ECE 805 – Machine Learning

Tutorial 2, Monday, January 30, 2023

---

1. In this method, we will minimize J by explicitly taking its derivatives with respect to the $\theta_j's$ and setting them to zero. Let's introduce some notation for doing calculus with matrices.

Training set: $x = \begin{bmatrix} (x^0)^T \\ (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^N)^T \end{bmatrix}$ Target values: $y = \begin{bmatrix} y^0 \\ y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}$

We aim to minimize J with respect to θ, where $\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix}$ and

$J(\theta) = \frac{1}{2}\sum_{i=0}^{m}\left(f_\theta(x^i) - y^i\right)^2$. $(Y = X\Theta)$

$Since\ f_\theta(x^i) = (x^i)^T\theta$, we can easily verify that

$$f_\theta - Y = \begin{bmatrix} f_\theta(x^{(0)}) - y^{(0)} \\ f_\theta(x^{(1)}) - y^{(1)} \\ f_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ f_\theta(x^{(N)}) - y^{(N)} \end{bmatrix} = \begin{bmatrix} (x^{(0)})^T\theta \\ (x^{(1)})^T\theta \\ (x^{(2)})^T\theta \\ \vdots \\ (x^{(N)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

$Using\ the\ fact\ that\ for\ a\ vector\ z, we\ have\ that\ z^Tz = \sum_i z_i^2$:

$$J(\theta) = \frac{1}{2}(X\Theta - Y)^T(X\Theta - Y) = \frac{1}{2}\sum_{i=0}^{m}\left(f_\theta(x^i) - y^i\right)^2$$

$Finally, to\ minimize\ J, lets\ find\ its\ derivatives\ with\ respect\ to\ \theta.$
$We\ know\ that\ \ \nabla_{A^T}\ tr\ ABA^TC = B^TA^TC^T + BA^TC$
$Hence,$

$$\begin{aligned}\nabla_\theta J(\theta) = \nabla_\theta \frac{1}{2}(X\Theta - Y)^T(X\Theta - Y) &= \frac{1}{2}\nabla_\theta(\Theta^TX^TX\Theta - \Theta^TX^TY - Y^TX\Theta + Y^TY) \\ &= \frac{1}{2}\nabla_\theta tr(\Theta^TX^TX\Theta - \Theta^TX^TY - Y^TX\Theta + Y^TY) \\ &= \frac{1}{2}\nabla_\theta(tr\ \Theta^TX^TX\Theta - 2\ tr\ Y^TX\Theta) = \frac{1}{2}(X^TX\Theta + X^TX\Theta - 2X^TY) \\ &= X^TX\Theta - X^TY\end{aligned}$$

$In\ the\ third\ step, we\ used\ the\ fact\ that\ the\ trace\ of\ a\ real\ number\ is$

$just\ the\ real\ number. The\ fourth\ step\ used\ the\ fact\ that\ tr\ A = tr\ A^T.$

$To\ minimize\ J, we\ set\ its\ derivatives\ to\ zero\ and\ obtain\ the\ normal\ equation:$

$$\frac{\partial J(\theta)}{\partial\theta} = X^TX\Theta - X^TY \Longrightarrow \frac{\partial J(\theta)}{\partial\theta} = 0 \Longrightarrow X^TX\Theta = X^TY$$

$Thus, the\ value\ of\ \Theta\ that\ minimized\ J(\theta)\ is\ given\ by\ the\ equation:$

$$\Longrightarrow \Theta = (X^TX)^{-1}X^TY$$

$Using\ \Theta = (X^T X)^{-1} X^T Y, we\ will\ find\ \theta\ that\ minimizes\ J.$

$$X = \begin{bmatrix}1\\3\\5\\7\end{bmatrix}, Y = \begin{bmatrix}6\\8\\9\\11\end{bmatrix} X = \begin{bmatrix}1 & 1\\1 & 3\\1 & 5\\1 & 7\end{bmatrix}$$

$$X^T = \begin{bmatrix}1 & 1 & 1 & 1\\1 & 3 & 5 & 7\end{bmatrix} \Rightarrow X^T X = \begin{bmatrix}1 & 1 & 1 & 1\\1 & 3 & 5 & 7\end{bmatrix} \begin{bmatrix}1 & 1\\1 & 3\\1 & 5\\1 & 7\end{bmatrix} = \begin{bmatrix}4 & 16\\16 & 84\end{bmatrix}$$

$$X^T Y = \begin{bmatrix}1 & 1 & 1 & 1\\1 & 3 & 5 & 7\end{bmatrix} \begin{bmatrix}6\\8\\9\\11\end{bmatrix} = \begin{bmatrix}34\\152\end{bmatrix}$$

$$(X^T X)^{-1} = \frac{1}{\det(X^T X)} \begin{bmatrix}84 & -16\\-16 & 4\end{bmatrix} = \frac{1}{80} \begin{bmatrix}84 & -16\\-16 & 4\end{bmatrix}$$

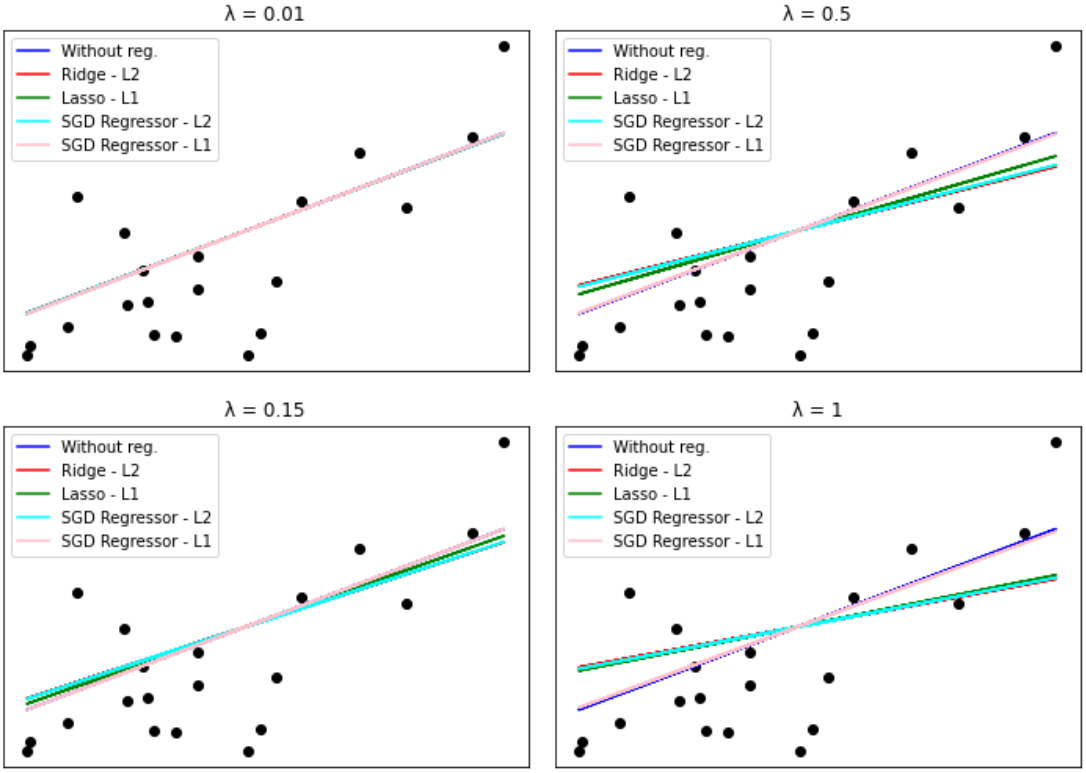
$$\det(X^T X) = 4 \cdot 84 - 16^2 = 336 - 256 = 80$$

$$\Theta = \frac{1}{80} \begin{bmatrix}84 & -16\\-16 & 4\end{bmatrix} \begin{bmatrix}34\\152\end{bmatrix} = \frac{1}{80} \begin{bmatrix}424\\64\end{bmatrix} \Rightarrow \Theta = \begin{bmatrix}5.3\\0.8\end{bmatrix}$$

Python script: ex1_tut2.py

2.

a. We will try out different (a) without, L1 and L2 regularization, and (b) different regularization parameters. The dataset to be used is the diabetes dataset from python datasets, which includes 10 features and 442 samples. We will use only one feature and linear regression model for visualization purposes.



As we can see from the above plots, when the regularization parameter is really small, the models act similar to the one without regularization. When λ increases, Ridge, Lasso and SGD Regressor – L2 perform the same, while SGD Regressor – L1 acts similar to the linear regression model without any regularization parameters.

Python script: ex2.py (includes all the models for visualization purposes)
ex2a_tut2.py (Linear regression model without regularization)
ex2b_tut2.py (Linear regression model with L2 regularization - Ridge)
ex2c_tut2.py (Linear regression model with L1 regularization - Lasso)

b. Following this, we will use the wine dataset which is a classic classification dataset. The dataset consists of 178 samples of 13 features. We will extract the mean square error (MSE) for a) Linear regression (without regularization), b) Linear Regression with L2 regularization – Ridge, c) Linear Regression with L1 regularization – Lasso, d) SGD Regressor – L2 regularization and e) SGD Regressor – L1 regularization. Also, we will compare the results between Linear Regression model and SGDRegressor in Python and discuss the differences.

Python script: ex2_MSE.py (includes all the models for extracting the MSE)

As we can see, when we integrate the regularization (L2) with Linear Regression model, the error is lower compared to the model without the regularization. Linear regression model with L1 regularization gives higher error from the model without regularization. When using the SGDRegressor with L2 regularization gives lower error than the model without regularization, but with L1 regularization we get even lower error. Different datasets give different errors, and if you re-run the script you will get a different error with the same dataset.

```
Mean squared error - without regularization: 0.104188
Mean squared error - L2 regularization: 0.076494
Mean squared error - L1 regularization: 0.089945
Mean squared error - SGD-L2: 0.089253
Mean squared error - SGD-L1: 0.086917
```

sklearn.linear_model.LinearRegression()

According to scikit-learn documentation, LinearRegression() is the ordinary least squares Linear Regression. LinearRegression() fits a linear model with coefficients w to minimize the residual sum of squares between the observed targets in the data set, and the targets predicted by the linear approximations. (It is to be noted that these linear approximations involves matrix operations)

Also, scikit-learn standard linear regression object is actually just a piece of code from scipy which is wrapped to give a predictor object. It basically uses Normal Equation to compute the minimizer analytically, and will give you a warning if the relevant matrix is non-invertible. In other words, Normal Equation is an analytical approach to Linear Regression with a Least Square Cost Function.

Also, if we wish to apply regularization techniques using LinearRegression() , we cannot do that directly by tuning the parameters of this class. We need to use Ridge() (This linear model addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients with l2 regularization) or Lasso() (It is a linear model that estimates sparse coefficients with l1 regularization) or ElasticNet() (It is a linear regression model trained with both l1 and l2 -norm regularization of the coefficients) classes of the scikit-learn library.

sklearn.linear_model.SGDRegressor()

The class SGDRegressor() implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties to fit linear regression models.

In stochastic gradient descent, we repeatedly run through the training set one data point at a time and update the parameters according to the gradient of the error with respect to each individual data point. In other words, gradient descent uses an iterative approach, starting with random values of coefficients and intercept and slowly improving them using derivatives.

The loss parameter of SGDRegressor() provides the opportunity to change the loss function being used. The default is set to squared_loss which refers to the ordinary least squares fit. The SGDRegressor() also provides a penalty parameter which basically acts as a regularization term and its default value is l2 that is Ridge Regression. It can also be set to l1 or elasticnet.

As we run the second code cell provided above multiple times, we will obtain slightly different values for loss each time. One should understand that as SGDRegressor() is an iterative approach, the parameters (that is coefficients and the intercept) obtained for the regression fit on every function call will differ slightly from one another. This can be prevented by fixing the random state of the model. To have reproducible output across multiple function calls, the parameter random_state can be set to an integer value while declaring the SGDRegressor() model.

When to use which class for Linear Regression model fitting?

It is to be understood at this point that Ordinary Least Squares being the analytical approach is not memory efficient when the size and/or the features of a data set increases. So, LinearRegression() approach is an effective and a time-saving option when one is working with a dataset with small features.

When it comes to memory efficiency, SGDRegressor() comes to the rescue. So, we can train SGDRegressor on the training data set, that does not fit into RAM. Also, we can update the SGDRegressor model with a new batch of data without retraining on the whole data set. So, SGDRegressor() approach is an effective one when one is working with the large data set, that is, large number of data points and/or features.

3. In this example, we will work on Logistic Regression and by using a manually constructed dataset with 2 classes and 5000 samples. Following this, we will use the iris dataset which includes 150 samples, 4 features and 3 classes. We will also try different (a) learning rates and (b) regularization constants.

Python script: ex3_tut2.py

Confusion matrix: