Question 1. Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to assignment on Gradescope, "¡¡title¿¿ Write-Up". If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results, "¡¡title¿¿ Code".
3. If there is a test set, submit your test set evaluation results, "¡¡title¿¿ Test Set".

After you've submitted your homework, watch out for the self-grade form.

(a) Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked with Weiran Liu. Because Prof. Sahai said I will not regret if I do the first question before the midterm, I will do it.

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

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.

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.

Signature: [illegible]

Question 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 $\vec{X}, \vec{Y} \in \mathbb{R}^d$. Any linear combination of the coordinates of the random vector $\vec{X}$ can be written as $\vec{a}^\top \vec{X}$, and similarly, a linear combination of the coordinates of the random vector $\vec{Y}$ can be written as $\vec{b}^\top \vec{Y}$. Note that $\vec{a}^\top \vec{X}, \vec{b}^\top \vec{Y} \in \mathbb{R}$ are scalar random variables.

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

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

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 Gausian 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 $\vec{X}$ of jointly Gaussian zero-men random variables, then we are going to pick a weight vector $\vec{w}$ that is aligned with $\mathbb{E}[\vec{X}Y]$. It is straightforward algebra to see that this direction maximizes the correlation coefficient $\rho(\frac{\vec{w}^\top}{\|\vec{w}\|}\vec{X}, Y)$.

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 $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$, we know that the singular values are given by the diagonal entries of Σ, 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 $\mathbf{A} = \sum_{i=1}^{d} \sigma_i \vec{u}_i \vec{v}_i^\top$, where the ith singular value is denoted by $\sigma_i = \Sigma_{ii}$ and $\vec{u}_i$ and $\vec{v}_i$ are the ith left and right singular vectors, respectively.

$$\begin{aligned}
\mathbf{A} &= \mathbf{U}\Sigma\mathbf{V}^\top \\
&= \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \dots & \vec{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_d \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vec{v}_1^\top \\ \vec{v}_2^\top \\ \vdots \\ \vec{v}_d^\top \end{bmatrix}
\end{aligned}$$

$$
\begin{aligned}
&= \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \dots & \vec{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 \vec{v}_1^\top \\ \sigma_2 \vec{v}_2^\top \\ \vdots \\ \sigma_d \vec{v}_d^\top \\ 0 \\ \vdots \end{bmatrix} \\
&= \sum_{i=1}^{d} \sigma_i \vec{u}_i \vec{v}_i^\top
\end{aligned}
$$

(b) **With the setup above, show that**
i) $\mathbf{A}^\top\mathbf{A}$ has i-th eigenvalue σ_i^2, with associated eigenvector $\vec{v}_i$.
ii) $\mathbf{A}\mathbf{A}^\top$ has i-th eigenvalue σ_i^2, with associated eigenvector $\vec{u}_i$.
Notice that both of the above matrices are symmetric.

Let's define $\Sigma^\top = [\Sigma_d, 0]$
i)

$$\begin{aligned}\mathbf{A}^\top\mathbf{A} &= \mathbf{V}\Sigma^\top\mathbf{U}^\top\mathbf{U}\Sigma\mathbf{V}^\top \\ &= \mathbf{V}\Sigma^\top\Sigma\mathbf{V}^\top \\ &= \mathbf{V}\begin{bmatrix}\Sigma_d & 0\end{bmatrix}\begin{bmatrix}\Sigma_d \\ 0\end{bmatrix}\mathbf{V}^\top \\ &= \mathbf{V}\Sigma_d^2\mathbf{V}^\top\end{aligned}$$

Here we have $\mathbf{A}^\top\mathbf{A}\mathbf{V} = \mathbf{V}\Sigma_d^2$. This is the matrix form definition of eigenvalues and eigenvectors. More specifically we have:

$$\mathbf{A}^\top\mathbf{A}\begin{bmatrix}\vec{v}_1^\top \\ \vec{v}_2^\top \\ \vdots \\ \vec{v}_d^\top\end{bmatrix} = \begin{bmatrix}\vec{v}_1^\top \\ \vec{v}_2^\top \\ \vdots \\ \vec{v}_d^\top\end{bmatrix}\begin{bmatrix}\sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_d\end{bmatrix}$$

$$\begin{bmatrix}\mathbf{A}^\top\mathbf{A}\vec{v}_1^\top \\ \mathbf{A}^\top\mathbf{A}\vec{v}_2^\top \\ \vdots \\ \mathbf{A}^\top\mathbf{A}\vec{v}_d^\top\end{bmatrix} = \begin{bmatrix}\sigma_1^2\vec{v}_1^\top \\ \sigma_2^2\vec{v}_2^\top \\ \vdots \\ \sigma_d^2\vec{v}_d^\top\end{bmatrix}$$

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

$$\begin{aligned}\mathbf{A}\mathbf{A}^\top &= \mathbf{U}\Sigma^\top\mathbf{V}^\top\mathbf{V}\Sigma\mathbf{U}^\top \\ &= \mathbf{U}\Sigma^\top\Sigma\mathbf{U}^\top \\ &= \mathbf{V}\begin{bmatrix}\Sigma_d & 0\end{bmatrix}\begin{bmatrix}\Sigma_d \\ 0\end{bmatrix}\mathbf{U}^\top \\ &= \mathbf{U}\Sigma_d^2\mathbf{U}^\top\end{aligned}$$

We can see it's the same as above so we have:

$$\mathbf{A}\mathbf{A}^\top\begin{bmatrix}\vec{u}_1^\top \\ \vec{u}_2^\top \\ \vdots \\ \vec{u}_d^\top\end{bmatrix} = \begin{bmatrix}\vec{u}_1^\top \\ \vec{u}_2^\top \\ \vdots \\ \vec{u}_d^\top\end{bmatrix}\begin{bmatrix}\sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_d\end{bmatrix}$$

$$\begin{bmatrix}\mathbf{A}^\top\mathbf{A}\vec{u}_1^\top \\ \mathbf{A}^\top\mathbf{A}\vec{u}_2^\top \\ \vdots \\ \mathbf{A}^\top\mathbf{A}\vec{u}_d^\top\end{bmatrix} = \begin{bmatrix}\sigma_1^2\vec{u}_1^\top \\ \sigma_2^2\vec{u}_2^\top \\ \vdots \\ \sigma_d^2\vec{u}_d^\top\end{bmatrix}$$

From the formula above, we can conclude that $\mathbf{A}\mathbf{A}^\top$ has i-th eigenvalue σ_i^2, with associated eigenvector $\vec{u}_i$.

(c) Use the first part to show that

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

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

Additionally, show that if A has a unique maximum singular value, then the maximizers $(\vec{u}^*, \vec{v}^*)$ 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 $\vec{u} \in \mathbb{R}^n : \|\vec{u}\|_2 = 1$ as a linear combination of left singular vectors $\{\vec{u}_i\}$ of the matrix A, and any vector $\vec{v} \in \mathbb{R}^d$ as a linear combination of the right singular vectors $\{\vec{v}_i\}$ of the matrix A.

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

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

To avoid confusion, I replaced $\vec{u}$ by $\vec{x}$ and $\vec{v}$ by $\vec{y}$

$$\begin{aligned}
\max_{\substack{\vec{x}:\|\vec{x}\|_2=1 \\ \vec{y}:\|\vec{y}\|_2=1}} \vec{x}^\top \mathbf{A}\vec{y} &= \max_{\substack{\vec{x}:\|\vec{x}\|_2=1 \\ \vec{y}:\|\vec{y}\|_2=1}} \vec{x}^\top \mathbf{U}\Sigma\mathbf{V}^\top \vec{y} \\
&= \max_{\substack{\vec{x}:\|\vec{x}\|_2=1 \\ \vec{y}:\|\vec{y}\|_2=1}} \vec{x}^\top \mathbf{U}\Sigma\mathbf{V}^\top \vec{y} \\
&= \max_{\substack{\vec{x}:\|\vec{x}\|_2=1 \\ \vec{y}:\|\vec{y}\|_2=1}} (\mathbf{U}^\top \vec{x})^\top \Sigma\mathbf{V}^\top \vec{y}
\end{aligned}$$

Because U and V are orthonormal matrices, they preserve the norms of $\vec{x}$ and $\vec{y}$. That is $\|\vec{x}\|_2 = \|\mathbf{U}^\top\vec{x}\|_2$ and $\|\vec{y}\|_2 = \|\mathbf{V}^\top\vec{y}\|_2$. Let's define two new vectors $\vec{m} = \mathbf{U}^\top\vec{x}$ and $\vec{n} = \mathbf{V}^\top\vec{y}$. The above optimization problem becomes:

$$\begin{aligned}
\max_{\substack{\vec{x}:\|\vec{x}\|_2=1 \\ \vec{y}:\|\vec{y}\|_2=1}} \vec{x}^\top \mathbf{A}\vec{y} &= \max_{\substack{\vec{m}:\|\vec{m}\|_2=1 \\ \vec{n}:\|\vec{n}\|_2=1}} \vec{m}^\top \Sigma \vec{n} \\
&= \max_{\substack{\vec{m}:\|\vec{m}\|_2=1 \\ \vec{n}:\|\vec{n}\|_2=1}} \sum_{i=1}^{d} \sigma_i m_i n_i
\end{aligned}$$

We just need to set $\vec{m} = e_1^n$ and $\vec{n} = e_1^d$, which gives us the maximum value of the formula above. This is because we assume that $\forall i \quad \sigma_i \geq \sigma_{i+1}$. This can be proved by

Holder inequality where we set $r_i = m_i * n_i$. Let's show it here:

$$\begin{aligned}
\max_{\substack{\vec{x}:\|\vec{x}\|_2=1\\ \vec{y}:\|\vec{y}\|_2=1}} \vec{x}^\top \mathbf{A}\vec{y} &= \max_{\substack{\vec{m}:\|\vec{m}\|_2=1\\ \vec{n}:\|\vec{n}\|_2=1}} \sum_{i=1}^{d} \sigma_i m_i n_i \\
&= \max_{\substack{\vec{m}:\|\vec{m}\|_2=1\\ \vec{n}:\|\vec{n}\|_2=1}} \sum_{i=1}^{d} \sigma_i r_i \\
&= \max_{\substack{\vec{m}:\|\vec{m}\|_2=1\\ \vec{n}:\|\vec{n}\|_2=1}} \|\sigma^\top r\| \\
&\le \|\sigma\|_\infty \|r\|_1 \\
&= \|\sigma\|_\infty \|m^\top n\|_1 \\
&\le \|\sigma\|_\infty \|m\|_2 \|n\|_2 \\
&= \|\sigma\|_\infty
\end{aligned}$$

The equality is achieved at $m = e_1$ and $n = e_1$.

If the maximum singular value is unique, we can compute $\vec{x} = \mathbf{U}^\top e_1^n = \vec{u}_1$ and $\vec{y} = \mathbf{V}^\top e_1^d = \vec{v}_1$. If the maximum singular value is not unique, the maximizers are not unique either, they can be linear combinations of the top eigenvectors corresponding to the maximum singular value.

(d)

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

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

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

Show that the canonical correlation analysis problem can be rewritten as

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

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

$$\begin{aligned}
\rho(\vec{a}^\top \vec{X}, \vec{b}^\top \vec{Y}) &= \frac{\mathbb{E}[\vec{a}^\top \vec{X} \vec{b}^\top \vec{Y}]}{\sqrt{\mathbb{E}[(\vec{a}^\top \vec{X})^2]\mathbb{E}[(\vec{b}^\top \vec{Y})^2]}} \\
&= \frac{\mathbb{E}[\vec{a}^\top \vec{X} \vec{Y}^\top \vec{b}]}{\sqrt{\mathbb{E}[\vec{a}^\top \vec{X} \vec{X}^\top \vec{a}]\mathbb{E}[\vec{b}^\top \vec{Y} \vec{Y}^\top \vec{b}]}} \\
&= \frac{\vec{a}^\top \mathbb{E}[\vec{X} \vec{Y}^\top] \vec{b}}{\sqrt{\vec{a}^\top \mathbb{E}[\vec{X} \vec{X}^\top] \vec{a} \vec{b}^\top \mathbb{E}[\vec{Y} \vec{Y}^\top] \vec{b}}} \\
&= \frac{\vec{a}^\top \Sigma_{XY} \vec{b}}{\sqrt{\vec{a}^\top \Sigma_{XX} \vec{a}}\sqrt{\vec{b}^\top \Sigma_{YY} \vec{b}}}
\end{aligned}$$

$$\begin{aligned}
\rho(\alpha\vec{a}^\top \vec{X}, \beta\vec{b}^\top \vec{Y}) &= \max_{\alpha\vec{a},\beta\vec{b}\in\mathbb{R}^d} \frac{\alpha\vec{a}^\top \Sigma_{XY} \beta\vec{b}}{(\alpha\vec{a}^\top \Sigma_{XX} \alpha\vec{a})^{1/2} \left(\beta\vec{b}^\top \Sigma_{YY} \beta\vec{b}\right)^{1/2}} \\
&= \max_{\vec{a},\vec{b}\in\mathbb{R}^d} \frac{\alpha\beta\vec{a}^\top \Sigma_{XY} \vec{b}}{(\alpha^2\vec{a}^\top \Sigma_{XX} \vec{a})^{1/2} \left(\beta^2\vec{b}^\top \Sigma_{YY} \vec{b}\right)^{1/2}} \\
&= \max_{\vec{a},\vec{b}\in\mathbb{R}^d} \frac{\alpha\beta\vec{a}^\top \Sigma_{XY} \vec{b}}{\alpha\left(\vec{a}^\top \Sigma_{XX} \vec{a}\right)^{1/2} \beta\left(\vec{b}^\top \Sigma_{YY} \vec{b}\right)^{1/2}} \\
&= \max_{\vec{a},\vec{b}\in\mathbb{R}^d} \frac{\vec{a}^\top \Sigma_{XY} \vec{b}}{(\vec{a}^\top \Sigma_{XX} \vec{a})^{1/2} \left(\vec{b}^\top \Sigma_{YY} \vec{b}\right)^{1/2}}
\end{aligned}$$

Therefore, we can conclude that if $(\vec{a}^*, \vec{b}^*)$ is a maximizer above, then $(\alpha\vec{a}^*, \beta\vec{b}^*)$ is a maximizer for any $\alpha, \beta > 0$.

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

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

One may do a similar computation for the whitened vector $\tilde{\vec{Y}} = \Sigma_{YY}^{-\frac{1}{2}}\vec{Y}$, 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:

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

We define $\vec{a} = \Sigma_{XX}^{-1/2}\vec{c}$ and $\vec{b} = \Sigma_{YY}^{-1/2}\vec{d}$

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

Because CCA is scaling invariant, we are only using the direction of $\vec{a}$ and $\vec{b}$. Without losing generality, we can aim for unitary vectors $\vec{c}$ and $\vec{d}$. So the original problem is converted to:

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

(f) Recall that the vectors $(\vec{a}^*, \vec{b}^*)$ 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

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}.$$

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

$$\begin{aligned}\rho^2 &= \max_{\substack{\vec{c}:\|\vec{c}\|_2=1\\ \vec{d}:\|\vec{d}\|_2=1}} \vec{c}^{\top}\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\vec{d}\vec{d}^{\top}\Sigma_{YY}^{-1/2}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}\vec{c}\\ &= \max_{\substack{\vec{c}:\|\vec{c}\|_2=1\\ \vec{d}:\|\vec{d}\|_2=1}} \vec{c}^{\top}\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\Sigma_{YY}^{-1/2}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}\vec{c}\end{aligned}$$

Now, let's define $A = \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}$, we have:

$$\rho^2 = \max_{\vec{c}:\|\vec{c}\|_2=1} \vec{c}^{\top}A\vec{c}$$

In part (c) we know:

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

We can set $\vec{u} = \vec{c} = \vec{v}$ and use the conclusion of part(c), which tells us ρ^2 is the maximum eigenvalue of the matrix

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}$$

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

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}.$$

From the previous part (c) we know that the solution $\vec{c}^*$ to the optimization problem is the maximum eigenvector of the matrix $A = \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}$. From part (e) we know $\vec{a} = \Sigma_{XX}^{-1/2}\vec{c}$ so $\vec{c} = \Sigma_{XX}^{1/2}\vec{a}$.

Therefore we know that $\vec{c}^* = \Sigma_{XX}^{1/2}\vec{a}^*$ is an eigenvector corresponding to the maximum eigenvalue of the matrix

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^{\top}\Sigma_{XX}^{-1/2}.$$

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

$$\Sigma_{YY}^{-1/2}\Sigma_{XY}^\top\Sigma_{XX}^{-1}\Sigma_{XY}\Sigma_{YY}^{-1/2}.$$

$$\begin{aligned}\rho^2 &= \max_{\substack{\vec{c}:\|\vec{c}\|_2=1\\ \vec{d}:\|\vec{d}\|_2=1}} \vec{d}^\top\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\vec{c}\vec{c}^\top\Sigma_{YY}^{-1/2}\Sigma_{XY}^\top\Sigma_{XX}^{-1/2}\vec{d} \\ &= \max_{\substack{\vec{c}:\|\vec{c}\|_2=1\\ \vec{d}:\|\vec{d}\|_2=1}} \vec{d}^\top\Sigma_{YY}^{-1/2}\Sigma_{XY}^\top\Sigma_{XX}^{-1}\Sigma_{XY}\Sigma_{YY}^{-1/2}\vec{d}\end{aligned}$$

Now, let's define $B = \Sigma_{YY}^{-1/2}\Sigma_{XY}^\top\Sigma_{XX}^{-1}\Sigma_{XY}\Sigma_{YY}^{-1/2}$, we have:

$$\rho^2 = \max_{\vec{d}:\|\vec{d}\|_2=1} \vec{d}^\top B\vec{d}$$

Follow the same argument as in part(g), we can conclude that $\vec{c}^* = \Sigma_{XX}^{1/2}\vec{a}^*$ is an eigenvector corresponding to the maximum eigenvalue of the matrix

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{XY}^\top\Sigma_{XX}^{-1/2}.$$

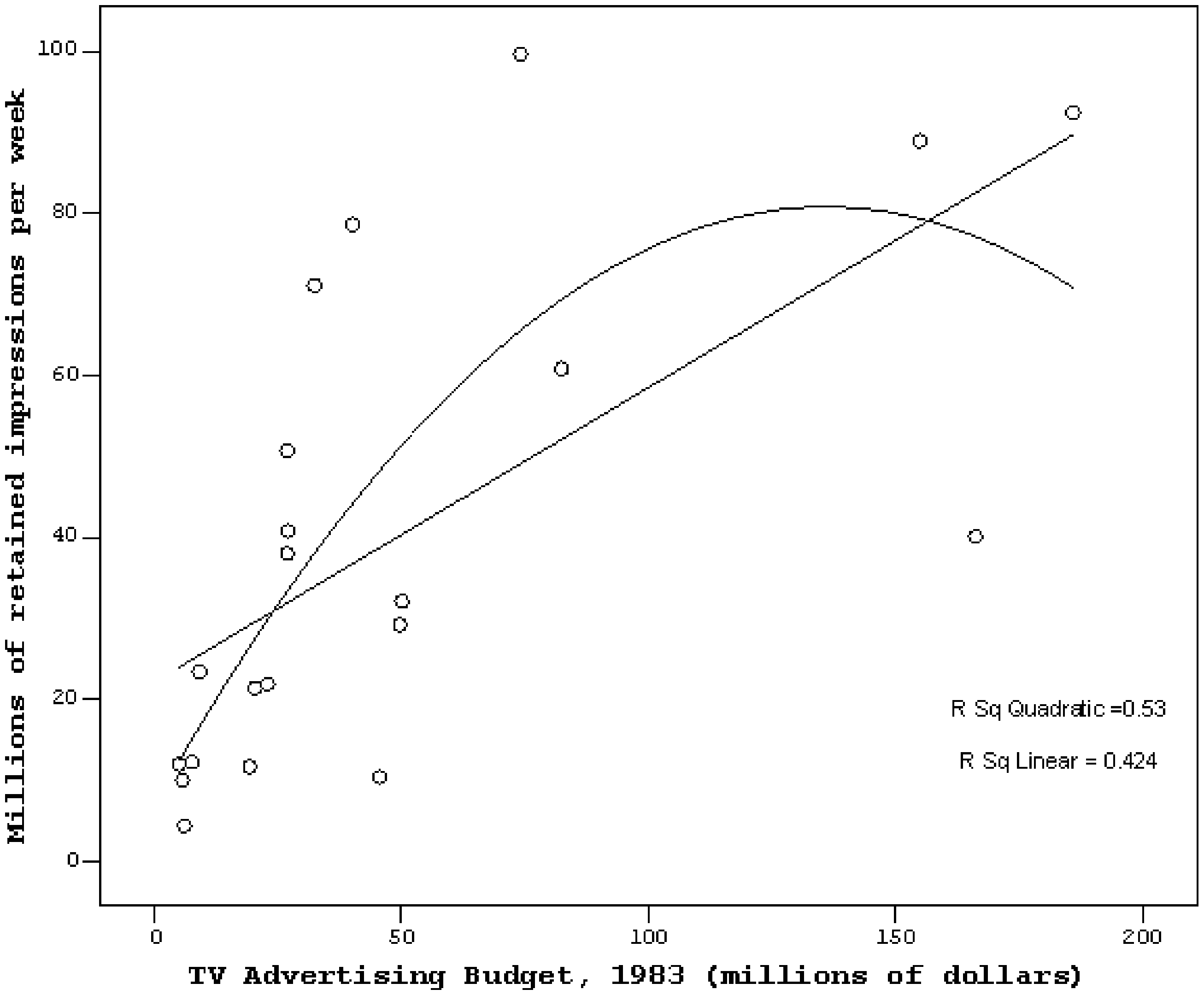
(i) Argue why such a CCA is not meaningful when the random vectors $\vec{X}$ and $\vec{Y}$ are uncorrelated, where by this we mean that $\mathsf{cov}(X_i, Y_j) = 0$ for all i, j.

If $\mathsf{cov}(X_i, Y_j) = 0$ for all i, j, then covariance matrix Σ_{XY} is a zero matrix. Therefore, ρ is always zero regardless of the direction we choose to project to.

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

The best way to use CCA in this case is to square each entry of $\vec{Y}$ to get $\vec{Y}^2$ and then apply CCA.

On the other hand, if we feed CCA with $\vec{X}$ and $\vec{Y}$, we will force us to find a linear relationship between $\vec{X}$ and $\vec{Y}$. This is inaccurate (see the example below).



http://www.pmean.com/06/images/QuadraticRegression01.gif

(k) **Why do you think that understanding CCA is relevant for machine learning?**

1. Dimension reduction. We can simplify our data into only two directions.

2. Normalization. Real world measurements can have different units and might be hard to compare. However, if we use CCA, we don't need to worry about units because it's scaling invariant. On the other hand, PCA is only rotating invariant.

3. Maximize prediction power. After whitening and decorrelation, we can use the transformed X to best predict Y.

Question 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:

```python
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 $\vec{a}$ and $\vec{b}$ that maximize the correlation between the projections $\tilde{\vec{x}} = \vec{a}^T\mathbf{x}$ and $\tilde{y} = \vec{b}^T\vec{y}$, From the previous problem, we know that this can be done by solving the problem

$$\rho = \max_{\vec{a},\vec{b}} \frac{\vec{a}^\top \Sigma_{XY} \vec{b}}{\left(\vec{a}^\top \Sigma_{XX} \vec{a}\right)^{1/2} \left(\vec{b}^\top \Sigma_{YY} \vec{b}\right)^{1/2}},$$

where Σ_{XX} and Σ_{YY} denote the covariance matrices of the $\mathbf{X}$ and $\mathbf{Y}$ images, and Σ_{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.

$$\bar{X} = \frac{1}{n}\mathbf{X}^\top \vec{1}\vec{1}^\top$$
$$\bar{y} = \frac{1}{n}\vec{y}^\top \vec{1}\vec{1}^\top$$
$$\Sigma_{XX} = (\mathbf{X} - \bar{X})^\top (\mathbf{X} - \bar{X})$$
$$\Sigma_{YY} = (\vec{y} - \bar{y})(\vec{y} - \bar{y})^\top$$
$$\Sigma_{XY} = (\mathbf{X} - \bar{X})(\vec{y} - \bar{y})$$

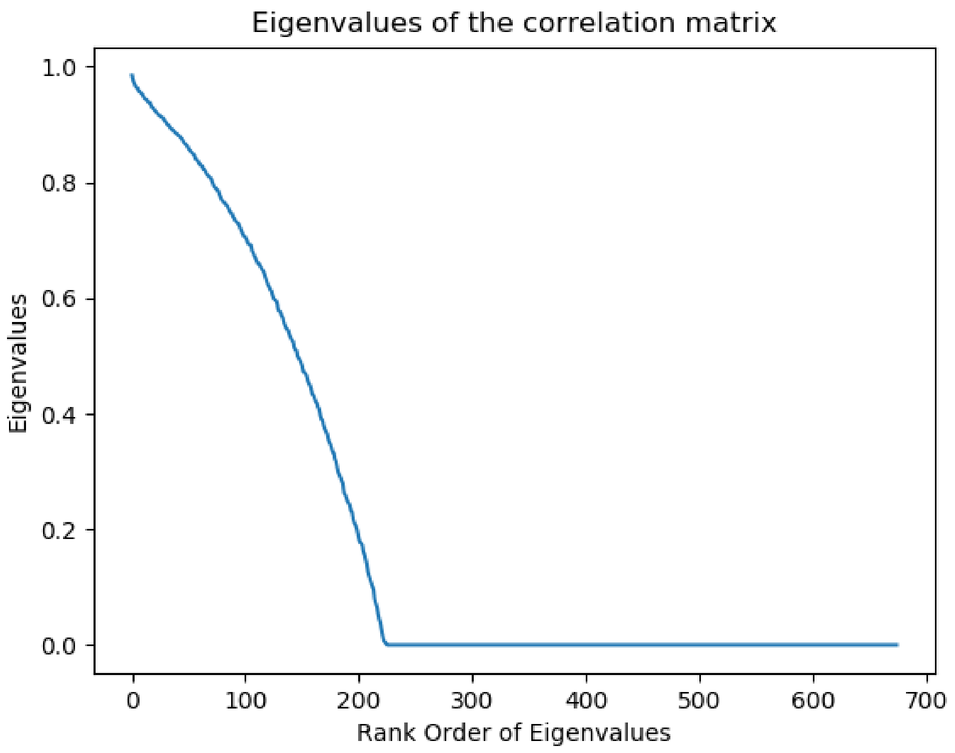
Below is the code I use to calculate the covariance matrices.

```
for i in range(num_data):
x_f = np.array([x_data[i]])
y_f = np.array([y_data[i]])
# TODO: COMPUTE COVARIANCE MATRICES
# BE CAREFUL: x_f is a row vector here
self.C_xx += x_f.T.dot(x_f)
self.C_yy += y_f.T.dot(y_f)
self.C_xy += x_f.T.dot(y_f)
```

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

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

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



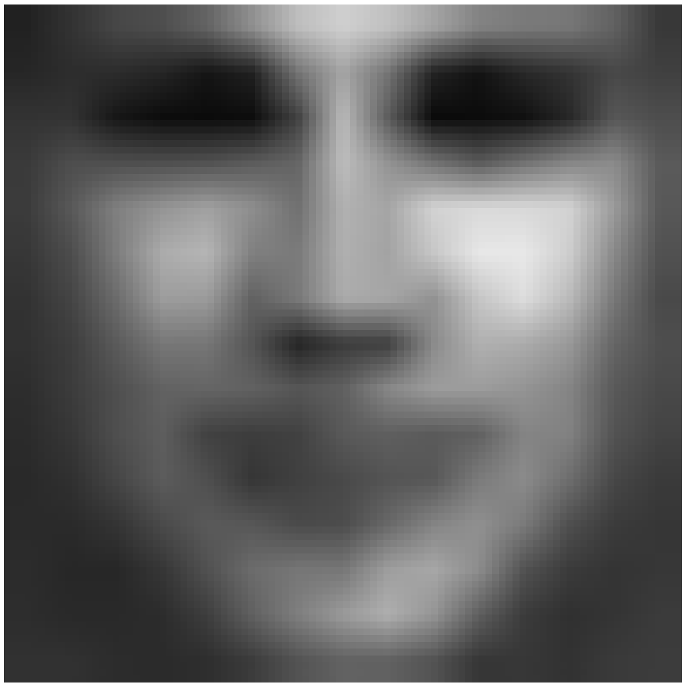
```
def solve_for_variance(self):
eigen_values = np.zeros((675,))
eigen_vectors = np.zeros((675, 675))
# TODO: COMPUTE CORRELATION MATRIX
corr_matrix = np.linalg.inv(sqrtm(self.C_xx + self.lmbda * np.eye(self.C_xx.shape[0]))).\
dot(self.C_xy).\
dot(np.linalg.inv(sqrtm(self.C_yy + self.lmbda * np.eye(self.C_yy.shape[0]))))
_, eigen_values, eigen_vectors = np.linalg.svd(corr_matrix)
return eigen_values, eigen_vectors


plt.figure()
plt.plot(eig_val)
plt.title('Eigenvalues of the correlation matrix')
plt.xlabel('Rank Order of Eigenvalues')
plt.ylabel('Eigenvalues')
plt.show()
```

(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 $\mathbf{U}\Sigma\mathbf{V}^T = \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}$, we can use the left hand singular vectors to form a projection.

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

Show/visualize the "face" corresponding to the first singular vector $\vec{u}_0$. 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)
```

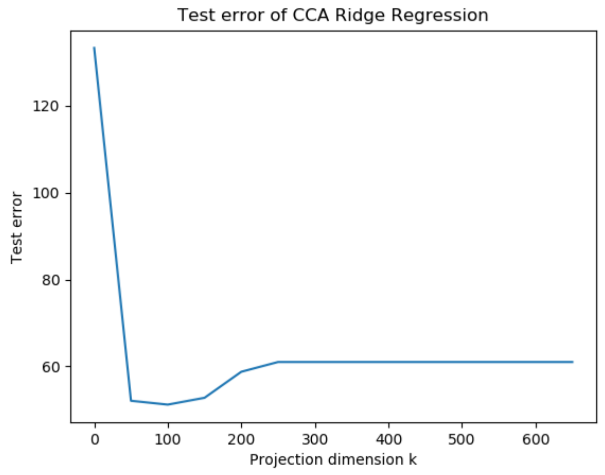


```
# Show face u0
ss_x = StandardScaler(with_std=False)
x_data = flatten_and_standardize(mooney.x_train)
ss_x.fit(x_data)
mooney.plot_image(ss_x.inverse_transform(eig_vec[:, 0]))
```

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



As you can see in my code, I had a lot of troubles keeping track of the dimensions.

```
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


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
# BE CAREFUL: x_f is a row vector here
self.C_xx += x_f.T.dot(x_f)
self.C_yy += y_f.T.dot(y_f)
self.C_xy += x_f.T.dot(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 = []

# 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_f is a row vector and X_proj is a column vector
X.append(np.reshape(X_proj.T.dot(x_f.T), (X_proj.shape[1])))
#X.append(np.zeros((X_proj.shape[0])))

Y = flatten_and_standardize(self.y_train)

#print('X.shape'+str(np.array(X).shape))
#print('Y.shape'+str(np.array(Y).shape))

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(np.reshape(X_proj.T.dot(x_f.T), (X_proj.shape[1])))
#X_test.append(np.zeros((X_proj.shape[0])))

Y_test = flatten_and_standardize(self.y_test)
#print('np.zeros((X_proj.shape[0]))'+str(np.zeros((X_proj.shape[0])).shape))
#print('X_test.shape'+str(np.array(X_test).shape))
#print('Y_test.shape'+str(np.array(Y_test).shape))

# 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)
```

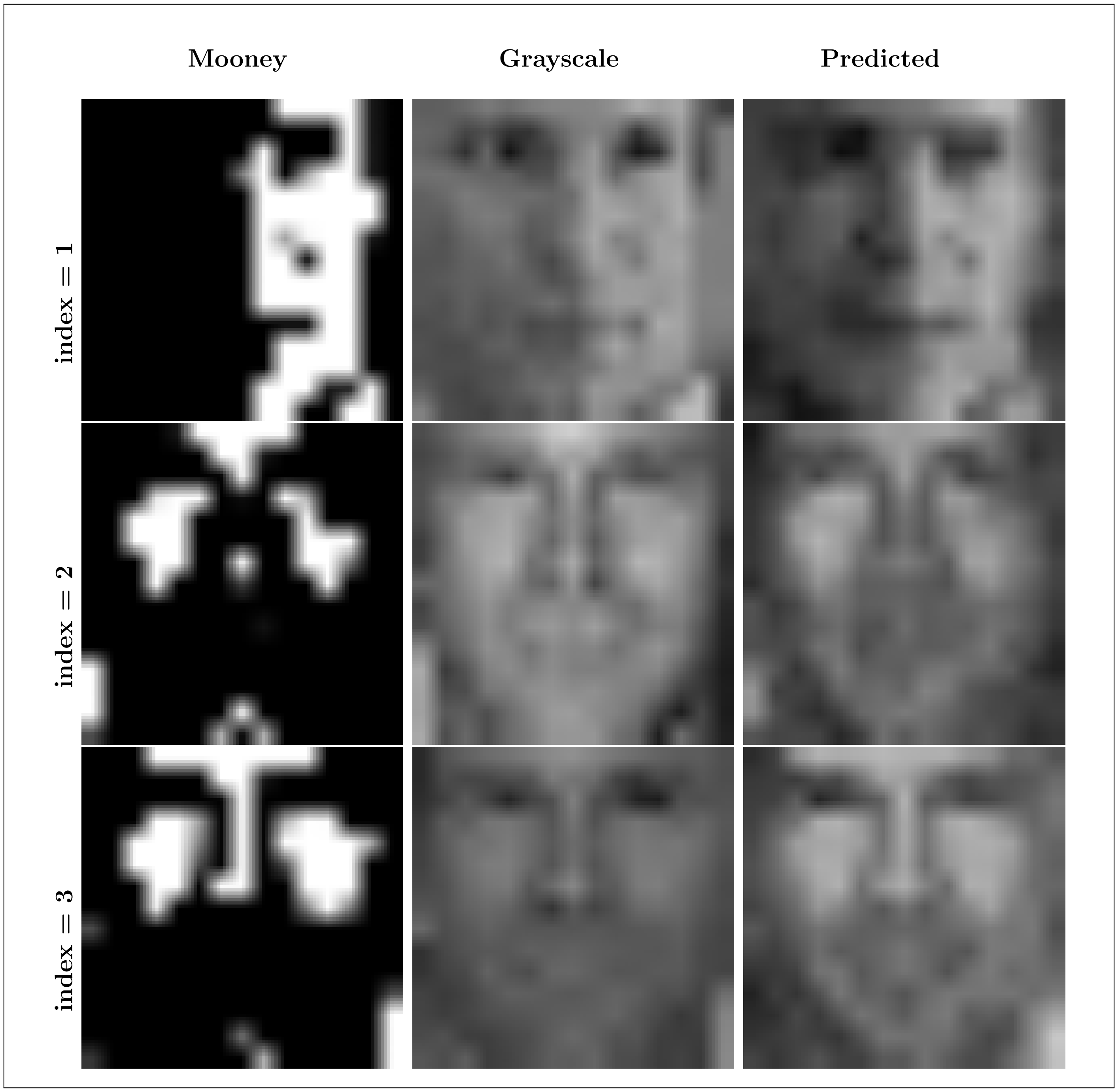
```
def compute_data_matrix(self):
X = flatten_and_standardize(self.x_train)
Y = flatten_and_standardize(self.y_train)
X_test = flatten_and_standardize(self.x_test)
Y_test = flatten_and_standardize(self.y_test)

# 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))
# TODO: COMPUTE CORRELATION MATRIX
corr_matrix = inv(sqrtm(self.C_xx + self.lmbda * np.eye(self.C_xx.shape[0]))).\
dot(self.C_xy).\
dot(inv(sqrtm(self.C_yy + self.lmbda * np.eye(self.C_yy.shape[0]))))
eigen_vectors, eigen_values, _ = svd(corr_matrix)
return eigen_values, eigen_vectors

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

def ridge_regression(self):
w_ridge = []

for i in range(self.y_dim):
# TODO: IMPLEMENT RIDGE REGRESSION
w_i = inv(self.X_ridge.T.dot(self.X_ridge)
+ self.lmbda * np.eye(self.X_ridge.shape[1])).\
dot(self.X_ridge.T).dot(self.Y_ridge[:, i])
w_ridge.append(np.reshape(w_i, (self.X_ridge.shape[1],)))
#w_ridge.append(np.zeros((self.X_ridge.shape[1],)))

self.w_ridge = np.vstack(w_ridge)
#print('self.y_dim'+str(self.y_dim))
#print('self.X_ridge'+str(np.array(self.X_ridge).shape))
#print('self.Y_ridge'+str(np.array(self.Y_ridge).shape))
#print('self.w_ridge'+str(np.array(self.w_ridge).shape))

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):
#print('X_ridge.T.shape'+str(X_ridge.T.shape))
#print('self.w_ridge.shape'+str(self.w_ridge.shape))
#print('Y_ridge.T.shape'+str(Y_ridge.T.shape))

prediction = np.matmul(self.w_ridge, X_ridge.T)

evaluation = Y_ridge.T - 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):
prediction = np.matmul(self.w_ridge, 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):
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):
x = x.astype("uint8")
x = cv2.resize(x, (100, 100))
cv2.imwrite('gt_face_' + str(count) + '.png', x)


if __name__ == '__main__':

mooney = Mooney()

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

# Plot eigenvalues
plt.figure()
plt.plot(eig_val)
plt.title('Eigenvalues of the correlation matrix')
plt.xlabel('Rank Order of Eigenvalues')
plt.ylabel('Eigenvalues')
#plt.show()

# Show face u0
ss_x = StandardScaler(with_std=False)
x_data = flatten_and_standardize(mooney.x_train)
ss_x.fit(x_data)
#mooney.plot_image(ss_x.inverse_transform(eig_vec[:, 0]))

proj = [0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 650]
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.figure()
plt.plot(proj, error_test)
plt.title('Test error of CCA Ridge Regression')
plt.xlabel('Projection dimension k')
plt.ylabel('Test error')
plt.show()

# 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)
```

(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 2: Example results with the input on the left and output on the right



index = 4

```
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


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
# BE CAREFUL: x_f is a row vector here
self.C_xx += x_f.T.dot(x_f)
self.C_yy += y_f.T.dot(y_f)
self.C_xy += x_f.T.dot(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 = []

# 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_f is a row vector and X_proj is a column vector
X.append(np.reshape(X_proj.T.dot(x_f.T), (X_proj.shape[1])))
#X.append(np.zeros((X_proj.shape[0])))

Y = flatten_and_standardize(self.y_train)

#print('X.shape'+str(np.array(X).shape))
#print('Y.shape'+str(np.array(Y).shape))

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(np.reshape(X_proj.T.dot(x_f.T), (X_proj.shape[1])))
#X_test.append(np.zeros((X_proj.shape[0])))

Y_test = flatten_and_standardize(self.y_test)
#print('np.zeros((X_proj.shape[0]))'+str(np.zeros((X_proj.shape[0])).shape))
#print('X_test.shape'+str(np.array(X_test).shape))
#print('Y_test.shape'+str(np.array(Y_test).shape))

# 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 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))
# TODO: COMPUTE CORRELATION MATRIX
corr_matrix = inv(sqrtm(self.C_xx + self.lmbda * np.eye(self.C_xx.shape[0]))).\
dot(self.C_xy).\
dot(inv(sqrtm(self.C_yy + self.lmbda * np.eye(self.C_yy.shape[0]))))
eigen_vectors, eigen_values, _ = svd(corr_matrix)
return eigen_values, eigen_vectors

def project_data(self, eig_val, eig_vec, proj=150):
```

```
# TODO: COMPUTE PROJECTION SINGULAR VECTORS
return eig_vec[:, 0:proj]

def ridge_regression(self):
w_ridge = []

for i in range(self.y_dim):
# TODO: IMPLEMENT RIDGE REGRESSION
w_i = inv(self.X_ridge.T.dot(self.X_ridge)
+ self.lmbda * np.eye(self.X_ridge.shape[1])).\
dot(self.X_ridge.T).dot(self.Y_ridge[:, i])
w_ridge.append(np.reshape(w_i, (self.X_ridge.shape[1],)))
#w_ridge.append(np.zeros((self.X_ridge.shape[1],)))

self.w_ridge = np.vstack(w_ridge)
#print('self.y_dim'+str(self.y_dim))
#print('self.X_ridge'+str(np.array(self.X_ridge).shape))
#print('self.Y_ridge'+str(np.array(self.Y_ridge).shape))
#print('self.w_ridge'+str(np.array(self.w_ridge).shape))

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):
#print('X_ridge.T.shape'+str(X_ridge.T.shape))
#print('self.w_ridge.shape'+str(self.w_ridge.shape))
#print('Y_ridge.T.shape'+str(Y_ridge.T.shape))

prediction = np.matmul(self.w_ridge, X_ridge.T)

evaluation = Y_ridge.T - 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, subfix=''):
for count, x in enumerate(self.X_test_ridge):
prediction = np.matmul(self.w_ridge, 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) + '_' + subfix + '.png', p)

for count, x in enumerate(self.x_test):
x = x.astype("uint8")
x = cv2.resize(x, (100, 100))
```

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

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


if __name__ == '__main__':

mooney = Mooney()

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

# Plot eigenvalues
plt.figure()
plt.plot(eig_val)
plt.title('Eigenvalues of the correlation matrix')
plt.xlabel('Rank Order of Eigenvalues')
plt.ylabel('Eigenvalues')
#plt.show()

# Show face u0
ss_x = StandardScaler(with_std=False)
x_data = flatten_and_standardize(mooney.x_train)
ss_x.fit(x_data)
#mooney.plot_image(ss_x.inverse_transform(eig_vec[:, 0]))

proj = [0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 650]
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.figure()
plt.plot(proj, error_test)
plt.title('Test error of CCA Ridge Regression')
plt.xlabel('Projection dimension k')
plt.ylabel('Test error')
#plt.show()

# COMPUTER REGRESSION WITH PROJECT AND OPTIMAL proj=k
opt_proj = proj[error_test.index(min(error_test))]
print('optimal degree of projection: '+str(opt_proj))
X_proj = mooney.project_data(eig_val, eig_vec, proj=opt_proj)
mooney.compute_projected_data_matrix(X_proj)
mooney.ridge_regression()
mooney.draw_images('proj')

# 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)
```

Question 4. [

(24 points)]Bias-Variance for Ridge Regression 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:

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

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.

$$\begin{aligned}
\text{Bias}^2(\widehat{w}_\lambda) &= (\mathbb{E}[\widehat{w}_\lambda] - w^*)^2 \\
&= (\mathbb{E}[\frac{S_{xy}}{s_x^2 + \lambda}] - w^*)^2 \\
&= (\mathbb{E}[\frac{\sum_{i=1}^n x_i Y_i}{s_x^2 + \lambda}] - w^*)^2 \\
&= (\mathbb{E}[\frac{\sum_{i=1}^n x_i (x_i w^* + Z_i)}{s_x^2 + \lambda}] - w^*)^2 \\
&= (\frac{\sum_{i=1}^n x_i (x_i * w^* + \mathbb{E}[Z_i])}{s_x^2 + \lambda} - w^*)^2 \\
&= (\frac{\sum_{i=1}^n x_i x_i w^*}{s_x^2 + \lambda} - w^*)^2 \\
&= (\frac{s_x^2}{s_x^2 + \lambda} w^* - w^*)^2 \\
&= (\frac{s_x^2}{s_x^2 + \lambda} - 1)^2 (w^*)^2
\end{aligned}$$

(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.

From 4a we know that:

$$\mathbb{E}[\widehat{w}_\lambda] = \frac{s_x^2}{s_x^2 + \lambda} w^*$$

$$\begin{aligned}
\widehat{w}_\lambda &= \frac{S_{xy}}{s_x^2 + \lambda} \\
&= \frac{\sum_{i=1}^n x_i(x_i w^* + Z_i)}{s_x^2 + \lambda} \\
&= \frac{\sum_{i=1}^n x_i x_i w^*}{s_x^2 + \lambda} + \frac{\sum_{i=1}^n x_i Z_i}{s_x^2 + \lambda} \\
&= \mathbb{E}[\widehat{w}_\lambda] + \frac{\sum_{i=1}^n x_i Z_i}{s_x^2 + \lambda}
\end{aligned}$$

Substitute into the equation above:

$$\begin{aligned}
\text{Var}(\widehat{w}_\lambda) &= \mathbb{E}[(\widehat{w}_\lambda - \mathbb{E}[\widehat{w}_\lambda])^2] \\
&= \mathbb{E}[(\mathbb{E}[\widehat{w}_\lambda] + \frac{\sum_{i=1}^n x_i Z_i}{s_x^2 + \lambda} - \mathbb{E}[\widehat{w}_\lambda])^2] \\
&= \mathbb{E}[(\frac{\sum_{i=1}^n x_i Z_i}{s_x^2 + \lambda})^2] \\
&= \frac{\mathbb{E}[(\sum_{i=1}^n x_i Z_i)^2]}{(s_x^2 + \lambda)^2} \\
&= \frac{\sum_{i=1}^n x_i^2}{(s_x^2 + \lambda)^2} \\
&= \frac{s_x^2}{(s_x^2 + \lambda)^2}
\end{aligned}$$

The last three lines are true because we know $Z_i \sim \mathbf{N}(0,1)$ is i.i.d.
Therefore we have $\mathbb{E}[Z_i Z_j] = 0 \quad \forall i \neq j$ and $\mathbb{E}[Z_i^2] = Var[Z_i] + \mathbb{E}[Z_i]^2 = 1$.

(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?**

Copy the solutions from part (a) and (b) here:

$$\text{Bias}^2(\widehat{w}_\lambda) = (1 - \frac{s_x^2}{s_x^2 + \lambda})^2 (w^*)^2$$
$$\text{Var}(\widehat{w}_\lambda) = \frac{s_x^2}{(s_x^2 + \lambda)^2}$$

(1) We can see that when λ increases, bias^2 will also increase but variance will decrease.
(2) $\lambda \to 0$: bias$^2 = 0$ and variance $= 1$
$\lambda \to$ inf : bias$^2 = (w^*)^2$ and variance $= 0$ (3) Bias is increasing and variance is decreasing. (4) Because one is increasing and the other is decreasing, we know that there exists such a λ that bias^2 + variance is minimized.

Question 5. [

(25 points)]Hospital

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 $\vec{X} \in \mathbb{R}^{n \times d}$, and associated real number labels $\vec{y} \in \mathbb{R}^n$, 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 $\widehat{y} = \vec{w}^\top \vec{x}$ 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: ($\vec{X}_B$ and $\vec{y}_B$) and are willing to share their learned model $\widehat{y} = \vec{w}_B^\top \vec{x}$ with you. In particular, Hospital B shares their entire Gaussian posterior distribution on $\vec{w}$ with you: $\mathbf{N}(\vec{w}_B, \Psi)$.

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

HINT: Recall that traditional ridge regression could be derived from a MAP perspective, where the parameter $\vec{w}$ 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?

$$\begin{aligned}\arg\max_{\vec{w}} \boldsymbol{L}(\vec{w}|X, \vec{y}, \vec{w}_B, \Psi^{-1}) &\sim e^{-\frac{\|\vec{y}-X\vec{w}\|_2^2}{2}} e^{-\frac{(\vec{w}-\vec{w}_B)^\top \Psi^{-1}(\vec{w}-\vec{w}_B)}{2}} \\ &\sim e^{-\|\vec{y}-X\vec{w}\|_2^2 - (\vec{w}-\vec{w}_B)^\top \Psi^{-1}(\vec{w}-\vec{w}_B)}\end{aligned}$$

We compute the log likelihood:

$$\arg\max_{\vec{w}} \log\left(\boldsymbol{L}(\vec{w}|X, \vec{y}, \vec{w}_B, \Psi^{-1})\right) \sim -\|\vec{y} - X\vec{w}\|_2^2 - (\vec{w} - \vec{w}_B)^\top \Psi^{-1}(\vec{w} - \vec{w}_B)$$
$$\arg\min_{\vec{w}} \log\left(\boldsymbol{L}(\vec{w}|X, \vec{y}, \vec{w}_B, \Psi^{-1})\right) \sim \|\vec{y} - X\vec{w}\|_2^2 + (\vec{w} - \vec{w}_B)^\top \Psi^{-1}(\vec{w} - \vec{w}_B)$$

Next we choose a new $\vec{w}^*$ to simplify our log likelihood function. To be more specific, we consider $\vec{w}^* = \Psi^{-1/2}(\vec{w} - \vec{w}_B)$. We can show that $\vec{w}^* \sim \mathbf{N}(0, \mathbf{I})$

$$\begin{aligned}&\vec{w}^*(\vec{w}^*)^\top \\ =&\Psi^{-1/2}(\vec{w} - \vec{w}_B)(\vec{w} - \vec{w}_B)\top\Psi^{-1/2} \\ =&\Psi^{-1/2}\Psi\Psi^{-1/2} \\ =&\mathbb{I}\end{aligned}$$

Substitute it into the log likelihood function above we have:

$$\begin{aligned}\arg\max_{\vec{w}^*} \log\left(\boldsymbol{L}(\vec{w}^*|X, \vec{y}, \vec{w}_B, \Psi^{-1})\right) &\sim \|\vec{y} - X(\Psi^{1/2}\vec{w}^* + \vec{w}_B)\|_2^2 + \|\vec{w}^*\|_2^2 \\ &\sim \|(\vec{y} - X\vec{w}_B) - X\Psi^{1/2}\vec{w}^*)\|_2^2 + \|\vec{w}^*\|_2^2\end{aligned}$$

Now we need to define new $\vec{y}^* = \vec{y} - X\vec{w}_B$ and $X^* = X\Psi^{1/2}$. The above function can be converted to the standard MAP form:

$$\arg\max_{\vec{w}^*} \log\big(\boldsymbol{L}(\vec{w}^*|X^*, \vec{w}_B, \Psi^{-1})\big) \sim \|\vec{y}^* - X^*\vec{w}^*\|_2^2 + \|\vec{w}^*\|_2^2$$

Now we write out the solution to ridge regression and convert all the variables back:

$$\begin{aligned}
\hat{\vec{w}}^* &= (X^{*\top}X^* + \mathbf{I})^{-1}X^{*\top}\vec{y}^* \\
\Psi^{-1/2}(\vec{w} - \vec{w}_B) &= (\Psi^{1/2}X^\top X\Psi^{1/2} + \mathbf{I})^{-1}\Psi^{1/2}X^\top(\vec{y} - X\vec{w}_B) \\
\vec{w} - \vec{w}_B &= \Psi^{1/2}(\Psi^{1/2}X^\top X\Psi^{1/2} + \mathbf{I})^{-1}\Psi^{1/2}X^\top(\vec{y} - X\vec{w}_B) \\
\vec{w} - \vec{w}_B &= (\Psi^{-1/2}\Psi^{1/2}X^\top X\Psi^{1/2}\Psi^{-1/2} + \Psi^{-1/2}\mathbf{I}\Psi^{-1/2})^{-1}X^\top(\vec{y} - X\vec{w}_B) \\
\vec{w} &= (X^\top X + \Psi^{-1})^{-1}X^\top(\vec{y} - X\vec{w}_B) + \vec{w}_B
\end{aligned}$$

(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?

Copy the solution to $\hat{\vec{w}}$ from the previous part:

$$\vec{w} = (X^\top X + \Psi)^{-1} X^\top (\vec{y} - X\vec{w}_B) + \vec{w}_B$$

We know X, $\vec{y}$ which are our data, and we also know $\vec{w}_B$ which is given by Hospital B. However, we don't know Ψ which is just a scalar in this question. Therefore, we can set it to be a hyperparameter.

$$\vec{w} = (X^\top X + \lambda)^{-1} X^\top (\vec{y} - X\vec{w}_B) + \vec{w}_B$$

If we trust Hospital B, we could make λ very large and penalize any deviation from $\vec{w}_B$. On the other hand, we have the freedom to set a different variance for $\vec{w}$ using our data.

We need to apply cross-validation to our own dataset and use validation error to determine the best λ.

Question 6. [
(24 points)]Ridge regression vs. PCA

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

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

has the following expression:

$$\widehat{\vec{w}}_{\text{ridge}} = diag(\frac{\sigma_i}{\lambda + \sigma_i^2})\mathbf{U}^\top\vec{y}. \tag{5}$$

Use $\widehat{y}_{test} = \vec{x}_{test}^\top \mathbf{V}\widehat{\vec{w}}_{\text{ridge}}$ to denote the resulting prediction for a hypothetical $\vec{x}_{test}$. Using (5) and the appropriate $\{\beta_i\}$, this can be written as:

$$\widehat{y}_{test} = \vec{x}_{test}^\top \sum_{i=1}^d \vec{v}_i\beta_i\vec{u}_i^\top\vec{y}. \tag{6}$$

What are the β_i for this to correspond to (5) from ridge regression?

$$\begin{aligned}
\widehat{y}_{test} &= \vec{x}_{test}^\top \mathbf{V}\widehat{\vec{w}}_{\text{ridge}} \\
&= \vec{x}_{test}^\top \mathbf{V} diag(\frac{\sigma_i}{\lambda+\sigma_i^2})\mathbf{U}^\top\vec{y} \\
&= \vec{x}_{test}^\top \begin{bmatrix}\vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_d\end{bmatrix} \begin{bmatrix} \frac{\sigma_1}{\lambda+\sigma_1^2} & & & \\ & \frac{\sigma_2}{\lambda+\sigma_2^2} & & \\ & & \ddots & \\ & & & \frac{\sigma_d}{\lambda+\sigma_d^2}\end{bmatrix} \begin{bmatrix}\vec{u}_1^\top \\ \vec{u}_2^\top \\ \dots \\ \vec{u}_d^\top\end{bmatrix} \vec{y} \\
&= \vec{x}_{test}^\top \begin{bmatrix}\vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_d\end{bmatrix} \begin{bmatrix} \frac{\sigma_1}{\lambda+\sigma_1^2}\vec{u}_1^\top \\ \frac{\sigma_2}{\lambda+\sigma_2^2}\vec{u}_2^\top \\ \dots \\ \frac{\sigma_d}{\lambda+\sigma_d^2}\vec{u}_d^\top \end{bmatrix} \vec{y} \\
&= \vec{x}_{test}^\top \begin{bmatrix} \vec{v}_1\frac{\sigma_1}{\lambda+\sigma_1^2}\vec{u}_1^\top \\ \vec{v}_2\frac{\sigma_2}{\lambda+\sigma_2^2}\vec{u}_2^\top \\ \dots \\ \vec{v}_d\frac{\sigma_d}{\lambda+\sigma_d^2}\vec{u}_d^\top \end{bmatrix} \vec{y}
\end{aligned}$$

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

$$= \vec{x}_{test}^{\top} \sum_{i=1}^{d} \vec{v}_i \beta_i \vec{u}_i^{\top} \vec{y}$$

We can see from the above equality, we have:

$$\beta_i = \frac{\sigma_i}{\lambda + \sigma_i^2}$$

(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 $\vec{x}_{test}$,

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

$$\widehat{y}_{test} = \vec{x}_{test}^\top \sum_{i=1}^{d} \vec{v}_i \beta_i \vec{u}_i^\top \vec{y}. \tag{7}$$

Give the β_i 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.

Intuitively, selecting the first k components can be achieved by setting the extra singular values to zeros. Therefore we have:

$$\beta_i = \begin{cases} \frac{\sigma_i}{0+\sigma_i^2} = \frac{1}{\sigma_i} & i \leq k \\ 0 \quad i > k \end{cases}$$

Now let's write it in the matrix form to double check the result:

$$\begin{aligned}
\widehat{y}_{test} &= \vec{x}_{test}^\top \sum_{i=1}^{d} \vec{v}_i \beta_i \vec{u}_i^\top \vec{y} \\
&= \vec{x}_{test}^\top \sum_{i=1}^{d} \vec{v}_i \frac{\sigma_i}{\lambda + \sigma_i^2} \vec{u}_i^\top \vec{y} \\
&= \vec{x}_{test}^\top \begin{bmatrix} \vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_d \end{bmatrix} \begin{bmatrix} \frac{\sigma_1}{\lambda+\sigma_1^2} & & & & & & \\ & \frac{\sigma_2}{\lambda+\sigma_2^2} & & & & & \\ & & \ddots & & & & \\ & & & \frac{\sigma_k}{\lambda+\sigma_k^2} & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{bmatrix} \begin{bmatrix} \vec{u}_1^\top \\ \vec{u}_2^\top \\ \dots \\ \vec{u}_d^\top \end{bmatrix} \vec{y} \\
&= \vec{x}_{test}^\top \begin{bmatrix} \vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_d \end{bmatrix} \begin{bmatrix} \frac{\sigma_1}{\lambda+\sigma_1^2} \vec{u}_1^\top \\ \frac{\sigma_2}{\lambda+\sigma_2^2} \vec{u}_2^\top \\ \dots \\ \frac{\sigma_d}{\lambda+\sigma_k^2} \vec{u}_k^\top \\ 0 \\ \vdots \\ 0 \end{bmatrix} \vec{y}
\end{aligned}$$

$$= \vec{x}_{test}^\top \begin{bmatrix} \vec{v}_1 \frac{\sigma_1}{\lambda+\sigma_1^2} \vec{u}_1^\top \\ \vec{v}_2 \frac{\sigma_2}{\lambda+\sigma_2^2} \vec{u}_2^\top \\ \cdots \\ \vec{v}_d \frac{\sigma_d}{\lambda+\sigma_k^2} \vec{u}_k^\top \\ 0 \\ \vdots \\ 0 \end{bmatrix} \vec{y}$$

$$= \vec{x}_{test}^\top \sum_{i=1}^{k} \vec{v}_i \beta_i \vec{u}_i^\top \vec{y}$$

We just need to set $\lambda = 0$ to convert it to the solution to OLS.

(c) (6 pts) For the following part, $d = 5$. The following $\vec{\beta} := (\beta_1, \ldots, \beta_5)$ (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 $\mathbf{X}, \vec{y}$ (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) $\vec{\beta} = (0.01, 0.1, 0.5, 0.1, 0.01)$

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

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

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

This $\vec{\beta}$ is the corresponding solution to λ-ridge regression. This is because β_i is not monotonic in this case. Only ridge regression can have this result.

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

This $\vec{\beta}$ is the corresponding solution to k-PCA-OLS. This is because only k-PCA-OLS can have zero entries for β.

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

This $\vec{\beta}$ is the corresponding solution to OLS. This is because all values are monotonically increasing as $\beta_i = \frac{1}{\sigma_i}$.

Question 7. [

(24 points)]Kernel PCA

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 $\vec{X} \in \mathbb{R}^{n\times\ell}$ 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 $\vec{x}_j \in \mathbb{R}^{\ell}$ as rows

$$\vec{X} = \begin{pmatrix} \vec{x}_1^\top \\ \vec{x}_2^\top \\ \vdots \\ \vec{x}_n^\top \end{pmatrix} \in \mathbb{R}^{n\times\ell}. \tag{8}$$

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

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

I would compute the eigenvectors of $XX^\top$ to get $\mathbf{U}$ if I only have access to $XX^\top$. This is because we have $\mathbf{X}\mathbf{X}^\top\mathbf{U} = \mathbf{U}\Sigma^\top\Sigma$, which is the matrix form of the eigenvectors definition .

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

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

$$\mathbf{X}\vec{x}_{test} = \begin{pmatrix} \langle \vec{x}_1, \vec{x}_{test} \rangle \\ \vdots \\ \langle \vec{x}_n, \vec{x}_{test} \rangle \end{pmatrix}. \tag{9}$$

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.

Define a vector $\vec{z} = [z_1, z_2, \ldots, z_l]^\top$. We also know the fact that (We use the compact form of the SVD so Σ is a square matrix):

$$\begin{aligned} \mathbf{X} &= \mathbf{U}\Sigma\mathbf{V}^\top \\ \mathbf{U}^\top\mathbf{X} &= \Sigma\mathbf{V}^\top \\ \Sigma^{-1}\mathbf{U}^\top\mathbf{X} &= \mathbf{V}^\top \end{aligned}$$

$$\begin{aligned} \vec{z} &= V^\top \vec{x}_{test} \\ &= \Sigma^{-1}\mathbf{U}^\top\mathbf{X}\vec{x}_{test} \\ &= \vec{f}(\vec{x}_{test}) \end{aligned}$$

Therefore we have:

$$z_i = \frac{1}{\sigma_i}\vec{u}_i^\top \mathbf{X}\vec{x}_{test}$$

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

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

1. Compute $XX^\top$ using kernel function $k(\vec{x}_i, \vec{x}_j)$.

$$XX^\top =$$

$$\begin{bmatrix} k(\vec{x}_1, \vec{x}_1) & k(\vec{x}_1, \vec{x}_2) & \ldots & k(\vec{x}_1, \vec{x}_d) \\ k(\vec{x}_2, \vec{x}_1) & \ddots & \ldots & \ldots \\ \vdots & \ldots & \ddots & k(\vec{x}_{d-1}, \vec{x}_d) \\ k(\vec{x}_2, \vec{x}_1) & \ldots & k(\vec{x}_1, \vec{x}_{d-1}) & k(\vec{x}_d, \vec{x}_d) \end{bmatrix}$$

2. Do SVD on $XX^\top$ and store the results as U for eigenvectors and Σ^2 for eigenvalues.

3. Take the square root of Σ^2 and then inverse it to get Σ^{-1}. Don't forget to truncate and disregard the extra parts of U.

4. Now we use our conclusion from the previous part and calculate $\vec{z}$. We need kernel function here too. Because we have:

$$z_i = \frac{1}{\sigma_i} \vec{u}_i^\top \begin{pmatrix} \langle \vec{x}_1, \vec{x}_{test} \rangle \\ \vdots \\ \langle \vec{x}_n, \vec{x}_{test} \rangle \end{pmatrix}$$

Question 8. [
(14 points)]Multiple Choice Questions

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 $\mathbf{X} = \mathbf{U\Sigma V}^\top$ is the singular value decomposition of $\mathbf{X}$ where $\sigma_i = \Sigma_{i,i}$ are the diagonal entries of Σ and satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$ while $\vec{u}_i$ and $\vec{v}_i$ are the ith columns of $\mathbf{U}$ and $\mathbf{V}$ respectively. **Which of the following is the rank k approximation to 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 $||\mathbf{X} - \mathbf{X}_k||_F^2$?

○ $\sum_{i=1}^k \sigma_i \vec{u}_i \vec{v}_{n-i}^\top$ [✓ $\sum_{i=1}^k \sigma_i \vec{u}_i \vec{v}_i^\top$] ○ $\sum_{i=d-k+1}^d \sigma_i \vec{u}_i \vec{v}_i^\top$ ○ $\sum_{i=1}^k \sigma_i \vec{u}_{n-i} \vec{v}_i^\top$

(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?**

○ $1, x_i$

[✓ $1, x_i, y_i, x_i^2, x_i y_i, y_i^2$]

○ $1, x_i, y_i$

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

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

[✓ $k(\vec{x}, \vec{y}) = \vec{x}^\top \vec{y}$]

[✓ $k(\vec{x}, \vec{y}) = (1 + \vec{x}^\top \vec{y})^p$ **for some degree p**]

○ $k(\vec{x}, \vec{y}) = k_1(\vec{x}, \vec{y}) - k_2(\vec{x}, \vec{y})$ **for valid kernels k_1 and k_2.**

[✓ $k(\vec{x}, \vec{y}) = e^{-\frac{1}{2}||\vec{x} - \vec{y}||_2^2}$]

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

○ **OLS**

○ **Ridge regression**

○ **Weighted Least Squares**

[✓ **TLS**]

(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 ($\vec{y} \in \mathbb{R}^n$). 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 X using $k \ll n$ random projections**]

[✓ **Preprocess X using PCA with $k \ll n$ components.**]

○ **Preprocess X using PCA with n components.**

○ **Add polynomial features**

○ **Use a kernel approach**

[✓ **Add a ridge penalty to OLS**]

[✓ **Do weighted least squares**]

(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**]

(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**

(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 **◯ Bias**

[✓ Validation Error] **[✓ Variance]**

(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 **◯ Bias**

[✓ Validation Error] **◯ Variance**

Question 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.

What's dictionary learning?

According to Wikipedia page `https://en.wikipedia.org/wiki/Sparse_dictionary_learning`, (sparse) dictionary learning is a method to find a sparse representation of the input data. As a beginner to the field, I would focus on a relatively simple example: K-SVD.

According to Wikipedia, K-SVD's goal is to find the corresponding D and R which minimize the objective function $\|X - DR\|_F^2$ where R has a less than T non-zero elements.To write it as an optimization problem, we have:

$$\min_{D,R} \|X - DR\|_F^2 \quad s.t \quad \forall i \|r_i\|_0 <= T$$

Recall Eckart-Young theorem which gives the best low-rank approximation:

$$\min_{M} \|X - M\|_F^2$$

The best rank(M)=k is given by$M = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^\top$, where SVD on X gives us: $X = \sum_{i=1}^{d} \sigma_i \vec{u}_i \vec{v}_i^\top$

Now we just need to keep the first k eigenvalues and their corresponding eigenvectors. Let's define $D = \mathbf{U\Sigma}$ and $R = \mathbf{V_k}^\top$. We can see that this is exactly the same as latent variable analysis I talked about last time.

Because the deadline is so tight for this homework, I will do an example of this in the next homework.