GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 6254 Fall 2022 Project #2

Assigned: 15 Sep
Due Date: 22 Sep

Solutions

Please contact the TAs for clarification on the instructions in the assignments.

1. In this problem we will compare the performance of traditional least squares, ridge regression, and the LASSO on a real-world dataset. We will use the "Boston House Prices" dataset which contains the median sale price (as of some point in the 1970's, when the dataset was created) of owner occupied homes in about 500 different neighborhoods in the Boston area, along with 13 features for each home that might be relevant. These features include factors such as measures of the crime rate; measures of school quality; various measures of density; proximity to things like highways, major employment centers, the Charles River; pollution levels; etc.[1]

   Follow the same steps as Project 1 to get on the COC-ICE cluster (and assuming you use the conda env ece6254 as defined in Project 1):

   - VPN into Georgia Tech (see Slide 7 of the ICE Orientation slides)

     `https://faq.oit.gatech.edu/content/how-do-i-get-started-campus-vpn/`

   - ssh <gt-userID>@coc-ice.pace.gatech.edu
   - module load anaconda3/2020.11
   - conda activate ece6254

   To get the Boston House Prices dataset in Python

   - conda install -c conda-forge scikit-learn
   - python
   - from sklearn.datasets import load_boston
   - boston = load_boston()
   - X = boston.data
   - y = boston.target

   In order to judge the quality of each approach, you should split the dataset into a training set and a testing set. The training set should consist of 400 observations, and you can use the remaining observations for testing.

[1] See `http://bit.ly/2lTueYY` for more details about this dataset.

Before training any of these algorithms, it is a good idea to "standardize" the data. By this, I mean that you should take each feature (i.e., each column of the matrix `X`) and subtract off its mean and divide by the standard deviation to make it zero mean and unit variance. Otherwise, the regularized methods will implicitly be placing bigger penalties on using features which just happen to be scaled to have small variance. You should determine how to "standardize" your training data by appropriately shifting/scaling each feature *using only the training data*, and then apply this transformation to both the training data and the testing data so that your learned function can readily be applied to the test set.

For all parts of the problem below, I would like you to submit your code.

(a) First, I would like you to evaluate the performance of least squares. You should implement this yourself using the equation we derived in class. Report the performance of your algorithm in terms of mean-squared error on the test set, i.e.,

$$\frac{1}{n_{\text{test}}}\|\boldsymbol{y}_{\text{test}} - \boldsymbol{X}_{\text{test}}\widehat{\boldsymbol{\theta}}\|_2^2.$$

Solution: Least squares results in an MSE of ≈ 38.2. *See the supplemental code for the full solution.*

(b) Next, using the formula derived in class, implement your own version of ridge regression. You will need to set the free parameter λ. You should do this using the training data in whatever manner you like (e.g., via a holdout set) – but you should *not* allow the testing dataset to influence your choice of λ. Report the value of λ selected and the performance of your algorithm in terms of mean-squared error on the test set.

Solution: Ridge regression with $\lambda \approx 350$ *results in an MSE of* ≈ 22. *See the supplemental code for the full solution.*

(c) Finally, I would like you to evaluate the performance of the LASSO. You do not need to implement this yourself. Instead, you can use scikit-learn's built in solver via

```python
from sklearn import linear_model
reg = linear_model.Lasso(alpha = ???) # Fill in alpha
reg.fit(Xtrain,ytrain)
reg.predict(Xtest)
```

Above, `alpha` corresponds to the λ parameter from the lecture notes. As in part (b), you will need to do something principled to choose a good value for this parameter. Report the value of `alpha` used in your code, the performance of your algorithm in terms of mean-squared error, and the number of nonzeros in $\boldsymbol{\theta}$. (You can get $\boldsymbol{\theta}$ via `reg.coef_`.)

Solution: When applying the LASSO to this problem, because of some quirks in the dataset, it is a little tough to determine the ideal `alpha`. *Depending on how you split your training data up to set this parameter, you might get a very different answer. My approach leads to setting* `alpha` *to be about 1.0, which results in an MSE of* ≈ 37 *using only three features. This is a slight improvement over least squares, and even though it might not be as good as ridge regression in terms of accuracy, it does have the advantage of using only 3 features. (If you are curious, these happen to be the features corresponding to the average number of rooms per dwelling, the student-teacher ratio in the local schools, and a measure of the income distribution in the area.) Interestingly, by setting* `alpha` *just slightly lower at 0.5, you get a reduction in the MSE to* ≈ 33.6 *using only two additional features (a total of five), although this choice was not suggested by my procedure for selecting* `alpha` *using the training data.*

2. In this problem I'd like you to use the following code to generate a dataset to evaluate various approaches to regression in the presence of outliers.

```python
import numpy as np
np.random.seed(2017)
n = 100
xtrain = np.random.rand(n)
ytrain = 0.25 + 0.5*xtrain + np.sqrt(0.1)*np.random.randn(n)
idx = np.random.randint(0,100,10)
ytrain[idx] = ytrain[idx] + np.random.randn(10)
```

The code above generates training data by selecting random values for the x_i's, then computing $f(x) = \frac{1}{4} + \frac{1}{2}x$ and adding a small amount of Gaussian noise to each observation. It then follows by creating some "outliers" in the y_i's by picking 10 random entries and adding a much larger amount of noise to just those elements.

In the problems below, you should find a linear fit to this data. In all of the methods below, there will be one or more parameters to set. You can do this manually using whatever approach you like. (Do not go crazy optimizing these, just tune the parameters until your estimate looks reasonable.)

(a) To begin, find a linear fit using the code for ridge regression that you produced in the first problem. Report the value of λ that you selected, and report the slope and intercept of your linear fit. Submit your code.

Solution: I selected $\lambda = 0.1$ *which resulted in an intercept of* ≈ 0.29 *and a slope of* ≈ 0.29*. See the supplemental code for the full solution.*

(b) Next, I would like you to find a linear fit using the Huber loss. This can be done via

```python
from sklearn import linear_model
reg = linear_model.HuberRegressor(epsilon = 1.35, alpha=0.001)
reg.fit(xtrain.reshape(-1,1),ytrain)
```

You have two parameters to choose here: ϵ (which controls the shape of the loss function and needs to be greater than 1.0) and α (the regularization parameter). Report the values of ϵ and α you selected, and report the slope and intercept of your linear fit (see `reg.intercept_ and reg.coef_`). Y Submit your code.

Solution: I selected $\epsilon = 1.16$ *and* $\alpha = 0.02$*, which resulted in an intercept of* ≈ 0.25 *and a slope of* ≈ 0.38*. See the supplemental code for the full solution.*