        ss_x.fit(x_data)
        x_data = ss_x.transform(x_data)

        ss_y.fit(y_data)
        y_data = ss_y.transform(y_data)

        for i in range(num_data):
            x_f = np.array([x_data[i]])
            y_f = np.array([y_data[i]])
            # TODO: COMPUTE COVARIANCE MATRICES
            self.C_xx += x_f.T @ x_f
            self.C_yy += y_f.T @ y_f
            self.C_xy += x_f.T @ y_f

        # DIVIDE BY THE NUMBER OF DATA POINTS
        self.C_xx = 1.0/float(num_data)*self.C_xx
        self.C_yy = 1.0/float(num_data)*self.C_yy
        self.C_xy = 1.0/float(num_data)*self.C_xy


    def compute_projected_data_matrix(self,X_proj):

        Y = []
        X = []

        Y_test = []
        X_test = []

        num_data = len(self.x_train)

        # LOAD TRAINING DATA
        for x in self.x_train:
            x_f = np.array([x.flatten()])
            # STANDARDIZE DATA
            x_f = standardized(x_f)
            # TODO: PROJECT DATA
            X.append((x_f @ X_proj)[0])

        Y = flatten_and_standardize(self.y_train)

        for x in self.x_test:
            x_f = np.array([x.flatten()])
            # STANDARDIZE DATA
            x_f = standardized(x_f)
            # TODO: PROJECT DATA
            X_test.append((x_f @ X_proj)[0])

        Y_test = flatten_and_standardize(self.y_test)

        # CONVERT TO MATRIX
        self.X_ridge = np.vstack(X)
        self.Y_ridge = np.vstack(Y)

        self.X_test_ridge = np.vstack(X_test)
        self.X_test_ridge = X_test
        self.Y_test_ridge = np.vstack(Y_test)

        return X

    def compute_data_matrix(self):
```

```
        X = flatten_and_standardize(self.x_train)
        Y = flatten_and_standardize(self.y_train)
        X_test = flatten_and_standardize(self.x_test)
        Y_test = flatten_and_standardize(self.y_test)

        # CONVERT TO MATRIX
        self.X_ridge = np.vstack(X)
        self.Y_ridge = np.vstack(Y)

        self.X_test_ridge = np.vstack(X_test)
        self.Y_test_ridge = np.vstack(Y_test)


    def solve_for_variance(self):
        eigen_values = np.zeros((675,))
        eigen_vectors = np.zeros((675, 675))

        # compute C_xx_inv_sqrt  and C_yy_inv_sqrt
        self.C_xx_inv_sqrt = inv(sqrtm(self.C_xx + 0.00001 * np.identity(len(self.C_
xx))))
        self.C_yy_inv_sqrt = inv(sqrtm(self.C_yy + 0.00001 * np.identity(len(self.C_
yy))))

        # plot sigular value of the matrix
        temp = self.C_xx_inv_sqrt @ self.C_xy @ self.C_yy_inv_sqrt
        u, s, v = svd(temp)
        plt.clf()
        plt.plot(range(len(s)), s)
        plt.ylabel('singular values')
        plt.savefig('b.png',format='png')

        # TODO: COMPUTE CORRELATION MATRIX
        evals, evecs = eig(temp @ temp.T)
        return evals, evecs

    def project_data(self, eig_val,eig_vec,proj=150):
        # TODO: COMPUTE PROJECTION SINGULAR VECTORS
        return eig_vec.T[0:proj].T

    def ridge_regression(self):
        # TODO: IMPLEMENT RIDGE REGRESSION
        w_ridge = inv(self.X_ridge.T @ self.X_ridge + 0.00001 * np.identity(len(self
.X_ridge[0]))) @ self.X_ridge.T @ self.Y_ridge

        self.w_ridge = w_ridge


    def plot_image(self, vector):
        vector = ((vector+1.0)/2.0)*255.0
        vector = np.reshape(vector,(15,15,3))
        p = vector.astype("uint8")
        p = cv2.resize(p,(100,100))
        count = 0

        cv2.imwrite('a_face_'+str(count)+'.png',p)

    def measure_error(self, X_ridge, Y_ridge):
        prediction = X_ridge @ self.w_ridge
        evaluation = Y_ridge - prediction

        #print(evaluation)

        dim,num_data = evaluation.shape

        error = []

        for i in range(num_data):
          # COMPUTE L2 NORM for each vector then square
          error.append(LA.norm(evaluation[:,i])**2)

        # Return average error
        return np.mean(error)
```

```
    def draw_images(self):
        for count, x in enumerate(self.X_test_ridge):
            if ( count < 5):
                prediction = np.matmul(self.w_ridge.T,x)
                prediction = ((prediction+1.0)/2.0)*255.0
                prediction = np.reshape(prediction,(15,15,3))
                p = prediction.astype("uint8")
                p = cv2.resize(p,(100,100))
                cv2.imwrite('face_'+str(count)+'.png',p)

        for count, x in enumerate(self.x_test):
            if ( count < 5 ):
                x = x.astype("uint8")
                x = cv2.resize(x,(100,100))
                cv2.imwrite('og_face_'+str(count)+'.png',x)

        for count, x in enumerate(self.y_test):
            if ( count < 5 ):
                x = x.astype("uint8")
                x = cv2.resize(x,(100,100))
                cv2.imwrite('gt_face_'+str(count)+'.png',x)

if __name__ == '__main__':

    mooney = Mooney()


    # (b) #
    mooney.load_data()
    mooney.compute_covariance_matrices()
    eig_val, eig_vec = mooney.solve_for_variance()

    # (c) #
    # project face to the first singular vector
    X_proj = mooney.project_data(eig_val, eig_vec, 1)
    X_p = mooney.compute_projected_data_matrix(X_proj)
    mooney.plot_image(X_proj)

    # (d)  & (e) #
    proj = [1,50,100,150,200,250,300,350,400,450,500,650]
    #proj = [100]
    error_test = []
    for p in proj:

        X_proj = mooney.project_data(eig_val,eig_vec,proj=p)

        # COMPUTE REGRESSION
        mooney.compute_projected_data_matrix(X_proj)
        mooney.ridge_regression()
        training_error = mooney.measure_error(mooney.X_ridge, mooney.Y_ridge)
        test_error = mooney.measure_error(mooney.X_test_ridge, mooney.Y_test_ridge)
        #mooney.draw_images()

        error_test.append(test_error)

    plt.clf()
    plt.plot(proj,error_test)
    plt.xlabel('k')
    plt.ylabel('test error')
    plt.savefig('d.png', format='png')

    # COMPUTE REGRESSION NO PROJECT
    mooney.compute_data_matrix()
    mooney.ridge_regression()
    mooney.draw_images()
    training_error = mooney.measure_error(mooney.X_ridge, mooney.Y_ridge)
    test_error = mooney.measure_error(mooney.X_test_ridge, mooney.Y_test_ridge)
```

4. (a) $E(\hat{w}_\lambda) = \frac{E[\sum_{i=1}^{n} x_i y_i]}{S_x^2+\lambda} = \frac{\sum_{i=1}^{n} x_i E[y_i]}{S_x^2+\lambda} = \frac{\sum_{i=1}^{n} x_i x_i w^*}{S_x^2+\lambda} = \frac{w^* S_x^2}{S_x^2+\lambda}$

$Bias^2(\hat{w}_\lambda) = (E[\hat{w}_\lambda] - w^*)^2 = \left(\frac{\lambda w^*}{S_x^2+\lambda}\right)^2$

(b) $Var(\hat{w}_\lambda) = E\left[\left(\hat{w}_\lambda - \frac{w^* S_x^2}{S_x^2+\lambda}\right)^2\right]$

$= E\left[\left(\frac{\sum x_i y_i - w^* \sum x_i x_i}{S_x^2+\lambda}\right)^2\right]$

$= E\left[\left(\frac{\sum x_i (y_i - x_i w^*)}{S_x^2+\lambda}\right)^2\right]$

$= E\left[\left(\frac{\sum x_i z_i}{S_x^2+\lambda}\right)^2\right]$

$= \frac{1}{(S_x^2+\lambda)^2} E[\sum (x_i z_i)]$

$= \frac{\sum x_i^2 E(z_i)^2}{(S_x^2+\lambda)^2}$

$= \frac{S_x^2}{(S_x^2+\lambda)^2}$

(c) When $\lambda \uparrow$, $Bias^2 \uparrow$, Variarve $\downarrow$

when $\lambda \downarrow$, bias $\downarrow$, varianre $\uparrow$

When $\lambda \to 0$, Bias $\to 0$, bias $\downarrow$, varience $\to 1$, varianre $\uparrow$.

When $\lambda \to \infty$, bias $\to w^*$, bias $\uparrow$, Varianre $\to 0$, Varianre $\downarrow$.

When bias = variarnre, $\left(\frac{\lambda w^*}{S_x^2+\lambda}\right)^2 = \frac{S_x^2}{(S_x^2+\lambda)^2}$, $\lambda = \frac{\sqrt{S_x^2}}{w^*}$, there is a

Variarnre / bias tradeoff.

5. (a) $P(\hat{w}|D) \propto P(D|\hat{w}) \cdot P(\hat{w})$

$P(D|\hat{w}) \propto \prod_i e^{-\frac{(y_i - x_i^T \hat{w})^2}{2}} \propto e^{-\frac{\sum \|y_i - x_i^T w\|^2}{2}} = e^{-\frac{\|y - Xw\|^2}{2}}$

$P(\hat{w}) \propto e^{-\frac{1}{2}(\hat{w} - \hat{w}_B)^T \Psi^{-1} (\hat{w} - \hat{w}_B)}$

$\log(P(\hat{w}|D)) = \min_{\hat{w}} \|y - Xw\|^2 + (w - w_B)^T \Psi^{-1} (w - w_B)$

$= \min_{w} \|y - Xw\|^2 + (w - w_B)^T \Psi^{-\frac{1}{2}} \Psi^{-\frac{1}{2}} (w - w_B)$

Let $v = \Psi^{-\frac{1}{2}} (w - w_B)$ $\quad w = \Psi^{\frac{1}{2}} v + w_B$

$y - Xw = y - X\Psi^{\frac{1}{2}} v - X w_B$

Let $\tilde{y} = y - X w_B$ $\quad \tilde{x} = X \Psi^{\frac{1}{2}}$

Our MAP problem becomes: $\min_v \|\tilde{y} - \tilde{x} v\|^2 + \|v\|^2$

$v = (\tilde{x}^T \tilde{x} + I)^{-1} \tilde{x}^T \tilde{y}$

$w = \Psi^{\frac{1}{2}} (\Psi^{\frac{1}{2}} X^T X \Psi^{\frac{1}{2}} + I)^{-1} \Psi^{\frac{1}{2}} X^T (y - X w_B) + w_B$

$= w_B + (X^T X + \Psi^{-1})^{-1} X^T (y - X w_B)$

(b) Assume the uncertainty from hospital B is $\sigma$.

$\log P(w|D) = \min_w \|y - Xw\|^2 + \frac{1}{\sigma^2} \|w - w_B\|^2$

Let $\tilde{w} = w - w_B$ $\quad \tilde{y} = y - X w_B$ $\quad \frac{1}{\sigma^2} = \lambda$

$\hat{w} = (X^T X + \lambda I)^{-1} X^T (y - X w_B) + w_B$

So $\lambda$ here is a hyperparameter, $\hat{w}$ can be estimated in this way similary to ridge regression

6. (a) $V \hat{W}_{ridge} = V \, diag\left(\frac{\sigma_i}{\lambda+\sigma_i^2}\right) U^T y$

$$= \sum_{i=1}^{d} \frac{\sigma_i}{\lambda+\sigma_i^2} \vec{v}_i \vec{u}_i^T y$$

$$\hat{y}_{test} = X_{test}^T V \hat{W}_{ridge}$$

$$= X_{test}^T \sum_{i=1}^{d} \frac{\sigma_i}{\lambda+\sigma_i^2} \vec{v}_i \vec{u}_i^T y$$

Let $\beta_i = \frac{\sigma_i}{\lambda+\sigma_i^2}$, $\hat{y}_{test}$ can be written as $\hat{y}_{test} = X_{test}^T \sum_{i=1}^{d} \vec{v}_i \beta_i \vec{u}_i^T \vec{y}$

(b) $SVD(X^TX) = SVD(V\Sigma^T U^T U \Sigma V^T) = V\Sigma^T\Sigma V^T = \sum_{i=1}^{d} \sigma_i^2 \vec{v}_i \vec{v}_i^T$

$V_k = [\vec{v}_1 \cdots \vec{v}_k]$

$X_k = X V_k = U\Sigma V^T V_k = \sum_{i=1}^{n} \sigma_i \vec{u}_i \vec{v}_i^T [\vec{v}_1 \cdots \vec{v}_k] = [\sigma_1 \vec{u}_1 \cdots \sigma_k \vec{u}_k]$

$w = (X_k^T X_k)^{-1} X_k^T y$

$$X_k^T X_k = \begin{bmatrix} \sigma_1 \vec{u}_1 \\ \vdots \\ \sigma_k \vec{u}_k \end{bmatrix} [\sigma_1 \vec{u}_1 \cdots \sigma_k \vec{u}_k] = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_k^2 \end{bmatrix}$$

$$(X_k^T X_k)^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2} & & \\ & \ddots & \\ & & \frac{1}{\sigma_k^2} \end{bmatrix}$$

$$\hat{y}_{test} = X_{test}^T V_k w = X_{test}^T [\vec{v}_1 \cdots \vec{v}_k] \begin{bmatrix} \frac{1}{\sigma_1^2} & & \\ & \ddots & \\ & & \frac{1}{\sigma_k^2} \end{bmatrix} \begin{bmatrix} \sigma_1 \vec{u}_1 \\ \vdots \\ \sigma_k \vec{u}_k \end{bmatrix} y$$

$$= X_{test}^T \sum_{i=1}^{k} \frac{1}{\sigma_i} \vec{v}_i \vec{u}_i^T y$$

Let $\beta_i = \begin{cases} \frac{1}{\sigma_i} & \text{for } i \le k \\ 0 & \text{for } i > k \end{cases}$ then $\hat{y}_{test} = X_{test}^T \sum_{i=1}^{d} \vec{v}_i \beta_i \vec{u}_i^T \vec{y}$

(c) (i) $\lambda$-ridge-regression (ii) k-PCA-OLS (iii): OLS

7. (a) $XX^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T$ ①

If we only had access to $XX^T$, we can do SVD for $XX^T$, and use Eq ①, we can see the $U$ matrix for $XX^T$ is the same as $U$ for $X$

(b) $X = U\Sigma V^T \qquad V^T = \Sigma^{-1} U^{-1} X$

$$= \begin{bmatrix} \frac{1}{\sigma_1} & & & 0 \\ & \ddots & & \\ & & \frac{1}{\sigma_\ell} & \end{bmatrix} \begin{bmatrix} \vec{u}_1^T \\ \vdots \\ \vec{u}_n^T \end{bmatrix} \begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix}$$

$$= \begin{bmatrix} \vec{u}_1^T/\sigma_1 \\ \vdots \\ \vec{u}_\ell^T/\sigma_\ell \end{bmatrix} \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

$$= \begin{bmatrix} u_{11}/\sigma_1 & u_{21}/\sigma_1 & \cdots & u_{n1}/\sigma_1 \\ u_{21}/\sigma_1 & u_{22}/\sigma_2 & \cdots & u_{n2}/\sigma_2 \\ \vdots & & & \\ u_{\ell 1}/\sigma_\ell & u_{\ell 2}/\sigma_\ell & & u_{n\ell}/\sigma_\ell \end{bmatrix} \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

$$v_j^T = \frac{1}{\sigma_j}\left(u_{j1} x_1^T + u_{j2} x_2^T + \cdots + u_{jn} x_n^T\right)$$

$$z_j = v_j^T x_{test} = \sum_{i=1}^{n} \frac{u_{ji}}{\sigma_j} x_i^T x_{test} = \begin{bmatrix} \frac{u_{j1}}{\sigma_j} & \frac{u_{j2}}{\sigma_j} & \cdots & \frac{u_{jn}}{\sigma_j} \end{bmatrix} \begin{pmatrix} \langle x_1, x_{test}\rangle \\ \vdots \\ \langle x_n, x_{test}\rangle \end{pmatrix}$$

(c) Let $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$, take the SVD of $X = U\Sigma V$, where $U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & & u_{2n} \\ & & \vdots & \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{bmatrix}$ $\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_\ell & \\ & & & 0 \end{bmatrix}$

define $V_k = [\vec{v}_1 \cdots \vec{v}_k]$, $X_k = X V_k = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} [\vec{v}_1 \cdots \vec{v}_k] = \begin{bmatrix} x_1^T\vec{v}_1 & x_1^T\vec{v}_2 & \cdots & x_1^T\vec{v}_k \\ x_2^T\vec{v}_1 & x_2^T\vec{v}_2 & \cdots & x_2^T\vec{v}_k \\ \vdots & & & \\ x_n^T\vec{v}_k & x_n^T\vec{v}_2 & \cdots & x_n^T\vec{v}_k \end{bmatrix}$

Here $(X_k)_{ij} = x_i^T \vec{v}_j = \vec{v}_j^T \vec{x}_i = \left[\frac{u_{j1}}{\sigma_j} \quad \frac{u_{j2}}{\sigma_j} \quad \cdots \quad \frac{u_{in}}{\sigma_j}\right] \begin{bmatrix} \langle x_1, x_i \rangle \\ \vdots \\ \langle x_n, x_i \rangle \end{bmatrix} = \sum_{m=1}^{n} \frac{u_{jm}}{\sigma_j} \langle x_m, x_i \rangle$

Using this kernal, we can calculate $X_k$.

Then we can calculate $w = (X_k^T X_k)^{-1} X_k^T y$. ②

Then we need to project $X_{test}$ to the first k principle components.

$$X_{test\_k} = V_k^T X_{test} = \begin{bmatrix} \vec{v}_1^T \\ \vdots \\ \vec{v}_k^T \end{bmatrix} X_{test} = \begin{bmatrix} \vec{v}_1^T X_{test} \\ \vdots \\ \vec{v}_k^T X_{test} \end{bmatrix}$$

Here, each $\vec{v}_i^T X_{test}$ can be calculated by $\sum_{m=1}^{n} \frac{u_{jm}}{\sigma_j} \langle x_m, X_{test} \rangle$. ③

The final prediction $y_{test} = X_{test\_k}\, w$, where $w$, $X_{test\_k}$ can be calculated by ② and ③.

8. (a) B. $\sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T$

(b) C. $1, x_i, y_i, x_i^2, x_i y_i, y_i^2$

(c) A. $k(x, y) = x^T y$ B. $k(x, y) = e^{-\frac{1}{2}\|x-y\|^2}$ C. $k(x, y) = (1 + x^T y)^p$

(d) D. TLS

(e) A. process X using $k \ll n$ random projections.
B. process X using PCA with $k \ll n$ components
E. use a kernal proach
F. Add a ridge penalty to OLS

(f) B. PCA

(g) A. training error B. validation error C. bias.

(h) B. validation error D. variance

(I) B. validation error.

9. Q: consider the problem of minimizing $f(x,y) = 4x^2 - 4xy + 2y^2$ using gradient descent method. Notice that the optimal solution is $(x,y) = (0,0)$. Suppose we start from the point $(x^0, y^0) = (2,3)$. Find the next point $(x', y')$.

A: $(x', y') = (x^0, y^0) - t \cdot \nabla f(x^0, y^0)$

$$\nabla f = \begin{pmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{pmatrix} = \begin{pmatrix} 8x - 4y \\ -4x + 4y \end{pmatrix} \qquad \nabla f(x^0, y^0) = \begin{pmatrix} 16 - 12 \\ -8 + 12 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Let $\theta(t) = f(x', y') = f\left[\begin{pmatrix} 2 \\ 3 \end{pmatrix} - \begin{pmatrix} 4t \\ 4t \end{pmatrix}\right] = f\left[\begin{pmatrix} 2-4t \\ 3-4t \end{pmatrix}\right]$

$$= 4 \times (2-4t)^2 - 4(2-4t)(3-4t) + 2(3-4t)^2$$

$$= 64t - 32 = 0 \qquad \Rightarrow t = \frac{1}{2}$$

So $(x', y') = \begin{pmatrix} 2 \\ 3 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 4 \\ 4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$