CS 534: Machine Learning
Emory University
Fall2024
Prof. Eugene Agichtein

Homework # 1

Mitchell Scott
(mtscot4)

1. **(Written) Ridge Regression (10 pts):** Show that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set. The augmented dataset is created by extending the centered matrix $\mathbf{X}$ with k additional rows, with the value $\sqrt{\lambda}\mathbf{I}$, and by augmenting the vector $\mathbf{y}$ with k zeros. In other words, we add artificial k data rows with corresponding response value zero; with one row (with one non-zero value) for each coefficient, so that the fitting procedure is forced to shrink the coefficients toward zero. You are asked to show that by adding these rows the ordinary regression will end up with the same coefficients as (regularized) ridge regression.

   Proof. Let

$$\mathbf{X} = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_k^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_k^{(2)} \\ \vdots & . & \vdots & \cdots & \vdots \\ x_0^{(N)} & x_1^{(N)} & x_2^{(N)} & \cdots & x_k^{(N)} \end{pmatrix} \tag{1}$$

   be the $N \times k$ centered matrix[1], where $x_i^{(j)}$ is the data of the i^{th} feature in the j^{th}. This means that we can define $\mathbf{C}$ as the augmented matrix

$$\mathbf{C} := \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{pmatrix} \tag{2}$$

$$\mathbf{Y} := \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}, \tag{3}$$

   where $\mathbf{0} \in \mathbb{R}^k$ vector of all 0's. Then applying the solution to the ordinary least squares problem, which can be solved by the Normal Equations, on the augmented matrix $\mathbf{C}$, we observe:

$$\mathbf{C}^\top\mathbf{C}\beta = \mathbf{C}^\top\mathbf{Y} \implies \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{pmatrix}^\top \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{pmatrix} \beta = \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{pmatrix}^\top \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} \tag{4}$$

$$\implies \begin{pmatrix} \mathbf{X}^\top & \sqrt{\lambda}\mathbf{I}^\top \end{pmatrix} \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{I} \end{pmatrix} \beta = \begin{pmatrix} \mathbf{X}^\top & \sqrt{\lambda}\mathbf{I}^\top \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} \tag{5}$$

$$\implies \left(\mathbf{X}^\top\mathbf{X} + \sqrt{\lambda}^2\mathbf{I}^2\right)\beta = \mathbf{X}^\top\mathbf{y} + \sqrt{\lambda}\mathbf{0} \tag{6}$$

$$\implies \left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)\beta = \mathbf{X}^\top\mathbf{y}, \tag{7}$$

[1]Although it is not explicitly stated, here I used the assumption that $N \gg k$, so the OLS solution is well defined. This is a practical assumption as we live in a data rich world, and typically we have fewer features than data.

which is precisely the regularized normal equations with regularization parameter λ. □

2. **Predicting Appliance Energy Usage using Linear Regression (40 pts)**

   (a) **(Written) How did you preprocess the data? Explain your reasoning for using this pre-processing.**

   Solution. In lecture, we are told that there are two main ways to preprocess data; the first of being normalization. This is when you do an affine transform of the data (a shift and a scale) so that the lowest value is mapped to 0, and the highest value is mapped to 1. This is to ensure that all of the data is on the same scale.

   However, when I went to office hours, I was told that the more common way is normalization of the data, where you take the data, find the mean and standard deviation to it and map that to a normal distribution centered at mean 0 and standard deviation 1. This again makes sure things are on similar scales, but also allows us to notice the difference between positive and negative, as compared to the mean. Since we were allowed to use `scikit-learn`, I normalized the data using the preprossessing class `StandardScalar()`. Following the example from the documention, I used the `fit` method on `trainx` and then I transformed `trainx` to itself through the normalization `transform` method, where it applied the learned μ, σ to the data. Then based on those learned values for the train data, and transforms `valx` and `testx` to the normalized versions of themselves with those values. It is important to note, that to prevent data leakage, we only fit on the train data.

   The above method worked for all of the arguments (columns of the matrix) except for the 'date' column. This is because the date data is stored as a string `'YYYY-MM-DD HH:MM:SS'`, which as a non-numeric argument cannot be normalized. The first step to resolve this is to use the PANDAS method `.todatetime()`, which will convert this date data into a more standard numeric format of `'M/D/YY H:MM'`. The data type for this was `datetime64[ns]`. Then to make sure that this was a numeric value (`float64`), I then used the PANDAS method `.tonumeric()`, which then converted that datatime feature to the number of nanoseconds since a predefined time in the past (called the 'UNIX epoch'). To compensate for that, I divided the answer by $6e10$ so that the data would be in minutes since the UNIX epoch. This data manipulation then allowed my `preprocesss_data` function to work on all arguments.

   (b) **(Code) Write a function `preprocess_data(trainx, valx, testx)` that does what you described in 2a above. If you do any feature extraction, you should do it outside of this function. Your function should return the preprocessed `'trainx'`, `'valx'`,`'testx'`.**

   Solution. Implimented in `q2.py`.

   (c) **(Code) Write a function `eval_linear1(trainx, trainy, valx, valy, testx, testy)` that takes in a training set, validation set, and test set, (each in the form of a numpy arrays), respectively, and trains a standard linear regression model only on the training data, and reports the RMSE and R^2 on the training set, validation set, and test set.**

	Linear 1	Linear 2
'train-rmse'	**0.90180**	0.91239
'train-r2'	**0.18675**	0.16755
'val-rmse'	0.89546	**0.81437**
'val-r2'	0.00268	**0.17512**
'test-rmse'	0.91832	**0.79325**
'test-r2'	-0.21168	**0.09589**

Figure 1: Comparing linear 1 and linear2, which was trained on both the test and val data set. The bolded result is the one with the lower rmse and the higher R^2 value.

Your function must return a dictionary with 6 keys, `'train-rmse'`, `'train-r2'`, `'val-rmse'`, `'val-r2'`, `'test-rmse'`, `'test-r2'` and the associated values are the numeric values (e.g., {`'train-rmse': 10.2`, `'train-r2': 0.3`, `'val-rmse': 7.2`, `'val-r2': 0.2`,`'test-rmse': 12.1`, `'test-r2': 0.4`}).

Solution. Implimented in `q2.py`.

(d) **(Code) Write a function `eval_linear2(trainx, trainy, valx, valy, testx, testy)` that takes in a training set, validation set, and test set, respectively, and trains a standard linear regression model using the training and validation data together and reports the RMSE and R^2 on the training set, validation set, and test set. Your function should follow the same output format specified above.**

Solution. Implimented in `q2.py`.

(e) **(Written) Report (using a table) the RMSE and R^2 between 2c and 2d on the energydata. How do the performances compare and what do the numbers suggest?**

Solution. We see in Tab. 1, that linear 1 out performs on the training set, but linear 2 outperforms on the validation and test data. These relaizations are two sides of the same coin. Since linear 1 was only trained on the training data, it was able to capture the nuances of that data better (but not by much) which allows it a narrow victory. On the flip side, since linear 1 had no access to lthe validation set, we see linear 2 out performs rather handily. Lastly, since linear 2 was trained on more data, it hopefully has developed more robust model parameters. This explains why it annahilates the linear 1 model in both rmse and R^2.

(f) **(Code) Write a Python function `eval_ridge(trainx, trainy, valx, valy, testx, testy, alpha)` that takes the regularization parameter, alpha, and trains a ridge regression model only on the training data. Your function should follow the same output format specified in (a) and (b).**

Solution. Implimented in `q2.py`.

(g) **(Code) Write a Python function `eval_lasso(trainx, trainy, valx, valy, testx, testy, alpha)` that takes the regularization parameter, alpha, and trains a lasso regression model only on the training data. Your function should follow the same output format specified in (a),**

	$\alpha = 0.0$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
'train-rmse'	**0.90180**	0.90180	0.90180	0.90180	0.90180	0.90180
'train-r2'	**0.18675**	0.18675	0.18675	0.18675	0.18675	0.18675
'val-rmse'	**0.89546**	0.89552	0.89559	0.89565	0.89571	0.89578
'val-r2'	**0.00268**	0.00253	0.00239	0.00225	0.00210	0.00196
'test-rmse'	**0.91832**	0.91847	0.91862	0.91877	0.91892	0.91907
'test-r2'	**-0.21168**	-0.21208	-0.21248	-0.21287	-0.21327	-0.21366

Figure 2: Comparing different values for α_{ridge}. The bolded result is the α value that produced the lower rmse and the higher R^2 value.

	$\alpha = 0.0$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
'train-rmse'	**0.90214**	0.96523	0.98563	1.0	1.0	1.0
'train-r2'	**0.18614**	0.06832	0.02852	0.0	0.0	0.0
'val-rmse'	**0.90449**	0.88558	0.89342	0.89678	0.89678	0.89678
'val-r2'	-0.01754	**0.02455**	0.00721	-0.00028	-0.00028	-0.00028
'test-rmse'	0.93860	**0.82445**	0.832132	0.83438	0.83438	0.83438
'test-r2'	-0.26580	**0.02337**	0.00508	-0.00030	-0.00030	-0.00030

Figure 3: Comparing different values for α_{lasso}. The bolded result is the α value that produced the lower rmse and the higher R^2 value.

(b), and (d).

Solution. Implimented in `q2.py`.

(h) **(Written) Report (using a table) the RMSE and R^2 for training, validation, and test for all the different alpha values you tried for both Ridge and Lasso regularizers. What are the optimal parameter values for alpha you would select for Ridge and Lasso, based on the validation data performance for each?**

Solution. Since I don't know the value of α_{ridge} or α_{lasso} *a priori*, I decided to run my numerical experment for $\alpha \in \{0.0^2, 0.1, 0.20.3, 0.4, 0.5\}$. The results for Ridge regression are visualized in Tab. 2. Since we see that bolded numbers are all in $\alpha_{ridge} = 0.0$, and get higher as α_{ridge} increases, we can specilate, that the $\hat{\alpha}_{ridge}$ is smaller than 0.1. In fact, when we rerun the experiment for a smaller range of α's and many more samples, and use the validation rmse as the metric, we observe that $\hat{\alpha}_{ridge} = 9.095945 \times 10^{-13}$ with `'val-rmse'` = 0.89546.

We then repeated the experiment above for the Lasso method, which has results summarized in Tab. 3. Since all of the bolded values are between 0.0 and 0.1, we refine our search in this range, and come to the conclusion that $\hat{\alpha}_{lasso} = 0.02792$ with `'val-rmse'` = 0.86110.

3. **(50pts) Predicting Appliance Energy Usage using SGD** Consider the Appliances energy prediction Data set from the previous problem. A template file, elastic.py, defines a class ElasticNet that takes in the regularization parameters, el

[2] Actually, I ran the experiment for 1e-5, which was so that each iteration of the Ridge and Lasso method would converge, but this was just to get as close to zero as we could.

(λ) alpha (α), eta (η) or the learning rate, the batch size (batch $\in [1, N]$), and epoch or the maximum number of epochs for training. These should be provided as initialization parameters when creating the object (i.e., elastic = new ElasticNet(el, alpha, eta, batch, epoch)). All of the functions below should be implemented as member methods of your class. You will implement ElasticNet in python and NumPy using stochastic gradient descent to train your model. For this problem, you ARE NOT allowed to use any existing toolbox/implementation (e.g., scikit- learn). The functions in 'elastic.py' will be tested against a different training, validation, and test set, so it should work for a variety of datasets and assume that the data has been appropriately pre-processed (i.e., do not do any standardization or scaling to the data prior to training the model). Similar to problem 2, any additional work outside of (Code) should be done in a separate file and submitted with the Code for full credit.

(a) **(Code) Implement the helper function `loss(B, X, Y,` λ`,` α`)`. The optimization problem is:**

$$\min f_0(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda\left(\alpha\|\beta\|_2^2 + (1-\alpha)\|\beta\|_1\right), \quad 0 \leqslant \alpha \leqslant 1 \tag{8}$$

In other words, given the coefficients, data and the regularization parameters, your function will calculate the loss $f_0(x)$ as shown in Eq (1). Note that the lambda and alpha values are used slightly differently here than in question 2h.

Solution. Implimented in `elastic.py`.

(b) **(Code) Implement the gradient function grad_step(x, y, beta, el, alpha, eta). You may find it helpful to derive the update for a single training sample and to consider proximal gradient descent for the $\|\beta\|_1$ portion of the objective function. As a reminder, given step size η and regularization parameter λ s.t $f(x) = g(x) + \lambda\|x\|_1$, the proximal update is:**

$$\text{prox}(x_i) = \begin{cases} x_i - \lambda\eta, & \text{if } x_i > \lambda\eta \\ 0, & \text{if } -\lambda\eta \leqslant x_i \leqslant \lambda\eta \\ x_i + \lambda\eta, & \text{if } x_i < \lambda\eta \end{cases} \tag{9}$$

Solution. Implimented in `elastic.py`.

(c) **(Code) Implement the Python function `train(self, x, y)` for your class that trains an elastic net regression model using stochastic gradient descent. Your function should return a dictionary where the key denotes the epoch number and the value of the loss associated with that epoch.**

Solution. Implimented in `elastic.py`.

(d) **(Code) Implement the Python function `coef(self)` for your class that returns the learned coefficients as a numpy array.**

Solution. Implimented in `elastic.py`.

(e) **(Code) Implement the Python function `predict(self, x)` that predicts the label for each training sample n vector x, and returns a**

numpy array y with the predictions for each sample. If x is a numpy m × d array, then y should be a numpy 1-d array of size m × 1.

Solution. Implimented in `elastic.py`.

(f) **(Written): Parameter exploration: For the optimal regularization parameters from ridge (λ_{ridge}) and lasso (λ_{lasso}) from problem 2h, and $\alpha = 0.5$, what are good learning rates for the dataset? Justify the selection by plotting the objective value ($f_0(\mathbf{x})$) on a graph for each learning rate, where the x-axis is the epochs and the y-axis is the objective, and a curve for each learning rate as shown in slides.**

Solution. Using what I know about optimization, we first would like the learning rate (equivalent to the step size in other iterative methods), the learning rate η should be less than 1.0 but larger than 1e-8. This is why I set my η values to be $1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1$, to hopefully have a learning rate that is both too small and too large, to replicate the figure on the slide.

In Fig. 4, we see that there is very little change for smaller values of η, and a decent amount of change for $\eta = 1$. We don't see the convergence that I would have hoped to see. We observe similar behavior in Fig. 5. Since we are seeing

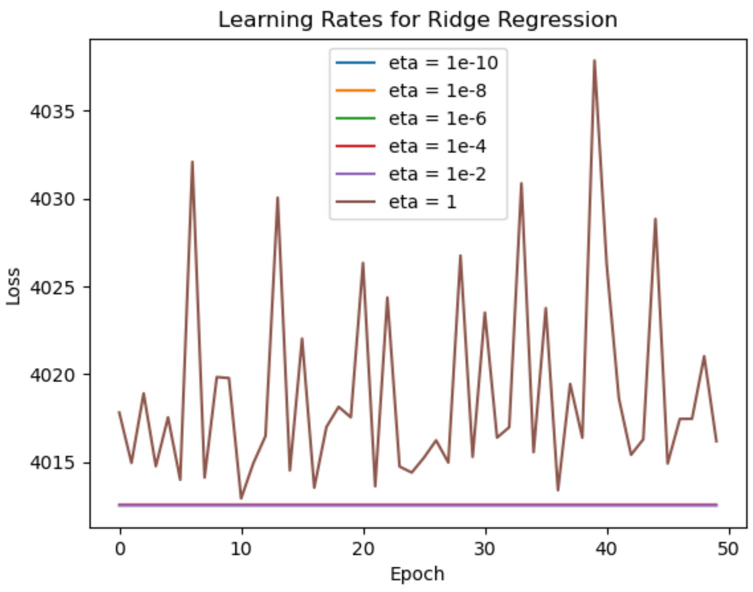

Figure 4: Different Learning Rates and their effect on Loss Function for the Ridge Regression.

teh same lackluster performance using these two values, it is causing me to believe that there is something wrong with my minibaches, more particularly, my averaging of the gradient function. When I run what I think is correct, we observe that computing the gradient specifically the matrix multiplication in the first term of the gradient function, we are getting overflow errors.

But, alas, this homework is already late, so for the sake of the next two questions, which rely on this problem, **we will use the heuristic that** $\hat{\eta} = 1e-6$ for our chosen learning rate. This fits into classical optimization, but also is what SciKit-Learn recommends for their learning rate.

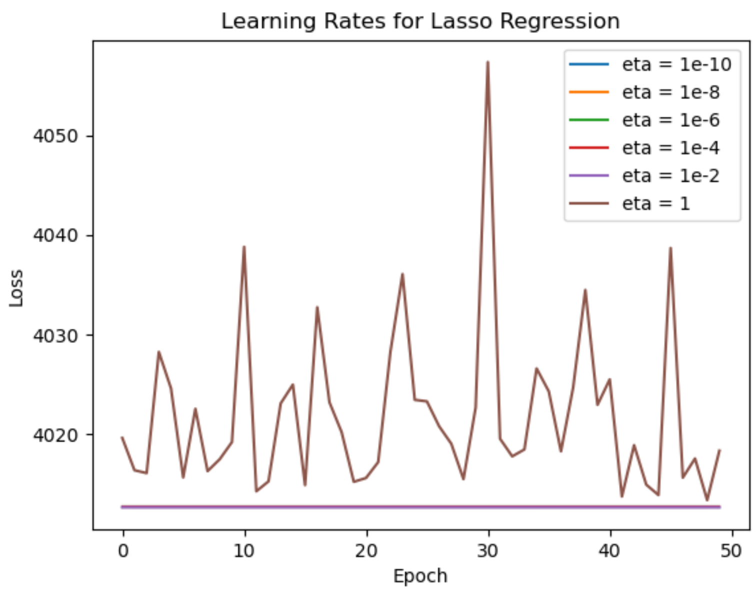

Figure 5: Different Learning Rates and their effect on Loss Function for the Lasso Regression.

(g) **(Written), Parameter exploration: Using the chosen learning rate above, explore the value of parameter α. For this, train the elastic net on the training set for each value of $\alpha \in [0, 1]$, increments of 0.1, and report in a table the values of RMSE and R^2 after predicting on training, validation and test set.**

Solution. Again, this data was generated using the assumption that $\hat{\eta} = 1e-6$. Since my data was messed up for the last part, I don't wan't to be double penalized. We can see the data presented in Tab. 6 doesn't appear to change all that much. Recall that this α parameter should interpolate between ridge and lasso regression. We notice that all of the values are the same for each α value. This tells me that there is most likely an issue with my regularization coefficient λ, or more likely the step size parameter η. While all of the data appears to be the same, we still get comparable values from the trials on the ridge and lasso regression encoded in question 1.

(h) **(Written) Final results: Report in a table the coefficients that yield the best elastic net model on the test data, and corresponding RMSE and R^2 . What conclusions can you draw compared to the best-performing model from "standard" Ridge regression and Lasso from part 2? Is there a notable difference in the model performance, or the final / best coefficients for each model?**

Solution. This question is a little hard for me to answer based on the the data presented in Tab. 6, as they are all the same. So to answer the questions, we will assume that $\hat{\alpha} = 0.4$. This means that the parameters that yielded the lowest test RMSE and R^2 are presented below: We do see that from part (2), the lowest RMSE and R^2 are 0.82445 and 0.02337, respectively. This is when $\lambda = 0.02792, \alpha = 1.0$, and the default $\eta = alpha = 0.0001$. We see that the optimized values from SciKit-learn do give about 9% lower RMSE, which is

	‘trRMSE’	‘trR2’	‘vaRMSE’	‘vaR2’	‘teRMSE’	‘teR2’
$\alpha = 0.0$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.1$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.2$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.3$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.4$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.5$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.6$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.7$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.8$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 0.9$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981
$\alpha = 1.0$	0.90180	0.18675	0.89546	0.08076	0.91832	0.08981

Figure 6: Comparing Different α vaslues. When $\alpha = 0.0$, then ElasticNet reduces to Ridge Regression; whereas, when $\alpha = 1.0$, ElasticNet reduces to Lasso Regression.

λ	α	η	‘testRMSE’	‘testR2’
0.02792	0.4*	1e-6†	0.91832	0.08981

Figure 7: Optimized Parameters for Elastic Net. * indicates there is a tie for all α values tried. † indicates that there is something seriously wrong with my code, so we set it to the heuristic set forth by SciKit-Learn forums.

to be expected because it is optimized for many different examples, and my ElasticNet is naively coded.

Lastly, in the `driver.py` function, you can see that my β values are printed out for both ridge and lasso regression, and we see that they are both dense vectors with similar weights for each atribute. The key compontent in both is the date data as it has a coefficient of 0.586 and 0.564, respectively. Both of these are very similar in sparcity and magnitude to the vector of β that the SciKit learn model shows for ridge regression. However, Lasso values are all really small or sparse, so it is hard to compare. Again, these are printed out in the driver file as well.

Acknowledgements

I would like to acknowledge that I attended both TA Swati’s and TA Ziyang’s office hours over the past week.