Project 1 Report

Sam Levine

Task 1A: (Warmup) Linear Regression on Synthetic Dataset, *from Scratch*

Objective

In Task 1A, our goal was to implement a basic linear regression model from scratch. This involved developing functions for the loss function, gradient descent algorithm, and testing the model's performance. All relevant code blocks and outputs can be found below as well.

Methods and Implementation:

1. Loss Function (`lossFunction`)

Purpose: To compute the empirical risk (mean squared error) for the linear regression model.

Implementation: The function accepts parameters (`theta`), feature matrix (`X`), and target vector (`y`). It computes the squared error between the predicted values (obtained using theta and `X`) and the actual target values (`y`). The mean of these squared errors represents the empirical risk.

The ridge regression risk function (as given in the original notebook), is defined as:

$$R(\mathbf{w}) = \mathbb{E}[(y - \mathbf{w}^\top x)^2] + \lambda \mathbf{w}^\top \mathbf{w}$$

$$\hat{R}_{\text{ridge}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \mathbf{x}_i\right)^2 + \lambda \mathbf{w}^\top \mathbf{w}$$

And this is how the gradient of the empirical risk function is derived:

$$\begin{aligned}
\hat{R}_{\text{ridge}}(\mathbf{w}) &= \frac{1}{n}(y - Xw)^\top (y - Xw) + \lambda \mathbf{w}^\top \mathbf{w} \\
&= \frac{1}{n}[y^\top y - 2w^\top X^\top y + w^\top X^\top X w] + \lambda \mathbf{w}^\top \mathbf{w} \\
\Rightarrow \nabla_w \hat{R}_{\text{ridge}}(\mathbf{w}) &= \frac{1}{n}[-2y^\top X + 2w^\top (X^\top X)] + 2\lambda w^\top \\
&= \frac{2}{n}[w^\top (X^\top X) - y^\top X] + 2\lambda w^\top
\end{aligned}$$

Results: `Loss at initial theta (zeros):  3.4740788381807883`

2. Gradient Descent (`gradientDescent`)

Purpose: To optimize the model parameters (theta) by minimizing the empirical risk.

Implementation: The function updates theta iteratively using the gradient descent rule (as given in the original notebook):

$$\hat{\mathbf{w}}_{k+1} \leftarrow \hat{\mathbf{w}}_k - \eta_k \left. \frac{\partial \hat{R}}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_k},$$

where η_k is the learning rate and $\frac{\partial \hat{R}}{\partial \mathbf{w}}$ is the gradient of the empirical risk function. The algorithm terminates when the loss values converge, as specified by the 'tolerance' parameter.

Results:

```
The regularized theta using ridge regression:
 [[1.73309706]
 [0.57451649]]
```

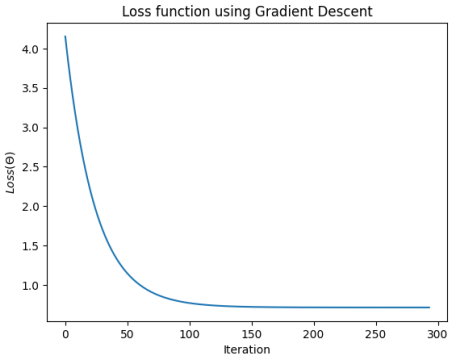

Figure 1: Your image caption here.

3. Results and Evaluation

The implementation of the `lossFunction` and `gradientDescent` successfully allowed us to train the ridge regression model. We can see that here:

Task 1B: Real Dataset: House Value Prediction

Objective

In Task 1B, we applied our ridge regression model to the California housing dataset with the aim of predicting house values in California districts. This dataset, sourced from the 1990 census data, is available through the scikit-learn package.

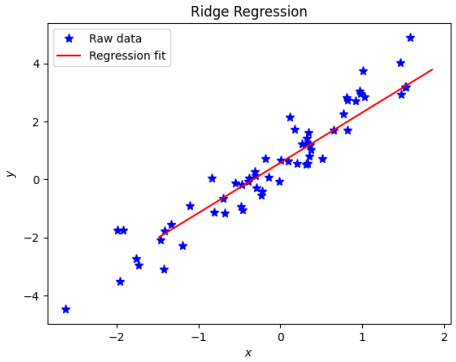

Figure 2: Your image caption here.

Methods and Implementation:

1. Training and evaluation (`train_and_eval`)

Purpose: The `train_and_eval` function is designed to fit the ridge regression model on the training data and subsequently evaluate its performance on a separate evaluation set. The purpose of this function is to understand how well the model generalizes to new, unseen data and to assess the impact of the regularization parameter λ on the model's performance.

Implementation: The function accepts training and evaluation datasets (`X_train`, `y_train`, `X_eval`, `y_eval`), along with a regularization parameter (`lambda_`). It begins by initializing the model parameters (`theta`) and setting the learning rate (`Eta`) and tolerance (`Tolerance`) for the gradient descent algorithm. The `gradientDescent` function is then called to train the model on the training data. After training, the model's performance is evaluated on the evaluation dataset by computing the mean squared error (MSE) between the predicted values and the actual values in `y_eval`. This process is repeated for different values of λ, and the test MSE is plotted as a function of λ to analyze the model's sensitivity to the regularization parameter.

Results: We use `plt.xscale('log')` to make our graph a little more useful. This plot shows how the test MSE varies with different values of λ.

2. Model selection via k-fold cross validation (`cross_validation`)

Purpose: The `cross_validation` function is designed to select the optimal value of the regularization parameter λ for the ridge regression model. This is achieved by performing k-fold cross validation, which helps in estimating the model's performance on unseen data and mitigating the risk of overfitting.

Implementation: The function performs 10-fold cross validation on the training data (`X_train`, `y_train`). The dataset is divided into 10 equal parts, and for each fold, the model is trained on 9 parts and evaluated on the remaining part using the `train_and_eval` function. This process is repeated for each fold,

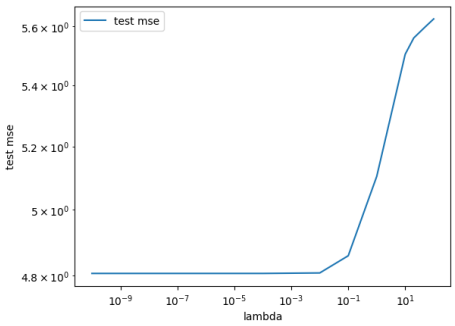

Figure 3: Your image caption here.

and the mean squared error (MSE) is calculated for each λ. The λ value that results in the lowest average MSE across all folds is selected as the optimal parameter.

Results: The cross-validation process yielded the following MSEs for different λ values:

```
cross validation with lambda:  1e-10
5.909225775211335
cross validation with lambda:  1e-06
5.909220212122046
cross validation with lambda:  0.0001
5.908576629218645
cross validation with lambda:  0.01
5.855085235007105
cross validation with lambda:  0.1
5.689339742402454
cross validation with lambda:  1
5.556405617534707
cross validation with lambda:  10
5.590139642676777
cross validation with lambda:  20
5.598313548518187
cross validation with lambda:  50
5.603882595976733
cross validation with lambda:  100
5.60767940413326
Best lambda:  1
```