CS 189 Introduction to Machine Learning
Fall 2018

HW2

This homework is due **Wednesday, September 12 at 10 p.m.**.

2 Prediction error of Ridge Regression

(a) Let A be a $d \times n$ matrix and B be a $n \times d$ matrix. For any $\mu > 0$, **show that** $(AB + \mu I)^{-1}A = A(BA + \mu I)^{-1}$, if $AB + \mu I$ and $BA + \mu I$ are invertible.

Solution: We begin with an equality

$$ABA + \mu A = ABA + \mu A.$$

Factoring both sides,

$$A(BA + \mu I) = (AB + \mu I)A.$$

Lastly, if if $AB + \mu I$ and $BA + \mu I$ are invertible, we can left-multiply by $(BA + \mu I)^{-1}$ and right-multiply by $(AB + \mu I)^{-1}$. This yields

$$(AB + \mu I)^{-1}A(BA + \mu I)(BA + \mu I)^{-1} = (AB + \mu I)^{-1}(AB + \mu I)A(BA + \mu I)^{-1}.$$

Simplifying,

$$(AB + \mu I)^{-1}A = A(BA + \mu I)^{-1},$$

as needed.

(b) Let $X \in \mathbb{R}^{n \times d}$ be n samples of d features, and $y \in \mathbb{R}^n$ be the corresponding n samples of the quantity that you would like to predict with regression. Let

$$\widehat{\theta}_\lambda = \arg\min_\theta \|X\theta - y\|_2^2 + \lambda\|\theta\|_2^2,$$

for $\lambda > 0$, be the solution to the ridge regression problem.

Show that $\widehat{\theta}_\lambda = X^\top(XX^\top + \lambda I)^{-1}y$.

Solution:

Start by taking the gradient of the loss function as follows:

$$\begin{aligned}\nabla_\theta(\|X\theta - y\|_2^2 + \lambda\|\theta\|_2^2) &= (X\theta - y)^\top X + \lambda\theta^\top \\ &= \theta^\top X^\top X - Y^\top X + \lambda\theta^\top \\ &= X^\top X\theta - X^\top y + \lambda\theta = 0\end{aligned}$$

Therefore,

$$\widehat{\theta}_\lambda = (X^\top X + \lambda I)^{-1} X^\top y$$

Using part a), we have $\theta = X^\top (XX^\top + \lambda I)^{-1} y$.

Recall that $(XX^\top + \lambda I)$ is positive definite and has real, positive eigenvalues when $\lambda > 0$. The invertibility of this matrix implies a unique solution for $\widehat{\theta}_\lambda$.

(c) Suppose X has the singular value decomposition $U\Sigma V^\top$, where $\Sigma = \text{diag}(s_1, \cdots, s_d)$, $s_i \geq 0$. Using part b), **show that $\widehat{\theta}_\lambda = VDU^\top y$, where D is a diagonal matrix to be determined.**

Solution: Notice that the SVD of $X^\top$ is $V\Sigma U^\top$. For convenience, write $\Sigma = \text{diag}(s_1, \cdots, s_2)$. By computation, we have

$$\begin{aligned}(X^\top X + \lambda I) &= V\Sigma U^\top U\Sigma V^\top + V(\lambda I)V^\top && (1)\\ &= V(\Sigma^2 + \lambda I)V^\top && (2)\end{aligned}$$

$(X^\top X + \lambda I)$ can thus be diagonalized into the form $V(\Sigma^2 + \lambda I)V^\top$, with $V\,\text{diag}(\frac{1}{s_i^2+\lambda})V^\top$ as its inverse. This allows us to write

$$\widehat{\theta}_\lambda = VDU^\top y$$

where

$$D = \text{diag}(\frac{s_i}{s_i^2 + \lambda})$$

(d) Let $\widehat{y}_\lambda = X\widehat{\theta}_\lambda$ be the predictions made by the ridge regressor $\widehat{\theta}_\lambda$. **Show that $\widehat{y}_\lambda = Py$, where P is a matrix to be determined. Comment on what P is doing to y.**

Solution: For convenience write $U = [u_1, \cdots, u_d]$ where u_i is an n-dimensional column vector. Note that u_i's are orthonormal.

$$\widehat{y}_\lambda = X\widehat{\theta}_\lambda = XVDU^\top y = U\Sigma DU^\top y = \sum_{i=1}^{d} u_i \frac{s_i^2}{s_i^2 + \lambda} u_i^\top$$

By the definition of PCA, we know that $XV = U\Sigma$ are the so-called principal components of the data matrix X. Thus ridge regression projects y onto these components, $s_i u_i$'s. It also shrinks the coefficients of the components, since $\lambda > 0$.

(e) Now suppose we have $y = X\theta_* + z$, where $\theta_* \in \mathbb{R}^d$ and $z = \mathcal{N}(0, \sigma^2 I) \in \mathbb{R}^n$ ($\sigma > 0$). Further suppose that X is an orthogonal matrix, that is, $X^\top X = I$.

$\mathbb{E}\|X(\widehat{\theta}_\lambda - \theta_*)\|^2$ is the expected squared difference between the predictions made by the ridge regressor $\widehat{y}_\lambda$ (see previous part) and y, where the expectation is taken with respect to the random variable z. ($\|\cdot\|$ denotes the $\ell 2$ norm.) We can think of this expression as the (in-sample) prediction error.

Show that $\mathbb{E}\|X(\widehat{\theta}_\lambda - \theta_*)\|^2 = \frac{1}{(1+\lambda)^2}\left(\lambda^2\|\theta_*\|^2 + d\sigma^2\right)$.

Solution: First we compute $\widehat{\theta}_\lambda - \theta_*$:

$$\widehat{\theta}_\lambda - \theta_* = (X^\top X + \lambda I)^{-1} X^\top y - \theta_* = ((1+\lambda)I)^{-1} X^\top (X\theta_* + z) - \theta_* = -\frac{\lambda}{1+\lambda}(\theta_*) + \frac{1}{1+\lambda} X^\top z$$

Since X is orthogonal, it is unitary invariant. Therefore,

$$\begin{aligned}\mathbb{E}\|X(\widehat{\theta}_\lambda - \theta_*)\|^2 &= \mathbb{E}\|\widehat{\theta}_\lambda - \theta_*\|^2 \\ &= \mathbb{E}\| - (\frac{\lambda}{1+\lambda})(\theta_*) + \frac{1}{1+\lambda}X^\top z\|^2 \\ &= \frac{\lambda^2}{(1+\lambda)^2}\|\theta_*\|^2 + \frac{1}{(1+\lambda)^2}d\sigma^2,\end{aligned}$$

since z is zero mean and $\mathbb{E}\|z\| = d\sigma^2$.

(f) **What is the λ^* that you should pick to minimize the prediction error you computed in part e)? Comment on how d, σ^2, and θ_* each affect the optimal choice of the regularization parameter λ.**

Solution: Differentiating $\mathbb{E}\|X(\widehat{\theta}_\lambda - \theta_*)\|^2 = \frac{\lambda^2}{(1+\lambda)^2}\|\theta_*\|^2 + \frac{1}{(1+\lambda)^2}d\sigma^2$ with respect to λ gives

$$\frac{2(\|\theta_*\|^2\lambda - d\sigma^2)}{(\lambda+1)^3}$$

Therefore, we have that

$$\lambda^* = d\sigma^2/\|\theta^*\|^2$$

Higher d (more features), higher σ (greater noise), and smaller norm of θ_* (smaller signal) all make us pick larger λ^*.

(g) **What is the prediction error $\mathbb{E}\|X(\widehat{\theta}_{\lambda_*} - \theta_*)\|^2$ using the optimized λ^*? Compare this prediction error to the prediction error for Ordinary Least Squares**

$$\mathbb{E}\|X(\widehat{\theta}_{OLS} - \theta_*)\|^2 = d\sigma^2,$$

which you computed in problem 5 f) of Homework 1.

Solution: By computation, we have that

$$\mathbb{E}\|X(\widehat{\theta}_{\lambda_*} - \theta_*)\|^2 = \frac{d\sigma^2\|\theta^*\|^2}{\|\theta^*\|^2 + d\sigma^2}$$

Since $\|\theta^*\|^2 < \|\theta^*\|^2 + d\sigma^2$, for $\sigma > 0$, we always have that $\mathbb{E}\|X(\widehat{\theta}_{\lambda_*} - \theta_*)\|^2 < \mathbb{E}\|X(\widehat{\theta}_{OLS} - \theta_*)\|^2$! Therefore, in this particular toy problem, ridge regression always has lower prediction error than OLS provided that we pick the optimal regularization parameter—which is not so easy in practice since it would require us to know σ^2 and $\|\theta_*\|$.

3 Independence and Multivariate Gaussians

As described in lecture, a covariance matrix $\Sigma \in \mathbb{R}^{N\times N}$ for a random variable $X \in \mathbb{R}^N$ with the following values, where $\text{cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]$ is the covariance between the i-th and j-th elements of the random vector X:

$$\Sigma = \begin{bmatrix} \text{cov}(X_1, X_1) & \dots & \text{cov}(X_1, X_n) \\ \dots & & \dots \\ \text{cov}(X_n, X_1) & \dots & \text{cov}(X_n, X_n) \end{bmatrix}. \tag{3}$$

Recall that the density of an N dimensional Multivariate Gaussian Distribution $\mathcal{N}(\mu, \Sigma)$ is defined as follows when Σ is positive definite:

$$f(x) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}. \tag{4}$$

Here, $|\Sigma|$ denotes the determinant of the matrix Σ.

(a) Consider the random variables X and Y in $\mathbb{R}$ with the following conditions.

(i) X and Y can take values $\{-1, 0, 1\}$.

(ii) When X is 0, Y takes values 1 and -1 with equal probability ($\frac{1}{2}$). When Y is 0, X takes values 1 and -1 with equal probability ($\frac{1}{2}$).

(iii) Either X is 0 with probability ($\frac{1}{2}$), or Y is 0 with probability ($\frac{1}{2}$).

Are X and Y uncorrelated? Are X and Y independent? Prove your assertions. *Hint:* Write down the joint probability of (X, Y) for each possible pair of values they can take.

Solution: Essentially, there are 4 possible points (X, Y) can be, all with equal probability ($\frac{1}{4}$): $\{(0, 1), (0, -1), (1, 0), (-1, 0)\}$, If graphed onto the Cartesian Plane, these point form "crosshairs".

To show that X and Y are uncorrelated, we need to prove:

$$E[(X - \mu_X)(Y - \mu_Y)] = E[X - \mu_X]E[Y - \mu_Y]$$
$$E[XY] = E[X]E[Y] = 0$$

Since for μ_X and μ_Y, we see that

$$E[X] = E[Y] = \frac{1}{2} * 0 + \frac{1}{2} * (\frac{1}{2} + \frac{-1}{2}) = 0$$

Notice for that whenever X is nonzero, Y is zero (vice versa). Thus, E[XY] = 0 since one of the terms is always zero, and we have shown that X and Y are uncorrelated. However, to show that X and Y are independent, we must show that:

$$P(X|Y) = P(X)$$

Unfortunately, this is not the case. $P(X = 0) = \frac{1}{2}$, but $P(X = 0|Y = 1) = 1$. Thus, X and Y are not independent.

(b) For $X = [X_1, \cdots, X_n]^\top \sim \mathcal{N}(\mu, \Sigma)$, **verify that if X_i, X_j are independent (for all $i \neq j$), then Σ must be diagonal, that is, X_i, X_j are uncorrelated.** **Solution:** Recall that if random variables Z, W are independent, we have $\mathbb{E}[ZY] = \mathbb{E}[Z]\mathbb{E}[Y]$. Thus we have the covariance $\mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)] = \mathbb{E}[X_i - \mu_i]\mathbb{E}[X_j - \mu_j] = 0 \cdot 0$ is 0, i.e. the X_i, X_j are uncorrelated.

(c) Let $N = 2$, $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, and $\Sigma = \begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix}$. Suppose $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma)$. **Show that X_1, X_2 are independent if $\beta = 0$.** Recall that two continuous random variables W, Y with joint density $f_{W,Y}$ and marginal densities f_W, f_Y are independent if $f_{W,Y}(w, y) = f_W(w)f_Y(y)$. **Solution:** Recall that the marginal density of two jointly Gaussian random variables is also Gaussian. In particular, we have that $X_1 \sim \mathcal{N}(\mu_1, \alpha)$ and $X_2 \sim \mathcal{N}(\mu_2, \gamma)$. Let's denote the marginal densities as $f_{X_1}(\cdot)$ and $f_{X_2}(\cdot)$.

Since $\beta = 0$, we may compute $\Sigma^{-1} = \begin{pmatrix} \alpha^{-1} & 0 \\ 0 & \gamma^{-1} \end{pmatrix}$.

Let's write out the joint density of X_1, X_2.

$$\begin{aligned} f_{X_1,X_1}(x_1, x_2) &= \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)} \\ &= \frac{1}{\sqrt{(2\pi)^2 \alpha\gamma}} e^{-\frac{1}{2}(\alpha^{-1}(x_1-\mu_1)^2 + \gamma^{-1}(x_2-\mu_2)^2)} \\ &= \frac{1}{\sqrt{(2\pi)\alpha a}} e^{-\frac{1}{2}(\alpha^{-1}(x_1-\mu_1)^2)} \cdot \frac{1}{\sqrt{(2\pi)\alpha\gamma}} e^{-\frac{1}{2}(\gamma^{-1}(x_2-\mu_2)^2)} \\ &= f_{X_1}(x_1) \cdot f_{X_2}(x_2) \end{aligned}$$

This proves that X_1, X_2 are independent if $\beta = 0$. Note that we don't need to verify that $f_{X_1}(x_1)$ and $f_{X_2}(x_2)$ are properly normalized (i.e. integrate to 1), since we can always shift around constant factors to ensure that this is the case.

(d) Consider a data point x drawn from a N-dimensional zero mean Multivariate Gaussian distribution $\mathcal{N}(0, \Sigma)$, as shown above. Assume that Σ^{-1} exists. **Prove that there exists matrix $A \in R^{N,N}$ such that $x^\top \Sigma^{-1} x = \|Ax\|_2^2$ for all vectors x. What is the matrix A?**

Solution:

Use the Spectral Decomposition Theorem to convert Σ into the following: U is a unitary matrix of orthonormal eigenvectors $\mathbf{e_i}$ $\forall i \in [0...N]$ and D is a diagonal matrix with eigenvalues λ_i $\forall i \in [0...N]$ located at indices corresponding to eigenvectors in U. Note that all the eigenvalues are greater 0 since Σ is positive semidefine (it is a covariance matrix). Indeed, for any $v \in \mathbb{R}^n$, $v^\top \Sigma v = \mathbb{E}[v^\top (XX^\top) v] = \mathbb{E}[\|Xv\|_2^2] \geq 0$ (but this was not necessary to show to receive full credit). In fact, since Σ is invertible, all eigenvalues of Σ are strictly positive, Hence, we may write

$$\Sigma = UDU^\top,$$

and therefore

$$\Sigma^{-1} = (UDU^\top)^{-1} = (U^\top)^{-1}D^{-1}U^{-1} = UD^{-1}U^\top.$$

This is because a unitary matrix U is such that $U^{-1} = U^\top$. Note that if the diagonal matrix D has values $d_{i,i}$ $\forall i$, then D^{-1} has value $\frac{1}{d_{i,i}}$ $\forall i$. Once again, since Σ was positive definite, the value $\frac{1}{d_{i,i}}$ exists.

Now, we decompose D^{-1} into its square-root by defining Q as a diagonal matrix with diagonal values $\frac{1}{\sqrt{d_{i,i}}}$. Verify that $QQ = D^{-1}$ and that $Q^\top = Q$. Thus, we have:

$$\Sigma^{-1} = UD^{-1}U^\top = UQQU^\top = UQQ^\top U^\top \tag{5}$$
$$\Sigma^{-1} = A^\top A, \tag{6}$$

where we've defined $A = (UQ)^\top$. Therefore,

$$x^\top \Sigma^{-1} x = x^\top A^\top A x = (Ax)^\top (Ax) = \|Ax\|_2^2. \tag{7}$$

Note that A is not necessarily unique, since if $A^\top A = \Sigma^{-1}$, then also $(QA)^\top QA = \Sigma^{-1}$ for any orthogonal Q.

(e) Let's constrain x to be on the unit sphere. In other words, the ℓ_2 norm (or magnitude) of vector x is 1 ($\|x\|_2 = 1$). In this case, **what are the maximum and minimum values of $\|Ax\|_2^2$? In other words, $\max_{x:\|x\|_2=1} \|Ax\|_2^2$ and $\min_{x:\|x\|_2=1} \|Ax\|_2^2$?**

Solution:

$x^\top \Sigma^{-1} x$ is a scalar written in vector quadratic form. It looks like an incomprehensible value, but when we convert it to $\|Ax\|_2^2$, we see that in reality its just the squared L2 norm of Ax, which measures the squared distance from the data vector x from the mean (in this case 0). Note that we can change the mean to be any arbitrary value without loss of generality.

Recall from Part B our decomposition for Σ^{-1}, which was as follows where U is a unitary matrix, D is a diagonal matrix.

$$\Sigma^{-1} = UD^{-1}U^\top = A^\top A \tag{8}$$

Note that $\|x\|_2 = 1$ and $\|Ux\|_2 = 1$ since unitary matrices are orthonormal and preserve magnitude. Define $q = Ux$, we have

$$\|Ax\|_2^2 = x^\top A^\top A x = x^\top U D^{-1} U^\top x = q^\top D^{-1} q \tag{9}$$

We can choose our x such that q will be any Euclidean Basis Vector $\mathbf{e_i}$ such that the ith element is 1 and all other elements are 0. Therefore, the maximum value that $\|Ax\|_2^2$ is $\frac{1}{\lambda_i}$, where λ_i is the minimum eigenvalue of Σ. The minimum value that $\|Ax\|_2^2$ is $\frac{1}{\lambda_j}$, where λ_j is the maximum eigenvalue of Σ.

(f) If we had $X_i \perp\!\!\!\perp X_j \; \forall i,j$ ($\perp\!\!\!\perp$ denotes independence), **what is the intuitive meaning for the maximum and minimum values of $\|Ax\|_2^2$?** Suppose you wanted to choose an x on the unit sphere to maximize the density function $f(x)$ in Eq (4); **what x should you choose?**

Solution:

As we showed in a previous part, if we have $X_i \perp\!\!\!\perp X_j \; \forall i,j$, then $cov(X_i,X_j) = 0 \; \forall i,j$, meaning that off diagonal terms for Σ are 0. Thus, we can find Σ^{-1} directly, as follows.

$$\Sigma_{i,j}^{-1} = \begin{cases} \frac{1}{\sigma_i^2} \text{ if } i = j \\ 0 \text{ else} \end{cases}$$

Therefore, if we have $X_i \perp\!\!\!\perp X_j \; \forall i,j$, the maximum value that $\|Ax\|_2^2$ is $\frac{1}{\sigma_i^2}$, where σ_i^2 is the minimum variance. The minimum value of $\|Ax\|_2^2$ is $\frac{1}{\sigma_j^2}$, where σ_j^2 is the maximum variance.

To maximize $f(x)$, we want the superscript above the exponent to be minimal since there is a negative sign. Thus, for $\|Ax\|_2^2$ to be minimal, we want to choose x to be the unit eigenvector corresponding to the maximal eigenvalue λ_j (i.e. maximum variance σ_j^2).

4 Blair and their giant peaches

Make sure to submit the code you write in this problem to "HW2 Code" on Gradescope.

Blair is a mage testing how long they can fly a collection of giant peaches. They has n training peaches – with masses given by $x_1, x_2, \dots x_n$ – and flies these peaches once to collect training data. The experimental flight time of peach i is given by y_i. They believes that the flight time is well approximated by a polynomial function of the mass

$$y_i \approx w_0 + w_1 x_i + w_2 x_i^2 \cdots + w_D x_i^D$$

where their goal is to fit a polynomial of degree D to this data. Include all text responses and plots in your write-up.

(a) **Show how Blair's problem can be formulated as a linear regression problem.**

Solution: The problem is to find the coefficients w_d such that the squared error is minimized:

$$\min_{w_0,\dots,w_D} \sum_i (w_0 + w_1 x_i + w_2 x_i^2 \cdots + w_D x_i^D - y_i)^2$$

Assume that we construct the following matrix:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^D \\ 1 & x_2 & x_2^2 & \dots & x_2^D \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^D \end{bmatrix}$$

Then the problem can be written as:

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

(b) You are given data of the masses $\{x_i\}_{i=1}^n$ and flying times $\{y_i\}_{i=1}^n$ in the "x_train" and "y_train" keys of the file `1D_poly.mat` with the masses centered and normalized to lie in the range $[-1, 1]$. **Write a script by completing `b.py` to do a least-squares fit (taking care to include a constant term) of a polynomial function of degree D to the data.** Letting f_D denote the fitted polynomial, **plot the average training error $R(D) = \frac{1}{n}\sum_{i=1}^n (y_i - f_D(x_i))^2$ against D in the range $D \in \{0, 1, 2, 3, \ldots, n-1\}$.** You may not use any library other than `numpy` and `numpy.linalg` for computation.

Solution:

See Figure 1 for the plot.

```python
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np
import scipy.io as spio


# There is numpy.linalg.lstsq, whicn you should use outside of this classs
def lstsq(A, b):
    return np.linalg.solve(A.T @ A, A.T @ b)


def main():
    data = spio.loadmat('1D_poly.mat', squeeze_me=True)
    x_train = np.array(data['x_train'])
    y_train = np.array(data['y_train']).T

    n = 20  # max degree
    err = np.zeros(n - 1)

    # fill in err
    for d in range(n - 1):
        D = d + 1
        for i in range(D + 1):
            if i == 0:
                Xf = np.array([1] * x_train.size)
            else:
                Xf = np.vstack([np.power(x_train, i), Xf])
        Xf = Xf.T

        w = lstsq(Xf, y_train)
        y_predicted = Xf @ w
        err[d] = (np.linalg.norm(y_train - y_predicted)**2) / n

    plt.plot(err)
    plt.xlabel('Degree of Polynomial')
    plt.ylabel('Training Error')
    plt.show()


if __name__ == "__main__":
    main()
```

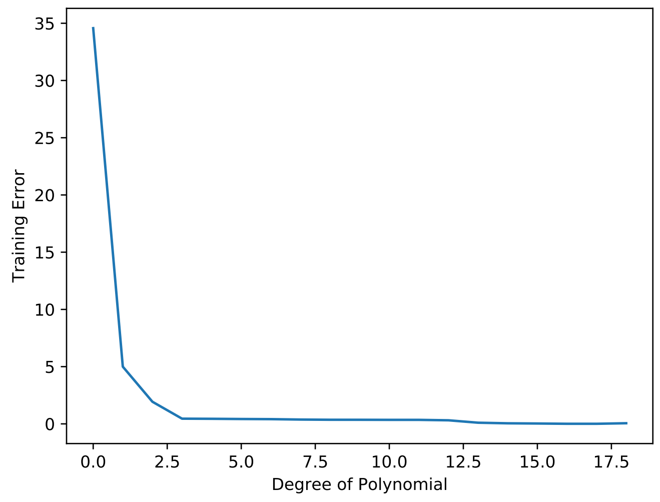

Figure 1: Training Error: **Result for 1D_poly.mat**

(c) **How does the average training error behave as a function of D, and why? What happens if you try to fit a polynomial of degree n with a standard matrix inversion method?**

Solution: The training error decreases since we have more degrees of freedom to fit the dataset. If we try to fit a polynomial of degree $n-1$, we will have enough parameters to fit the data exactly, so the training error will be zero. Using a polynomial of degree n results in a non-invertible data matrix, and so we cannot use a standard matrix inversion method to find our solution.

(d) Blair has taken CS189 so decides that they needs to run another experiment before deciding that their prediction is true. They runs another fresh experiment of flight times using the same peaches, to obtain the data with key "y_fresh" in 1D_POLY.MAT. Denoting the fresh flight time of peach i by $\tilde{y}_i$, **by completing `c.py`, plot the average error $\tilde{R}(D) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - f_D(x_i))^2$ for the same values of D as in part (b) using the polynomial approximations f_D also from the previous part. How does this plot differ from the plot in (b) and why?**

Solution: The plots are shown in Figure 2. The plots are different from the training errors since the data was generated afresh. While increasing the polynomial degree served to fit the noise in the training data, we cannot hope to fit noise by using the same model for fresh data. Hence, our performance degrades as we increase polynomial degree, with a minimum seen at the true model order.

```python
#!/usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np
import scipy.io as spio


# There is numpy.linalg.lstsq, whicn you should use outside of this classs
def lstsq(A, b):
    return np.linalg.solve(A.T @ A, A.T @ b)

```

```

def main():
    data = spio.loadmat('1D_poly.mat', squeeze_me=True)
    x_train = np.array(data['x_train'])
    y_train = np.array(data['y_train']).T
    y_fresh = np.array(data['y_fresh']).T

    n = 20  # max degree
    err_train = np.zeros(n - 1)
    err_fresh = np.zeros(n - 1)

    # fill in err_fresh and err_train
    for d in range(n - 1):
        D = d + 1
        for i in range(D + 1):
            if i == 0:
                Xf = np.array([1] * n)
            else:
                Xf = np.vstack([np.power(x_train, i), Xf])
        Xf = Xf.T

        w = lstsq(Xf, y_train)
        err_train[d] = (np.linalg.norm(y_train - Xf @ w)**2) / n
        err_fresh[d] = (np.linalg.norm(y_fresh - Xf @ w)**2) / n

    plt.figure()
    plt.ylim([0, 6])
    plt.plot(err_train, label='train')
    plt.plot(err_fresh, label='fresh')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

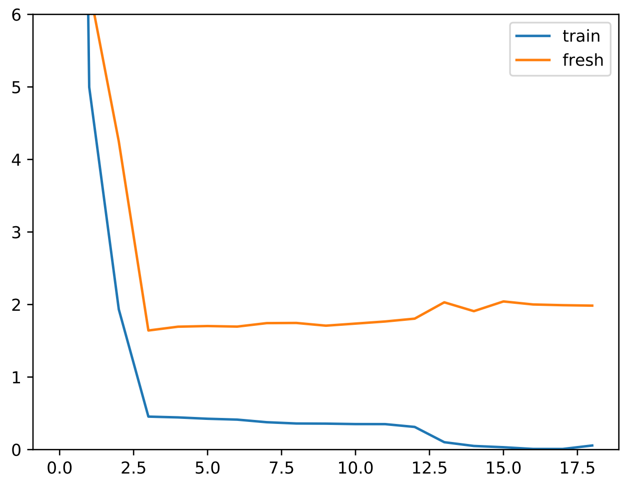

Figure 2: Fresh Error yfresh: **Result for 1D_poly.mat**

(e) **How do you propose using the two plots from parts (b) and (d) to "select" the right polynomial model for Blair?**

Solution: The right model is the one that minimizes the error in the fresh dataset. The minimizer is at $D = 4$.

(f) Blair has a new hypothesis – the flying time is actually a function of the mass, smoothness, size, and sweetness of the peach, and some multivariate polynomial function of all of these parameters. A D-multivariate polynomial function looks like

$$f_D(\mathbf{x}) = \sum_j \alpha_j \prod_i x_i^{p_{ji}},$$

where $\forall j : \sum_i p_{ji} \le D$. Here α_j is the scale constant for jth term and p_{ji} is the exponent of x_i in jth term. The data in `polynomial_regression_samples.mat` (100000×5) with columns corresponding to the 5 attributes of the peach. **Use 4-fold cross-validation to decide which of $D \in \{0, 1, 2, 3, 4, 5, 6\}$ is the best fit for the data provided**. For this part, compute the polynomial coefficients via ridge regression with penalty $\lambda = 0.1$, instead of ordinary least squares. You are not allowed to use any library other than `numpy` and `numpy.linalg`. Write your implementation by completing `fg.py`.

Solution: Please refer to the next part for the general implementation.

(g) Now **redo the previous part, but use 4-fold cross-validation on all combinations of $D \in \{1, 2, 3, 4, 5, 6\}$ and $\lambda \in \{0.05, 0.1, 0.15, 0.2\}$** - this is referred to as a grid search. **Find the best D and λ that best explains the data using ridge regression. Print the average training/validation error per sample for all D and λ.** Again, write your implementation by completing `fg.py`.

Solution: Minimum of cross validation error happens at $D = 4$ and $\lambda = 0.15$.

```
Average train error:
[[0.05856762013 0.05856762066 0.05856762225 0.05856762491 0.05856762862]
 [0.05848920002 0.05848948229 0.0584902034  0.05849122    0.05849243261]
 [0.05846063295 0.05848702853 0.05848893458 0.05849036454 0.05849178744]
 [0.05839260718 0.05848700898 0.05848892445 0.05849035741 0.05849178171]
 [0.05831118703 0.05848700861 0.05848892426 0.05849035728 0.05849178161]
 [0.05815215946 0.0584870086  0.05848892426 0.05849035728 0.05849178161]]
Average valid error:
[[0.05857468619 0.05857468536 0.0585746856  0.05857468691 0.05857468927]
 [0.05852250667 0.05852112813 0.05852038752 0.05852010745 0.05852016181]
 [0.05854617135 0.0585210268  0.05852032221 0.05852006042 0.0585201252 ]
 [0.05860046897 0.05852102354 0.05852032035 0.05852005896 0.05852012386]
 [0.05869725681 0.05852102357 0.05852032036 0.05852005896 0.05852012386]
 [0.05887534948 0.05852102357 0.05852032036 0.05852005896 0.05852012386]]
```

In the following implementation, we run cross-validation to find errors for each D separately, by varying the value of λ. This is purely for pedagogical purposes. You can combine these two steps by running through D in an outer loop. Note that we can collect all of these errors, and then choose the model that minimizes the cross-validation error.

```python
import matplotlib.pyplot as plt
import numpy as np
import scipy.io as spio
```

```

data = spio.loadmat('polynomial_regression_samples.mat', squeeze_me=True)
data_x = data['x']
data_y = data['y']
Kc = 4  # 4-fold cross validation
KD = 6  # max D = 6
LAMBDA = [0, 0.05, 0.1, 0.15, 0.2]

feat_x = 0


def lstsq(A, b, lambda_=0):
    return np.linalg.solve(A.T @ A + lambda_ * np.eye(A.shape[1]), A.T @ b)


def assemble_feature(x, D):
    n_feature = x.shape[1]
    Q = [(np.ones(x.shape[0]), 0, 0)]
    i = 0
    while Q[i][1] < D:
        cx, degree, last_index = Q[i]
        for j in range(last_index, n_feature):
            Q.append((cx * x[:, j], degree + 1, j))
        i += 1
    return np.column_stack([q[0] for q in Q])


def fit(D, lambda_):
    Ns = int(data_x.shape[0] * (Kc - 1) / Kc)  # training
    Nv = int(Ns / (Kc - 1))  # validation

    Etrain = np.zeros(4)
    Evalid = np.zeros(4)
    for c in range(4):
        valid_x = feat_x[c * Nv:(c + 1) * Nv]
        valid_y = data_y[c * Nv:(c + 1) * Nv]
        train_x = np.delete(feat_x, list(range(c * Nv, (c + 1) * Nv)), axis=0)
        train_y = np.delete(data_y, list(range(c * Nv, (c + 1) * Nv)))

        w = lstsq(train_x, train_y, lambda_=lambda_)
        Etrain[c] = np.mean((train_y - train_x @ w)**2)
        Evalid[c] = np.mean((valid_y - valid_x @ w)**2)

    return np.mean(Etrain), np.mean(Evalid)


def main():
    np.set_printoptions(precision=11)
    Etrain = np.zeros((KD, len(LAMBDA)))
    Evalid = np.zeros((KD, len(LAMBDA)))
    for D in range(KD):
        global feat_x
        feat_x = assemble_feature(data_x, D + 1)
        for i in range(len(LAMBDA)):
            Etrain[D, i], Evalid[D, i] = fit(D + 1, LAMBDA[i])

    print('Average train error:', Etrain, sep='\n')
    print('Average valid error:', Evalid, sep='\n')

    D, i = np.unravel_index(Evalid.argmin(), Evalid.shape)
    print("D =", D + 1)
    print("lambda =", LAMBDA[i])


if __name__ == "__main__":
    main()
```