CSE 546: Homework 1

Due on October 14, 2016

Brian de Silva

Collaborators: Kathleen Champion, Scott Moe, Kelsey Maass.

Problem 1

Let X_1 and X_2 be independent, continuous random variables uniformly distributed on [0,1]. Let $X = \max(X_1, X_2)$. Compute

1. $E(X)$.
2. $\mathrm{Var}(X)$.
3. $\mathrm{Cov}(X, X_1)$.

Solution:

1. Since X_1 and X_2 are independent and are uniformly distributed on [0,1], their joint probability density function is the identity (on $[0,1] \times [0,1]$). Hence

$$E(x) = \int_0^1 \int_0^1 \max(x_1, x_2) dx_1 dx_2 \tag{1}$$
$$= \int_0^1 \int_{x_2}^1 x_1 dx_1 dx_2 + \int_0^1 \int_{x_1}^1 x_2 dx_2 dx_1 \tag{2}$$
$$= 2 \int_0^1 \int_{x_2}^1 x_1 dx_1 dx_2 \tag{3}$$

since the two integrals are equivalent. Completing the calculation yields

$$\begin{aligned} E(x) &= 2 \int_0^1 \int_{x_2}^1 x_1 dx_1 dx_2 \\ &= \int_0^1 x_1^2 \Big|_{x_2}^1 dx_2 \\ &= \int_0^1 \left(1 - x_2^2\right) dx_2 \\ &= \left(x - x^3/3\right)\Big|_0^1 \\ &= \frac{2}{3}. \end{aligned}$$

Problem 1 continued on next page...

2. We use the identity $\mathrm{Var}(X) = E(X^2) - E(X)^2$. We have that

$$\begin{aligned}
E(X^2) &= \int_0^1 \int_0^1 \max(x_1, x_2)^2 dx_1 dx_2 \\
&= \int_0^1 \int_{x_2}^1 x_1^2 dx_1 dx_2 + \int_0^1 \int_{x_1}^1 x_2^2 dx_2 dx_1 \\
&= 2\int_0^1 \int_{x_2}^1 x_1^2 dx_1 dx_2 \\
&= \frac{2}{3}\int_0^1 x_1^3 \Big|_{x_2}^1 dx_2 \\
&= \frac{2}{3}\int_0^1 \left(1 - x_2^3\right) dx_2 \\
&= \frac{2}{3}\left(x - x^4/4\right)\Big|_0^1 \\
&= \frac{1}{2}.
\end{aligned}$$

   Using the identity above, we obtain

$$\mathrm{Var}(X) = E(X^2) - E(X)^2 = \frac{1}{2} - \left(\frac{2}{3}\right)^2 = \frac{1}{18}.$$

3. For this problem we use the identity $\mathrm{Cov}(X, X_1) = E(XX_1) - E(X)E(X_1)$ so we need to compute $E(X_1)$ and $E(XX_1)$.

$$E(X_1) = \int_0^1 x_1 dx = \frac{1}{2}x^2\Big|_0^1 = \frac{1}{2}.$$

   To find $E(XX_1)$, we first need the joint probability density function for the two random variables, f_{X,X_1}. To get this we use the identity $f_{X,X_1}(x,y) = f_{X|X_1}(x|y)f_{X_1}(y)$. The PDF for X_1 is simply the identity on $[0,1]$, so we must now find $f_{X|X_1}(x|y)$. We have

$$\begin{aligned}
f_{X|X_1}(x|y) &= \frac{d}{dx}F_{X|X_1}(x|y) \\
&= \frac{d}{dx}P(X \le x | X_1 \le y) \\
&= \frac{d}{dx}P(\max(X_1, y) \le x | X_1 \le y) \\
&= \frac{d}{dx}P(X_1 \le x, y \le x | X_1 \le y) \\
&= \frac{d}{dx}P(X_1 \le x | X_1 \le y)P(y \le x | X_1 \le y) \\
&= \frac{d}{dx}P(X_1 \le x | X_1 \le y)P(y \le x) \\
&= \frac{d}{dx}P(X_1 \le x | X_1 \le y)x \\
&= \frac{d}{dx}\left(x \begin{cases} x/y & x \le y \\ 1 & x > y \end{cases}\right) \\
&= \frac{d}{dx}\begin{cases} x^2/y & x \le y \\ x & x > y \end{cases} \\
&= \begin{cases} 2x/y & x < y \\ 1 & x > y \end{cases}.
\end{aligned}$$

Hence

$$\begin{aligned}
E(XX_1) &= \int_0^1 \int_0^1 xy f_{X|X_1}(x|y)dxdy \\
&= \int_0^1 \int_0^y xy\frac{2x}{y}dxdy + \int_0^1 \int_y^1 xydxdy \\
&= \int_0^1 \int_0^y 2x^2 dxdy + \frac{1}{2}\int_0^1 yx^2\Big|_y^1 dy \\
&= \frac{2}{3}\int_0^1 y^3 dy + \frac{1}{2}\int_0^1 \left(y - y^3\right) dy \\
&= \frac{1}{6}\, y^4\big|_0^1 + \frac{1}{2}\left(\frac{1}{2}y^2 - \frac{1}{4}y^4\right)\Big|_0^1 \\
&= \frac{7}{24}.
\end{aligned}$$

Finally, we conclude that

$$\mathrm{Cov}(XX_1) = E(X_1X) - E(X)E(X_1) = \frac{7}{24} - \frac{1}{2}\cdot\frac{2}{3} = -\frac{1}{24}$$

Problem 2

Imagine we have a gumball machine which dispenses a random number of gumballs each time you insert a quarter. Assume that the number of gumballs dispensed is Poisson distributed (i.i.d) with parameter λ. Curious and sugar-deprived, you insert 8 quarters one at a time and record the number of gumballs dispensed on each trial:

Trial	1	2	3	4	5	6	7	8
Gumballs	4	1	3	5	5	1	3	8

Let $G = (G_1, G_2, \ldots, G_n)$ be a random vector where G_i is the number of gumballs dispensed on trial i:

1. Give the log-likelihood function of G given λ.
2. Compute the MLE for λ in the general case.
3. Compute the MLE for λ using the observed G.

Solution:

1. Since each trial is independent, the probability of a particular G is the product of the probabilities of each of its entries:
$$P(G|\lambda) = \prod_{i=1}^{n} \frac{\lambda^{G_i}}{G_i!}e^{-\lambda}.$$

Taking the natural log of this and using the properties of logarithms we obtain

$$\begin{aligned}
\ln P(G|\lambda) &= \ln\left(\prod_{i=1}^{n} \frac{\lambda^{G_i}}{G_i!} e^{-\lambda}\right) \\
&= \ln\left(e^{-\lambda} \prod_{i=1}^{n} \frac{\lambda^{G_i}}{G_i!}\right) \\
&= \ln(e^{-n\lambda}) + \ln\left(\prod_{i=1}^{n} \frac{\lambda^{G_i}}{G_i!}\right) \\
&= -n\lambda + \sum_{i=1}^{n} \ln\left(\frac{\lambda^{G_i}}{G_i!}\right) \\
&= -n\lambda + \sum_{i=1}^{n} (G_i \ln(\lambda) - \ln(G_i!)).
\end{aligned}$$

This is the log-likelihood of G.

2. To find the MLE we take the derivative of the log-likelihood with respect to λ and set it equal to zero to find the max of the likelihood (which is the same as the max of the log-likelihood since ln is an increasing function).

$$\begin{aligned}
\frac{d}{d\lambda}\left(\ln(P(G|\lambda))\right) &= \frac{d}{d\lambda}\left(-n\lambda + \sum_{i=1}^{n} (G_i \ln(\lambda) - \ln(G_i!))\right) \\
&= -1 + \frac{1}{\lambda} \sum_{i=1}^{n} G_i.
\end{aligned}$$

Setting this to zero and solving for λ, we find that the MLE for λ is

$$\lambda_{MLE} = \frac{1}{n} \sum_{i=1}^{n} G_i.$$

3. Plugging in the particular case given in the table above, we have

$$\lambda_{MLE} = \frac{4+1+3+5+5+1+3+8}{8} = \frac{15}{4}.$$

Problem 3

Assume we have n training examples, $(X_1, y_1), (X_2, y_2), /dots, (X_n, y_n)$, where $X_i \in \mathbb{R}^d$. In matrix form we can write

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots y_n \end{bmatrix}, \; X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{bmatrix}, \; w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}, \text{ and } \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}.$$

Assume ϵ_i is normally distributed with variance σ^2. The maximum likelihood estimate of the model parameters w is given by the *Normal equations*:

$$\hat{w} = (X^T X)^{-1} X^T Y.$$

Problem 3 continued on next page...

Define $\hat{Y}$ to be the vector of predictions using $\hat{w}$ if we were to plug in the original training set X:

$$\begin{aligned}\hat{Y} &= X\hat{w} \\ &= X(X^TX)^{-1}X^TY \\ &= HY,\end{aligned}$$

where we define $H = X(X^TX)^{-1}X^T$ (the *Hat Matrix*). $\hat{w}$ minimizes the sum of squared errors:

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

The Leave-One-Out Cross Validation score is defined to be:

$$\text{LOOCV} = \sum_{i=1}^{n}\left(y_i - \hat{y}_i^{(-i)}\right)^2,$$

where $\hat{Y}^{(-i)}$ is the estimator of Y after removing the i-th observation (i.e. it minimizes $\sum_{j\neq i}\left(y_i - \hat{y}_i^{(-i)}\right)^2$).

1. What is the time complexity of computing the LOOCV score naively?
2. Write $\hat{y}_i$ in terms of H and Y.
3. Show that $\hat{Y}^{(-i)}$ is also the estimator which minimizes SSE for Z where
$$Z_j = \begin{cases} y_j, & j \neq i \\ \hat{y}_i^{(-i)}, & j = i \end{cases}$$
4. Write $y_i^{(-i)}$ in terms of H and Z.
5. Show that $\hat{y}_i - y_i^{(-i)} = H_{ii}y_i - H_{ii}y_i^{(-i)}$.
6. Show that
$$\text{LOOCV} = \sum_{i=1}^{n}\left(\frac{y_i - y_i^{(-i)}}{1 - H_{ii}}\right)^2.$$
What is the algorithmic complexity of computing the LOOCV score using this formula?

Solution:

1. The matrix-matrix product and matrix inversion involved in the computation of H cost many more operations than the matrix-vector products and vector-vector operations. The latter are at most $\mathcal{O}(Nd)$. Note that at any time we may choose to do a matrix-vector product rather than a matrix-matrix product, we choose the matrix-vector product. At each iteration of the naive LOOCV algorithm we must create a modified version of X, call it X_i, with the i-th row removed. This matrix is $(N-1)\times d$. Since this matrix changes at each iteration, we must recompute a modified H each time (otherwise we could save our result and perhaps save some time). $X_i^TX_i$ is a $d\times d$ matrix. Each of its d^2 entries is a dot product between two length $N-1$ vectors, so the complexity of its computation is $\mathcal{O}(Nd^2)$. Since it is $d\times d$, its inversion costs $\mathcal{O}(d^3)$ operations. If d is much less than N, then the cost of these two computations is $\mathcal{O}(Nd^2)$. But if $d\sim N$, then the cost is $\mathcal{O}(Nd^2+d^3)$, which is really still $\mathcal{O}(Nd^2)$. The remaining operations are much cheaper and do not factor into the complexity as N and d become large.

   Since we perform N iterations of the above steps, the overall complexity is either $\mathcal{O}(N^2d^2)$, with the constant differing depending on how close to N d is.

2. $\hat{y}_i$ is the i-th component of $\hat{Y} = HY$, so $\hat{y}_i = (HY)_i = H_i Y$, where H_i is the i-th row of H.

3. Recall that $\hat{Y}^{(-i)}$ minimizes
$$\sum_{j \neq i} \left(y_j - \hat{y}_j^{(-i)} \right)^2.$$
The SSE for Z, given an estimator $\hat{W}$ is
$$\begin{aligned} \|Z - \hat{W}\|_2^2 &= \sum_j (z_j - \hat{w}_j)^2 \\ &= \sum_{j \neq i} (y_j - \hat{w}_j)^2 + (y_j - \hat{w}_j)^2. \end{aligned}$$
If we choose $\hat{W}$ to be $\hat{Y}^{(-i)}$ then this expression reduces to
$$\sum_{j \neq i} (y_j - \hat{y}_j)^2.$$
Since $\hat{Y}^{(-i)}$ minimizes this quantity and the term that dropped out was nonnegative, it follows that it minimizes the SSE for Z.

4. $\hat{y}_i^{(-i)}$ is the i-th component of $\hat{Y}^{(-i)}$, which minimizes the SSE for Z, and therefore satisfies $\hat{Y}^{(-i)} = HZ$. Hence $\hat{y}_i^{(-i)} = (HZ)_i = H_i Z$, where H_i is the i-th row of H.

5. By the previous parts, we have
$$\hat{y}_i - \hat{y}_i^{(i)} = (HY)_i - (HZ)_i = [H(Y - Z)]_i = H_{ii}(y_i - \hat{y}_i^{(i)}),$$
since $Y - Z = (0, 0, \ldots, 0, y_i - \hat{y}_i^{(i)}, 0, \ldots, 0)^T$.

6. From the previous result, we deduce
$$\begin{aligned} \hat{y}_i - \hat{y}_i^{(i)} &= H_{ii}(y_i - \hat{y}_i^{(i)} \\ \implies \hat{y}_i^{(-i)}(H_{ii} - 1) &= H_i i y_i - \hat{y}_i \\ \implies \hat{y}_i^{(-i)} &= \frac{H_{ii} y_i - \hat{y}_i}{H_{ii} - 1}. \end{aligned}$$
Substituting this into the LOOCV score, we obtain
$$\begin{aligned} \sum_{i=1}^n \left(y_i - \hat{y}_i^{(-i)} \right)^2 &= \left(y_i - \frac{H_{ii} y_i - \hat{y}_i}{H_{ii} - 1} \right)^2 \\ &= \sum_{i=1}^n \left(\frac{-y_i \hat{+} y_i}{H_{ii} - 1} \right)^2 \\ &= \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{1 - H_{ii}} \right)^2. \end{aligned}$$
In order to compute the LOOCV score with this formula, we require $\hat{Y}$ and H (though finding $\hat{Y}$ essentially necessitates that we compute H). From our earlier analysis, we saw that computing H required $\mathcal{O}(Nd^2)$. Since we no longer need to solve N of these problems, the cost is simply $\mathcal{O}(Nd^2)$ (evaluation of the simplified formula for the LOOCV score is cheap compared to the computation of $\hat{Y}$ and H).

Problem 4

The loss function to be optimized under ridge regression is

$$E_R = \sum_{i=1}^{n} \left(y_i - \left(\hat{w}_0 + x^{(j)} \hat{w} \right) \right)^2 + \lambda \|\hat{w}\|_2^2.$$

The loss function to be optimized under LASSO regression is

$$E_L = \sum_{i=1}^{n} \left(y_i - \left(\hat{w}_0 + x^{(j)} \hat{w} \right) \right)^2 + \lambda \|\hat{w}\|_1.$$

1. Discuss briefly how chosing too small a λ affects the magnitude of the following quantities. Please describe the effects for both ridge and LASSO, or state why the effects will be the same.
   (a) The error on the training set.
   (b) The error on the testing set.
   (c) The elements of $\hat{w}$.
   (d) The number of nonzero elements of $\hat{w}$.
2. Now discuss briefly how choosing too large a λ affects the magnitude of the same quantities in the previous question. Again describe the effects for both ridge and LASSO, or state why the effects will be the same.

Solution:

1. (a) In both cases choosing too small a λ causes the error on the training set to decrease. You are allowing the least-squares coefficients to be almost as large as they want and you are likely overfitting the training data if λ is too small.
   (b) In both cases we expect the error on the testing set to increase since you are overfitting the data in the training set.
   (c) For very small λ it is almost as if you are not placing any penalty on large elements of $\hat{w}$, so in both cases the entries will likely be large. The entries of $\hat{w}$ may be a bit smaller in the ridge regression case since their magnitudes are squared when training the model.
   (d) The number of nonzero elements in $\hat{w}$ will be large in both instances (and perhaps marginally larger in the ridge regression model) since sparsity is not really being penalized with such a small λ.
2. (a) In both cases the error on the training set will increase. We are over-penalizing large coefficients and not leaving enough flexibility in the model to fit the training data at all. Weighting the added regularity constraint so highly means that most of the effort will go towards minimizing the corresponding 1 and 2 norm terms, rather than the square error, making for a poor fit of the data.
   (b) The error on the testing set will likely increase in both cases as well. The models will be much more influenced by their attempts to create small/sparse weights than they are by the actual training data. This means they will not really fit the data in either the training set or the testing set.
   (c) Both methods will produce weight vectors with small magnitude entries. The magnitudes of these entries are very heavily penalized, so they will all be tiny, if not zero. The entries in the vector produced by ridge regression will be uniformly small, since any element larger than 1 in magnitude

Problem 4 continued on next page...

has that magnitude increased when it is squared in the calculation of E_R. The LASSO weights, on the other hand, will be either 0, or small. But the nonzero entries will not be quite as small as with ridge regression (for the same λ).

(d) Ridge regression will produce a vector $\hat{w}$ with very small entries, but few of them will be nonzero. LASSO will select a weight vector which is very sparse. Most of its entries will be 0 if λ is large enough.

Problem 5

5.1 Bayes Optimal Prediction

Suppose we have a distribution over pairs (X, Y) where Y is a scalar and X lives in an arbitrary set $\mathcal{X}$. The square loss for a function $f : \mathcal{X} \to \mathbb{R}$ can be defined as:

$$L(f) = \mathbb{E}_{X,Y}[(Y - f(X))^2],$$

where the expectation is with respect to the underlying distribution on (X, Y).

1. Show that:
$$L(f) = \mathbb{E}_X[(E[Y|X] - f(X))^2] + \mathbb{E}_{X,Y}[(Y - E[Y|X])^2]$$
where $E[Y|X]$ is the conditional expectation of Y given X.

2. The *Bayes optimal* regressor f_{Bayes} is the function which minimizes $L(f)$ over the class of all functions. Provide a natural expression for f_{Bayes}, and write down $L(f_{Bayes})$.

5.2 Bias-Variance-Noise Error Decomposition

Suppose we are given a training set $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ sampled from this underlying distribution and a class of functions $\mathcal{F} = \{f\}$ and that the estimator we provide, using $\mathcal{T}$ lives in this class, i.e. $\hat{f} \in \mathcal{F}$. The *empirical risk minimizer* (ERM) is defined as

$$\hat{f} = \text{argmin}_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2.$$

The "best in class" function f^* is:

$$f^* = \text{argmin}_{f \in \mathcal{F}} L(f).$$

1. Let $\bar{f}(X) = E_{\mathcal{T}}[\hat{f}(X)]$ where the expectation is taken with respect to the randomness in our training set, i.e. $\bar{f}(\cdot)$ is the expected function with respect to the randomness in our training set. Show that

$$\begin{aligned}\mathbb{E}_{\mathcal{T}}[L(\hat{f})] &= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}[(Y - \hat{f}(X))^2] \\ &= \mathbb{E}_{\mathcal{T}}\mathbb{E}_X[(\bar{f}(X) - \hat{f}(X))^2] + \mathbb{E}_X[(\bar{f}(X) - E[Y|X])^2] + \mathbb{E}_{X,Y}[(Y - E[Y|X])^2].\end{aligned}$$

2. Interpret these terms as the variance, the bias, and the noise, respectively.

3. For the ERM, is it the case that $f^* = \bar{f}$?

4. Suppose the $\mathcal{F}$ is finite. For the ERM, in the limit of a large sample size N, what do you think $\bar{f}$ and $\hat{f}$ tend to? Why?

Solution:

5.1 Bayes Optimal Prediction

1. In manipulating the conditional expectations we use the following identities (along with linearity):

   - $\text{Var}(X) = E[X^2] + E[X]^2$
   - $E[f(Z)|Z] = f(Z)$
   - $E(f(Z)Y|Z) = f(Z)E[Y|Z]$
   - $E[E[X|\mathcal{H}]] = E[X]$

$$\begin{aligned}
L(f) &= \mathbb{E}_{X,Y}[(Y - f(X))^2] \\
&= \mathbb{E}_{X,Y}[Y^2 - 2Yf(X) + f(X)^2|X] \\
&= \mathbb{E}_{X,Y}[Y^2|X] - 2\mathbb{E}_X[f(X)Y|X] + \mathbb{E}_X[f(X)^2|X] \\
&= \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2 \middle| X\right] + E[Y|X]^2 - 2f(X)E[Y|X] + f(X)^2 \\
&= \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right] + \mathbb{E}_X[E[Y|X]^2 - 2f(X)E[Y|X] + f(X)^2|X] \\
&= \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right] + \mathbb{E}_X[(E[Y|X] - f(X))^2|X] \\
&= \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right] + \mathbb{E}_X[(E[Y|X] - f(X))^2].
\end{aligned}$$

2. Given a data set (X, Y), f_{Bayes} minimizes

$$L(f) = \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right] + \mathbb{E}_X[(E[Y|X] - f(X))^2].$$

   Notice that this is equivalent to minimizing

$$\mathbb{E}_X[(E[Y|X] - f(X))^2]$$

   since our choice of f has no effect on the $\mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right]$ term of the square loss. Furthermore, $\mathbb{E}_X[(E[Y|X] - f(X))^2] \geq 0$ because it is the expectation of a nonnegative function. It is clear, then, that $f_{Bayes} = E[Y|X]$ since $EY|X$ is a function of X and $\mathbb{E}_X[(E[Y|X] - E[Y|X])^2] = 0$. For this f_{Bayes}, the square loss is

$$L(f_{Bayes}) = \mathbb{E}_{X,Y}\left[(Y - E[Y|X])^2\right].$$

5.2 Bias-Variance-Noise Error Decomposition

1. We compute

$$\begin{aligned}
\mathbb{E}_{\mathcal{T}} L(\hat{f}) &= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}\left[\left(Y-\hat{f}(X)\right)^2\right] \\
&= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}\left[\left(Y-\hat{f}(X)\right)^2 \middle| X\right] \\
&= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}\left[Y^2 - 2\hat{f}(X) + \hat{f}(X)^2 \middle| X\right] \\
&= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}\left[Y^2 \middle| X\right] - 2\mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}[\hat{f}(X)Y|X] + \mathbb{E}_X\mathbb{E}_T\left[\hat{f}(X)^2 \middle| X\right] \\
&= \mathbb{E}_{\mathcal{T}}\mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2 \middle| X\right] + \mathbb{E}_{\mathcal{T}}E[Y|X]^2 - 2\mathbb{E}_T\hat{f}(X)\mathbb{E}_{X,Y}[Y|X] \\
&\quad + \mathbb{E}_X\mathbb{E}_{\mathcal{T}}\left[\left(\hat{f}(X)-\bar{f}(X)\right)^2 \middle| X\right] + \mathbb{E}_X\mathbb{E}_{\mathcal{T}}[\bar{f}(X)]^2 \\
&= \mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2\right] + E[Y|X]^2 - 2\bar{f}(X)E[Y|X] \\
&\quad + \mathbb{E}_T\mathbb{E}_X\left[\left(\bar{f}(X)-\hat{f}(X)\right)^2 \middle| X\right] + E[\bar{f}(X)]^2 \\
&= \mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2\right] + \left(E[Y|X]-\bar{f}(X)\right)^2 + +\mathbb{E}_T\mathbb{E}_X\left[\left(\bar{f}(X)-\hat{f}(X)\right)^2\right] \\
&= \mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2\right] + \mathbb{E}_X\left[\left(E[Y|X]-\bar{f}(X)\right)^2 \middle| X\right] + \mathbb{E}_T\mathbb{E}_X\left[\left(\bar{f}(X)-\hat{f}(X)\right)^2\right] \\
&= \mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2\right] + \mathbb{E}_X\left[\left(\bar{f}(X)-E[Y|X]\right)^2\right] + \mathbb{E}_T\mathbb{E}_X\left[\left(\bar{f}(X)-\hat{f}(X)\right)^2\right].
\end{aligned}$$

2. The first term,

$$\mathbb{E}_T\mathbb{E}_X\left[\left(\bar{f}(X)-\hat{f}(X)\right)^2\right]$$

tells us how much we can expect the function our model produces to differ from the "average" function that is produced, given the randomness in our training data. This is synonymous with the variance in the models we produce (it is the expectation of the square difference between an object we compute, and its own expectation).

The next term,

$$\mathbb{E}_X\left[\left(\bar{f}(X)-E[Y|X]\right)^2\right]$$

expresses the expected amount our average estimator will differ from the quantity it is trying to approximate–$E[Y|X]$. This is just the bias in our estimator. If it is unbiased, then this expectation should be 0.
The final term,

$$\mathbb{E}_{X,Y}\left[(Y-E[Y|X])^2\right]$$

characterizes the amount by which Y differs from its expectation, given X, on average. This is a property of data itself which gives us a measure of how hard it is to guess Y given some data X. The more noise that is present in the data, the harder this will be and the larger this quantity will be. Therefore we can identify this term with the noise inherent in the data.

3. For the ERM, it is not necessarily true that $f^* = \hat{f}$. To see why, consider the following scenario. Suppose we generate training data by randomly sampling a fourth degree polynomial, only taking two

samples for our training data (so $N = 2$). Furthermore, supppose we take our test space $\mathcal{F}$ to be the set of all polynomials of degree at most four. It is easy to see that f^* is simply the original polynomial we sampled. However, each of our models generated with the ERM would match only the linear part of the true underlying function, with the higher degree terms ($x^2,\ x^3,\ x^4$) being chosen to be zero or arbitrarily. The average ERM function over all training sets (of cardinality two) would not average out to the correct fourth degree polynomial, f^*.

4. If $\mathcal{F}$ is finite, then for any given training data we select our estimator from a finite set of fixed functions, $\{f_1, f_2, \ldots, f_k\}$, using the ERM. Note that one of these functions must be f^*. In the limit of very large sample size, a few things happen. First, $\bar{f}$ tends to f^*. This occurs because the very large (random) sample size means that in each instance, the space is reasonably well sampled, so there is less difference between different sets of training data. Since f^* does the best job of minimizing $L(f)$, as the sample size grows, the likelihood that the ERM causes us to choose f^* approaches 1. This implies that $\bar{f} = f^*$. This also means that the probability that $\hat{f}$ is f^* for any given training set tends toward 1 as the sample size increases. Hence $\bar{f}$ and $\hat{f}$ both tend to f^* in the limit of large sample size N.

Problem 6

Supose $Y \in \{0, 1\}$ and let us consider the classification problem of recognizing if a digit is 2 or not using linear regression. Let $Y = 1$ for all the 2's digits in the training set, and use $Y = 0$ for all other digits.
Build a linear classifier by minimizing:

$$\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - w \cdot x_i)^2 + \lambda \|w\|^2$$

using the training set training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Use appropriate regularization as needed. Let x_i be a vector of the pixel values of the image.
For the purpose of classification, we can labela digit as a 2 if $w \cdot x$ is larger than some threshold.

1. Based on your training set, choose a reasonable threshold for classification. What is your 0/1 loss on the training set? What is your square loss on the training set?

2. On the test set, what is your 0/1 loss? What is your square loss?

3. Why might linear regression be a poor idea for classification?

4. Recall your answer for what the Bayes optimal predictor is for the square loss. What is the Bayes optimal predictor using the square loss for classification?

Solution:

1. Using the training set, we tested multiple thresholds to see which produced the best classification rate for our chosen value of $\lambda \approx 61.73$ (as suggested in Murphy). This ended up being about 0.3799. Our 0/1 loss on the training set was 58130 correct and 1870 incorrect. Our square loss was 0.0311667.

2. On the test set, using the same parameters as above, our 0/1 loss was 9666 correct and 334 incorrect and our square loss was 0.0334. The code for this problem is in the "hw1-6" file.

3. Linear regression does not do an especially good job of classifying because its approximations are continuous. In classification problems, you try to take input and map it to a discrete set of output

(classes). In this case we would like regression to map input to either 1 (if the image is of a 2) or 0 (if the image is of another digit). However, the approximation generated by linear regression is a linear hyperplane. Furthermore, if we give our regression model arbitrarily large input, it can produce approximations that are arbitrarily far from 0 or 1.

4. The optimal Bayes predictor using the square loss for classification can be defined as follows. Given a threshold $\tau \in (0, 1)$, let w satisfy

$$w = \text{argmax}_{\hat{w}} P(\hat{w} \cdot x > \tau | x \text{ is an image of a 2}).$$

Then

$$f_{Bayes} = \begin{cases} 1 & w \cdot x \, \tau \\ 0 & \text{otherwise} \end{cases}.$$

Problem 7

7.1 Time complexity and making your code fast

Algorithm: Naive Coordinate Descent

while *not converged* **do**

  $w_0 \leftarrow \sum_{i=1}^{N} \left(y_i - \sum_j w_j X_{ij} \right) / N$;

  for $k \in \{1, 2, \ldots, d\}$ **do**

    $a_k \leftarrow 2 \sum_{i=1}^{N} X_{ik}^2$;

    $c_k \leftarrow 2 \sum_{i=1}^{N} X_{ik} \left(y_i - (w_0 + \sum_{j \neq k} w_j X_{ij}) \right)$;

    $$w_k \leftarrow \begin{cases} (c_k + \lambda)/a_k & c_k < -\lambda \\ 0 & c_k \in [-\lambda, \lambda] \\ (c_k - \lambda)/a_k & c_k > \lambda \end{cases};$$

  end

end

Algorithm 1: Naive Coordinate Descent

Assume we are using a sparse matrix representation, so that a dot product takes time proportional to the number of non-zero entries.

1. Define $y_i = \boldsymbol{X}_i w + w_0$. Simplify the update rules for w_0 and the computation for c_k in Algorithm 1 using y.

2. Let $\|X\|_0$ be the number of nonzero entries in X. What is the time complexity to compute $\hat{y}$?

3. What is the time complexity to update w_0 when $\hat{y}$ is not already computed? What if $\hat{y}$ is already computed?

4. Let $z_j = \sum_i I(X_{ij} \neq 0)$ be the number of nonzero elements in the j-th column of X. What is the time complexity to update w_j when $\hat{y}$ is already computed?

5. Let $\hat{y}_i^{(t)} = X_i w^{(t)} + w_0^{(t)}$, and assume we update $w_0^{(t)}$ to $w_0^{(t+1)}$ using the rule above. Let $\hat{y}_i^{(t+1)} = X_i w^{(t)} + w_0^{(t+1)}$ be the new prediction after updating. Express $\hat{y}_i^{(t+1)}$ in terms of $\hat{y}_i^{(t)}$. What is the complexity to calculate $\hat{y}_i^{(t+1)}$ when $\hat{y}_i^{(t)}$ is already computed?

6. Let $\hat{y}_i^{(t)} = X_i w^{(t)} + w_0^{(t)}$, and assume we update $w_k^{(t)}$ to $w_k^{(t+1)}$ using the rule above. Let $\hat{y}_i^{(t+1)} = \sum_{j \neq k} w_j^{(t)} X_{ij} + w_k^{(t+1)} X_{ik} + w_0^{(t)}$ be the new prediction after updating. Express $\hat{y}_i^{(t+1)}$ in terms of $\hat{y}_i^{(t)}$. What is the complexity to calculate $\hat{y}_i^{(t+1)}$ when $\hat{y}_i^{(t)}$ is already computed?

7. What is the per iteration complexity of this algorithm?

7.3 Try out your work on synthetic data

Suppose the $x \in \mathbb{R}^d,\ y \in \mathbb{R},\ k < d$, and pairs of data (x_i, y_i) are generated independently according to the model

$$y_i = w_0^* + w_1^* x_{i,1} + w_2^* x_{i,2} + \cdots + w_k^* x_{i,k} + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is some Gaussian noise.

1. Let $N = 50,\ d = 75,\ k = 5$ and $\sigma = 1$. Generate some data by drawing each element of $X \in \mathbb{R}^{N \times d}$ from a standard normal distribution $\mathcal{N}(0, 1)$. Let $w_0^* = 0$ and create w^* by setting the first k elements to ± 10 and the remaining elements to 0. Finally, generate a Gaussian noise vector ϵ with variance σ^2 and form $y = Xw^* + w_0^* + \epsilon$.

   With your synthetic data, solve multiple Lasso problems on a regularization path, starting at λ_{max} and decreasing λ by a constant ratio until few features are chosen correctly. Compare the sparsity pattern of your Lasso solution $\hat{w}$ to that of the true model parameters w^*. Record values for precision and recall.

   How well are you able to discover the true nonzeros? Comment on how λ affects these results and include plots of precision and recall vs. λ.

2. Change σ to 10, regenerate the data, and solve the Lasso problem using a value of λ that worked well when $\sigma = 1$. How are precision and recall affected? How might you change λ in order to achieve better precision or recall?

7.4 Become a data scientist at Yelp

1. Solve Lasso to predict the number of useful votes a Yelp review will receive. Use the first 4000 samples for training, the next 1000 samples for validation, and the remaining samples for testing.

   Starting at λ_{max}, run Lasso on the training set, decreasing λ using previous solutions as initial conditions to each problem. Stop when you have considered enough λ's that, based on validation error, you can choose a good solution with confidence (for instance, when validation error begins increasing or stops decreasing significantly). At each solution, record the root-mean-squared-error (RMSE) on training and validation data. In addition, record the number of nonzeros in each solution.

   Plot the RMSE values together on a plot against λ. Separately plot the number of nonzeros as well.

2. Find the λ that acheives best validation performance, and test your model on the remaining set of test data. What RMSE value do you obtain?

3. Inspect your solution and take a look at the 10 features with weights largest in magnitude. List the names of these features and their weights, and comment on if the weights generally make sense intuitively.

4. Repeat parts 1, 2, and 3 using the data matrix and labels for predicting the score of a review. Use the first 30,000 examples for training and divide the remaining samples between validation and testing as before.

Solution:

7.1 Time complexity and making your code fast

1. We can express the update formula for w_0 in terms of simpler operations:

$$\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sum_j w_j X_{ij}\right) &= \frac{1}{N}\sum_{i=1}^{N}(y_i - X_i w) \\
&= \frac{1}{N}\sum_{i=1}^{N}(y_i - X_i w + w_0 - w_0) \\
&= \frac{1}{N}\sum_{i=1}^{N}((y_i - \hat{y}_i) + w_0) \\
&= \frac{1}{N}\sum_{i=1}^{N}((y_i - \hat{y}_i)) + w_0 \\
&= \mu(y - \hat{y}) + w_0,
\end{aligned}$$

   where $\mu(x)$ is the mean of the entries in the vector x. The simplified update is then

$$w_0 \leftarrow \mu(y - \hat{y}) + w_0.$$

   We can simplify the expression for c_k as well:

$$\begin{aligned}
2\sum_{i=1}^{N} X_{ik}\left(y_i - \left(w_0 + \sum_{j\neq k} w_j X_{ij}\right)\right) &= 2\sum_{i=1}^{N} X_{ik}\left(y_i - (w_0 + X_i w - w_k X_{ik})\right) \\
&= 2\sum_{i=1}^{N} X_{ik}\left(y_i - \hat{y}_i + w_k X_{ik}\right) \\
&= 2\sum_{i=1}^{N}\left(X_{ik}(y_i - \hat{y}_i) + w_k X_{ik}^2\right) \\
&= 2X(:,k)\cdot(y - \hat{y}) + 2w_k\sum_{i=1}^{N} X_{ik}^2 \\
&= 2X(:,k)\cdot(y - \hat{y}) + 2w_k a_k,
\end{aligned}$$

   where $X(:,k)$ is the k-th column of X and $\cdot$ denotes the Euclidean inner product. Hence the new update is

$$c_k \leftarrow 2X(:,k)\cdot(y - \hat{y}) + 2w_k a_k.$$

2. The overall update for $\hat{y}$ is

$$\hat{y} \leftarrow Xw + \vec{w}_0,$$

   where $\vec{w}_0$ is understood to be a vector with all its entries w_0. The cost of updating $\hat{y}$ is then $\mathcal{O}(\|X\|_0 + N)$ ($\|X\|_0$ from the matrix vector product and N additions for adding the $\vec{w}_0$ term).

3. If we do not already have $\hat{y}$ computed, then we must use the more expensive update formula

$$w_0 \leftarrow \mu(Y - Xw)$$

which requires $O(\|X\|_0 + N)$ operations. If we do have $\hat{y}$ then we simply need to compute the difference vector $y - \hat{y}$ and its average, so the cost is $\mathcal{O}(N)$.

4. To update w_k we need to first find a_k and c_k. a_k is the square of the 2-norm of a column of A, so it costs $\mathcal{O}(z_k)$ operations. To find c_k we perform a dot product and a subtraction of two length N vectors. However, since we are dotting the difference $y - \hat{y}$ with a column of X, we can take advantage of sparsity and only perform the z_k subtractions corresponding to the components that are dotted with nonzero entries of $X(:, k)$. The dot product and subtraction then cost only $\mathcal{O}(z_k)$ operations. The other operations are all constant-time complexity, so the overall complexity is $\mathcal{O}(z_k)$.

5. We have

$$\begin{aligned}
\hat{y}^{(t+1)} &= Xw^{(t)} + \vec{w}_0^{(t+1)} \\
&= Xw^{(t)} + \vec{\mu}(y - \hat{y}^{(t)}) + \vec{w}_0^{(t)} \\
&= \hat{y}^{(t)} + \vec{\mu}(y - \hat{y}^{(t)}) \\
&= \hat{y}^{(t)} + (\vec{w}_0^{(t+1)} - \vec{w}_0^{(t)}),
\end{aligned}$$

assuming we kept $w_0^{(t)}$. Otherwise we can compute $\vec{\mu}(y - \hat{y}^{(t)})$. If we already have access to $\hat{y}^{(t)}$, then the cost is just $\mathcal{O}(N)$, for the vector addition.

6. Using the relations given above, we obtain

$$\begin{aligned}
\hat{y}_i^{(t+1)} &= \sum_{j \neq k} w_j^{(t)} X_{ij} + w_k^{(t+1)} X_{ik} + w_0^{(t)} \\
&= \sum_{j} w_j^{(t)} X_{ij} + w_k^{(t+1)} X_{ik} - w_k^{(t)} X_{ik} + w_0^{(t)} \\
&= X_i w^{(t)} + \left(w_k^{(t+1)} - w_k^{(t)} \right) X_{ik} + w_0^{(t)} \\
&= \hat{y}_i^{(t)} + \left(w_k^{(t+1)} - w_k^{(t)} \right).
\end{aligned}$$

If we save the difference $w_k^{(t+1)} - w_k^{(t)}$ from when we update $w_k^{(t)}$, we get the cheap update formula

$$\hat{y}^{(t+1)} = \hat{y}^{(t)} + \left(w_k^{(t+1)} - w_k^{(t)} \right) X(:, k).$$

Assuming we already have $\hat{y}^{(t)}$, the complexity of this update is $\mathcal{O}(z_k)$ (z_k to multiply the z_k coefficients in $X(:, k)$ by a scalar, and then z_k additions to update the appropriate entries in $\hat{y}^{(t)}$).

7. Putting everything together, the total complexity of a loop of the algorithm is $\mathcal{O}(dN + \|X\|_0)$. The updates outisde the inner for-loop cost $\mathcal{O}(N + \|X\|_0)$ operations (the $\mathcal{O}(\|X\|_0)$ term comes from the matrix-vector multiplication Xw in the recomputation of $\hat{y}$ at the beginning of each iteration). An iteration of the inner loop costs $\mathcal{O}(z_k + N)$ (the difference $y - \hat{y}$ in the computation of c_k costs N operations). Summing over all d iterations gives $\mathcal{O}(\sum_k z_k + dN) = \mathcal{O}(\|X\|_0 + dN)$ operations.

7.3 Try out your work on synthetic data

1. Our implementation of the Lasso method (using the above coordinate descent algorithm) does a good job of discovering the nonzero elements of w^* for a range of λ values. The code for problem 7 is contained in the "hw1-7" file.

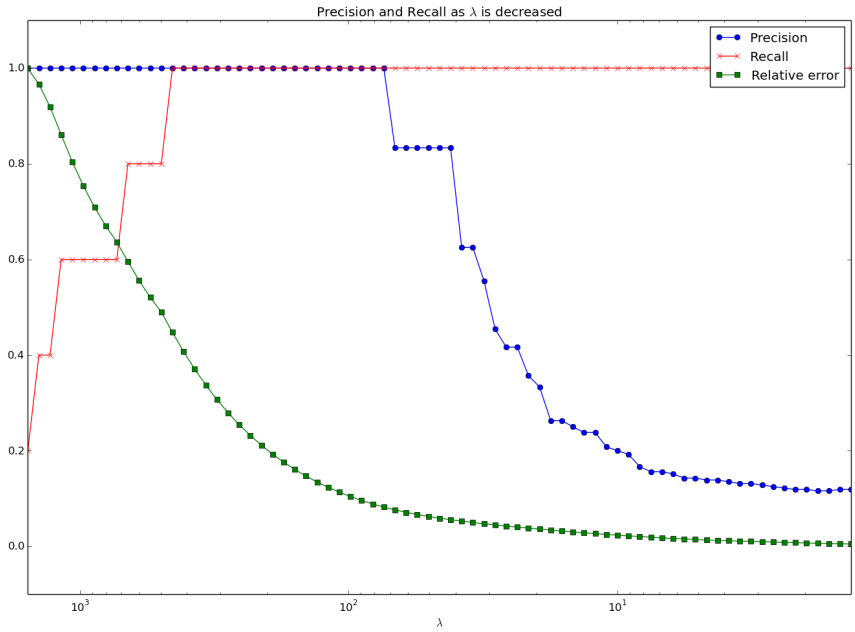

Figure 1: Precision, recall, and relative error of Lasso coefficients as λ is decreased (low variance test data)

Figure 1 shows a typical plot of the precision, recall, and relative error of the coefficients produced by Lasso along a regularization path. We began the regularization paramter, λ at λ_{max}, the minimum value which causes all the coefficients to be zero. We then divided it by 1.1 and recomputed the coefficients, precision, recall, and relative error. The relative error used was the 2-norm of the difference vector between the approximation and the true data divided by the 2-norm of the data.
As λ is varied the qualitative behavior of the solutions is as we would expect. For λ large, the relative error is high, and the method is able to detect few of the nonzero coefficients of the true solution (low recall). However, the few sparse coefficients it does detect are correct (high precision). As λ is decreased and the sparsity constraint is weakened, the relatie error decreases, the method begins correctly locating more and more of the nonzero entries (the region where both the precision and recall are 1). Eventually the sparsity constraint is so loose that the algorithm begins creating false positives–it chooses incorrect coefficients to make nonzero. However, it continues to identify the true nonzero coefficients, even if they are just a small fraction of the total number of nonzero entries it chooses. Eventually the constraint is extremely weak and the method makes many terms nonzero and the relative error approaches a minimum. Ideally one should choose λ somewhere in the region where precision and recall are both 1, i.e. where the method is correctly identifying exactly the nonzero entries. Choosing a smaller λ in this range will decrease the error on the training set, but may run the risk of finding too many nonzero elements.

2. Based on multiple trials using $\sigma = 1$, we decided to use $\lambda = 340$. This choice of λ typically produced perfect precision and recall scores. When we change σ to 10 and run the method, we start getting precision scores that bounce around between .45454545 and .8333333. The recall scores are still perfect 1's. It seems that since the data has higher variance, Lasso erroneously picks up on extra features that are not actually important. This means that our sparsity constraint is not strong enough to prevent this from happening. In order to improve the precision, we should increase λ. Changing λ to 600 seems to strike a good balance between helping the precision and not hurting the recall too much. With this

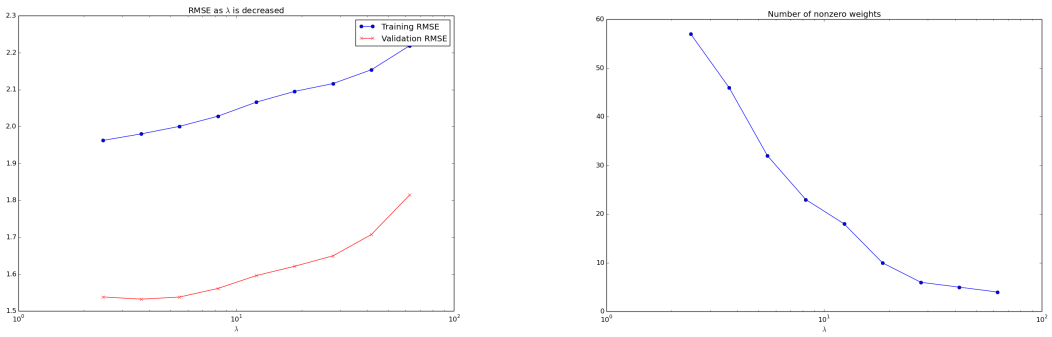

Figure 2: Left: RMSE on useful vote prediction as λ is varied. Right: Number of nonzero entries in Lasso weight vector for vote prediction as λ is varied.

choice, the precision and recall are typically both 1, but sometimes one or both will drop down to about 0.8, depending on the particular random test data.

7.4 - Become a data scientist at Yelp

1. Since using λ_{max} results in a coefficient vector of all 0's, we have not included the results of this run in the plots. The results reported here were generated by starting at $\lambda_{max}/2$, solving the Lasso problem on the training data, noting quantities of interest, then feeding the solution back into the Lasso algorithm, but with λ decreased by a factor of 1.5. We repeat this process until the RMSE on the validation data begins to increase.

   Figure 2 shows the RMSE on the training and validation sets as a function of λ as well as the number of nonzero weights in each instance. Note that these two plots show λ increasing as we move from left to right. As expected, the training error decreases as λ is decreased. Somewhat unexpectedly, the error on the validation set is lower than that of the training error, perhaps because it is contains about a quarter as many samples as the training set. In any case, after λ becomes small enough, the RMSE for the validation set begins to increase. It is then that we end our regularization path and select a particular λ to define our model. The RMSE on the validation set at this λ is a little over 1.5. Looking at the right plot in Figure 2, we see that the number of nonzero elements of the Lasso weight vector increases as λ decreases. This is simply because a smaller λ corresponds to a smaller penalty on nonzero elements. At the point when the validation RMSE levels off there are between 45 and 50 nonzero weights.

2. Using the regularization parameter $\lambda = 3.81944444$ (the λ after which the error began to increase on the validation set) we obtained a RMSE error of about 2.370476 on the training data.

3. The 10 features with weights largest in magnitude are given below along with their respective weights.

Problem 7 continued on next page...

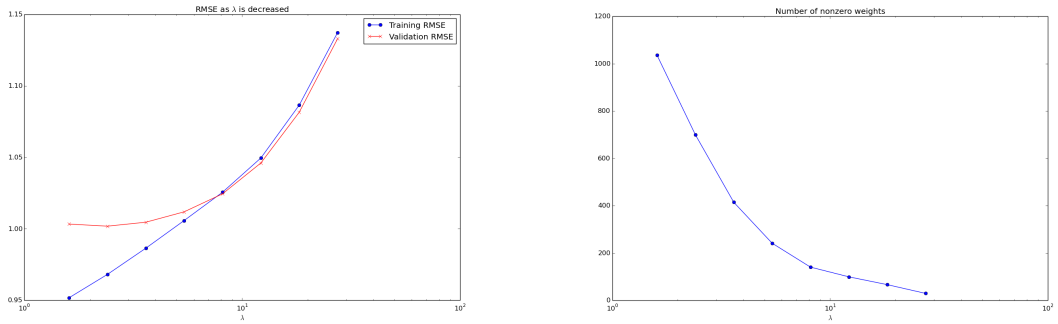

Figure 3: Left: RMSE on review score prediction as λ is varied. Right: Number of nonzero entries in Lasso weight vector for review score prediction as λ is varied.

Feature	**Weight**
sqrt(ReviewNumCharacters*UserCoolVotes)	97.1599141947
sqrt(UserCoolVotes*BusinessNumStars)	30.2084851755
sq(ReviewNumWords*UserNumReviews)	-29.1892051433
UserNumReviews*BusinessLatitude	-27.336940468
sqrt(UserFunnyVotes*InPhoenix)	19.72481161
ReviewNumWords*ReviewNumLineBreaks	13.8648759176
ReviewInFall*InGlendale	11.9915368908
UserUsefulVotes*InScottsdale	11.5698548997
BusinessNumReviews*InGlendale	11.1039128691
sq(UserUsefulVotes*IsRestaurant)	10.0336457219

Most of these labels make qualitative sense. Posts from users who have been frequently voted cool by other users and which are long are likely to be marked helpful. It is also intuitive that factors such as the number of stars a business has and the number of words and line breaks in a review (indicative of a detailed review) would influence the number of useful votes a review receives. Some of the features appear to be more geographic in nature, e.g. "BusinessLatitude" or "InPheonix".

4. Using the star data, we repeat the above steps to attempt to predict review scores. We used the same procedures as above, with a λ_{max} appropriate for this new problem. We used the first 30,000 examples for training, the next 10,000 for validation, and the last 5,000 for testing.

   Figure 3 summarizes our results. On the left we have the RMSE on the prediction and validation sets as λ is reduced. Initially (for larger λ) the RMSEs are similar, but as λ becomes small, the RMSE for the validation data begins to level out, while the RMSE of the training data continues to decrease. This is a sign that the model is beginning to overfit the training data, so we halt the process once the error on the validation set starts to grow. For this problem, this occurs just after $\lambda = 2.40834045$, where the RMSE on the validation set is about 1. On the right of Figure 3 we have the number of nonzero entries in the weight vector as a function of λ. As expected, shrinking λ increases the number of such entries. Near the optimal λ there are around 700 "important" entries.

   For the best value of λ we found, $\lambda = 2.40834045$, we achieve an RMSE of about 1.02112064 on the test set, which is about what we were able to achieve on the validation set.

   The 10 features with weights largest in magnitude are given in the following table:

Feature	**Weight**
great	27.2464360655
not	-25.3241844478
best	22.93949326
amazing	18.3310656044
rude	-16.3626730612
love	15.3474277295
the worst	-15.1828346249
delicious	15.1303879469
awesome	14.7674913791
no	-13.400891341

These features are even more intuitive than those discovered in the previous problem. The features with large positive weights are extremely positive adjectives, and are likely parts of reviews that are praising businesses. The features with large negative weights are those with words that are generally negative, such as "rude" or "the worst". The presence of these words tends to be associated with lower review scores.