CS 189 Introduction to Machine Learning
Spring 2018 HW6

Your self-grade URL is http://eecs189.org/self_grade?question_ids=1_1,1_2,2_1,2_2,2_3,2_4,2_5,2_6,2_7,2_8,2_9,2_10,2_11,3_1,3_2,3_3,3_4,3_5,4_1,4_2,4_3,5_1,5_2,6_1,6_2,6_3,7_1,7_2,7_3,8_1,8_2,8_3,8_4,8_5,8_6,8_7,8_8,8_9,9.

This homework is due **Sunday, March 4 at 10pm.**

2 Canonical Correlation Analysis

The goal of canonical correlation analysis (CCA) is to find linear combinations that maximize the correlation of two random vectors. We are given two zero-mean random vectors $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^d$. Any linear combination of the coordinates of the random vector $\mathbf{X}$ can be written as $\mathbf{a}^\top \mathbf{X}$, and similarly, a linear combination of the coordinates of the random vector $\mathbf{Y}$ can be written as $\mathbf{b}^\top \mathbf{Y}$. Note that $\mathbf{a}^\top \mathbf{X}, \mathbf{b}^\top \mathbf{Y} \in \mathbb{R}$ are scalar random variables.

The goal of CCA can be summarized as solving the following optimization problem

$$\rho = \max_{\mathbf{a},\mathbf{b} \in \mathbb{R}^d} \rho(\mathbf{a}^\top \mathbf{X}, \mathbf{b}^\top \mathbf{Y}). \tag{1}$$

where the correlation coefficient $\rho(P, Q)$ between two zero-mean scalar random variables P and Q is defined by

$$\rho(P, Q) = \frac{\mathbb{E}[PQ]}{\sqrt{\mathbb{E}[P^2]\mathbb{E}[Q^2]}}.$$

The zero-mean jointly-Gaussian case expresses why we care about correlation. Two jointly Gaussian zero-mean random variables that are uncorrelated are also independent of each other. Furthermore, if we want to best estimate (in a mean-squared sense) a particular scalar zero Gaussian random variable Y based on a vector $\mathbf{X}$ of jointly Gaussian zero-men random variables, then we are going to pick a weight vector $\mathbf{w}$ that is aligned with $\mathbb{E}[\mathbf{X}Y]$. It is straightforward algebra to see that this direction maximizes the correlation coefficient $\rho(\frac{\mathbf{w}^\top}{\|\mathbf{w}\|}\mathbf{X}, Y)$.

In this problem, we will work our way towards finding a solution to the problem of canonical correlation analysis using the singular value decomposition. In parts (a) to (c), we derive useful results regarding the singular value decomposition. In the later parts, you will use this result and see that canonical correlation analysis is equivalent to the singular value decomposition in a rotated and stretched coordinate system.

(a) Let $n \geq d$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with full column-rank and singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we know that the singular values are given by the diagonal entries of $\mathbf{\Sigma}$, and

the left singular vectors are the columns of the matrix $\mathbf{U}$ and the right singular vectors are the columns of the matrix $\mathbf{V}$. Note that both the matrices $\mathbf{U}$ and $\mathbf{V}$ have orthonormal columns.

Show that $\mathbf{A} = \sum_{i=1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, where the ith singular value is denoted by $\sigma_i = \mathbf{\Sigma}_{ii}$ and $\mathbf{u}_i$ and $\mathbf{v}_i$ are the ith left and right singular vectors, respectively.

Solution: There are many ways to compute the answer to this problem; we illustrate one of them. We assume that we have the full SVD $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{d \times d}$.

Begin by computing the matrix $\mathbf{U}\mathbf{\Sigma}$, and note that $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{\Sigma} \in \mathbb{R}^{n \times d}$. Think of this matrix multiplication as d vector multiplications; we therefore have

$$(\mathbf{U}\mathbf{\Sigma})_i = \mathbf{U} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \sigma_i \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sigma_i \mathbf{u}_i,$$

where as before, we have used A_i to denote the ith column of a matrix $\mathbf{A}$. Stacking these up, we have

$$\mathbf{U}\mathbf{\Sigma} = \begin{bmatrix} | & & | \\ \sigma_1 \mathbf{u}_1 & \cdots & \sigma_d \mathbf{u}_d \\ | & & | \end{bmatrix}.$$

Now notice that a matrix product can be computed using outer products. In other words, we have $\mathbf{A}\mathbf{B}^\top = \sum_i \mathbf{A}_i \mathbf{B}_i^\top$ (write out a simple 2×2 example to see exactly why). Therefore, we have

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top.$$

(b) **With the setup above, show that**

i) $\mathbf{A}^\top \mathbf{A}$ has i-th eigenvalue σ_i^2, with associated eigenvector $\mathbf{v}_i$.

ii) $\mathbf{A}\mathbf{A}^\top$ has i-th eigenvalue σ_i^2, with associated eigenvector $\mathbf{u}_i$.

Notice that both of the above matrices are symmetric.

Solution: (i) From the above part, we have

$$\mathbf{A}^\top \mathbf{A} = \Big(\sum_{i=1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top\Big)^\top \Big(\sum_{j=1}^{d} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top\Big)$$

$$= (\sum_{i=1}^{d} \sigma_i \mathbf{v}_i \mathbf{u}_i^\top)(\sum_{j=1}^{d} \sigma_i \mathbf{u}_j \mathbf{v}_j^\top).$$

Now notice that $\mathbf{u}_i^\top \mathbf{u}_j = 0$ unless $i = j$, in which case $\mathbf{u}_i^\top \mathbf{u}_i = 1$. Therefore, expanding the above multiplication, we see that only the terms where $i = j$ remain, and we have

$$\mathbf{A}^\top \mathbf{A} = \sum_{i=1}^{d} \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^\top.$$

Consequently, we have

$$\mathbf{A}^\top \mathbf{A} \mathbf{v}_j = \sum_{i=1}^{d} \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^\top \mathbf{v}_j = \sigma_i^2 \mathbf{v}_j,$$

where we have used the fact that $\mathbf{v}_i^\top \mathbf{v}_j = 0$ unless $i = j$, in which case $\mathbf{v}_j^\top \mathbf{v}_j = 1$.

In words, we have shown that $\mathbf{v}_j$ is an eigenvector of $\mathbf{A}^\top \mathbf{A}$ with associated eigenvalue σ_j^2. This holds for all $j = \{1, 2, \ldots, d\}$.

ii) The second part proceeds exactly as above, where we show that

$$\begin{aligned}\mathbf{A}\mathbf{A}^\top &= (\sum_{i=1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top)(\sum_{j=1}^{d} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top)^\top \\ &= (\sum_{i=1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top)(\sum_{j=1}^{d} \sigma_i \mathbf{v}_j \mathbf{u}_j^\top).\end{aligned}$$

Now notice that $\mathbf{v}_i^\top \mathbf{v}_j = 0$ unless $i = j$, in which case $\mathbf{v}_i^\top \mathbf{v}_i = 1$. Therefore, expanding the above multiplication, we see that only the terms where $i = j$ remain, and we have

$$\mathbf{A}\mathbf{A}^\top = \sum_{i=1}^{d} \sigma_i^2 \mathbf{u}_i \mathbf{u}_i^\top.$$

Consequently, we have

$$\mathbf{A}\mathbf{A}^\top \mathbf{u}_j = \sum_{i=1}^{d} \sigma_i^2 \mathbf{u}_i \mathbf{u}_i^\top \mathbf{u}_j = \sigma_i^2 \mathbf{u}_j,$$

where we have used the fact that $\mathbf{u}_i^\top \mathbf{u}_j = 0$ unless $i = j$, in which case $\mathbf{u}_j^\top \mathbf{u}_j = 1$.

In words, we have shown that $\mathbf{u}_j$ is an eigenvector of $\mathbf{A}\mathbf{A}^\top$ with associated eigenvalue σ_j^2. This holds for all $j = \{1, 2, \ldots, d\}$.

Alternatively, we could have used the matrix definition of the SVD and the spectral theorem, and this answer will also get full credit.

(c) **Use the first part to show that**

$$\sigma_1(\mathbf{A}) = \max_{\substack{\mathbf{u}:\|\mathbf{u}\|_2=1 \\ \mathbf{v}:\|\mathbf{v}\|_2=1}} \mathbf{u}^\top \mathbf{A} \mathbf{v},$$

where $\sigma_1(\mathbf{A})$ is the maximum singular value of $\mathbf{A}$.

Additionally, show that if $\mathbf{A}$ has a unique maximum singular value, then the maximizers $(\mathbf{u}^*, \mathbf{v}^*)$ above are given by the first left and right singular vectors, respectively.

Hint 1: You may or may not find the following fact useful: We can express any $\mathbf{u} \in \mathbb{R}^n : \|\mathbf{u}\|_2 = 1$ as a linear combination of left singular vectors $\{\mathbf{u}_i\}$ of the matrix $\mathbf{A}$, and any vector $\mathbf{v} \in \mathbb{R}^d$ as a linear combination of the right singular vectors $\{\mathbf{v}_i\}$ of the matrix $\mathbf{A}$.

Hint 2: You may find the following results: For any two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, we have

- Cauchy-Schwarz inequality: $|\mathbf{a}^\top \mathbf{b}| \leq \|\mathbf{a}\|_2 \|\mathbf{b}\|_2$, with equality only when $\mathbf{b}$ is a scaled version of $\mathbf{a}$.
- Holder's inequality: $|\mathbf{a}^\top \mathbf{b}| \leq \|\mathbf{a}\|_1 \|\mathbf{b}\|_\infty$. Here, the ℓ_1 and ℓ_∞ norms of a vector $\mathbf{v}$ are defined by $\|\mathbf{v}\|_1 = \sum_i |v_i|$, and $\|\mathbf{v}\|_\infty = \max_i |v_i|$.
  - Let us say the vector $\mathbf{b}$ is fixed; then one way to achieve equality in the Holder inequality is to have: Let i be such that $|b_i| = \|\mathbf{b}\|_\infty$. Set $a_i = \|\mathbf{a}\|_1$, and $a_j = 0$ for all $j \neq i$.

Solution: Using the hint, we may write a unit norm vector $\mathbf{u} = \sum_{i=1}^n a_i \mathbf{u}_i$, where $\mathbf{u}_i$ are the n left singular vectors of the matrix $\mathbf{A}$. Notice that since $\mathbf{u}$ is a unit norm vector, we must have $\|\mathbf{u}\|_2^2 = \sum_{i=1}^n a_i^2 = 1$. Similarly, $\mathbf{v} = \sum_{i=1}^d b_i \mathbf{v}_i$, where $\mathbf{v}_i$ are the d right singular vectors of the matrix $\mathbf{A}$ and $\|\mathbf{v}\|_2^2 = \sum_{i=1}^d b_i^2 = 1$.

Alternatively, notice that for any $\mathbf{u} : \|\mathbf{u}\|_2 = 1$, we have $\|\mathbf{U}^\top \mathbf{u}\|_2 = \|\mathbf{u}\|_2 = 1$ by unitary invariance of the ℓ_2 norm (since $\mathbf{U}^\top \in \mathbb{R}^{n \times n}$ has orthonormal columns). Similarly, we have $\|\mathbf{V}^\top \mathbf{v}\|_2 = \|\mathbf{v}\|_2 = 1$. Since $\mathbf{U}^\top$ and $\mathbf{V}^\top$ are full rank matrices, maximizing over $\mathbf{u}$ and $\mathbf{v}$ is equivalent to maximizing over a linear combination of them, and so we have

$$\max_{\substack{\mathbf{u}:\|\mathbf{u}\|_2=1 \\ \mathbf{v}:\|\mathbf{v}\|_2=1}} \mathbf{u}^\top \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^\top \mathbf{v} = \max_{\substack{\mathbf{a}:\|\mathbf{a}\|_2=1 \\ \mathbf{b}:\|\mathbf{b}\|_2=1}} \mathbf{a}^\top \boldsymbol{\Sigma} \mathbf{b}.$$

Now notice that $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times d}$ has its last $n - d$ rows equal to zero. Therefore, expanding out the last expression similarly to the first part, we have reduced the problem to solving

$$\max_{\substack{\mathbf{a}:\|\mathbf{a}\|_2=1 \\ \mathbf{b}:\|\mathbf{b}\|_2=1}} \sum_{i=1}^d \sigma_i a_i b_i.$$

Now, we see that there is no value in setting any entry of $\mathbf{a}$ or $\mathbf{b}$ to be negative, since σ_i is always positive. We will therefore assume that $a_i \geq 0$ and $b_i \geq 0$ for all i. Also, denote $a_i b_i = c_i$, and collect the c_i values into a d-dimensional vector $\mathbf{c}$. Similarly, collect the σ_i values into a vector $\boldsymbol{\sigma}$. We therefore have that for any unit norm vectors $\mathbf{u}, \mathbf{v}$, the following inequality holds:

$$\mathbf{u}^\top \mathbf{A} \mathbf{v} = \mathbf{c}^\top \boldsymbol{\sigma} \leq \|\mathbf{c}\|_1 \|\boldsymbol{\sigma}\|_\infty = \|\mathbf{c}\|_1 \sigma_1(\mathbf{A}),$$

where we have used Holder's inequality.

Now, note that $\|\mathbf{c}\|_1 = \mathbf{a}^\top \mathbf{b}$ since all the entries were assumed to be positive, and using Cauchy Schwarz inequality, we know that $\mathbf{a}^\top \mathbf{b} \leq \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 = 1$. We have thus shown that $\mathbf{u}^\top \mathbf{A} \mathbf{v} \leq \sigma_1(\mathbf{A})$ for all unit norm vectors $\mathbf{u}$ and $\mathbf{v}$.

In order to show that the maximum is attained, choose $\mathbf{u} = \mathbf{u}_1$, and $\mathbf{v} = \mathbf{v}_1$, and note that this corresponds to choosing $\mathbf{a}$ and $\mathbf{b}$ to be the first basis vector in $\mathbb{R}^d$, which we denote by $\mathbf{e}_1$. Thus, we have

$$\mathbf{u}_1 \mathbf{A} \mathbf{v}_1 = \mathbf{s}_1 \mathbf{\Sigma} \mathbf{e}_1 = \sigma_1(\mathbf{A}).$$

We have thus shown that the maximum $\sigma_1(A)$ is indeed attained by the claimed maximizers $\mathbf{u}^* = \mathbf{u}_1$ and $\mathbf{v}^* = \mathbf{v}_1$.

(d) Let us now look at the canonical correlation analysis problem, where we are given the covariance of two zero-mean random vectors $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^d$ as follows:

$$\mathbb{E}[\mathbf{X}\mathbf{X}^\top] = \mathbf{\Sigma}_{XX}, \quad \mathbb{E}[\mathbf{Y}\mathbf{Y}^\top] = \mathbf{\Sigma}_{YY}, \quad \text{and,} \quad \mathbb{E}[\mathbf{X}\mathbf{Y}^\top] = \mathbf{\Sigma}_{XY}.$$

The goal is to find two directions/unit-vectors $\mathbf{a}$ and $\mathbf{b}$ so that the resulting scalar random variables $\mathbf{a}^\top \mathbf{X}$ and $\mathbf{b}^\top \mathbf{Y}$ have maximal correlation coefficient $\rho(\mathbf{a}^\top \mathbf{X}, \mathbf{b}^\top \mathbf{Y})$.

Show that the canonical correlation analysis problem can be rewritten as

$$\rho = \max_{\mathbf{a},\mathbf{b} \in \mathbb{R}^d} \frac{\mathbf{a}^\top \mathbf{\Sigma}_{XY} \mathbf{b}}{\left(\mathbf{a}^\top \mathbf{\Sigma}_{XX} \mathbf{a}\right)^{1/2} \left(\mathbf{b}^\top \mathbf{\Sigma}_{YY} \mathbf{b}\right)^{1/2}}.$$

Conclude that if $(\mathbf{a}^*, \mathbf{b}^*)$ is a maximizer above, then $(\alpha \mathbf{a}^*, \beta \mathbf{b}^*)$ is a maximizer for any $\alpha, \beta > 0$. We see that scaling the vectors $\mathbf{a}$ or $\mathbf{b}$ does not affect their correlation coefficient.

Solution:

Using the fact that $\mathbf{b}^\top Y$ is a scalar, we have $\mathbf{b}^\top Y = Y^\top \mathbf{b}$. We can now apply linearity of expectation to see that

$$\mathbb{E}[(\mathbf{a}^\top X)(\mathbf{b}^\top Y)] = \mathbb{E}[\mathbf{a}^\top X Y^\top \mathbf{b}] = \mathbf{a}^\top \mathbb{E}[X Y^\top] \mathbf{b} = \mathbf{a}^\top \mathbf{\Sigma}_{XY} \mathbf{b}.$$

By a similar argument, we have

$$\mathbb{E}[(\mathbf{a}^\top X)(\mathbf{a}^\top X)] = \mathbf{a}^\top \mathbf{\Sigma}_{XX} \mathbf{a}, \text{ and}$$
$$\mathbb{E}[(\mathbf{b}^\top Y)(\mathbf{b}^\top Y)] = \mathbf{b}^\top \mathbf{\Sigma}_{YY} \mathbf{b}.$$

Substituting these quantities into the definition of the correlation coefficient, we have

$$\rho = \max_{\mathbf{a},\mathbf{b} \in \mathbb{R}^d} \frac{\mathbf{a}^\top \mathbf{\Sigma}_{XY} \mathbf{b}}{\left(\mathbf{a}^\top \mathbf{\Sigma}_{XX} \mathbf{a}\right)^{1/2} \left(\mathbf{b}^\top \mathbf{\Sigma}_{YY} \mathbf{b}\right)^{1/2}}.$$

In order to see the last conclusion, simply scale $\mathbf{a}$ and $\mathbf{b}$ by α and β respectively, and notice that

$$\frac{(\alpha\mathbf{a})^\top \mathbf{\Sigma}_{XY}(\beta\mathbf{b})}{\left((\alpha\mathbf{a})^\top \mathbf{\Sigma}_{XX}(\alpha\mathbf{a})\right)^{1/2}\left((\beta\mathbf{b})^\top \mathbf{\Sigma}_{YY}(\beta\mathbf{b})\right)^{1/2}} = \frac{\mathbf{a}^\top \mathbf{\Sigma}_{XY}\mathbf{b}}{\left(\mathbf{a}^\top \mathbf{\Sigma}_{XX}\mathbf{a}\right)^{1/2}\left(\mathbf{b}^\top \mathbf{\Sigma}_{YY}\mathbf{b}\right)^{1/2}}$$

via cancellation in the numerator and denominator. Thus the maximization problem does not change either.

(e) Let us simplify our optimization problem. Assume that the covariance matrices $\mathbf{\Sigma}_{XX}$ and $\mathbf{\Sigma}_{YY}$ are full-rank. We choose matrices $\mathbf{\Sigma}_{XX}^{-\frac{1}{2}}$, $\mathbf{\Sigma}_{YY}^{-\frac{1}{2}}$ to whiten the vectors $\mathbf{X}$ and $\mathbf{Y}$ respectively. Note that for the vector $\tilde{\mathbf{X}} = \mathbf{\Sigma}_{XX}^{-\frac{1}{2}}\mathbf{X}$, we have

$$\mathbb{E}[\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top] = \mathbb{E}[(\Sigma_{XX}^{-\frac{1}{2}}\mathbf{X})(\Sigma_{XX}^{-\frac{1}{2}}\mathbf{X})^\top] = \mathbf{I}$$

One may do a similar computation for the whitened vector $\tilde{\mathbf{Y}} = \mathbf{\Sigma}_{YY}^{-\frac{1}{2}}\mathbf{Y}$, and conclude that its covariance matrix is identity too! Such a whitening step simplifies our computations, both at algebraic and conceptual level.

Rewrite the optimization problem from the previous part, using the aforementioned whitening in the following form:

$$\rho = \max_{\substack{\mathbf{c}:\|\mathbf{c}\|_2=1 \\ \mathbf{d}:\|\mathbf{d}\|_2=1}} \mathbf{c}^\top \mathbf{\Sigma}_{XX}^{-1/2}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-1/2}\mathbf{d}.$$

Solution: Using the results from section (d) we can write the optimization problem as follows:

$$\begin{aligned}
\rho &= \max_{\mathbf{a},\mathbf{b}\in\mathbb{R}^d} \frac{\mathbf{a}^\top \mathbf{\Sigma}_{XY}\mathbf{b}}{\left(\mathbf{a}^\top \mathbf{\Sigma}_{XX}\mathbf{a}\right)^{1/2}\left(\mathbf{b}^\top \mathbf{\Sigma}_{YY}\mathbf{b}\right)^{1/2}} \\
&= \max_{\mathbf{a},\mathbf{b}\in\mathbb{R}^d} \frac{(\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})^\top \mathbf{\Sigma}_{XX}^{-\frac{1}{2}}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-\frac{1}{2}}(\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})}{\left((\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})^\top \mathbf{\Sigma}_{XX}^{-\frac{1}{2}}\mathbf{\Sigma}_{XX}\mathbf{\Sigma}_{XX}^{-\frac{1}{2}}(\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})\right)^{1/2}\left((\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})^\top \mathbf{\Sigma}_{YY}^{-\frac{1}{2}}\mathbf{\Sigma}_{YY}\mathbf{\Sigma}_{YY}^{-\frac{1}{2}}(\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})\right)^{1/2}} \\
&= \max_{\mathbf{a},\mathbf{b}\in\mathbb{R}^d} \frac{(\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})^\top \mathbf{\Sigma}_{XX}^{-\frac{1}{2}}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-\frac{1}{2}}(\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})}{\left((\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})^\top (\mathbf{\Sigma}_{XX}^{\frac{1}{2}}\mathbf{a})\right)^{1/2}\left((\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})^\top (\mathbf{\Sigma}_{YY}^{\frac{1}{2}}\mathbf{b})\right)^{1/2}} \\
&= \max_{\mathbf{c},\mathbf{d}\in\mathbb{R}^d} \frac{\mathbf{c}^\top \mathbf{\Sigma}_{XX}^{-\frac{1}{2}}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-\frac{1}{2}}\mathbf{d}}{\left(\mathbf{c}^\top \mathbf{c}\right)^{1/2}\left(\mathbf{d}^\top \mathbf{d}\right)^{1/2}}
\end{aligned}$$

From the previous part, we know that scaling c and d does not change the objective. Consequently, we may assume that c and d have unit norm, and so

$$\rho = \max_{\substack{\mathbf{c}:\|\mathbf{c}\|_2=1 \\ \mathbf{d}:\|\mathbf{d}\|_2=1}} \mathbf{c}^\top \mathbf{\Sigma}_{XX}^{-1/2}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-1/2}\mathbf{d}.$$

(f) Recall that the vectors $(\mathbf{a}^*, \mathbf{b}^*)$ denote maximizers in the previous part of the problem. Using the simplification and the above parts **show that**

ρ^2 is the maximum eigenvalue of the matrix

$$\boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1}\boldsymbol{\Sigma}_{XY}^{\top}\boldsymbol{\Sigma}_{XX}^{-1/2}.$$

Hint: An appropriate change of variables may make your life easier.

Solution:

Let us denote $\mathbf{A} = \boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}$. From part (c), we know that the value of ρ is the maximum singular value of the matrix $\mathbf{A}$. We can see that $\mathbf{A}\mathbf{A}^\top = \boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1}\boldsymbol{\Sigma}_{XY}^{\top}\boldsymbol{\Sigma}_{XX}^{-1/2}$ and using part (b) it follows that ρ^2 is the maximum eigenvalue of the matrix $\mathbf{A}\mathbf{A}^\top$.

(g) Following the previous part's setup, **show that** $\mathbf{c}^* = \boldsymbol{\Sigma}_{XX}^{1/2}\mathbf{a}^*$ is an eigenvector corresponding to the maximum eigenvalue of the matrix

$$\boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1}\boldsymbol{\Sigma}_{XY}^{\top}\boldsymbol{\Sigma}_{XX}^{-1/2}.$$

Solution: From part (c) $\mathbf{c}^* = \boldsymbol{\Sigma}_{XX}^{1/2}\mathbf{a}^*$ is the maximal left vector of $\mathbf{A}$. We now apply part (b), which states that $\mathbf{c}^*$ is the maximal eigenvector of the matrix

$$\mathbf{A}\mathbf{A}^\top = \boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}\left(\boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}\right)^\top.$$

(h) Following the previous part's setup, **show that** $\mathbf{d}^* = \boldsymbol{\Sigma}_{YY}^{1/2}\mathbf{b}^*$ is an eigenvector corresponding to the maximum eigenvalue of the matrix

$$\boldsymbol{\Sigma}_{YY}^{-1/2}\boldsymbol{\Sigma}_{XY}^{\top}\boldsymbol{\Sigma}_{XX}^{-1}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}.$$

Solution: Similarly, from part (c) $\mathbf{d}^* = \boldsymbol{\Sigma}_{YY}^{1/2}\mathbf{b}^*$ is the maximal right vector of $\mathbf{A}$ and from (b) $\mathbf{d}^*$ is the maximal eigenvector of the matrix

$$\mathbf{A}^\top\mathbf{A} = \left(\boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}\right)^\top\boldsymbol{\Sigma}_{XX}^{-1/2}\boldsymbol{\Sigma}_{XY}\boldsymbol{\Sigma}_{YY}^{-1/2}.$$

(i) **Argue why such a CCA is not meaningful when the random vectors $\mathbf{X}$ and $\mathbf{Y}$ are uncorrelated, where by this we mean that $\mathsf{cov}(X_i, Y_j) = 0$ for all i, j.**

Solution: Since $\boldsymbol{\Sigma}_{XY} = 0$, CCA returns $\rho = 0$ and $\mathbf{a} = \mathbf{b} = \mathbf{0}$, and is therefore useless.

(j) Suppose you happen to know that $\mathbf{X}$ and $\mathbf{Y}^2$ (where a squared-vector $\mathbf{Y}^2$ is defined by squaring each entry of the vector $\mathbf{Y}$) share a linear relationship, **how could you still use CCA effectively with the given data?**

Solution: If we know that there is a non-linear relationship between the variables that is quadratic, we may perform CCA on the variables $\mathbf{X}$ and $\mathbf{Z} = \mathbf{Y}^2$, between which we expect a linear relationship to exist. This is similar to the trick we played in moving from linear regression to polynomial regression.

(k) **Why do you think that understanding CCA is relevant for machine learning? Solution:** Canonical Correlation Analysis (CCA) is a fundamental statistical technique for characterizing the linear relationships between two multidimensional variables. As such it allows learning features that may exist in multiple sources. For example one can be given two sources recording, one audio and one video, both sources might be noisy in different ways, but both encode the same speech pattern.

This is a linear modeling approach that then inspires us for nonlinear neural net architectures that have bottlenecks.

3 Mooney Reconstruction

In this problem, we will try to restore photos of celebrities from Mooney photos, which are binarized faces. In order to do this, we will leverage a large training set of grayscale faces and Mooney faces.

Producing a face reconstruction from a binarized counterpart is a challenging high dimensional problem, but we will show that we can learn to do so from data. In particular, using the power of Canonical Correlation Analysis (CCA), we will reduce the dimensionality of the problem by projecting the data into a subspace where the images are most correlated.

Images are famously redundant and well representable in lower-dimensional subspaces as the eigenfaces example in class showed. However, here our goal is to relate two different kinds of images to each other. Let's see what happens.

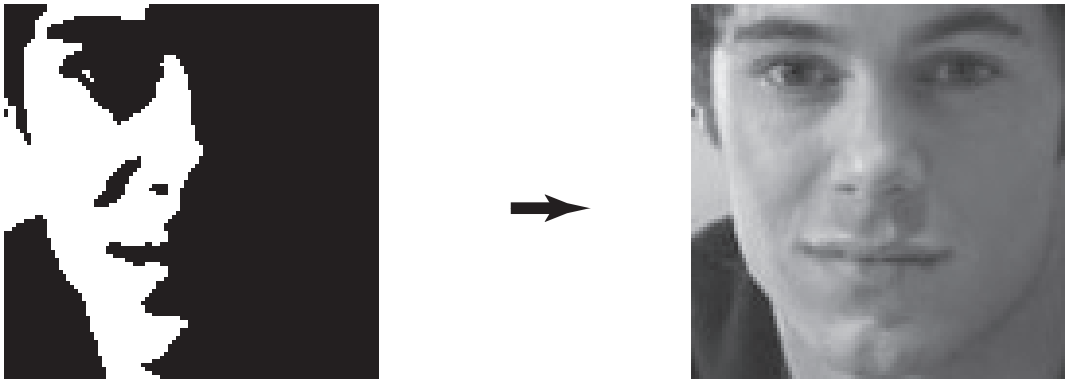


Figure 1: A
binarized Mooney image of an face being restored to its original grayscale image.

The following datasets will be used for this project: X_train.p, Y_train.p, X_test.p and Y_test.p. The training data X_train.p contains 956 binarized images, where each image $\mathbf{X}_i \in \mathbb{R}^{15\times15\times3}$. The test data X_test.p contains 255 binarized images with the same dimensionality. Y_train.p contains 956 corresponding grayscale images, where $\mathbf{Y}_i \in \mathbb{R}^{15\times15\times3}$. Y_test.p contains 255 grayscale images that correspond to X_test.p.

Through out the problem we will assume that all data points are flattened and standardize as follows:

$$x \mapsto (x/255) * 2.0 - 1.0 \quad \text{and} \quad y \mapsto (y/255) * 2.0 - 1.0.$$

Note that this standardization step on loading the data does not ensure that the resulting images have zero mean. So removing the mean from the images is an additional processing step.)

Please use only the following libraries to solve the problem in Python 3.0:

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

(a) We use CCA to find pairs of directions $\mathbf{a}$ and $\mathbf{b}$ that maximize the correlation between the projections $\tilde{\mathbf{x}} = \mathbf{a}^T\mathbf{x}$ and $\tilde{y} = \mathbf{b}^T\mathbf{y}$, From the previous problem, we know that this can be done by solving the problem

$$\rho = \max_{\mathbf{a},\mathbf{b}} \frac{\mathbf{a}^\top \boldsymbol{\Sigma}_{XY} \mathbf{b}}{\left(\mathbf{a}^\top \boldsymbol{\Sigma}_{XX} \mathbf{a}\right)^{1/2} \left(\mathbf{b}^\top \boldsymbol{\Sigma}_{YY} \mathbf{b}\right)^{1/2}},$$

where $\boldsymbol{\Sigma}_{XX}$ and $\boldsymbol{\Sigma}_{YY}$ denote the covariance matrices of the $\mathbf{X}$ and $\mathbf{Y}$ images, and $\boldsymbol{\Sigma}_{XY}$ denotes the covariance between $\mathbf{X}$ and $\mathbf{Y}$. Note that unlike in the previous problem, we are not given the covariance matrices and must estimate them from the images samples of $\mathbf{X}$ and $\mathbf{Y}$.

Write down how to estimate the three covariance matrices from finite samples of data and implement the code for it.

In order to properly compute the covariance matrices you have to 1) standardize the data, 2) subtract the mean and 3) scale by the number of data points at the end.

Note throughout the homework, we shall use the function StandardScaler to subtract the mean from the data and subsequently add it back.

Solution:

```
def compute_covariance_matrices(self):

  #USE STANDARD SCALAR TO DO MEAN SUBTRACTION
  #NOTE WE TURN OFF STD SCALING
  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 = (x_f/ 255.0) * 2.0 - 1.0
    y_f = (y_f/ 255.0) * 2.0 - 1.0

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

  #CACULATE THE MEAN USING SKLEARN STANDARDSCALAR
  ss_x.fit(x_data)
  #SUBTRACT THE MEAN FROM THE DATA POINTS
  x_data = ss_x.transform(x_data)

  #CACULATE THE MEAN USING SKLEARN STANDARDSCALAR ON THE Y COMPONENTS
  ss_y.fit(y_data)

  #SUBTRACT THE MEAN FROM THE DATA POINTS
  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]])

    #COMPUTE COVARIANCE MATRIX
    self.C_xx += np.dot(x_f.T,x_f)
    self.C_yy += np.dot(y_f.T,y_f)

    self.C_xy += np.dot(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
```

(b) We know from the previous problem that we are interested in the maximum singular value of the matrix $\mathbf{\Sigma}_{XX}^{-1/2}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-1/2}$, and that this corresponds to the maximum correlation coefficient ρ. Now, however, **plot the full "spectrum" of singular values of the matrix**

$$(\mathbf{\Sigma}_{XX} + \lambda\mathbf{I})^{-1/2}\mathbf{\Sigma}_{XY}(\mathbf{\Sigma}_{YY} + \lambda\mathbf{I})^{-1/2}.$$

For numerical issues we need to add a very small scalar times the identity matrix to the covariance terms. Set $\lambda = 0.00001$.

Solution:

```
  def solve_for_variance(self):
    lmbda = 0.00001
    #COMPUTE INVERTED SQUARE ROOT FOR EACH MATRIX
    A = inv(sqrtm(self.C_xx+lmbda*np.eye(self.x_dim)))
    B = inv(sqrtm(self.C_yy+lmbda*np.eye(self.y_dim)))

    #MULTIPLY THE MATRICES WITH \mat \Sigma_{XY}
    C = np.matmul(A,np.matmul(self.C_xy,B))

    #COMPUTE THE SINGULAR VALUES
    u,s,d = svd(C)

    #PLOT THE SINGULAR VALUES
    plt.plot(s)
    plt.xlabel('Directions')
    plt.ylabel('Singular Values')
    plt.show()

    singular_values = s
    singular_vectors = u


    return singular_values,singular_vectors
```

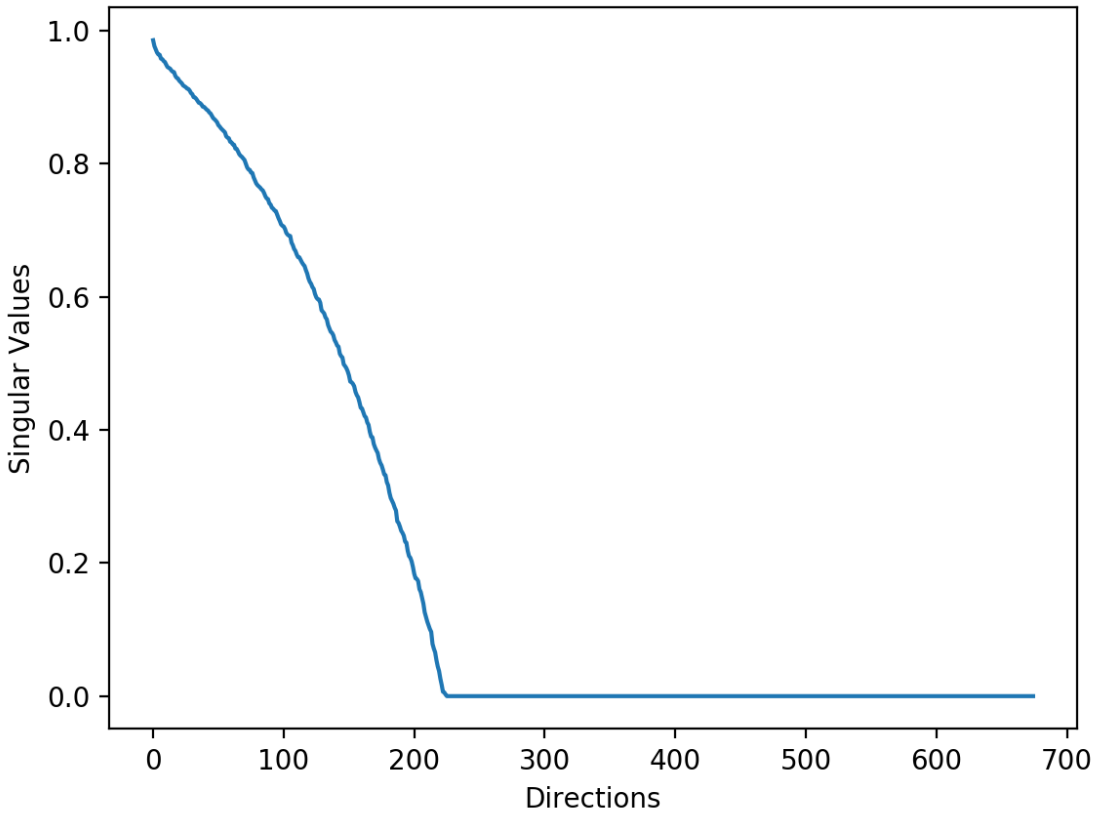


Figure 2: The Singular Values for the Correlation Matrix plotted from largest to smallest.

(c) You should have noticed from the previous part that we have some singular value decay. It therefore makes sense to only consider the top singular value(s), as in CCA. Let us now try to project our images $\mathbf{X}$ on to the subspace spanned by the top k singular vectors. Given the SVD $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{\Sigma}_{XX}^{-1/2}\mathbf{\Sigma}_{XY}\mathbf{\Sigma}_{YY}^{-1/2}$, we can use the left hand singular vectors to form a projection.

$$\mathbf{P}_k = \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \dots & \mathbf{u}_{k-1} \end{bmatrix},$$

Show/visualize the "face" corresponding to the first singular vector $\mathbf{u}_0$. Use the following code for the visualization:

```
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('singular_face.png',p)
```

Solution:

We plot the first singular vector of U. Note: make sure to add the mean back to the data before you draw it with $plot\_image$.

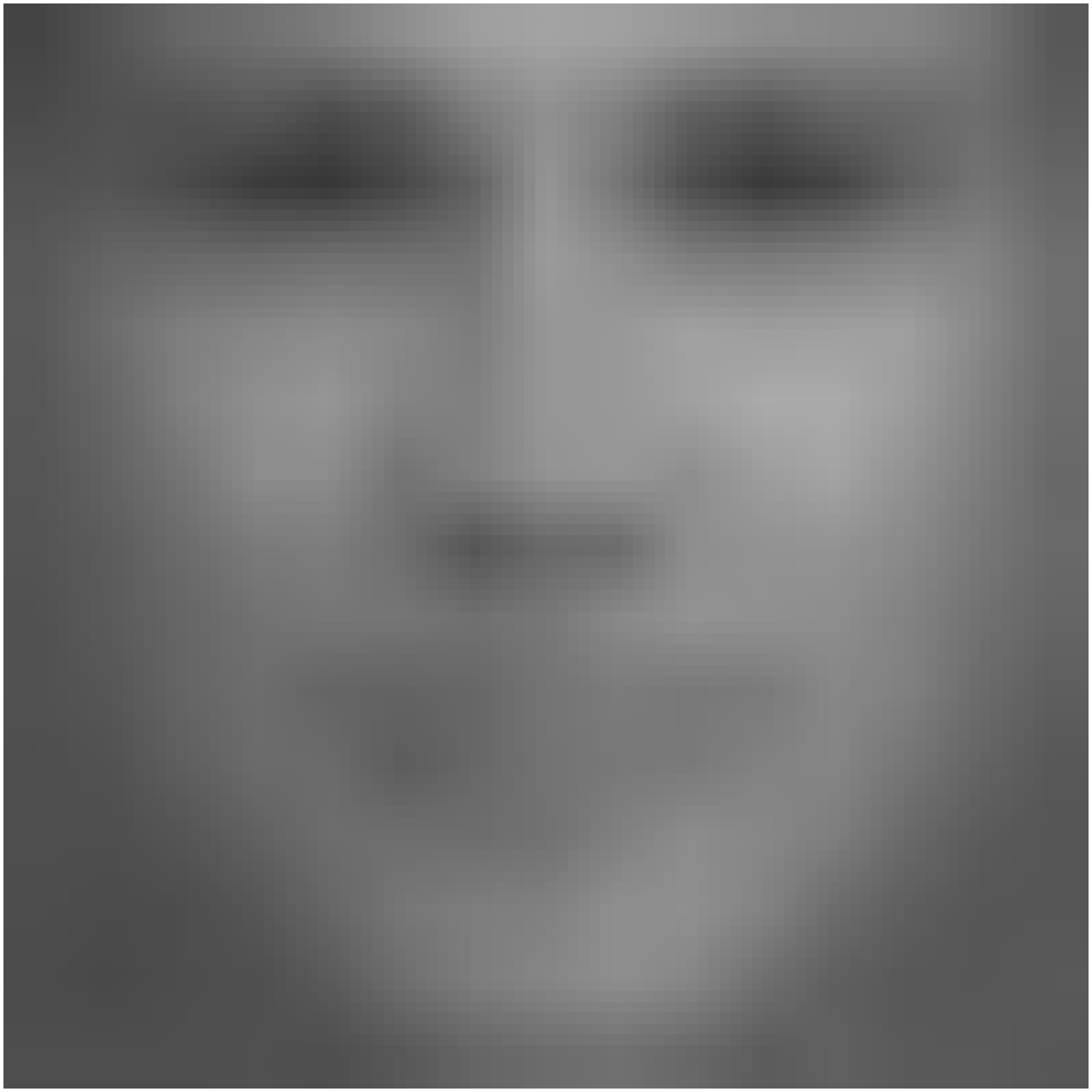


Figure 3: The first component of the CCA

```
def draw_singular_face(self,C):
  ###C is the Correlation Matric Computed in Part B###
  u,s,d = svd(C)

  singular_vectors = u

  ###ADD THE COMPUTED MEAN ON THE X TRAINING DATA BACK TO THE PREDICTED VECTOR###
```

```
  sing_vec = self.ss_x.inverse_transform(singular_vectors[:,0])

  self.plot_image(sing_vec)
```

(d) We will now examine how well the projected data helps generalization when performing regression. You can think of CCA as a technique to help learn better features for the problem. We will use ridge regression regression to learn a mapping, $\mathbf{W} \in \mathbb{R}^{k \times 675}$ $(15 * 15 * 3 = 675)$, from the projected binarized data to the grayscale images. The binarized images are placed in matrix $\mathbf{X} \in \mathbb{R}^{956 \times 675}$

$$\min_{\mathbf{W}} ||(\mathbf{X}\mathbf{P}_k)\mathbf{W} - \mathbf{Y}||_2^2 + \lambda ||\mathbf{W}||_F^2.$$

Implement Ridge Regression with $\lambda = 0.00001$. Plot the Squared Euclidean test error for the following values of k (the dimensions you reduce to):

$k = \{0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 650\}$**.**

Solution:

As shown, in the plot below $k = 50$ yields the best prediction on our held out test set. The intuition for this is that CCA reduced the space of our problem, so the learner only had to consider a smaller subset when making the prediction. However, if k becomes too small, the learner does not have enough information and the error goes up. In these solutions, we computed the data points with the mean subtracted from the data. The code below starts with the function *sweep_projection*

```

def compute_projected_data_matrix(self,X_proj):

  Y = []
  X = []

  Y_test = []
  X_test = []

  #LOAD TRANING DATA
  for x in self.x_train:

    x_f = np.array([x.flatten()])
    #STANDARD DATA
    x_f = (x_f/ 255.0) * 2.0 - 1.0

    #SUBTRACT MEAN
    x_f = self.ss_x.transform(x_f)

    #PROJECT DATA
    x_p = np.matmul(X_proj,x_f.T)

    X.append(x_p[:,0])

  for y in self.y_train:

    y_f = np.array([y.flatten()])

    #STANDARDIZE DATA
```

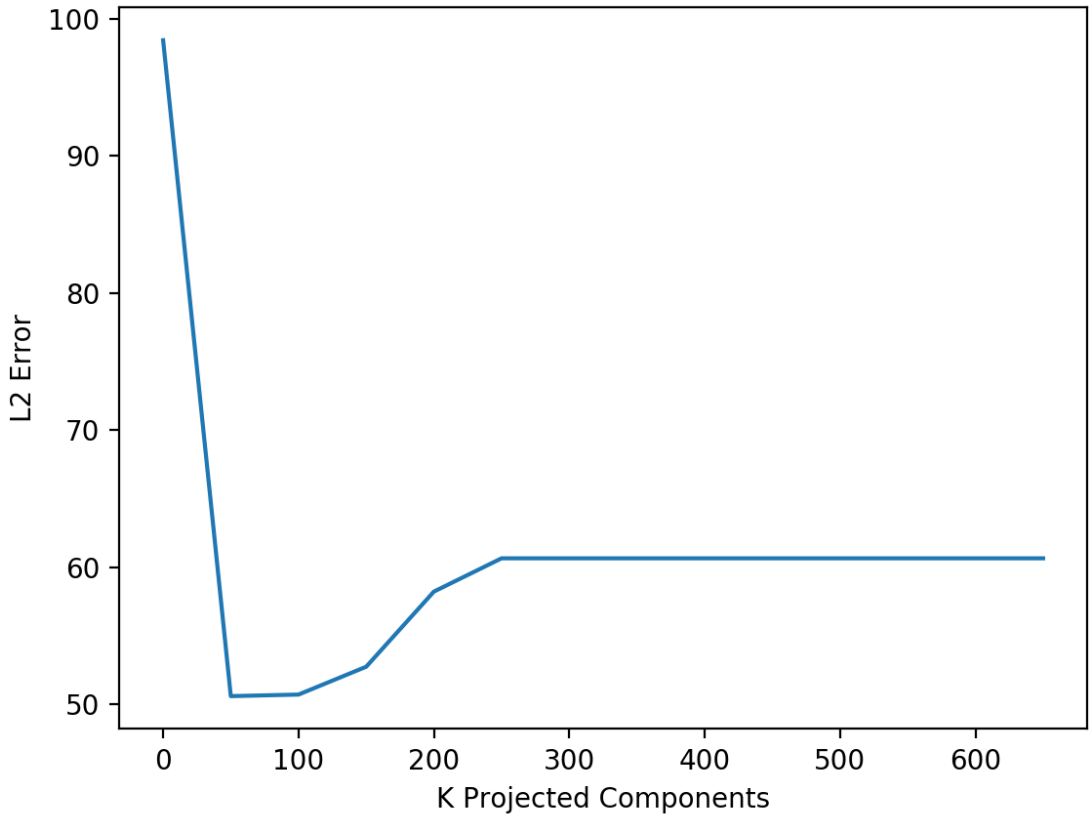


Figure 4: The error on the held out set versus how many projected components were considered. This plots the error normalized by the number of data-points, but unnormalized plots are fine.

```
    y_f = (y_f/ 255.0) * 2.0 - 1.0

    #SUBTRACT MEAN
    y_f = self.ss_y.transform(y_f)
    Y.append(y_f[0,:])

  for x in self.x_test:

    x_f = np.array([x.flatten()])
    #STANDARDIZE DATA
    x_f = (x_f/ 255.0) * 2.0 - 1.0
    #SUBTRACT MEAN
    x_f = self.ss_x.transform(x_f)

    #PROJECT DATA
    x_p = np.matmul(X_proj,x_f.T)
    X_test.append(x_p[:,0])


  for y in self.y_test:

    y_f = np.array([y.flatten()])
    #STANDARDIZE DATA
    y_f = (y_f/ 255.0) * 2.0 - 1.0
```

```
    #SUBTRACT MEAN
    y_f = self.ss_y.transform(y_f)

    Y_test.append(y_f[0,:])

  #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 ridge_regresion(self):

  #PERFORM RIDGE REGRESSION
  X_G = np.matmul(self.X_ridge.T,self.X_ridge)

  lmbda = 0.00001
  A = X_G+lmbda*np.eye(X_G.shape[0])

  p_1 = LA.inv(A)

  w_ridge = []
  for i in range(self.y_dim):

    p_2 = np.matmul(self.Y_ridge[:,i],self.X_ridge)
    ans = np.matmul(p_2,p_1)
    w_ridge.append(ans)

  #RETURNED LEARNED WEIGHTS
  self.w_ridge = np.vstack(w_ridge)

def measure_error_on_test(self):

    #MAKE A PREDICTION ON THE PROJECTED DATA
    prediction = np.matmul(self.w_ridge,self.X_test_ridge.T)

    #MEASURE HOW CLOSE THE PREDICTION IS TO THE TRUE ANSWER
    evaluation = self.Y_test_ridge.T - prediction

    dim,num_data = evaluation.shape

    error = []

    #COMPUTE THE L2 NORM
    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 project_data(self, sing_vec,k=100):

  #EXCTRACT FIRST K  SINGULAR VECTORS
  X_proj = sing_vec[:,0:k]

  return X_proj.T

def sweep_projection(self):

  ###CACULATE COVARAINCE MATRICES (PART A)###
  compute_covariance_matrices()

  ####COMPUTE CORRELATION MATRIC (PART B)###
  sing_val, sing_vec = solve_for_variance()
```

```
122
123   proj = [0,50,100,150,200,250,300,350,400,450,500,650]
124   error_test = []
125
126
127   for k in proj:
128
129     ####COMPUTE MATRIX P
130     X_proj = hw5_sol.project_data(sing_val,sing_vec,proj=k)
131
132
133
134     #####COMPUTE PROJECTED DATA####
135     hw5_sol.compute_projected_data_matrix(X_proj)
136     #COMPUTE REGRESSION
137     hw5_sol.ridge_regresion()
138
139     #COMPUTE TEST ERROR
140     test_error = hw5_sol.measure_error_on_test()
141
142     error_test.append(test_error)
143
144   #PLOT PROJECTION VERSUS ERROR
145
146   plt.plot(proj,error_test)
147
148   plt.ylabel('L2 Error')
149   plt.xlabel('K Projected Components')
150   plt.show()
```

(e) **Try running the learned model on 4 of the images in the test set and report the results. Give both the binarized input, the true grayscale, and the output of your model.** You may use the code from previous part to visualize the images.

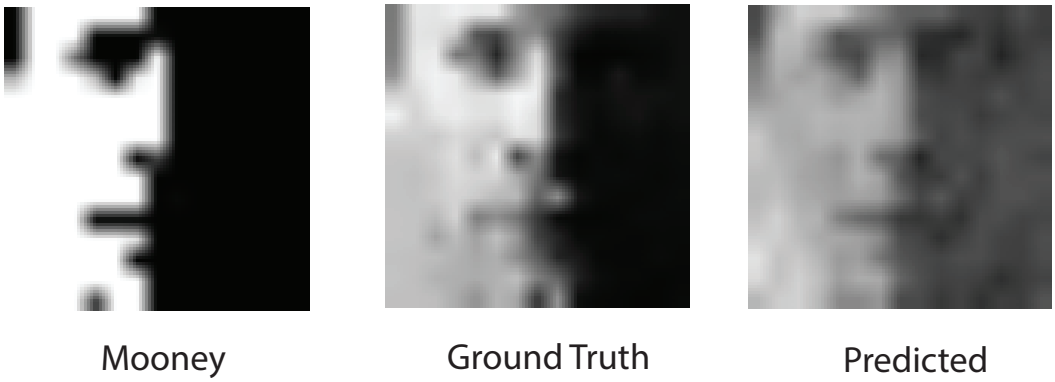


Figure 5: Example results with the input on the left and output on the right

Solution: The images shown below kind of match the ground truth image. Since, we are using a linear mapping on raw pixel values it is hard to expect better results. State of the art performance on this task is achieved using neural networks, as is common in a lot of vision problem. Note, in order to generate the new image make sure to add the mean back to the original image.

```
1 def draw_images(self):
```

```
    count = 0
    for x in self.X_test_ridge:

      prediction = np.matmul(self.w_ridge,x)
      ###ADD THE COMPUTED MEAN BACK TO THE PREDICTED VECTOR###
      prediction = self.ss_y.inverse_transform(prediction)
      self.plot_image(prediction,count)
      count += 1
    count = 0


    for x in self.x_test:

      x = x.astype("uint8")

      x = cv2.resize(x,(100,100))

      cv2.imwrite('og_face_'+str(count)+'.png',x)


      count+=1

    for x in self.y_test:
      x = x.astype("uint8")

      x = cv2.resize(x,(100,100))

      cv2.imwrite('gt_face_'+str(count)+'.png',x)


      count+=1
```

4 Bias-Variance for Ridge Regression (24 points)

Consider the scalar data-generation model:

$$Y = xw^* + Z$$

where x denotes the scalar input feature, Y denotes the scalar noisy measurement, $Z \sim \mathcal{N}(0,1)$ is standard unit-variance zero-mean Gaussian noise, and w^* denotes the true generating parameter that we would like to estimate.

We are given a set of n training samples $\{x_i, y_i\}_{i=1}^n$ that are generated by the above model with i.i.d. Z_i and distinct x_i. Our goal is to fit a linear model and get an estimate $\widehat{w}$ for the true parameter w^*. For all parts, assume that x_i's are given and fixed (not random).

For a given training set $\{x_i, y_i\}_{i=1}^n$, the ridge-regression estimate for w^* is defined by

$$\widehat{w}_\lambda = \arg\min_{w\in\mathbb{R}} \lambda w^2 + \sum_{i=1}^{n}(y_i - x_i w)^2 \qquad \text{with } \lambda \geq 0.$$

For the rest of the problem, assume that this has been solved and written in the form:

$$\widehat{w}_\lambda = \frac{S_{xy}}{s_x^2 + \lambda} \tag{2}$$

Figure 6: Example results with the input on the left and output on the right

where $S_{xy} = \sum_{i=1}^{n} x_i Y_i$ and $s_x^2 = \sum_{i=1}^{n} x_i^2$.
(This is given, no need to rederive it).

(a) (8 pts) **Compute the squared-bias of the ridge estimate $\widehat{w}_\lambda$ defined as follows**

$$\text{Bias}^2(\widehat{w}_\lambda) = (\mathbb{E}[\widehat{w}_\lambda] - w^*)^2. \tag{3}$$

It is fine if your answer depends on w^* or s_x, but it should not depend directly or indirectly on the realizations of the random Z noise. (So, no S_{xy} allowed.)

Hint: First compute the expectation of the estimate $\widehat{w}_\lambda$ over the noises Z in the observation.

Solution: To compute the expectation, we note that 1) the ridge-estimate is random because the observation Y is random, and 2) that $E[Y_i] = E[x_i w^* + Z_i] = x_i w^*$. Using these two facts and the linearity of expectation, we have

$$\begin{aligned}
E[\widehat{w}_\lambda] = E\left[\frac{\sum_{i=1}^n x_i Y_i}{\sum_{i=1}^n x_i^2 + \lambda}\right] &= \frac{1}{\sum_{i=1}^n x_i^2 + \lambda}\left[\sum_{i=1}^n E[x_i Y_i]\right] \\
&= \frac{1}{\sum_{i=1}^n x_i^2 + \lambda}\left[\sum_{i=1}^n x_i E[Y_i]\right] \\
&= \frac{1}{\sum_{i=1}^n x_i^2 + \lambda}\left[\sum_{i=1}^n x_i^2 w^*\right] \\
&= w^*\left(\frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda}\right).
\end{aligned}$$

We now compute the squared-bias as follows:

$$\begin{aligned}
(E[\widehat{w}_\lambda - w^*])^2 &= \left(w^*\left(\frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda}\right) - w^*\right)^2 \\
&= (w^*)^2 \frac{\lambda^2}{(\sum_{i=1}^n x_i^2 + \lambda)^2} \\
&= (w^*)^2 \frac{\lambda^2}{(s_x^2 + \lambda)^2}.
\end{aligned}$$

(b) (8 pts) **Compute the variance of the estimate $\widehat{w}_\lambda$ which is defined as**

$$\text{Var}(\widehat{w}_\lambda) = \mathbb{E}[(\widehat{w}_\lambda - \mathbb{E}[\widehat{w}_\lambda])^2]. \tag{4}$$

Hint: It might be useful to write $\widehat{w}_\lambda = \mathbb{E}[\widehat{w}_\lambda] + R$ for some random variable R.

Solution: We have

$$\begin{aligned}\widehat{w}_\lambda = \frac{\sum_{i=1}^n x_i Y_i}{\lambda + \sum_{i=1}^n x_i^2} &= \frac{\sum_{i=1}^n x_i^2 w^* + x_i Z_i}{\lambda + \sum_{i=1}^n x_i^2} \\ &= E[\widehat{w}_\lambda] + \underbrace{\frac{1}{\lambda + \sum_{i=1}^n x_i^2} \sum_{i=1}^n x_i Z_i}_{R},\end{aligned}$$

where R denotes the random variable defined in the hint.

Thus, we have

$$\begin{aligned}E(\widehat{w}_\lambda - E[\widehat{w}_\lambda])^2 &= \frac{1}{(\lambda + \sum_{i=1}^n x_i^2)^2} E(\sum_{i=1}^n x_i Z_i)^2 \\ &= \frac{1}{(\lambda + \sum_{i=1}^n x_i^2)^2} \left(\sum_{i=1}^n x_i^2 E(Z_i^2) + \sum_{i \neq j} x_i x_j E[Z_i Z_j] \right) \\ &= \frac{\sum_{i=1}^n x_i^2}{(\lambda + \sum_{i=1}^n x_i^2)^2} \\ &= \frac{s_x^2}{(\lambda + s_x^2)^2}.\end{aligned}$$

since $E(Z_i^2) = 1$ and $E(Z_i Z_j) = 0$ for $i \neq j$ as Z_i is independent of Z_j.

(c) (8 pts) **Describe how the squared-bias and variance of the estimate $\widehat{w}_\lambda$ change as we change the value of λ. What happens as $\lambda \to 0$? $\lambda \to \infty$? Is the bias increasing or decreasing? Is the variance increasing or decreasing? In what sense is there a bias/variance tradeoff?**

Solution: We have

$$\text{Bias}^2(\widehat{w}_\lambda) = (w^*)^2 \frac{\lambda^2}{(\sum_{i=1}^n x_i^2 + \lambda)^2} = (w^*)^2 \frac{1}{(s_x^2/\lambda + 1)^2}$$

$$\text{Var}(\widehat{w}_\lambda) = \frac{\sum_{i=1}^n x_i^2}{(\lambda + \sum_{i=1}^n x_i^2)^2} = \frac{s_x^2}{(\lambda + s_x^2)^2}$$

and thus clearly squared-bias increases with increase in λ and takes value 0 at 0. In fact, $\lambda = 0$ corresponds to the OLS case and you may recall that OLS for linear models is unbiased. (Students are not expected to make this observation in the exam, but it is stated here as a take away message.) Furthermore, for $\lambda \to \infty$, we have that bias-squared $\to (w^*)^2$. We can directly observe this fact, since $\lambda \to \infty$ implies that the ridge estimate would be 0 and hence the bias would be w^* and consequently the bias-squared would be $(w^*)^2$.

For the variance, we see that increasing λ reduces variance. This is intuitively correct too, because larger penalty forces the weight to shrink towards zero thereby reducing its scale and hence the variance too!

Thus, we see that a larger penalty in ridge-regression increases the squared-bias for the estimate and reduces the variance, and thus we observe a trade-off.

5 Hospital (25 points)

You work at hospital A. Your hospital has collected patient data to build a model to predict who is likely to get sepsis (a bad outcome). Specifically, the data set contains the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, and associated real number labels $\mathbf{y} \in \mathbb{R}^n$, where n is the number of patients you are learning from and d is the number of features describing each patient. You plan to fit a linear regression model $\widehat{y} = \mathbf{w}^\top \mathbf{x}$ that will enable you to predict a label for future, unseen patients (using their feature vectors).

However, your hospital has only started collecting data a short time ago. Consequently the model you fit is not likely to be particularly accurate. Hospital B has exactly the same set up as your hospital (i.e., their patients are drawn from the same distribution as yours and they have the same measurement tools). For privacy reasons, Hospital B will not share their data. However, they tell you that they have trained a linear model on their own sepsis-relevant data: ($\mathbf{X}_B$ and $\mathbf{y}_B$) and are willing to share their learned model $\widehat{y} = \widehat{\mathbf{w}}_B^\top \mathbf{x}$ with you. In particular, Hospital B shares their entire Gaussian posterior distribution on $\mathbf{w}$ with you: $\mathcal{N}(\widehat{\mathbf{w}}_B, \mathbf{\Psi})$.

(a) (10 pts) Assume that we use the posterior from Hospital B as our own prior distribution for $\mathbf{w} \sim \mathcal{N}(\widehat{\mathbf{w}}_B, \mathbf{\Psi})$. Suppose that our Hospital A model is given by $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$, where the noise, $\boldsymbol{\epsilon}$, has an assumed distribution $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. **Derive the MAP estimate $\widehat{\mathbf{w}}$ for $\mathbf{w}$ using Hospital A's data $\mathbf{X}, \mathbf{y}$ and the prior information from Hospital B.**

HINT: Recall that traditional ridge regression could be derived from a MAP perspective, where the parameter $\mathbf{w}$ *has a zero mean Gaussian prior distribution with a scaled identity covariance. How could you use reparameterization (i.e. change of variables) for the problem here?*

Solution:

The overall point of the Hospital problem was to highlight the equivalence between a prior distribution on parameter(s) and "pseudo" data. Stating a prior can in general be reformulated as pretending to have seen some made up data, often referred to as pseudo-data. This point was highlighted in homework. In general, the tighter the prior distribution (e.g. the less variance), the more sure we are that it is "correct". Thus it also corresponds to having seen more "pseudo" data.

In part (a) the question is really just asking you to crank through the MAP estimate. But the pedagogical insight was to appreciate that in the hospital setting, the "pseudo" data actually corresponded to real data and helped us get around a privacy problem! One could derive it from "scratch" as done in lecture, with the specific prior given in the question, as follows (but there is a much easier solution we go over next):

From first principles solution:

Let D be the dataset, i.e. $\mathbf{X}$ and $\mathbf{y}$. According to the Bayes' Rule and ignoring the terms not related to $\mathbf{w}$, we have:

$$\begin{aligned}\log p(\mathbf{w}|D) &\propto \log p(D|\mathbf{w}) + \log p(\mathbf{w}) \\ &\propto -(\mathbf{X}\mathbf{w}-\mathbf{y})^\top(\mathbf{X}\mathbf{w}-\mathbf{y}) - (\mathbf{w}-\mathbf{w}_B)^\top\mathbf{\Psi}_B^{-1}(\mathbf{w}-\mathbf{w}_B) \\ &\propto (-\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} + 2\mathbf{y}^\top\mathbf{X}\mathbf{w}) - (\mathbf{w}^\top\mathbf{\Psi}_B^{-1}\mathbf{w} - 2\mathbf{w}_B{}^\top\mathbf{\Psi}_B^{-1}\mathbf{w}) \\ &\propto -\mathbf{w}^\top(\mathbf{X}^\top\mathbf{X} + \mathbf{\Psi}_B^{-1})\mathbf{w} + 2(\mathbf{y}^\top\mathbf{X} + \mathbf{w}_B{}^\top\mathbf{\Psi}_B^{-1})\mathbf{w}\end{aligned} \tag{5}$$

Setting the derivative of $\log p(\mathbf{w}|D)$ to zero and solve the equation, we get:
$\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X} + \mathbf{\Psi}_B^{-1})^{-1}(\mathbf{X}^\top\mathbf{y} + \mathbf{\Psi}_B^{-1}\mathbf{w}_B)$

However, an easier way to solve this question was to simply do a change of variables on w such that its prior became zero-mean and we could just read off the solution derived in lecture for MAP with arbitrary prior variance. In particular let $\mathbf{v} = \mathbf{w} - \mathbf{w}_b$ (i.e. $\mathbf{w} = \mathbf{v} + \mathbf{w} =$), then we have

$$\begin{aligned}\log p(\mathbf{w}|D) &\propto \log p(D|\mathbf{w}) + \log p(\mathbf{w}) \\ &= \log N(\mathbf{y}; \mathbf{X}(\mathbf{v}+\mathbf{w}_B), \mathbf{I}) + \log N(\mathbf{v}; 0, \Psi)\end{aligned} \tag{6}$$

From which we read off that $\hat{w} = w_B + (X^TX + \Psi^{-1})^{-1}X^T(y - Xw_b)$, appealing to the formula we derived in class and appearing in the notes.

A third solution would have been to use the change of variables

$$\mathbf{w} = \Psi^{1/2}\mathbf{v} + \mathbf{w}_B$$

to reduce to standard ridge with $\lambda = 1$ and read the solution off directly as done in lecture–see the lecture notes only without the $\mathbf{w}_B$ offset term.

(b) (15 pts) Now, for simplicity, consider $d = 1$ so that the w is a scalar parameter. Suppose that instead of giving you their posterior distribution, Hospital B only gave you their mean $\widehat{w}_B$. How can you use this information to help fit your model? **Describe in detail how you should use your own hospital's patient data and combine it with the mean $\widehat{w}_B$ from Hospital B in a procedure to find your own $\widehat{w}$ for predicting sepsis in Hospital A.**

Hint 1: You might want to consider introducing an appropriate hyperparameter and doing what you usually do with hyperparameters.

Hint 2: What does the λ hyperparameter in ridge-regression correspond to from a probabilistic perspective?

Solution:

This problem was asking students to remember that an unknown variance term in a prior corresponds to a hyperparameter. After all λ in traditional ridge regression corresponded to the reciprocal of the prior variance. This suggests that we should introduce a hyperparameter ψ into the problem for the unknown variance. Once we have done this, we can do what we always do with hyperparameters: use cross-validation to pick an appropriate value for them. Putting these ideas together, we get the solution:

(a) Divide the data $\mathbf{X}$ and $\mathbf{y}$ from Hospital A into k folds. (For example, we could even have $k = n$ here.)

(b) For different potential values of ψ do the following:

i. Repeat over the k folds:

A. Let the training data $\mathbf{X}_{train}$ be all of $\mathbf{X}$ except the j-th fold. Similarly, let the training data $\mathbf{y}_{train}$ be all of $\mathbf{y}$ except the j-th fold.

B. Use the modified ridge regression of Part (a) using the current potential value for the hyperparameter ψ to fit a candidate $\widehat{w}_\psi$ using $\mathbf{X}_{train}$ and $\mathbf{y}_{train}$.

C. Evaluate the validation error on the validation data representing the j-th fold of $\mathbf{X}$ and $\mathbf{y}$.

ii. Average together the validation error (squared errors) for the different fold to get an average validation error estimate for this particular value for the hyperparameter ψ.

(c) Choose the hyperparameter value ψ with the lowest average validation error.

(d) Retrain the model using the chosen value for ψ on the entire training data $\mathbf{X}$ and $\mathbf{y}$ to get our final $\widehat{w}$ for hospital A.

We also would accept other ways of using a hyperparameter. For example, realizing that the effect of the hyperparameter in the scalar case is simply to "shrink" the OLS estimate towards $\widehat{w}_B$, it would also be reasonable to directly define the amount of interpolation between the two as the hyperparameter. The important thing is that the validation data for evaluating hyperparameters be chosen in a way that is not cross-contaminated with data being used to set the parameters themselves. So, it is still vital that the OLS estimate be done with different data and cross-validation be used.

Given that the context here is data poor, it does make sense to use k-fold cross-validation instead of simply splitting the data into a training and validation set once. However on the exam, no points were deducted for just using a simple validation approach.

6 Ridge regression vs. PCA (24 points)

Assume we are given n training data points $(\mathbf{x}_i, y_i)$. We collect the target values into $\mathbf{y} \in \mathbb{R}^n$, and the inputs into the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ where the rows are the $d-$dimensional feature vectors $\mathbf{x}_i^\top$ corresponding to each training point. Furthermore, assume that $\frac{1}{n}\sum_{i=1}^n \mathbf{x_i} = \mathbf{0}$, $n > d$ and $\mathbf{X}$ has rank d.

In this problem we want to compare two procedures: The first is ridge regression with hyperparameter λ, while the second is applying ordinary least squares after using PCA to reduce the feature dimension from d to k (we give this latter approach the short-hand name k-PCA-OLS where k is the hyperparameter).

Notation: The singular value decomposition of $\mathbf{X}$ reads $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times d}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$. We denote by $\mathbf{u}_i$ the n-dimensional column vectors of $\mathbf{U}$ and by $\mathbf{v}_i$ the $d-$dimensional column vectors of $\mathbf{V}$. Furthermore the diagonal entries $\sigma_i = \Sigma_{i,i}$ of $\mathbf{\Sigma}$ satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d > 0$. For notational convenience, assume that $\sigma_i = 0$ for $i > d$.

(a) (6 pts) It turns out that the ridge regression optimizer (with $\lambda > 0$) in the $\mathbf{V}$-transformed coordinates

$$\widehat{\mathbf{w}}_{\text{ridge}} = \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{V}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

has the following expression:

$$\widehat{\mathbf{w}}_{\text{ridge}} = \text{diag}(\frac{\sigma_i}{\lambda + \sigma_i^2})\mathbf{U}^\top\mathbf{y}. \tag{7}$$

Use $\widehat{y}_{test} = \mathbf{x}_{test}^\top \mathbf{V}\widehat{\mathbf{w}}_{\text{ridge}}$ to denote the resulting prediction for a hypothetical $\mathbf{x}_{test}$. Using (7) and the appropriate scalar $\{\beta_i\}$, this can be written as:

$$\widehat{y}_{test} = \mathbf{x}_{test}^\top \sum_{i=1}^d \mathbf{v}_i \beta_i \mathbf{u}_i^\top \mathbf{y}. \tag{8}$$

What are the $\beta_i \in \mathbb{R}$ for this to correspond to (7) from ridge regression?

Solution:

The resulting prediction for ridge reads

$$\hat{\mathbf{y}}_{\text{ridge}} = \mathbf{x}^\top \mathbf{V}\,\text{diag}\left(\frac{\sigma_i}{\lambda + \sigma_i^2}\right)\mathbf{U}^\top\mathbf{y}$$

$$= \mathbf{x}^\top \sum_{i=1}^d \frac{\sigma_i}{\lambda + \sigma_i^2}\mathbf{v}_i\mathbf{u}_i^\top\mathbf{y}$$

Therefore we have $\beta_i = \frac{\sigma_i}{\lambda + \sigma_i^2}$ for $i = 1, \ldots, d$.

(b) (12 pts) Suppose that we do k-PCA-OLS — i.e. ordinary least squares on the reduced k-dimensional feature space obtained by projecting the raw feature vectors onto the $k < d$ principal components of the covariance matrix $\mathbf{X}^\top \mathbf{X}$. Use $\widehat{y}_{test}$ to denote the resulting prediction for a hypothetical $\mathbf{x}_{test}$,

It turns out that the learned k-PCA-OLS predictor can be written as:

$$\widehat{y}_{test} = \mathbf{x}_{test}^\top \sum_{i=1}^{d} \mathbf{v}_i \beta_i \mathbf{u}_i^\top \mathbf{y}. \tag{9}$$

Give the $\beta_i \in \mathbb{R}$ coefficients for k-PCA-OLS. Show work.

Hint 1: some of these β_i will be zero. Also, if you want to use the compact form of the SVD, feel free to do so if that speeds up your derivation.

Hint 2: some inspiration may be possible by looking at the next part for an implicit clue as to what the answer might be.

Solution: The OLS on the k-PCA-reduced features reads

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{V}_k\mathbf{w} - \mathbf{y}\|_2^2$$

where the columns of $\mathbf{V}_k$ consist of the first k eigenvectors of $\mathbf{X}$.

In the following we use the compact form SVD, that is note that one can write

$$\begin{aligned}\mathbf{X} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V} \\ &= \mathbf{U}_d\mathbf{\Sigma}_d\mathbf{V}\end{aligned}$$

where $\mathbf{\Sigma}_d = \mathrm{diag}(\sigma_i)$ for $i = 1, \ldots, d$ and $\mathbf{U}_d$ are the first d columns of $\mathbf{U}$. In general we use the notation $\mathbf{\Sigma}_k = \mathrm{diag}(\sigma_i)$ for $i = 1, \ldots, k$.

Apply OLS on the new matrix $\mathbf{X}\mathbf{V}_k$ to obtain

$$\begin{aligned}\widehat{\mathbf{w}}_{\text{PCA}} &= [(\mathbf{X}\mathbf{V}_k)^\top(\mathbf{X}\mathbf{V}_k)]^{-1}(\mathbf{X}\mathbf{V}_k)^\top\mathbf{y} \\ &= [\mathbf{V}_k^\top\mathbf{V}\mathbf{\Sigma}_d^2\mathbf{V}^\top\mathbf{V}_k]^{-1}\mathbf{V}_k^\top\mathbf{X}^\top\mathbf{y} \\ &= \mathbf{\Sigma}_k^{-1}\mathbf{U}_k^\top\mathbf{y} = \widetilde{\mathbf{\Sigma}}_k^{-1}\mathbf{U}^\top\mathbf{y}\end{aligned}$$

where $\widetilde{\mathbf{\Sigma}}_k = \begin{pmatrix}\mathbf{\Sigma}_k & 0\end{pmatrix}$

The resulting prediction for PCA reads (note that you need to project it first!)

$$\begin{aligned}\widehat{\mathbf{y}}_{\text{PCA}} &= \mathbf{x}^\top\mathbf{V}_k\widehat{\mathbf{w}}_{\text{PCA}} \\ &= \mathbf{x}^\top\mathbf{V}_k\mathbf{\Sigma}_k^{-1}\mathbf{U}_k^\top\mathbf{y} \\ &= \mathbf{x}^\top\sum_{i=1}^{k}\frac{1}{\sigma_i}\mathbf{v}_i\mathbf{u}_i^\top\mathbf{y}\end{aligned}$$

and hence $\beta_i = \frac{1}{\sigma_i}$ if $i \leq k$ and $\beta_i = 0$ for $i = k+1, \ldots, d$.

(c) (6 pts) For the following part, $d = 5$. The following $\boldsymbol{\beta} := (\beta_1, \ldots, \beta_5)$ (written out to two significant figures) are the results of OLS (i.e. what we would get from ridge regression in the limit $\lambda \to 0$), λ-ridge-regression, and k-PCA-OLS for some $\mathbf{X}, \mathbf{y}$ (identical for each method) and $\lambda = 1, k = 3$. **Write down which procedure was used for each of the three sub-parts below.**

We hope this helps you intuitively see the connection between these three methods.

Hint: It is not necessary to find the singular values of $\mathbf{X}$ *explicitly, or to do any numerical computations at all.*

(i) $\boldsymbol{\beta} = (0.01, 0.1, 0.5, 0.1, 0.01)$

(ii) $\boldsymbol{\beta} = (0.01, 0.1, 1, 0, 0)$

(iii) $\boldsymbol{\beta} = (0.01, 0.1, 1, 10, 100)$

Solution: Ridge, 3-PCA-OLS, OLS.

Reasoning: The prediction for OLS is the same as for PCA with $k = d$.

$$\hat{\mathbf{y}}_{OLS} = \mathbf{x}^\top \sum_{i=1}^{d} \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^\top \mathbf{y}$$

Putting all pieces together, we can thus see that PCA does "hard shrinkage" or "hard cutoff" (i.e. sets to zero) of the last $k+1, \ldots, d$ coefficients β_i, whereas ridge regression does "soft shrinkage" (i.e. shrinks towards zero) of the coefficients.

7 Kernel PCA (24 points)

In lectures, discussion, and homework, we learned how to use PCA to do dimensionality reduction by projecting the data to a subspace that captures most of the variability. This works well for data that is roughly Gaussian shaped, but many real-world high dimensional datasets have underlying

low-dimensional structure that is not well captured by linear subspaces. However, when we lift the raw data into a higher-dimensional feature space by means of a nonlinear transformation, the underlying low-dimensional structure once again can manifest as an approximate subspace. Linear dimensionality reduction can then proceed. As we have seen in class so far, kernels are an alternate way to deal with these kinds of nonlinear patterns without having to explicitly deal with the augmented feature space. This problem asks you to discover how to apply the "kernel trick" to PCA.

Let $\mathbf{X} \in \mathbb{R}^{n \times \ell}$ be the data matrix, where n is the number of samples and ℓ is the dimension of the raw data. Namely, the data matrix contains the data points $\mathbf{x}_j \in \mathbb{R}^\ell$ as rows

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix} \in \mathbb{R}^{n \times \ell}. \tag{10}$$

(a) (5 pts) **Compute $\mathbf{X}\mathbf{X}^\top$ in terms of the singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ where $\mathbf{U} \in \mathbb{R}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times \ell}$ and $\mathbf{V} \in \mathbb{R}^{\ell \times \ell}$.** Notice that $\mathbf{X}\mathbf{X}^\top$ is the matrix of pairwise Euclidean inner products for the data points. **How would you get $\mathbf{U}$ if you only had access to $\mathbf{X}\mathbf{X}^\top$?**

Solution: By plugging in the compact SVD decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and using $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ we get

$$\mathbf{X}^\top\mathbf{X} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top.$$

Similarly with $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$ we get

$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top.$$

Notice from the last line that $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^\top$ with eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_\ell^2$ where $\sigma_1, \sigma_2, \ldots, \sigma_d$ are the singular values of $\mathbf{X}$ and can therefore be computed by performing an eigendecomposition of $\mathbf{X}\mathbf{X}^\top$.

(b) (7 pts) Given a new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, one central use of PCA is to compute the projection of $\mathbf{x}_{test}$ onto the subspace spanned by the k top singular vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$.

Express the scalar projection $z_j = \mathbf{v}_j^\top \mathbf{x}_{test}$ onto the j-th principal component as a function of the inner products

$$\mathbf{X}\mathbf{x}_{test} = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{x}_{test} \rangle \\ \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_{test} \rangle \end{pmatrix}. \tag{11}$$

Assume that all diagonal entries of $\mathbf{\Sigma}$ are nonzero and non-increasing, that is $\sigma_1 \geq \sigma_2 \geq \cdots > 0$.

Hint: Express $\mathbf{V}^\top$ in terms of the singular values $\mathbf{\Sigma}$, the left singular vectors $\mathbf{U}$ and the data matrix $\mathbf{X}$. If you want to use the compact form of the SVD, feel free to do so.

Solution: By multiplying the compact SVD $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ on both sides with $\mathbf{U}^\top$, we get $\mathbf{U}^\top\mathbf{X} = \mathbf{\Sigma}\mathbf{V}^\top$ and multiplying both sides of the new equation with $\mathbf{\Sigma}^{-1}$, we obtain

$$\mathbf{V}^\top = \mathbf{\Sigma}^{-1}\mathbf{U}^\top\mathbf{X}.$$

Therefore we get

$$z_j = \mathbf{v}_j^\top \mathbf{x}_{test} = \frac{1}{\sigma_j}\mathbf{u}_j^\top \mathbf{X}\mathbf{x}_{test}$$

.

(c) (12 pts) How would you define kernelized PCA for a general kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ (to replace the Euclidean inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$)? For example, the RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\delta^2}\right)$.

Describe this in terms of a procedure which takes as inputs the training data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^\ell$ and the new test point $\mathbf{x}_{test} \in \mathbb{R}^\ell$, and outputs the analog of the previous part's z_j coordinate in the kernelized PCA setting. You should include how to compute $\mathbf{U}$ from the data, as well as how to compute the analog of $\mathbf{X}\mathbf{x}_{test}$ from the previous part.

Invoking the SVD or computing eigenvalues/eigenvectors is fine in your procedure, as long as it is clear what matrix is having its SVD or eigenvalues/eigenvectors computed. The kernel $k(\cdot, \cdot)$ can be used as a black-box function in your procedure as long as it is clear what arguments it is being given.

Solution: For kernelizing PCA, we replace inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with $k(\mathbf{x}_i, \mathbf{x}_j)$ and $\langle \mathbf{x}_i, \mathbf{x}_{test} \rangle$ with $k(\mathbf{x}_i, \mathbf{x}_{test})$, the procedure is then:

(a) Obtain the vectors $\mathbf{u}_j$ as eigenvectors from the eigendecomposition of the kernelized counterpart of the Gram matrix: $\mathbf{K} \in \mathbb{R}^{n \times n}$ with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The eigenvalues should be sorted in decreasing order. They are all non-negative real numbers because of the properties of kernels — the K matrix must be positive semi-definite.

(b) Kernelize the inner products $z_j = \frac{1}{\sigma_j}\mathbf{u}_j^\top \mathbf{X}\mathbf{x}_{test}$ from the previous part by using:

$$z_j = \frac{1}{\sigma_j}\mathbf{u}_j^\top \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_{test}) \\ k(\mathbf{x}_2, \mathbf{x}_{test}) \\ \vdots \\ k(\mathbf{x}_n, \mathbf{x}_{test}) \end{pmatrix}, \tag{12}$$

where the σ_j are the square roots of the eigenvalues for the martix K above generated by using the kernel on all pairs of training points. Because these are non-negative real numbers, the square root is well defined.

8 Multiple Choice Questions (14 points)

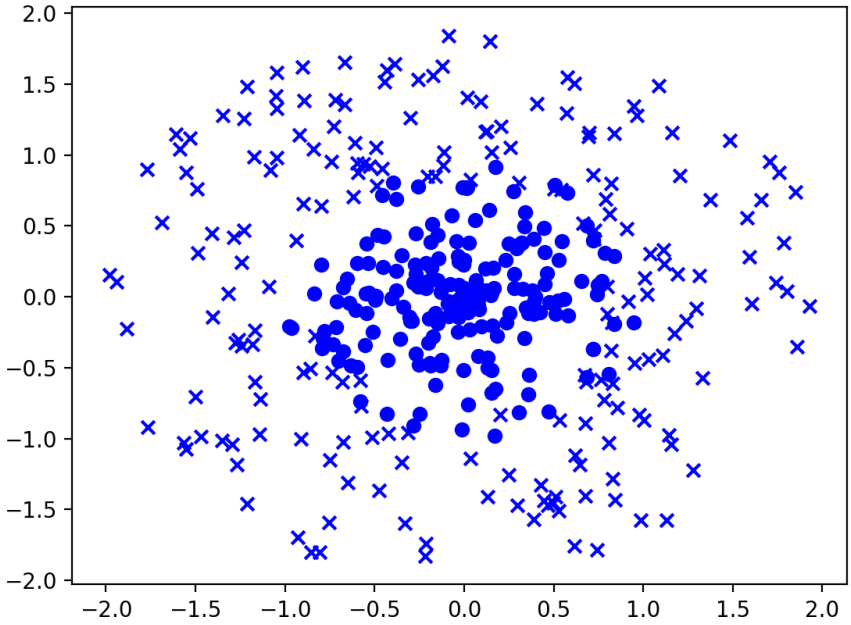
For these questions, select **all** the answers which are correct. You will get full credit for selecting all the right answers. On some questions, real-valued partial credit will be assigned. You will be graded on your **best seven of nine, so feel free to skip up to two of them.**

(a) Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $n \geq d$. Suppose $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the singular value decomposition of $\mathbf{X}$ where $\sigma_i = \Sigma_{i,i}$ are the diagonal entries of $\mathbf{\Sigma}$ and satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$ while $\mathbf{u}_i$ and $\mathbf{v}_i$ are the ith columns of $\mathbf{U}$ and $\mathbf{V}$ respectively. **Which of the following is the rank k approximation to $\mathbf{X}$ that is best in the Froebenius norm.** That is, which low rank approximation, $\mathbf{X}_k$, for $\mathbf{X}$ yields the lowest value for $||\mathbf{X} - \mathbf{X}_k||_F^2$?

◯ $\sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_{n-i}^\top$ ◯ $\sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ ◯ $\sum_{i=d-k+1}^{d} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ ◯ $\sum_{i=1}^{k} \sigma_i \mathbf{u}_{n-i} \mathbf{v}_i^\top$

Solution: The solution comes from the Eckhart-Young theorem which states we should use the singular vector expansion from 1 to k, $\sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$

(b) Consider a simple dataset of points $(x_i, y_i) \in \mathbb{R}^2$, each associated with a label b_i which is -1 or $+1$. The dataset was generated by sampling data points with label -1 from a disk of radius 1.0 (shown as filled circles in the figure) and data points with label $+1$ from a ring with inner radius 0.8 and outer radius 2.0 (shown as crosses in the figure). **Which set of polynomial features would be best for performing linear regression, assuming at least as much data as shown in the figure?**



◯ $1, x_i$

◯ $1, x_i, y_i$

◯ $1, x_i, y_i, x_i^2, x_i y_i, y_i^2$

◯ $1, x_i, y_i, x_i^2, x_i y_i, y_i^2, x_i^3, y_i^3, x_i^2 y_i, x_i y_i^2$

Solution: Because the decision boundary was created by circles, and because there was enough data to justify modelling circles, we should use precisely order two polynomial features: $1, x_i^2, x_i y_i, y_i^2$. (Aside: If we had not specified that the points were created by circles, it could have been reasonable also add in possibly up to cubic polynomials, but this was considered incorrect here. Even if we had specified that the boundaries were created by circles, but

had given only say two data points, then even a second order polynomial would not have made sense.)

(c) **Which of the following is a valid kernel function** for vectors of the same length, $\mathbf{x}$ and $\mathbf{y}$?

- ◯ $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$
- ◯ $k(\mathbf{x}, \mathbf{y}) = e^{-\frac{1}{2}||\mathbf{x}-\mathbf{y}||_2^2}$
- ◯ $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^\top \mathbf{y})^p$ for some degree p
- ◯ $k(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y}) - k_2(\mathbf{x}, \mathbf{y})$ for valid kernels k_1 and k_2.

Solution: All except $k(\mathbf{x}, \mathbf{y}) = k_1(x, y) - k_2(x, y)$ for valid kernels k_1 and k_2. This is not a valid kernel because valid kernel functions are not in general closed under subtraction, rather they are closed only under positive, linear combinations.

(d) During training of your model, both independent variables in the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and dependent target variables $\mathbf{y} \in \mathbb{R}^n$ are corrupted by noise. At test time, the data points you are computing predictions for, $\mathbf{x}_{test}$, are noiseless. **Which method(s) should you use to estimate the value of $\hat{\mathbf{w}}$ from the training data in order to make the most accurate predictions $\mathbf{y}_{test}$ from the noiseless test input data, $\mathbf{X}_{test}$?** Assume that you make predictions using $\mathbf{y}_{test} = \mathbf{X}_{test}\hat{\mathbf{w}}$.

- ◯ OLS
- ◯ Ridge regression
- ◯ Weighted Least Squares
- ◯ TLS

Solution: TLS since the test data has no noise while the training data does. This means that we are looking for the true relationship (as opposed to the best predictor for the training data) and total least squares is the one that does this.

Because TLS was shown to be equivalent to ridge-regression with a negative regularizer, points were not taken off for marking that as well.

(e) Assume you have n input data points, each with d high quality features ($\mathbf{X} \in \mathbb{R}^{n \times d}$) and associated labels ($\mathbf{y} \in \mathbb{R}^n$). Suppose that $d \gg n$ and that you want to learn a linear predictor. **Which of these approaches would help you to avoid overfitting?**

- ◯ Preprocess $\mathbf{X}$ using $k \ll n$ random projections
- ◯ Preprocess $\mathbf{X}$ using PCA with $k \ll n$ components.
- ◯ Preprocess $\mathbf{X}$ using PCA with n components.
- ◯ Add polynomial features
- ◯ Use a kernel approach
- ◯ Add a ridge penalty to OLS
- ◯ Do weighted least squares

Solution: The goal here is simple dimensionality reduction for overfitting avoidance. As the earlier problem in the exam showed, ridge regression can be viewed as a softer form of k-PCA-OLS where the different dimensions are downweighted softly rather than as a strict truncation. So both PCA and ridge are clearly valid approaches to reduce overfitting.

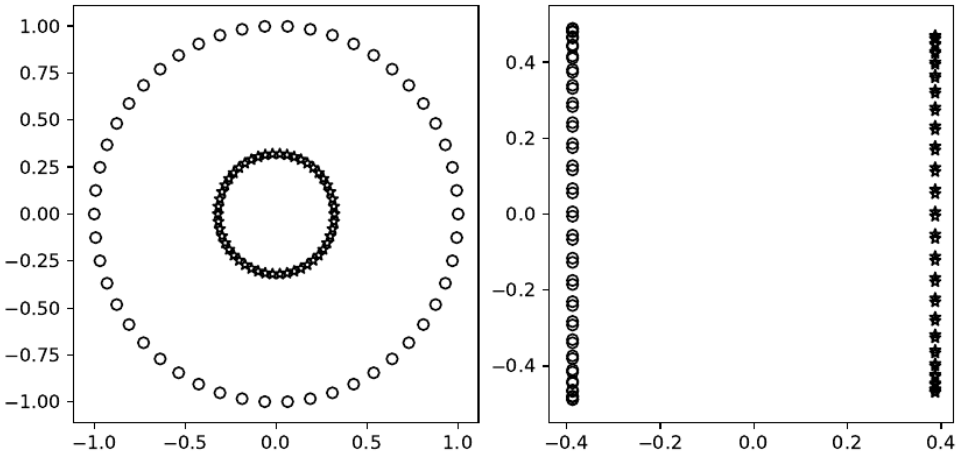
However, because the problem said that these are high-quality features, what you know about random projection also applies and so they too can be useful for dimensionality reduction.

The other answers are mostly wrong. If you use n components of PCA, there are still too many parameters relative to the data points. And so some overfitting will still occur.

Adding polynomial features makes the overfitting issue worse and not better, while weighing samples doesn't help us in any way.

We did not penalize for also marking kernel approaches since you know from lecture that the right kernel can also regularize because the kernel approach serves the same purpose as choosing features and priors together.

(f) **Which methods could yield a transformation to go from the two-dimensional data on the left to the two-dimensional data on the right?**



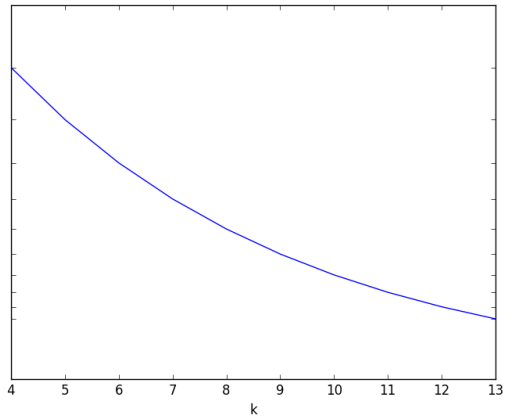
- ◯ Random projections
- ◯ PCA
- ◯ Use of a kernel
- ◯ Adding polynomial features

Solution:

The plot here was literally obtained by doing RBF kernel-PCA on the data and choosing the two dominant components. PCA is clearly being used, but also something that allows us to get nonlinear relationships. Either the right kernel or polynomial features would suffice since circles are involved.

Random projections would not help here since they are linear.

(g) Your friend is training a machine learning model to predict disease severity based on k different health indicators. She generates the following plot, where the value of k is on the x axis.



Which of these might the y axis represent?

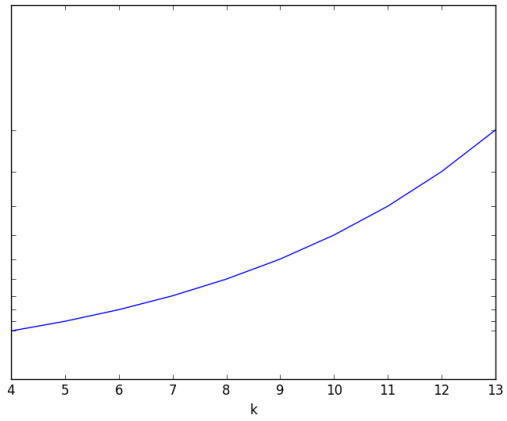
- ◯ Training Error
- ◯ Validation Error
- ◯ Bias
- ◯ Variance

Solution: As k increases, the number of features in the ML model increases.

Full credit was given to anyone who responded **Training Error, Bias, Validation Error** or **Training Error, Bias**.

It is most important to recognize that training error and bias decrease with the number of features in the ML model. The reason that validation error is a correct answer is because it may be the case that the behavior changes after $k = 13$. Given this graph, it is not possible to know for sure.

(h) Your friend is training a machine learning model to predict disease severity based on k different health indicators. She generates the following plot, where the value of k is on the x axis.



Which of these might the y axis represent?

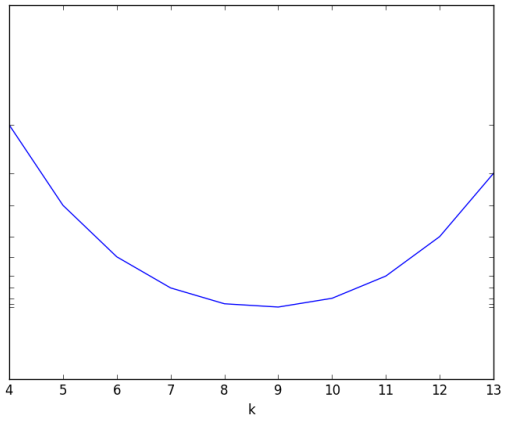
- ◯ Training Error
- ◯ Validation Error
- ◯ Bias
- ◯ Variance

Solution: As k increases, the number of features in the ML model increases.

Full credit was given to anyone who responded **Variance, Validation Error** or **Variance**.

It is most important to recognize that variance increases with the number of features in the ML model. The reason that validation error is a correct answer is because it may be the case that the behavior changes before $k = 4$. Given this graph, it is not possible to know for sure.

(i) Your friend is training a machine learning model to predict disease severity based on k different health indicators. She generates the following plot, where the value of k is on the x axis.



Which of these might the y axis represent?

- ◯ Training Error
- ◯ Validation Error
- ◯ Bias
- ◯ Variance

Solution: As k increases, the number of features in the ML model increases.

Full credit was given to anyone who responded **Validation Error**.

9 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze,

Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.