CS 189 Introduction to Machine Learning
Fall 2017 HW5

This homework is due **Friday, September 29 at 10pm.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

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

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it:

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.

2 Properties of Convex Functions

In lecture, we will introduce gradient descent as a powerful general optimization tool which, under most conditions, will converge to a local minimum of an objective function. However, a local minimum may not always be a good solution. When a function is convex, its local minima are global minima. Thus, gradient descent is an especially powerful tool for numerically optimizing convex functions. It is very useful to recognize when a function is convex or not.

Before we get to a convex function, though, let's talk about convex *sets*. A convex set is a set S where

$$x_1 \in S, x_2 \in S \implies \lambda x_1 + (1-\lambda)x_2 \in S,$$

where $0 \leq \lambda \leq 1$. In words, a convex set is a set S where x_1 in S and x_2 in S implies that everything "in between" x_1 and x_2 is also in S.

There are several equivalent ways to define a convex *function*. Here are three:

- A function f is convex if

  $$\lambda f(x_1) + (1-\lambda) f(x_2) \geq f(\lambda x_1 + (1-\lambda)x_2)$$

  for any x_1 and x_2 in the domain of f and $0 \leq \lambda \leq 1$.
- A function f is convex if the set $S_f = \{(x,y) | x \in \mathbb{R}^n, y \in \mathbb{R}, y \geq f(x)\}$ is convex. The set S_f is called the *epigraph* of the function f. It is all the points that lie "above" the curve f.
- A function f is convex if its Hessian matrix (the matrix of second partial derivatives) is positive semi-definite everywhere on the domain of f. This definition assumes the Hessian matrix exists everywhere on the domain, which is not true for non-differentiable functions.

(a) **Give an example of a function f from $\mathbb{R}$ to $\mathbb{R}$ which is convex, but not differentiable at at least one point in the domain of f.**

Solution: One solution is the absolute value function: $f(x) = |x|$. It is not differentiable at 0, but the function is still convex. Proof:

$$\begin{aligned} f(\lambda x_1 + (1-\lambda)x_2) &= |\lambda x_1 + (1-\lambda)x_2| \\ &\leq |\lambda x_1| + |(1-\lambda)x_2| \\ &= \lambda |x_1| + (1-\lambda)|x_2| \\ &= \lambda f(x_1) + (1-\lambda) f(x_2). \end{aligned} \tag{1}$$

This highlights the importance of having several definitions of convexity.

(b) **Compute the Hessian of the function f which maps a vector x to its loss in the ordinary least squares problem.** As a reminder, the function f is

$$f(x) = ||Ax - y||^2,$$

where $x \in \mathbb{R}^d$ and A and y are constants. This is the function we minimize to get the optimal solution x^*. **Argue that the Hessian of f is positive semi-definite.**

Solution: To take the Hessian, we first expand

$$||Ax-y||^2 = x^TA^TAx - y^TAx - x^TA^Ty + y^Ty = x^TA^TAx - 2y^TAx + y^Ty.$$

To compute the Hessian, we would like to compute $\frac{\partial^2 f}{\partial x \partial x^T}$. In general,

$$\frac{\partial^2}{\partial x \partial x^T}(x^TMx + b^Tx + c) = M^T + M.$$

So, in this case,

$$\frac{\partial^2 f}{\partial x \partial x^T} = (A^TA)^T + A^TA = 2A^TA.$$

To see the above conclusion from first principles, note that the Hessian matrix has ijth entry given by $\frac{\partial^2 f}{\partial x_i x_j}$. Clearly, all terms of f that have degree 1 disappear in the Hessian, since we are taking two derivatives. Hence, $\frac{\partial^2 f}{\partial x_i x_j} = \frac{\partial^2 (x^\top A^\top Ax)}{\partial x_i x_j}$. Now notice that $x^\top A^\top Ax = \sum_{k,\ell}(A^\top A)_{k\ell}x_k x_\ell$, and so

$$\frac{\partial^2 (x^\top A^\top Ax)}{\partial x_i x_j} = (A^\top A)_{ij} + (A^\top A)_{ji}.$$

Consequently, $\nabla^2 f(x) = (A^TA)^T + A^TA = 2A^TA$.

Recall that a matrix M is PSD if $v^TMv \geq 0$ for all v. In this case, if we take $v^T 2A^TAv$ we can write it as

$$v^T 2A^TAv = 2(Av)^T(Av) = 2||Av||^2 \geq 0.$$

Thus, $2A^TA$ is PSD.

(c) **Prove that if f is convex and if x_0 is a local minimum of f, then x_0 is also a global minimum.** Remember that a local minimum is defined as follows: if x_0 is a local minimum, then there exists an ε such that

$$||x_0 - x|| < \varepsilon \implies f(x) \geq f(x_0).$$

(Hint: first express clearly what does it mean for a function to have a global minimum at x_0.)

Solution: We will use a proof by contradiction. Assume that x_0 is a local minimum but not a global minimum. Assume that x_1 is a point in the domain of f such that $f(x_1) < f(x_0)$. We apply the definition of convexity using x_0 and x_1. Assume that $\lambda > 0$:

$$f(\lambda x_1 + (1-\lambda)x_0) \leq \lambda f(x_1) + (1-\lambda)f(x_0) < \lambda f(x_0) + (1-\lambda)f(x_0) = f(x_0).$$

When λ is smaller than $\frac{\varepsilon}{||x_1 - x_0||}$, consider the point

$$x_2 = \lambda x_1 + (1-\lambda)x_0.$$

We have

$$||x_2 - x_0|| = ||\lambda x_1 + (1-\lambda)x_0 - x_0|| = ||\lambda x_1 - \lambda x_0|| = \lambda ||x_1 - x_0|| < \varepsilon.$$

This contradicts the definition of a local minimum because x_2 is within ε of x_0 but $f(x_2) < f(x_0)$, so our assumption that x_0 is a local minimum but not a global minimum must be wrong.

(d) If f and g are convex functions, consider the functions h defined below. **Either prove that h is always convex (for any f and g) or provide a counter-example (where f and g are convex, but h is not)**:

(i) $h(x) = f(x) + g(x)$
(ii) $h(x) = \min\{f(x), g(x)\}$
(iii) $h(x) = \max\{f(x), g(x)\}$
(iv) $h(x) = f(g(x))$

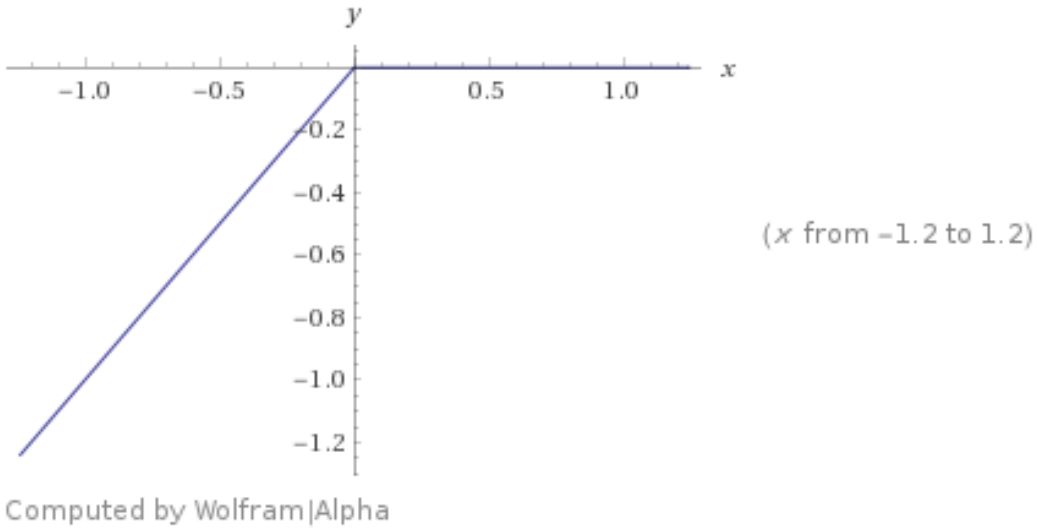
Solution:

(i) This is true.

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) &= f(\lambda x_1 + (1-\lambda)x_2) + g(\lambda x_1 + (1-\lambda)x_2) \\ &\leq \lambda f(x_1) + (1-\lambda) f(x_2) + \lambda g(x_1) + (1-\lambda) g(x_2) \\ &= \lambda (f(x_1) + g(x_1)) + (1-\lambda)(f(x_2) + g(x_2)) \\ &= \lambda h(x_1) + (1-\lambda) h(x_2) \end{aligned} \tag{2}$$

(ii) This is false. Let $f(x) = x$ and $g(x) = 0$. Both f and g are linear functions, so they are convex. Consider the case where $\lambda = \frac{1}{2}$ and $x_1 = -1$ and $x_2 = 1$. We have

$$h(\lambda x_1 + (1-\lambda)x_2) = h(0) = 0 > \lambda h(x_1) + (1-\lambda) h(x_2) = \frac{-1}{2}.$$



(iii) This is true. A useful property to recall here is that for any a, b, c, d, we know

$$\max\{a+b, c+d\} \leq \max\{a+b, a+d, c+b, c+d\} = \max\{a,c\} + \max\{b,d\}.$$

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) &= \max\{f(\lambda x_1 + (1-\lambda)x_2), g(\lambda x_1 + (1-\lambda)x_2)\} \\ &\leq \max\{\lambda f(x_1) + (1-\lambda) f(x_2), \lambda g(x_1) + (1-\lambda) g(x_2)\} \\ &\leq \lambda \max\{f(x_1), g(x_1)\} + (1-\lambda) \max\{f(x_2), g(x_2)\} \\ &\leq \lambda h(x_1) + (1-\lambda) h(x_2) \end{aligned} \tag{3}$$

Another way to prove this is to notice that the epigraph of h is the intersection of the epigraphs of f and g. The intersection of two convex sets is convex, so that is sufficient to prove h convex.

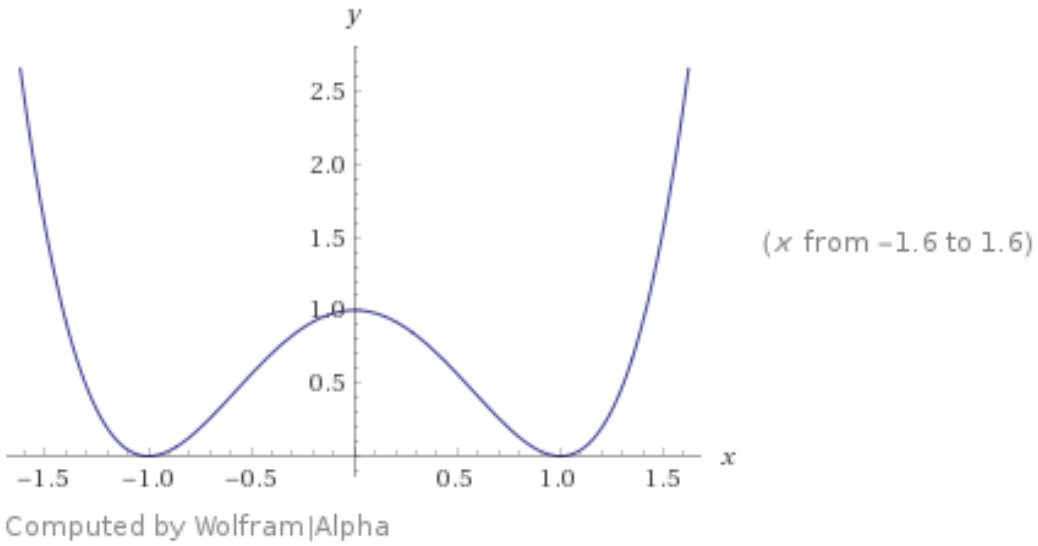
(iv) This is false. Let $f(x) = x^2$ and let $g(x) = x^2 - 1$. Both f and g are convex because their second derivative is non-negative everywhere. But

$$h(x) = (x^2 - 1)^2.$$

Its second derivative is

$$h''(x) = 12x^2 - 4,$$

which is negative when $|x| < \frac{1}{\sqrt{3}}$. This statement *is* true if f is a non-decreasing function.



3 Canonical Correlation Analysis

In this problem, we will work our way through the singular value decomposition, and show how it helps yield a solution to the problem of canonical correlation analysis.

(a) Let $n \geq d$. For a matrix $A \in \mathbb{R}^{n \times d}$ having full column rank and singular value decomposition $A = U\Sigma V^\top$, we know that the singular values are given by the diagonal entries of Σ, and the left (resp. right) singular vectors are the columns of the matrix U (resp. V). Both U and V have orthonormal columns.

Show that $A = \sum_{i=1}^{d} \sigma_i u_i v_i^\top$, where the ith singular value is denoted by $\sigma_i = \Sigma_{ii}$ and u_i and 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 $A = U\Sigma V^\top$, where $U \in \mathbb{R}^{n \times n}$, $\Sigma \in \mathbb{R}^{n \times d}$, and $V \in \mathbb{R}^{d \times d}$.

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

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

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

$$U\Sigma = \begin{bmatrix} | & & | \\ \sigma_1 u_1 & \cdots & \sigma_d u_d \\ | & & | \end{bmatrix}.$$

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

$$U\Sigma V^\top = \sum_{i=1}^{d} \sigma_i u_i v_i^\top.$$

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

i) $A^\top A$ **has** i**th eigenvalue** $\lambda_i = \sigma_i^2$**, with associated eigenvector** v_i**.**

i) $AA^\top$ **has** i**th eigenvalue** $\lambda_i = \sigma_i^2$**, with associated eigenvector** u_i**.**

Notice that both of the above matrices are symmetric.

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

$$\begin{aligned} A^\top A &= \Big(\sum_{i=1}^{d} \sigma_i u_i v_i^\top\Big)^\top \Big(\sum_{j=1}^{d} \sigma_j u_j v_j^\top\Big) \\ &= \Big(\sum_{i=1}^{d} \sigma_i v_i u_i^\top\Big)\Big(\sum_{j=1}^{d} \sigma_i u_j v_j^\top\Big). \end{aligned}$$

Now notice that $u_i^\top u_j = 0$ unless $i = j$, in which case $u_i^\top u_i = 1$. Therefore, expanding the above multiplication, we see that only the terms where $i = j$ remain, and we have

$$A^\top A = \sum_{i=1}^{d} \sigma_i^2 v_i v_i^\top.$$

Consequently, we have

$$A^\top A v_j = \sum_{i=1}^{d} \sigma_i^2 v_i v_i^\top v_j = \sigma_i^2 v_j,$$

where we have used the fact that $v_i^\top v_j = 0$ unless $i = j$, in which case $v_j^\top v_j = 1$.

In words, we have shown that v_j is an eigenvector of $A^\top A$ 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

$$\begin{aligned} AA^\top &= (\sum_{i=1}^{d} \sigma_i u_i v_i^\top)(\sum_{j=1}^{d} \sigma_j u_j v_j^\top)^\top \\ &= (\sum_{i=1}^{d} \sigma_i u_i v_i^\top)(\sum_{j=1}^{d} \sigma_i v_j u_j^\top). \end{aligned}$$

Now notice that $v_i^\top v_j = 0$ unless $i = j$, in which case $v_i^\top v_i = 1$. Therefore, expanding the above multiplication, we see that only the terms where $i = j$ remain, and we have

$$AA^\top = \sum_{i=1}^{d} \sigma_i^2 u_i u_i^\top.$$

Consequently, we have

$$AA^\top u_j = \sum_{i=1}^{d} \sigma_i^2 u_i u_i^\top u_j = \sigma_i^2 u_j,$$

where we have used the fact that $u_i^\top u_j = 0$ unless $i = j$, in which case $u_j^\top u_j = 1$.

In words, we have shown that u_j is an eigenvector of $AA^\top$ 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**

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

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

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

Hint 1: You can express any $u : \|u\|_2 = 1$ as a linear combination of left singular vectors of the matrix A, and similarly with v and right singular vectors. You may or may not find this hint useful.

Hint 2: You may find the following facts that hold for any two vectors $a, b \in \mathbb{R}^d$ useful: Cauchy-Schwarz inequality: $|a^\top b| \leq \|a\|_2 \|b\|_2$, with equality when b is a scaled version of a.

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

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

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

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

Now notice that $\Sigma \in \mathbb{R}^{n \times d}$ 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

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

Now, we see that there is no value in setting any entry of a or 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 c. Similarly, collect the σ_i values into a vector σ. We therefore have that for any unit norm vectors u, v, the following inequality holds:

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

where we have used Holder's inequality.

Now, note that $\|c\|_1 = a^\top b$ since all the entries were assumed to be positive, and using Cauchy Schwarz inequality, we know that $a^\top b \leq \|a\|_2 \|b\|_2 = 1$. We have thus shown that $u^\top A v \leq \sigma_1(A)$ for all unit norm vectors u and v. In order to show that the maximum is attained, choose $u = u_1$, and $v = v_1$, and note that this corresponds to choosing a and b to be the first basis vector in $\mathbb{R}^d$, which we denote by e_1. Thus, we have

$$u_1 A v_1 = e_1 \Sigma e_1 = \sigma_1(A).$$

We have thus shown that the maximum $\sigma_1(A)$ is indeed attained by the claimed maximizers $u^* = u_1$ and $v^* = v_1$.

(d) Define the correlation coefficient between two scalar zero-mean random variables P and Q as

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

Let us now look at the canonical correlation analysis problem, where we are given two (say, zero mean) random vectors $X, Y \in \mathbb{R}^d$, with covariance (and variance) given by

$$\begin{aligned}\mathbb{E}[XX^\top] &= \Sigma_{XX}\\ \mathbb{E}[YY^\top] &= \Sigma_{YY}\\ \mathbb{E}[XY^\top] &= \Sigma_{XY}.\end{aligned}$$

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

The goal of CCA is to find linear combinations that maximize the correlation. In other words, we want to solve the problem

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

Show that the problem can be rewritten as

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

Conclude that if (a^*, b^*) is a maximizer above, then $(\alpha a^*, \beta b^*)$ is a maximizer for any $\alpha, \beta > 0$.

Solution:

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

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

By a similar argument, we have

$$\begin{aligned}\mathbb{E}[(a^\top X)(a^\top X)] &= a^\top \Sigma_{XX} a, \text{ and}\\ \mathbb{E}[(b^\top Y)(b^\top Y)] &= b^\top \Sigma_{YY} b.\end{aligned}$$

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

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

In order to see the last conclusion, simply scale a and b by α and β respectively, and notice that

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

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

(e) Assume that the covariance matrices Σ_{XX} and Σ_{YY} are full rank, and denote the maximizers in the above problem by (a^*, b^*). Use the above parts to **show that**

i) ρ^2 is the maximum eigenvalue of the matrix

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

ii) $\Sigma_{XX}^{1/2}a^*$ is the maximal eigenvector of the matrix

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

iii) $\Sigma_{YY}^{1/2}b^*$ is the maximal eigenvector of the matrix

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

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

Solution:

We use the change of variables $c = \Sigma_{XX}^{1/2}a$, and $d = \Sigma_{YY}^{1/2}b$. Since the matrices Σ_{XX} and Σ_{YY} are invertible, we have $a = \Sigma_{XX}^{-1/2}c$, and $b = \Sigma_{YY}^{-1/2}d$. Substituting these expressions into the definition of ρ above, we have

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

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

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

Once we have written it in this form, we can use previous parts. From part (c), we know that the value of ρ is the maximum singular value of the matrix $\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}$, and that $c^* = \Sigma_{XX}^{1/2}a^*$ and $d^* = \Sigma_{YY}^{1/2}b^*$ are the maximal left and right singular vectors of this matrix, respectively.

We now apply part (b), which states that c^* is the maximal eigenvector of the matrix

$$\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\left(\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\right)^\top.$$

Similarly, d^* is the maximal eigenvector of the matrix

$$\left(\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}\right)^\top \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2},$$

and ρ^2 is the maximum eigenvalue of both of these matrices.

(f) **Argue why "vanilla" CCA is useless when the random vectors X and Y are uncorrelated, where by this we mean that $\mathrm{cov}(X_i, Y_j) = 0$ for all i, j.**. If you happen to know that X and Y^2 (where this is defined by squaring each entry of Y) share a linear relationship, **how might you modify the CCA procedure to account for this?**

Solution: $\Sigma_{XY} = 0$, so CCA returns $\rho = a = b = 0$, and is therefore useless. However, if we know that there is a non-linear relationship between the variables that is quadratic, we may perform CCA on the variables X and $Z = Y^2$, 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.

4 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.

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 $x_i \in \mathbb{R}^{15\times15\times3}$. The test data X_test.p contains 255 binarized images with the same dimensionality. Y_train.p contains 956 corresponding grayscale images, where $y_i \in \mathbb{R}^{15\times15\times3}$. 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:

$$x = (x/255) * 2.0 - 1.0$$

(Note, however, that this standardization during loading the data does not remove the need to do actual mean removal during processing.)

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

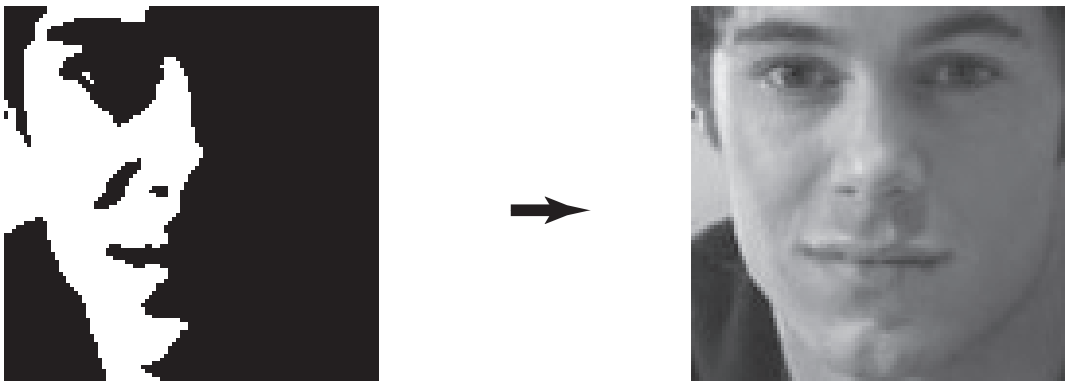


Figure 1: A
binarized Mooney image of an face being restored to its original grayscale image.

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

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

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

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 had 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 used 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]])
```

```
59      y_f = np.array([y_data[i]])

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

65      self.C_xy += np.dot(x_f.T, y_f)

67    #DIVIDE BY THE NUMBER OF DATA POINTS
      self.C_xx = 1.0/float(num_data)*self.C_xx
69    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 $\Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}$, and that this corresponds to the maximum correlation coefficient ρ.

Now, however, **plot the full "spectrum" of singular values of the matrix** $(\Sigma_{XX}+\lambda I)^{-1/2}\Sigma_{XY}(\Sigma_{YY}+\lambda I)^{-1/2}$**.** 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):
2   lmbda = 0.00001
    #COMPUTE INVERTED SQUARE ROOT FOR EACH MATRIX
4   A = inv(sqrtm(self.C_xx+lmbda*np.eye(self.x_dim)))
    B = inv(sqrtm(self.C_yy+lmbda*np.eye(self.y_dim)))
6
    #MULTIPLY THE MATRICES WITH \Sigma_{XY}
8   C = np.matmul(A,np.matmul(self.C_xy,B))

10  #COMPUTE THE SINGULAR VALUES
    u,s,d = svd(C)
12
    #PLOT THE SINGULAR VALUES
14  plt.plot(s)
    plt.xlabel('Directions')
16  plt.ylabel('Singular Values')
    plt.show()
18
    singular_values = s
20  singular_vectors = u


22
    return singular_values,singular_vectors
```

(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 x on to the subspace spanned by the top k singular values. Given that the SVD $U\Sigma V^T = \Sigma_{XX}^{-1/2}\Sigma_{XY}\Sigma_{YY}^{-1/2}$, we can use the left hand singular vectors to form a projection.

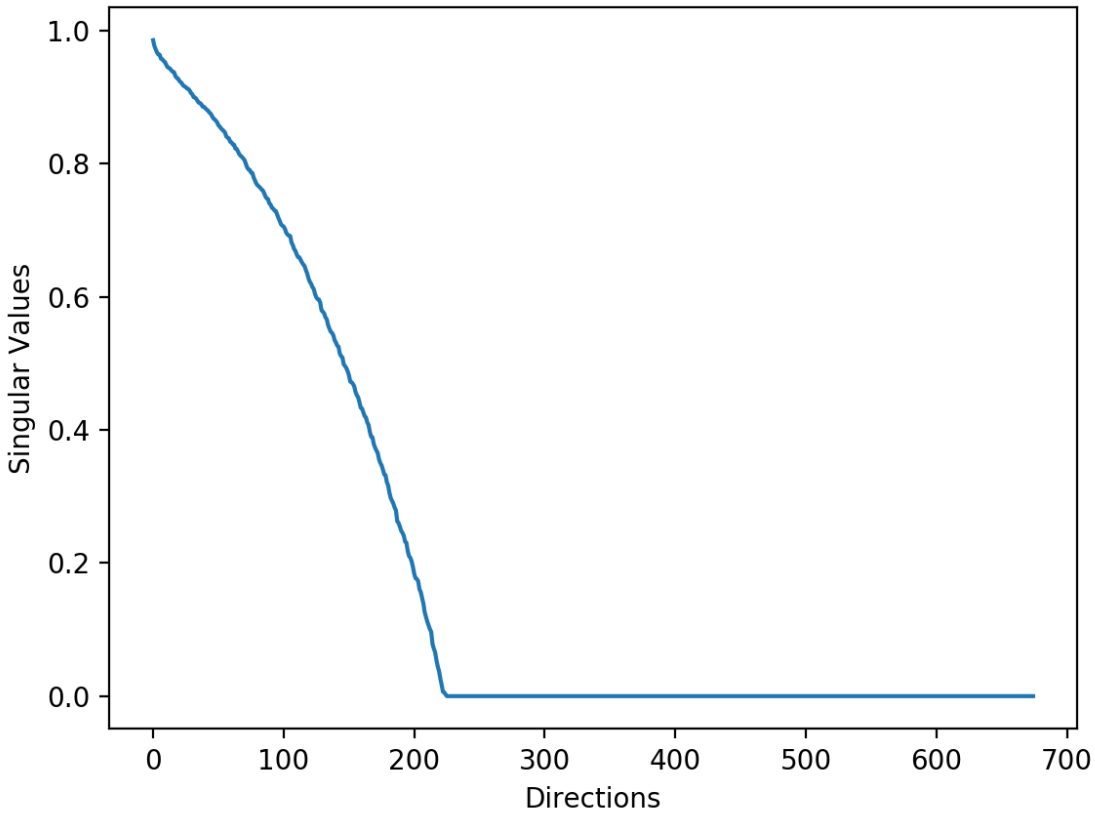


Figure 2: The Singular Values for the Correlation Matrix plotted from largest to smallest.

$$P_k = \begin{bmatrix} u_0, & u_1, & \dots & u_{k-1} \end{bmatrix},$$

Show/visualize the "face" corresponding to the first singular vector 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*.

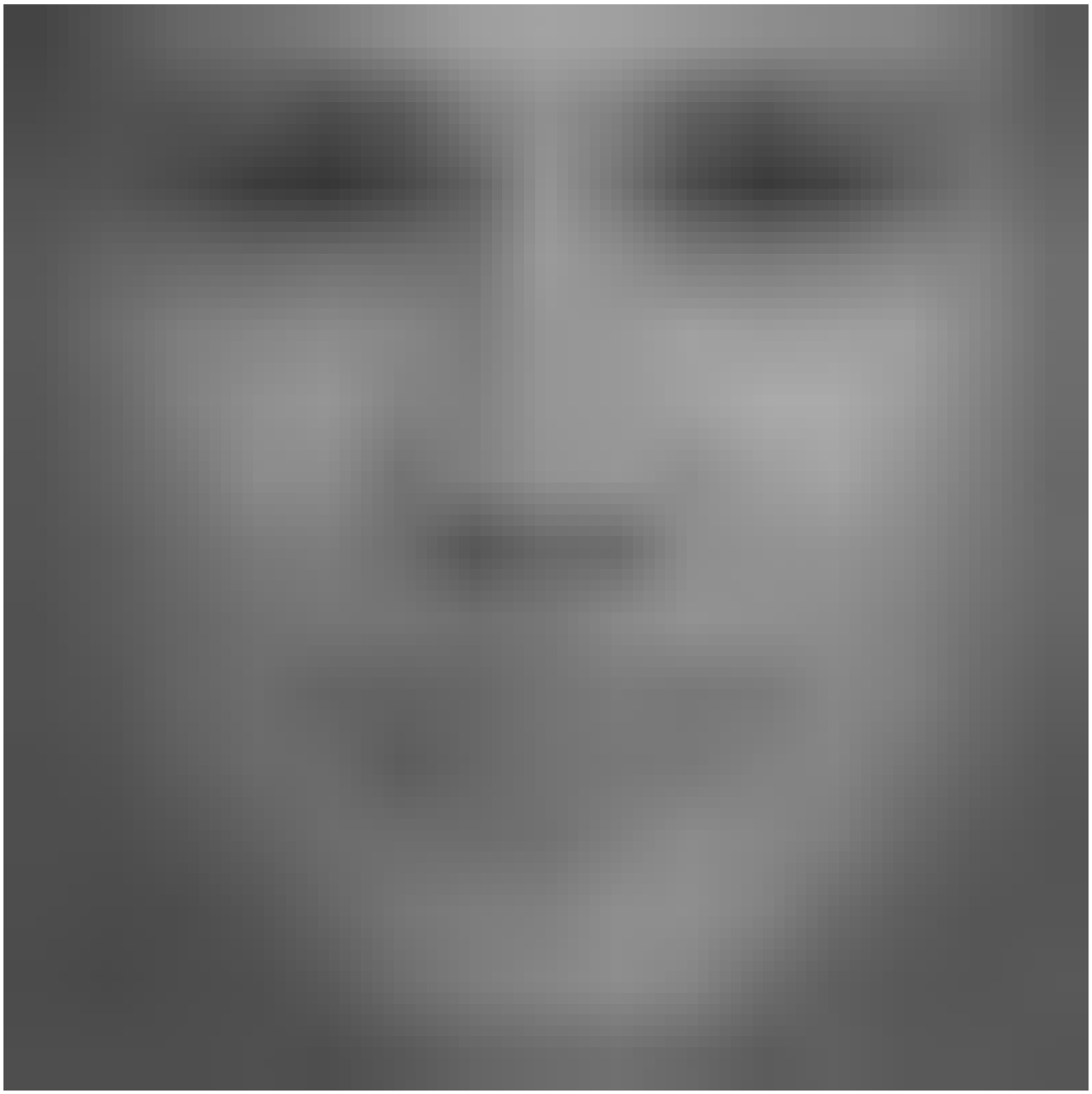


Figure 3: The first component of the CCA

```
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, $w \in \mathbb{R}^{k \times 675}$, from the projected binarized data to the grayscale images. The binarized images are placed in matrix $X \in \mathbb{R}^{956 \times 675}$

$$\min_w ||(XP_k)w - Y||_2^2 + \lambda ||w||_2^2$$

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*

```
1
def compute_projected_data_matrix(self,X_proj):
3
  Y = []
5 X = []

7  Y_test = []
   X_test = []
9
  #LOAD TRANING DATA
11 for x in self.x_train:

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

17    #SUBTRACT MEAN
      x_f = self.ss_x.transform(x_f)
19
      #PROJECT DATA
21    x_p = np.matmul(X_proj,x_f.T)

23    X.append(x_p[:,0])

25 for y in self.y_train:

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

29    #STANDARDIZE DATA
```

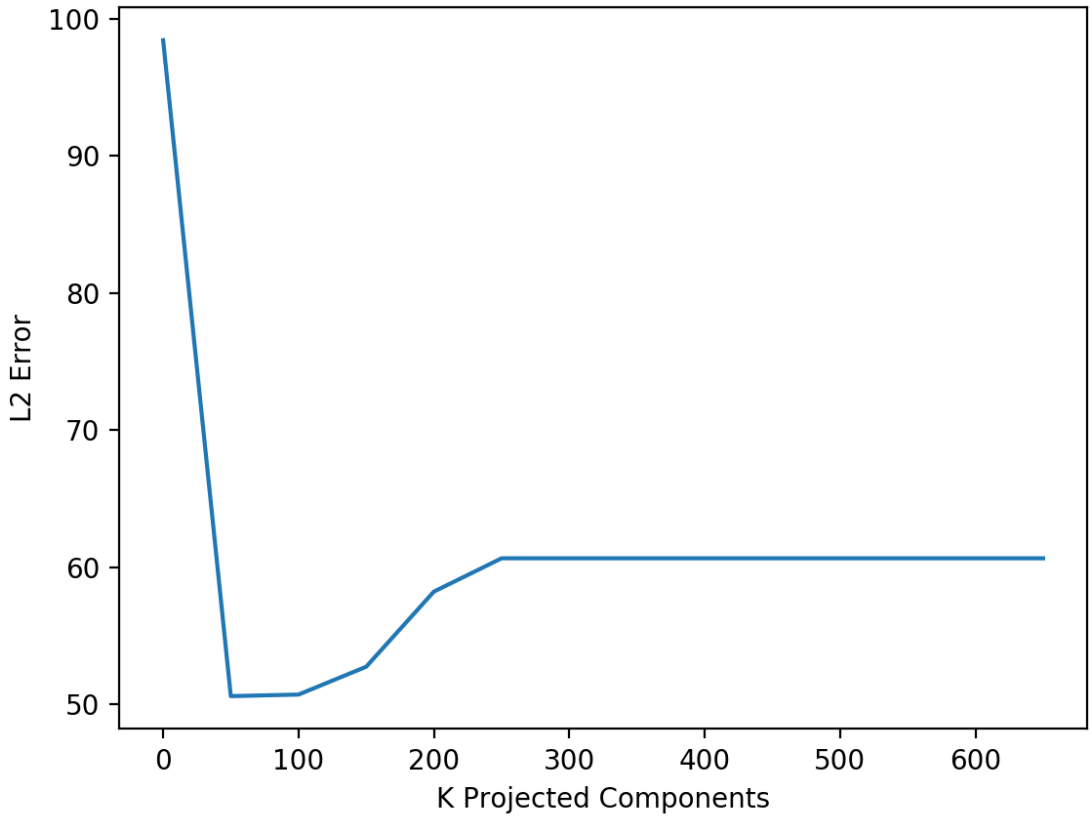


Figure 4: The error on the held out set versus how many projected components were considered. This plots the error normalized by the number of data-points, but unnormalized plots are fine.

```
    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:
```

```
51      y_f = np.array([y.flatten()])
        #STANDARDIZE DATA
53      y_f = (y_f/ 255.0) * 2.0 - 1.0
        #SUBTRACT MEAN
55      y_f = self.ss_y.transform(y_f)

57      Y_test.append(y_f[0,:])

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


63
      self.X_test_ridge = np.vstack(X_test)
65    self.Y_test_ridge = np.vstack(Y_test)

67  def ridge_regresion(self):

69    #PERFORM RIDGE REGRESSION
      X_G = np.matmul(self.X_ridge.T, self.X_ridge)
71
      lmbda = 0.00001
73    A = X_G+lmbda*np.eye(X_G.shape[0])

75    p_1 = LA.inv(A)

77    w_ridge = []
      for i in range(self.y_dim):
79
        p_2 = np.matmul(self.Y_ridge[:,i], self.X_ridge)
81      ans = np.matmul(p_2,p_1)
        w_ridge.append(ans)
83
      #RETURNED LEARNED WEIGHTS
85    self.w_ridge = np.vstack(w_ridge)

87  def measure_error_on_test(self):

89      #MAKE A PREDICTION ON THE PROJECTED DATA
        prediction = np.matmul(self.w_ridge, self.X_test_ridge.T)
91
        #MEASURE HOW CLOSE THE PREDICTION IS TO THE TRUE ANSWER
93      evaluation = self.Y_test_ridge.T - prediction

95      dim,num_data = evaluation.shape

97      error = []

99      #COMPUTE THE L2 NORM
        for i in range(num_data):
101
          #COMPUTE L2 NORM FOR EACH VECTOR THEN SQUARE
103       error.append(LA.norm(evaluation[:,i])**2)

105     #RETURN AVERAGE ERROR
        return np.mean(error)
107
  def project_data(self, sing_vec,k=100):
109
```

```
    #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()

    proj = [0,50,100,150,200,250,300,350,400,450,500,650]
    error_test = []


    for k in proj:

        ####COMPUTE MATRIX P_k
        X_proj = hw5_sol.project_data(sing_val,sing_vec,proj=k)



        #####COMPUTE PROJECTED DATA####
        hw5_sol.compute_projected_data_matrix(X_proj)
        #COMPUTE REGRESSION
        hw5_sol.ridge_regresion()

        #COMPUTE TEST ERROR
        test_error = hw5_sol.measure_error_on_test()

        error_test.append(test_error)

    #PLOT PROJECTION VERSUS ERROR

    plt.plot(proj,error_test)

    plt.ylabel('L2 Error')
    plt.xlabel('K Projected Components')
    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.** Note: You can use the code from above to visualize the images.

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.

```
def draw_images(self):
    count = 0
```

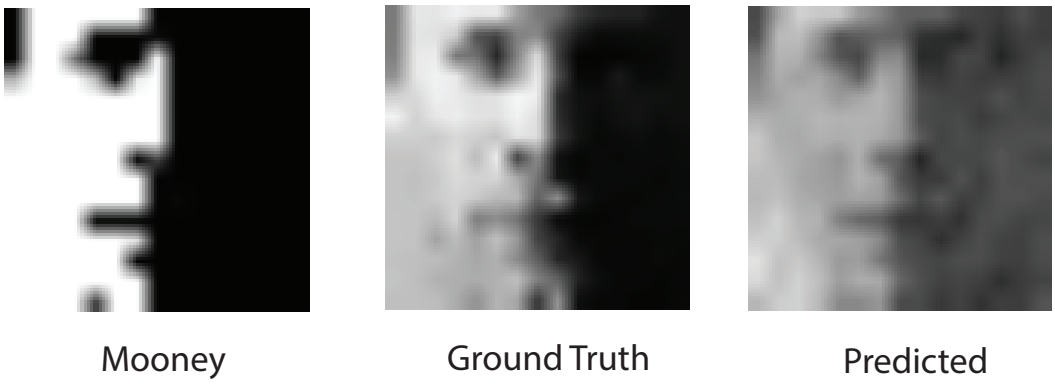


Figure 5: Example results with the input on the left and output on the right

```
    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
```

5 Fruits!

The goal of the problem is help the class build a dataset for image classification, which will be used later in the course to classify fruits and vegetables. Please pick ten of the following fruits:

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

1. Apple
2. Banana
3. Oranges

4. Grapes
5. Strawberry
6. Peach
7. Cherry
8. Nectarine
9. Mango
10. Pear
11. Plum
12. Watermelon
13. Pineapple

Take two pictures of each specific fruit, for a total of 20 fruit pictures, against any background such that the fruit is centered in the image and the fruit takes up approximately a third of the image in area; see below for examples. Save these pictures as .png files. **Do not** save the pictures as .jpg files, since these are lower quality and will interfere with the results of your future coding project. Place all the images in a folder titled *data*. Each image should be titled *[fruit name]_[number].png* where [number]$\in \{0, 1\}$. Ex: apple_0.png, apple_1.png, banana_0.png, etc ... (the ordering is irrelevant). Please also include a file titled *rich_labels.txt* which contain entries on new lines prefixed by the file name *[fruit name]_[number]*, followed by a description of the image (maximum of eight words) with only a space delimiter in between. Ex: apple_0 one fuji red apple on wood table. To turn in the folder compress the file to a .zip and upload it to Gradescope.

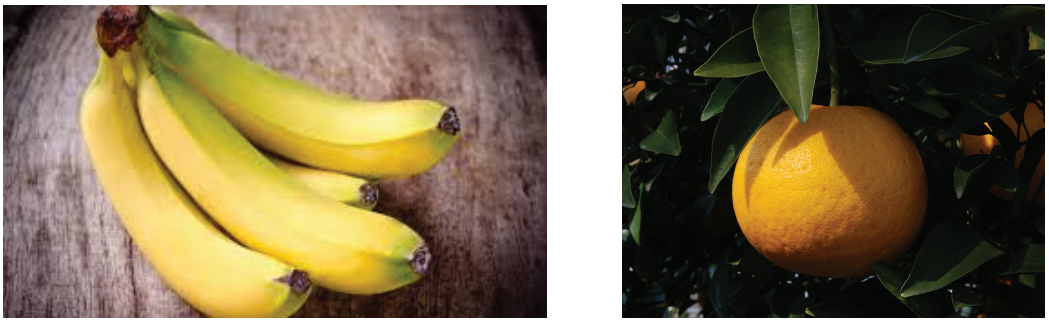


Figure 7: Example of properly centered images of four bananas against a table (left) and one orange on a tree (right).

Please keep in mind that data is an integral part of Machine Learning. A large part of research in this field relies heavily on the integrity of data in order to run algorithms. It is, therefore, vital that your data is in the proper format and is accompanied by the correct labeling not only for your grade on this section, but for the integrity of your data.

Note that if your compressed file is over 100 MB, you will need to downsample the images. You can use the following functions from skimage to help.

```
from skimage.io import imread, imsave
from skimage.transform import resize
```

6 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn 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.