Machine Learning Project Report

Ahmed Tlili, Cyrine Akrout, Ahmed Zguir
EPFL

April 20, 2024

1 Introduction

In this project we introduced 3 classes: `LinearRegression`, `LogisticRegression` and `KNN`. The class main is a script that utilizes these classes to train a model on a specified set and predicts on given samples. We normalized the training and test set using the Z-score normalization then we added a bias. If `--test` is passed as parameter we output the classification metrics on the Test Set. Otherwise, we output these metrics on the Validation set which is 20% of the training set. We introduced a new argument `--KFold_plot` which is by default set to false in order to plot the MSE curve for regression tasks and the accuracy curves for classification tasks and find the optimal hyperparameters using cross validation. Above each figure, you can find the terminal command used to plot.

2 Linear Regression

2.1 Model Training

The model employs the normal equation to compute the weight vector, incorporating a regularization term when $\lambda > 0$ to perform ridge regression. This approach allows for a direct calculation of the optimal weights, minimizing the regularized mean squared error through the equation:

$$W = (X^T X + \lambda I)^{-1} X^T Y$$

We don't directly compute the inverse matrix but instead we use numpy linear algebra solver for numerical stability.

2.2 Hyperparameter Tuning

We experimented with various values of λ to find the optimal balance (through KFold cross validation with $K = 5$).

```
$ python main.py --method
linear_regression --task center_locating
--KFold_plot
```

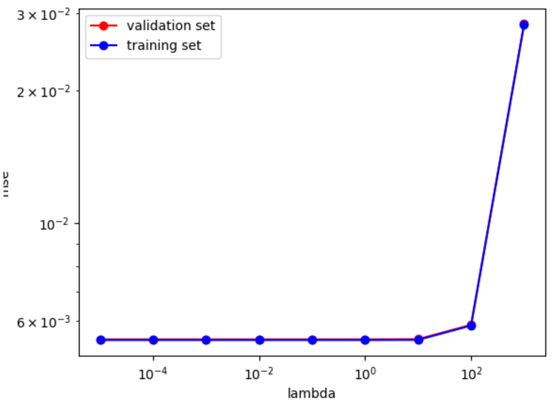

Figure 1: Impact of Regularization Strength λ on Linear Regression Performance.

We can see both curves overlap. We find that the mean square error in this case becomes bigger as we increase λ.
So we will choose $\lambda = 0$.

3 Logistic Regression

3.1 Model Training

Training involves converting labels to one-hot encoded vectors, initializing weights, and iteratively updating these weights to minimize the loss function. The update step at each iteration is guided by the learning rate and the gradient of the loss with respect to the weights. The process incorporates a softmax function to compute class probabilities and uses these probabilities to calculate the cross-entropy loss, thereby enabling the model to handle multiclass classification effectively.

In the provided implementation of the logistic regression classifier, an essential optimization technique has been employed to enhance numerical stability:

```
scores -= np.max(scores, axis = 1,
                 keepdims = True)
```

This step adjusts the scores by subtracting the maximum score in each row before applying the exponential function. This operation ensures that the highest value in each set of scores is 0, significantly reducing the risk of exponential overflow, which can lead to NaN values during computation. Importantly, this optimization does not alter the outcome of the softmax function, as the subtraction of a constant value from all scores does not affect the relative differences between them.

3.2 Hyperparameter Tuning

```
$ python main.py --method
logistic_regression --task
breed_identifying --KFold_plot
```

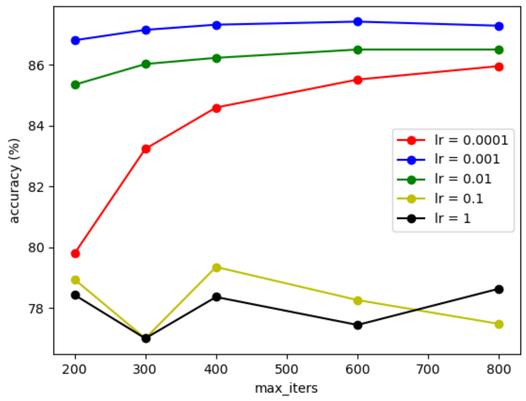

Figure 2: Impact of Learning Rate and Maximum Iterations on Model Accuracy (Validation Set).

We clearly reach maximum accuracy with $lr = 0.001$ and $iters_{max} = 600$.

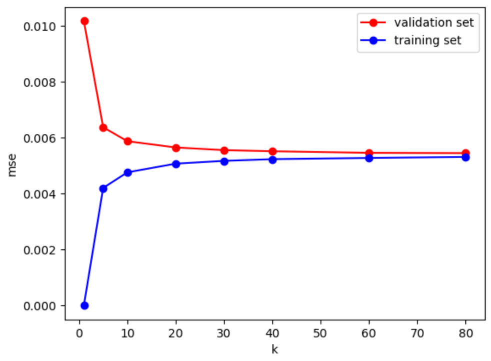

Figure 3: Finding the Optimal k: Performance Analysis of kNN (regression).

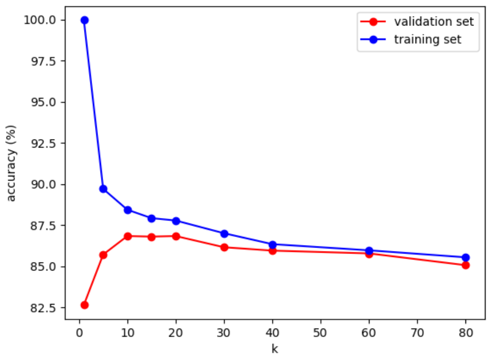

Figure 4: Finding the Optimal k: Performance Analysis of kNN (classification).

For regression, we take $k = 30$ and for classification $k_{\text{optimal}} = 20$.

4 K-Nearest Neighbors (KNN)

4.1 Model Training

In regression tasks, the prediction is the average of the labels of the k nearest neighbors. This approach ensures simplicity and computational efficiency while still leveraging the collective knowledge of neighboring data points. $\hat{y} = \frac{1}{k}\sum_{i=1}^{k} y_{NNi}$

Now for classification, the kNN algorithm identifies the k nearest neighbors and determines the output by selecting the most frequent label among them. This process supports multi-class classification and is effective in capturing the local structure of the data by relying on the majority vote.

4.2 Hyperparameter Tuning

```
$ python main.py --method knn --task
center_locating --KFold_plot
$ python main.py --method knn --task
breed_identifying --KFold_plot
```

5 Conclusion

Finally this is the overall performance after the optimisation of the hyperparameters and running each of the following commands :

```
(1) $ python main.py --method
linear_regression --lmda 0 --task
center_locating --test
(2) $ python main.py --method
logistic_regression --lr 1e-3 --max_iters
600 --task breed_identifying --test
(3) $ python main.py --method knn --task
center_locating --K 30 --test
(4) $ python main.py --method knn --task
breed_identifying --K 20 --test
```

-	Linear Reg	Logistic Reg	KNN
Breed	-	(2)86.2%	(4)86.2%
Center	(1)0.005(MSE)	-	(3)0.005(MSE)

Table 1: Benchmark on Test Data

Which is slightly better than the benchmark given in the Project Description.