CSE 546 HW1

DAVID P. FLEMING

Contents

Date: October 14^{th}, 2016.

Question 1

1.1. The expectation value for $max(X_1, X_2)$ is given by

$$E[X] = \int max(X_1, X_2) f(x) dx_1 dx_2 \tag{1}$$

where $f(x) = 1$ is the PDF for the uniform random variates. We consider two regions, one below the line defined by $X = X_1 = X_2$ and one above said line for our integrations. This gives us the following integral:

$$\begin{aligned} E[X] &= \int_0^1 \int_{x_2}^1 x_1 dx_1 dx_2 + \int_0^1 \int_{x_1}^1 x_2 dx_2 dx_1 \\ &= \int_0^1 (\frac{1}{2} - \frac{1}{2}x_2^2) dx_2 + \int_0^1 (\frac{1}{2} - \frac{1}{2}x_1^2) dx_1 \\ &= (\frac{1}{2}x_2 - \frac{1}{6}x_2^2)|_0^1 + (\frac{1}{2}x_1 - \frac{1}{6}x_1^2)|_0^1 \\ &= \frac{2}{3} \end{aligned} \tag{2}$$

1.2. Since

$$Var[X] = E[X^2] - E[X]^2 \tag{3}$$

and we now know $E[X] = 2/3$, we solve the integral presented above but with the integrand squared as follows

$$\begin{aligned} E[X^2] &= \int_0^1 \int_{x_2}^1 x_1^2 dx_1 dx_2 + \int_0^1 \int_{x_1}^1 x_2^2 dx_2 dx_1 \\ &= \int_0^1 (\frac{1}{3} - \frac{1}{3}x_2^3) dx_2 + \int_0^1 (\frac{1}{3} - \frac{1}{3}x_1^3) dx_1 \\ &= (\frac{1}{3}x_2 - \frac{1}{12}x_2^4)|_0^1 + (\frac{1}{3}x_1 - \frac{1}{12}x_1^4)|_0^1 \\ &= \frac{1}{2} \end{aligned} \tag{4}$$

which when combined with the definition for $Var[X]$ gives

$$Var[X] = \frac{1}{2} - \left(\frac{2}{3}\right)^2 = \frac{1}{18} \tag{5}$$

1.3. Since

$$Cov[X, X_1] = E[XX_1] - E[X]E[X_1] \tag{6}$$

and we know $E[X]$ from 1.1 and $E[X_1] = 1/2$ is the trivial result for the uniform distribution from $[0, 1]$, we need only calculate the first term as follows:

$$\begin{aligned} E[XX_1] &= \int_0^1 \int_{x_2}^1 x_1^2 dx_1 dx_2 + \int_0^1 \int_{x_1}^1 x_2 x_1 dx_2 dx_1 \\ &= \int_0^1 (\frac{1}{3} - \frac{1}{3}x_2^3) dx_2 + \int_0^1 x_1(\frac{1}{2} - \frac{1}{2}x_1^2) dx_1 \\ &= (\frac{1}{3}x_2 - \frac{1}{12}x_2^4)|_0^1 + (\frac{1}{4}x_1^2 - \frac{1}{8}x_1^4)|_0^1 \\ &= \frac{3}{8} \end{aligned} \tag{7}$$

giving us

$$Cov[X, X_1] = E[XX_1] - E[X]E[X_1] = \frac{3}{8} - \frac{2}{3}\frac{1}{2} = \frac{1}{24} \tag{8}$$

Note: For this question, I collaborated with Matt Wilde.

Question 2

2.1. For the log-likelihood of G given λ and i.i.d. samples, we have

$$\begin{aligned} LL &= \log\left(P(G|\theta)\right) \\ &= \Pi_i^n P(G_i|\theta) \\ &= \log\left(\frac{\lambda^{\Sigma_i^n k_i} e^{-\lambda n}}{\Pi_i^n k_i!}\right) \\ &= \Sigma_i^n k_i \log\lambda - \lambda n - \log\Pi_i^n k_i! \end{aligned} \tag{9}$$

2.2. To compute the MLE for λ in general,

$$\frac{\partial}{\partial\lambda}\left[LL\right] = 0 \tag{10}$$

which yields

$$\begin{aligned} 0 &= \frac{\partial}{\partial\lambda}\left(\Sigma_i^n k_i \log\lambda - \lambda n - \log\Pi_i^n k_i!\right) \\ &= \frac{\Sigma_i^n k_i}{\lambda} - n \\ \hat{\lambda}_{MLE} &= \frac{\Sigma_i^n k_i}{n} \end{aligned} \tag{11}$$

2.3. For the observed set G, I use Eqn. 11 to compute λ_{MLE} as

$$\hat{\lambda}_{MLE} = \frac{4+1+3+5+5+1+3+8}{8} = 3.75 \tag{12}$$

Note: For this question, I collaborated with Matt Wilde.

Question 3

3.1. To compute the LOOCV score natively, one must compute $\hat{y}$ $N-1$ times where $\hat{y}$ is the prediction vector computed via Ordinary Least-Squares (OLS). Therefore, the time complexity of computing LOOCV will be $\mathcal{O}(N*\mathcal{O}(OLS))$. To compute $\mathcal{O}(OLS)$, I will assume the naive algorithm for matrix inversion which has complexity $\mathcal{O}(d^3)$ for a $d \times d$ matrix and assume the naive algorithm for matrix multiplication which has complexity $\mathcal{O}(nmp)$ for the matrix product of $n \times m$ and $m \times p$ matrices.

For OLS, $\hat{Y_{OLS}} = X(X^TX)^{-1}X^TY$ where X is $N \times d$ and Y is $N \times 1$. The complexities of taking multiple matrix products sum since they happen sequentially. If we take the pairwise matrix products to compute $\hat{Y_{OLS}}$, we get $\mathcal{O}(Nd^2) + \mathcal{O}(Nd^2) + \mathcal{O}(dN^2) + \mathcal{O}(N^2)$. Combining those terms with the $\mathcal{O}(d^3)$ complexity addition from matrix inversion, I get a total complexity of $\mathcal{O}(d^3 + (1+2d)N^2 + Nd^2)$ which asymptotes to $\mathcal{O}(d^3+dN^2+Nd^2)$. If $N > d$, this term can be reduced further however I make no assumptions about whether d or N is larger. To compute LOOCV, we must compute $\hat{Y}$ $N-1$ times giving a total complexity of $\mathcal{O}(Nd^3+dN^3+N^2d^2)$.

3.2. Since in matrix notation we have $\hat{Y} = HY$, I can write

$$\hat{y_i} = \sum_{j=1}^{N} H_{ij}y_j \tag{13}$$

where $\hat{y}$ has length N.

3.3. To show that $\hat{Y}^{(-i)}$ is the estimator which minimizes SSE for the given Z_j, first I substitute Z_j for y_i in the given expression for SSE to get the following

$$SSE = \sum_{j=1}^{N}(Z_j - \hat{y_j})^2. \tag{14}$$

Now, I expand the above expression for SSE using the definition of Z_j

$$SSE = \sum_{j\neq i}^{N}(y_j - \hat{y_j})^2 + (\hat{y_i}^{(-i)} - \hat{y_i})^2 \tag{15}$$

which reduces to the original SSE with an additional $i = j$ term. Therefore $\hat{Y}^{(-i)}$ must be the estimator which minimizes the SSE for Z.

3.4. Analogous to the answer to Question 3.2, I can write $\hat{Y}^{(-i)} = HZ$ or in summation notation,

$$\hat{y_i}^{(-i)} = \sum_{j=1}^{N} H_{ij}Z_j \tag{16}$$

3.5. To show that $\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii}y_i - H_{ii}\hat{y}_i^{(-i)}$, I will take the left-hand side (LHS) of the equation and expand it with my definitions for $\hat{y}_i$ and $\hat{y}_i^{(-i)}$ from my answers to 3.2 and 3.4 as follows

$$\begin{aligned} \hat{y}_i - \hat{y}_i^{(-i)} = \\ \sum_{j=1}^{N} H_{ij}y_j - \sum_{j=1}^{N} H_{ij}Z_j = \\ \sum_{j\neq i}^{N} H_{ij}y_j + H_{ii}y_i - \sum_{j\neq i}^{N} H_{ij}y_j - H_{ii}\hat{y}_i^{(-i)} = \\ H_{ii}y_i - H_{ii}\hat{y}_i^{(-i)} = RHS \end{aligned} \tag{17}$$

as the $j \neq i$ term in Z_j allows us to cancel the summation terms, giving us the RHS as posited.

3.6. To derive the expected result, first I will rearrange $\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii}y_i - H_{ii}\hat{y}_i^{(-i)}$ as follows

$$\begin{aligned} \hat{y}_i - \hat{y}_i^{(-i)} &= H_{ii}y_i - H_{ii}\hat{y}_i^{(-i)} \\ \hat{y}_i - H_{ii}y_i &= \hat{y}_i^{(-i)} - H_{ii}\hat{y}_i^{(-i)} \\ \hat{y}_i - H_{ii}y_i &= (1 - H_{ii})\hat{y}_i^{(-i)} \\ \hat{y}_i^{(-i)} &= (\hat{y}_i - H_{ii}y_i)\frac{1}{1 - H_{ii}}. \end{aligned} \tag{18}$$

I now insert this expression for $\hat{y}_i^{(-i)}$ into the expression for LOOCV and reduce to derive the following

$$\begin{aligned} LOOVC &= \sum_{i=1}^{N}(y_i - \hat{y}_i^{(-i)})^2 \\ &= \sum_{i=1}^{N}\left(y_i - (\hat{y}_i - H_{ii}y_i)\frac{1}{1 - H_{ii}}\right)^2 \\ &= \sum_{i=1}^{N}\left(y_i\frac{1 - H_{ii}}{1 - H_{ii}} - \frac{\hat{y}_i}{1 - H_{ii}} + \frac{H_{ii}y_i}{1 - H_{ii}}\right)^2 \\ &= \sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - H_{ii}}\right)^2. \end{aligned} \tag{19}$$

With the updated formula, we can estimate the naive algorithmic complexity of computing the LOOCV by noting that we need only compute $\hat{Y_{OLS}}$ once this time. We get the H term as a part of the $\hat{Y}$ computation. Therefore, the algorithmic

complexity of the LOOCV using the new formula is equal to that of the OLS I derived above and is given by $\mathcal{O}(d^3 + dN^2 + Nd^2)$.

Question 4

Note: Both both subquestions 1 and 2, I assume that the too small or too large λs bias the model to a too complex or too simple model relative to the optimum model complexity, respectively.

4.1: Too Small a λ.

4.1.a. For both LASSO and Ridge Regression (RR), the error on the training set would decrease as the penalty term for both regression techniques would be effectively negligible. With small λ, the regularization penalty is also small causing both regression techniques to tend towards least squares regression (LSR) and allow for more complex models which overfit the training data and hence leading to smaller training set error. For example, too small of a λ could push both LASSO and RR to favor a high-order polynomial model when the underlying data is linear.

4.1.b. For both LASSO and RR, too small a λ leads to overfitting on the training set. With a model overfit on the training set, it will do a poor job of generalizing to new data and hence will poorly fit the testing set leading to larger testing error.

4.1.c. For both LASSO and RR, too small a λ yields too small of a complexity penalty causing both algorithms to tend towards the LSR solution. In this case, $\hat{\omega}$ could get large via overfitting. RR, however, will likely predict larger $\hat{\omega}$ than LASSO as RR's l_2 norm penalty primarily seeks to control the magnitude of the weight vector, hence biasing towards larger values in this case as the penalty decreases, while LASSO seeks a sparse solution.

4.1.d. With LASSO, too small of a λ would yield more non-zero parameters since a smaller regularization penalty will prevent many of the elements of $\hat{\omega}$ from being set to 0 under LASSO's sharp l_1 norm penalty which tries to make $\hat{\omega}$ sparse. For RR, too small of a λ would have little effect on the number of non-zero parameters as RR's regularization penalty deals with constraining the magnitude of $\hat{\omega}$ as opposed to forcing it to be sparse.

4.2: Too Large a λ.

4.2.a. For both regression techniques, too large of a λ will yield larger training set error as the regularization penalty will select for simpler models that could poorly fit the data. For example, too large of a λ could push both LASSO and RR to favor a linear model when the underlying data is quadratic.

4.2.b. For both LASSO and RR, too large of a λ will yield larger testing set error since the models again will be biased towards lower-than-optimal model complexity and hence will likely yield a poor fit to future data similar to why the error on the training set is also larger in this case.

4.2.c. For both LASSO and RR, too large of a λ will yield smaller $\hat{\omega}$ as the larger regularization penalty restricts the weight vector. RR, however, will likely predict smaller $\hat{\omega}$ than LASSO as RR's l_2 norm penalty primarily seeks to control the magnitude of the weight vector, hence biasing towards smaller values in this case, while LASSO seeks a sparse solution.

4.2.d. For LASSO, a larger λ will yield a sparser solution via its l_1 norm regularization penalty and hence will have fewer nonzero elements in $\hat{\omega}$ than RR. In the large λ limit, RR will also yield fewer nonzero elements in $\hat{\omega}$ only because its large regularization penalty makes coefficients in its weight vector small and hence potentially pushing some towards 0.

Note: For this question, I collaborated with Matt Wilde.

Question 5

5.1. For the following derivation, I will make use of the following conditional probability identities

$$E_{X,Y}[arg] = E_X[E_{Y|X}[arg]] \tag{20}$$

for some arbitrary argument arg and

$$E_{Y|X}[Y] = E[Y|X] \tag{21}$$

which is the Law of Iterated Expectations.

Now, I first write $L(f)$ and add a form of 0 and expand the terms as follows

(22)
$$\begin{aligned} L(f) &= E_{X,Y}[(Y - f(X))^2] \\ &= E_{X,Y}[((Y - E[Y|X]) + (E[Y|X] - f(X)))^2] \\ &= E_{X,Y}[(Y - E[Y|X])^2 + (E[Y|X] - f(X))^2 - 2(Y - E[Y|X])(E[Y|X] - f(X))]. \end{aligned}$$

To simplify Eqn. 22, I will split it up into 3 terms. Note that the first term, $E_{X,Y}[(Y - E[Y|X])^2]$ is the second term in the answer, so I will leave that be.

The second term can be simplified as follows

$$\begin{aligned} & E_{X,Y}[(E[Y|X] - f(X))^2] \\ &= E_X[E_{Y|X}[(E[Y|X] - f(X))^2]] \\ &= E_X[(E[Y|X] - f(X))^2] \end{aligned} \tag{23}$$

via Eqn. 20 and since both $E[Y|X]$ and $f(X)$ are constant with respect to (wrt) integration over Y. Now, we have

(24)
$$E_{X,Y}[(Y-E[Y|X])^2]+E_X[(E[Y|X]-f(X))^2]-E_{X,Y}[2(Y-E[Y|X])(E[Y|X]-f(X))]$$

and need only shown that the third term in the above relation goes to 0 to complete the proof which I do below (Note: I use f instead of f(X) for brevity)

$$\begin{aligned}
&E_{X,Y}[(Y - E[Y|X])(E[Y|X] - f(X))] \\
&= E_{X,Y}[YE[Y|X] - Yf - E[Y|X]^2 + fE[Y|X]] \\
&= E_{X,Y}[YE[Y|X] - E[Y|X]^2 + fE[Y|X] - Yf] \qquad (25) \\
&= E_X[E_{Y|X}[YE[Y|X]] - E_{Y|X}[E[Y|X]^2] + E_{Y|X}[fE[Y|X]] - E_{Y|X}[Yf]] \\
&= E_X[E_{Y|X}[Y]E[Y|X] - E[Y|X]^2 + fE[Y|X] - E_{Y|X}[Y]f]
\end{aligned}$$

where I used the fact that both f and $E[Y|X]$ do not depend on Y and hence $E_{Y|X}[E[Y|X]] = E[Y|X]$ and $E_{Y|X}[f] = f$. Now using Eqn. 21,

$$\begin{aligned}
&E_X[E_{Y|X}[Y]E[Y|X] - E[Y|X]^2 + fE[Y|X] - E_{Y|X}[Y]f] \\
&= E_X[E[Y|X]^2 - E[Y|X]^2 + fE[Y|X] - E[Y|X]f] \qquad (26) \\
&= 0
\end{aligned}$$

and completing the proof.

5.2. Given the expression derived above for the square loss for a function f, the Bayes optimal regressor $f_{Bayes} = E[Y|X]$. This expression for f_{Bayes} is the optimal regressor because the best function one can learn from the data is the expected value of the label given the data, $E[Y|X]$. With the expression for f_{Bayes}, I get

$$\begin{aligned}
L(f_{Bayes}) &= E_X[E[Y|X] - f_{Bayes}^2] + E_{X,Y}[(Y - E[Y|X])^2] \\
&= E_X[(E[Y|X] - E[Y|X])^2] + E_{X,Y}[(Y - E[Y|X])^2] \qquad (27) \\
&= E_{X,Y}[(Y - E[Y|X])^2].
\end{aligned}$$

This expression for $L(f_{Bayes})$ represents the inherent uncertainty in the data which sets the error limit a model cannot improve upon.

Question 6

Note: The following scripts (attached) include all code used to solve this question: `hw1_6.py, mnist_utils.py, regression_utils.py, ridge_utils.py, validation.py`

Here, the I build a linear classifier by minimizing the following Ridge Regression Loss:

$$\min_\omega = \frac{1}{N}\Sigma_{i=1}^{N}(y_i - \omega \cdot x_i)^2 + \lambda||\omega||^2. \qquad (28)$$

In general, the RR regression weight vector can be solved for analytically via the following

$$\hat{\omega_{ridge}} = (\lambda I_D + X^T X)^{-1} X^T y \tag{29}$$

however the matrix inversion required is not only computationally expensive but also can be numerically unstable. Instead, I minimize Equation 27 by following the formulae in the textbook (see Murphy Section 7.5.2) which involves centering the data and augmenting X with the Cholesky decomposition of the precision matrix and y with a vector of 0s. This yields the MAP estimate

$$\hat{\omega_{ridge}} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \tilde{y}. \tag{30}$$

6.1. The hyperparameters of my model, the regularization parameter λ and the threshold parameter c, were simultaneously fit for along a regularization grid. Since my training set was sufficiently large with $N = 60,000$ samples, I randomly partitioned it into a validation set with $N_{val} = 6,000$ and a training set with $N_{train} = 54,000$. I iterated over a two-dimensional grid of thresholds $c \in [-0.2, 1]$ in 20 linear bins and $\lambda \in [10^{-5}, 10^{10}]$ in 20 logarithmic bins. For each (c, λ) pair, I fit the ridge regression on the training set then tested it on the validation data. I used the validation predictions to compute the 0-1 loss and stored it for each grid point. I found the optimal $c = 0.368$ and $\lambda = 4.3 \times 10^7$ from the minimum value of the 0-1 loss grid. With these optimal hyperparameters, I then fit the ridge regression model on the entire $N = 60,000$ sample training set.

For estimating the losses, I compute the 0-1 loss as

$$\sum_{i=1}^{N} I(y_i \neq \hat{y}_i) \tag{31}$$

where y_i is the label for if the digit is a 2 (1) or not (0) and $\hat{y}_i$ is the predicted label of whether the digit is a 2 (1) or not (0). The predicted label is set to 1 if $X_i w > c$ and 0 otherwise for a threshold value c I estimated above. I compute the square loss as

$$\sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{32}$$

where y_i is the label for if the digit is a 2 (1) or not (0) and $\hat{y}_i$ is the predicted $X_i \omega + w_0$ floating point value produced by the linear model.

After training my model, I find a 0/1 loss of 1892 and a square loss of 2290.8 on the training set. Note that these losses do not seem terribly high for my classifier given that there are of order 6,000 2's in the MNIST training set. However, even a naive classifier that says everything is not a 2 would be correct roughly 90% of the time, therefore I expect my model that does a simple linear regression to do slightly better.

6.2. On the testing set, I evaluated my model from fitting the training set and apply it to the testing input images to find a 0/1 loss of 316 and a square loss of 383.4. Note that it appears that my error on the testing set is actually less than that of the training set! This, however, is not the case since the training set contains 60,000 samples while the testing set contains 10,000. If the size of the set over which the model is evaluated is taken into account, my model performs worse on the testing set and hence the testing error is larger as is expected for classifying with a linear regression model.

6.3. Linear regression is a poor idea for classification since the slope of the predicted line decides which input vector is labeled a 0 and which is labeled a 1. Instead of ascribing a probability to each estimate, it instead checks to see if it's above or below a given threshold. This model does not successfully generalize to new data since the slope of the classifying line is dependent on the training data and new points could dramatically shift the slope of that line, changing which class a given point belongs to. Instead when using a linear classifier, one should use logistic regression as it has the desirable outcome of predicting the label via a probability. That is, if the logistic regressor predicts a probability $p > 0.5$, the sample is assigned a label of 1.

6.4. Recalling what my answer for the Bayes optimal predictor for the square loss, the Bayes optimal predictor using square loss for classification is $E[y|x, D]$. Since I used a linear regressor in this question, I can replace y to get $X^T E[\omega|x, D]$ where X can be removed from the expectation value since it is constant over the integral.

Note: For this question, I collaborated with Matt Wilde.

Question 7

7.1.

7.1.1. If we define

$$\hat{y_i} = X_i\omega + w_0, \tag{33}$$

we can rearrange that to get

$$X_i\omega = \hat{y_i} - w_0. \tag{34}$$

Combining that expression with the update rule for w_0 from Algorithm 1, we find

$$w_0^{(t+1)} = \sum_{i=1}^{N}(y_i - \hat{y_i}^{(t)})/N + w_0^{(t)} \tag{35}$$

by substituting out the $X_i\omega$ term in favor of the expression with $\hat{y_i} - w_0$ since $X_i\omega = \sum_j \omega_j X_{ij}$.

Similarly for the new c_k update rule, we can insert

$$w_0 = \hat{y_i} - X_i\omega. \tag{36}$$

into the following expression for c_k

$$c_k = 2\sum_{i=1}^{N} X_{ik}(y_i - (w_0 + \sum_{j\neq k} \omega_j X_{ij})) \tag{37}$$

where

$$-\sum_{j} X_{ij}\omega_j + \sum_{j\neq k} \omega_j X_{ij} = -X_{ik}\omega_k \tag{38}$$

allows us to simplify the expression into

$$c_k = 2\sum_{i=1}^{N} X_{ik}(y_i - \hat{y}_i + X_{ik}\omega_k). \tag{39}$$

This final expression no longer includes the summation over j.

7.1.2. The dot product of a sparse matrix and a vector takes time proportional to the number of nonzero entries in said sparse matrix. Therefore if $||X||_0$ is the number of nonzero entries in X, the time complexity to compute $\hat{y} = X\omega + w_0$ is $\mathcal{O}(||X||_0 + N)$.

7.1.3. If $\hat{y}$ is not already computed, it takes $\mathcal{O}(N||X||_0)$ to compute $\hat{y}$ then $\mathcal{O}(N)$ to loop over all entries of y and $\hat{y}$ to compute w_0 yielding a total complexity $\mathcal{O}(N + N||X||_0)$ since we still need only to compute $\hat{y}$ once. If $\hat{y}$ has already been computed, the time complexity is only $\mathcal{O}(N)$.

7.1.4. Define z_j to be the number of nonzero elements in the $j-th$ column of X. If $\hat{y}$ is already computed, the time complexity to compute $\hat{\omega}$ is $\mathcal{O}(z_j)$ as one needs to loop over only z_j elements while computing both a_k and c_k while computing w_k since the dot product takes time proportional to the number of nonzero entries, z_j.

7.1.5. I can rearrange the expression for $\hat{y}_i^{(t)}$ to get the following

$$X_i\omega^{(t)} = \hat{y}_i^{(t)} - w_0^{(t)}. \tag{40}$$

I insert the above expression for $X_i\omega^{(t)}$ into the equation for $\hat{y}_i^{(t+1)}$ to find the corresponding update rule for the new prediction

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + w_0^{(t+1)} - w_0^{(t)} \tag{41}$$

where $w_0^{(t+1)}$ was found via the update rule derived in Question 7.1.1 and $w_0^{(t)}$ was the value of w_0 at the start of the current iteration.

If $\hat{y}^{(t)}$ is already computed, the complexity to calculate $\hat{y}^{(t+1)}$ is simply $\mathcal{O}(N)$ as one must loop over the N elements of $\hat{y}^{(t+1)}$ for the computation.

7.1.6. First, define

$$\hat{y}_i^{(t+1)} = \sum_{j \neq k} \omega_j^{(t)} X_{ij} + \omega_k^{(t+1)} X_{ik} + w_0^{(t)} \tag{42}$$

to be the new prediction after updating. Again, I note that

$$X_i \omega^{(t)} = \hat{y}_i^{(t)} - w_0^{(t)} \tag{43}$$

which we can insert that into the above new prediction since $\sum_j \omega_j^{(t)} X_{ij} = X_i \omega^{(t)}$ to get

$$\hat{y}_i^{(t+1)} = \hat{y_i}^{(t)} - w_0^{(t)} - \omega_k^{(t)} X_{ik} + \omega_k^{(t+1)} X_{ik} + w_0^{(t)} \tag{44}$$

to be the new prediction after updating. This expression reduces to the new update rule for $\hat{y}_i^{(t+1)}$ in terms of $\hat{y}_i^{(t)}$

$$\hat{y}_i^{(t+1)} = \hat{y_i}^{(t)} + X_{ik}(\omega_k^{(t+1)} - \omega_k^{(t)}). \tag{45}$$

If $\hat{y}_i^{(t)}$ is already computed, the time complexity required to compute $\hat{y}_i^{(t+1)}$ is $\mathcal{O}(N)$ as one must loop over all N elements in $\hat{y}_i^{(t+1)}$.

7.1.7. Combining all the rules derived in 7.1.1 - 7.1.6, we have the Efficient Coordinate Descent Algorithm for LASSO regression. To estimate the per iteration complexity of the algorithm, we note that within the while loop there are two regimes to consider: within the for loop over all d features and outside said loop. Outside the loop, the first calculation of $\hat{y}$ is $\mathcal{O}(||X||_0 + N)$ as shown in 7.1.2. The update rules for w_0 and the second update of $\hat{y}$ both take $\mathcal{O}(N)$ yielding a net $\mathcal{O}(||X||_0 + N)$ for the operations outside of the for loop. Within the for loop over all d features, the complexity for updating $\hat{y}$ is $\mathcal{O}(N)$ as this operation involves looping over N elements to perform simple arithmetic operations. The complexity to update $\hat{\omega_k}$ is $\mathcal{O}(z_j)$. Given that a maximum number of non-zero elements for a given column X_i is N, the asymptotic complexity to update $\hat{\omega}$ is $\mathcal{O}(N)$. Therefore, the operations within the loop have a complexity of $\mathcal{O}(2N)$ which asymptotes to $\mathcal{O}(N)$. These loop operations occur d times over the course of the loop yielding a net for loop complexity of $\mathcal{O}(Nd)$. This complexity then combines with the outside of the loop complexity to yield a per iteration complexity of $\mathcal{O}(Nd + ||X||_0 + N)$. When we only consider the behavior for large dN or large $||X||_0$, this gives a per iteration complexity of the algorithm of $\mathcal{O}(Nd + ||X||_0)$.

7.2. My `python` implementations of both Algorithm 1 and Algorithm 2 are in the attached file `lasso_utils.py` and are named `fit_lasso and fit_lasso_fast`, respectively. Each function includes the offset term w_0 that is not regularized, have the option to take initial conditions for both ω and w_0 and can handle dense or sparse X using the `scipy.sparse` package.

7.3.

7.3.1. In the file `regression_utils.py`, the function `generate_norm_data` generates the synthetic data set according to the procedure outlines in Question 7.3. To optimize over λ, I ran a regularization path starting at $\lambda_{max} = 5000$. This value of λ_{max} was a factor of 3 or so larger than $\lambda_{max} = 2||X^T(y - \bar{y})||_\infty$ in order to demonstrate that too large of a regularization penalty yields a null weight vector. For each iteration, I decreased λ by a factor of 2 and initialized my LASSO algorithm with the results of the previous iteration. For the first iteration, I assumed that ω was null. For each iteration, I compared the sparsity pattern of my solution $\hat{w}$ to the true value I generated, ω_{true} by recording the precision and recall. The results of this regularization path for data generated with $\sigma = 1$ and for data with $\sigma = 10$ is shown in Fig. 1.

For data generated with $\sigma = 1$ (solid curves in Fig. 1), a $\lambda \sim 10^3$ gives perfect precision and recall. Larger λs yield a larger regularization penalty which enforces a sparser solution, causing correct non-zero values to go to 0 resulting in worse recall and precision. Smaller λs have little effect on recall. The LASSO algorithm tends towards ordinary least squares regression for smaller λ and hence is still able to correctly determine the true non-zero values giving the observed perfect recall. In this regime, however, the reduce penalty term also allows terms in $\hat{\omega}$ that should be 0 to become non-zero due to the regression algorithm overfitting the inherent noise in the data. The additional non-zero terms arising from overfitting the noise drives the precision down as λ decreases as the number of correct non-zeros in $\hat{\omega}$ stays the same while the total number of non-zero terms grows.

7.3.2. I reran the same regularization analysis for the data generated with $\sigma = 10$ (dashed curves in Fig. 1) as I did for the $\sigma = 1$ data over the same λ regularization path allowing for a direct comparison of the algorithm's performance for a given λ. In general, I found that the recall was relatively unchanged between the two fits for all λ. This makes sense since too large a λ drives all $\hat{\omega}$ to 0 while smaller λ allows for the linear regression to recover the correct non-zero values of ω_{true}. The precision for the fits on the data generated with $\sigma = 10$, however, was much worse than the fits over the $\sigma = 1$ data for almost all values of λ. The enhanced noise in that synthetic dataset makes overfitting an issue and hence underscores the importance of regularization. At larger λ, the precision is reasonable as the regularization penalty prevents the LASSO algorithm from overfitting and producing more incorrect non-zero terms in $\hat{\omega}$. Smaller λ, however, yield a weaker regularization penalty and hence the algorithm overfits, producing many incorrect non-zero terms in $\hat{\omega}$ in giving poor precision.

In order to achieve the optimal precision and recall for this noisy data, one should run a regularization path as before while monitoring recall. Once recall has increased to 1, that λ_{best} should be returned as the optimal regularization coefficient. For $\lambda < \lambda_{best}$, the regularization penalty would be too low allowing noise to produce spurious features yielding incorrect non-zero terms in $\hat{\omega}$. For the

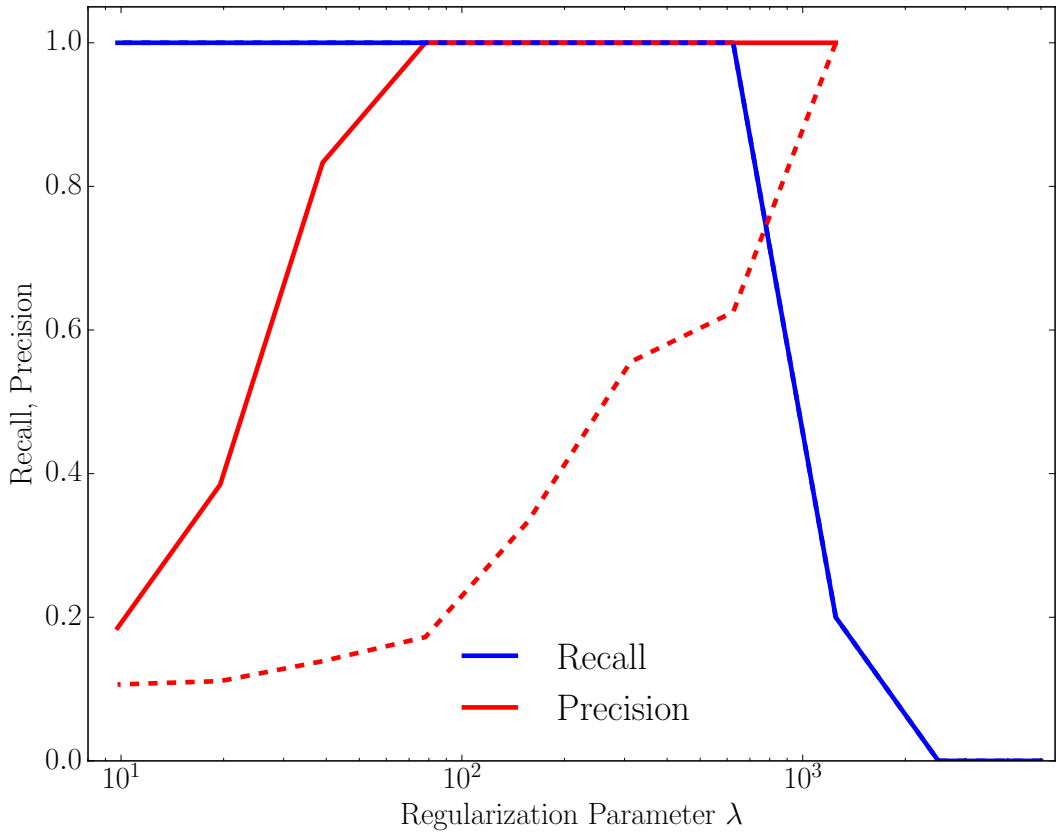

FIGURE 1. Precision (red curves) and recall (blue curves) as a function of the regularization coefficient λ over a regularization path. Solid curves represent data generated with $\sigma = 1$ while dashed curves represent noisy data generated assuming $\sigma = 10$. For a given λ, the fitting the noisy $\sigma = 10$ data using the LASSO algorithm yields less precise answers but comparable recall relative to the $\sigma = 1$ data

general case where ω_{true} is not known, one should bias towards larger λ for noisy data to prevent overfitting.

Note: On this question, I collaborated with Matt Wilde, Serena Liu, and Janet Matsen.