Introduction to Machine Learning Project Report (MS1)

Arda Çınar Demirtaş, 384868 - Hojoon Kim, 385098 - Yiğit Narter, 384870

1. Introduction

The project aims to apply the machine learning techniques we learned in the course to a subset of the Stanford Dogs dataset. Our objective is first to accurately identify the breed of dogs depicted in images, and second, to determine the precise center point of each dog within the image. To achieve this, we employ linear regression and logistic regression for regression and classification tasks respectively, as well as k-nearest neighbors (KNN) for both tasks. We evaluate the performance of these methods using appropriate evaluation metrics and present our findings in the subsequent sections.

2. Method

2.1 Data Preparation

We implemented three data preparations: validation set, adding bias, normalization.

When the argument "test" is not set, which means it's not a test, we create a validation set from the training set. We also added an argument as "val_ratio" that determines the ratio for the validation set.

For linear regression, we augment the dataset with an additional dimension for biases to facilitate matrix multiplication. All features should be normalized for the kNN algorithm, and for this we used z-score standardization, which transforms values such that the results have mean 0 and standard deviation 1. The same normalization technique was used for logistic regression as well, since normalizing the data provides significantly better accuracy results.

2.2 Methodology

We implemented the regularized linear regression algorithm for a regression task. Linear regression provides a closed-form solution for determining the model's weights and outputs mathematically. By utilizing the sum of squares of the weights as a regularizer, ridge regression also achieves a closed-form. During the training phase, our approach involves obtaining optimal weights through matrix computation. Subsequently, during the testing phase, predictions are made by multiplying the weights with the input data. The main hyperparameter is lambda, which indicates the impact of the regularization term.

We conducted multi-class logistic regression for the classification task. We use the softmax function to transform predictions by the model into probabilities across multiple classes, ensuring they sum to one. For prediction, the class with highest probability among all is selected, and that becomes the class assignment for the new input. For training, to measure the error between predicted probabilities and ground truth labels (both of which are one-hot encoded), we employed the multi-class cross-entropy loss function. As logistic regression does not have a closed-form solution, we employed gradient descent for optimization on the said loss function of the weights of our model..

We also implemented the k-nearest neighbors (kNN) algorithm for both classification and regression tasks. kNN is an algorithm that essentially operates by finding the k training samples with minimum distances to a test sample based on a chosen distance metric, which is Euclidean distance in our case. For classification, the algorithm assigns the majority class label among the k neighbors, while for regression, it computes the mean of the k nearest neighbors' target values. The main hyperparameter of kNN is k, representing the number of neighbors to consider during prediction.

To assess algorithm performance, we employed accuracy and macro F1-score metrics for classification, and mean squared error for regression.

2.3 Hyperparameter Search and Visualization

Another addition was to visualize the hyperparameter evaluations for all three methods. We added an argument as "plt" that enables this visualization.

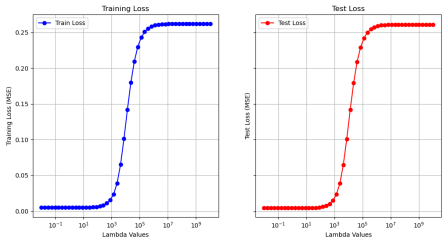
For linear regression, our implementation iterates lambda values in a logarithmic scale. For logistic regression, our implementation iterates over different learning rates in a logistic scale. For kNN, our implementation iterates over a range of k values specified by args.K.

For the center locating task, it computes the mean squared error (MSE) for train and test sets across different hyperparameters and plots them. For the breed identifying task, it computes accuracy and F1-score for train and test sets across different learning rates for logistic regression and different k values for KNN, and plots them in separate subplots.

3. Experiment and Results

We ran the main.py code on the Stanford Dogs dataset to evaluate the performance of the methods we implemented.

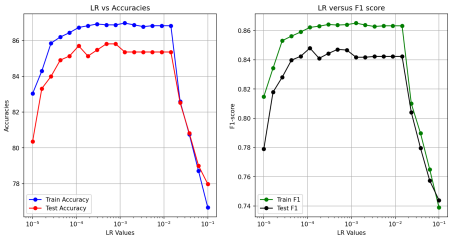
To test the linear regression algorithm for regression, we varied the hyperparameter lambda from 0.01 to 10^{10} in a logarithmic scale and visualized the metrics.



[Figure 1: Loss for Linear Regression]

Figure 1 illustrates that lower values of lambda correspond to lower losses for both training and testing. Thus, we can infer that a lambda value of 0 is optimal.

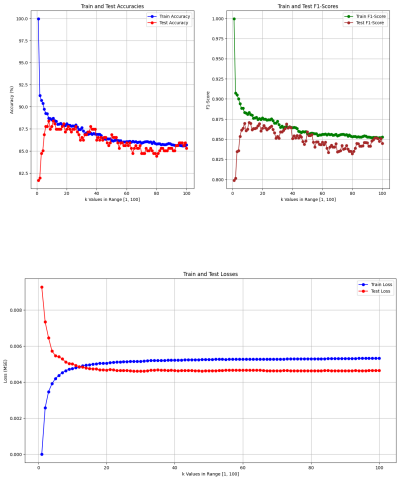
For logistic regression, a hyperparameter search was done to test 20 different learning rates from 10^{-5} to 10^{-2} in a logistic scale for gradient descent. Figure 2 shows these learning rates and their effects on accuracy and F1 score after performing gradient descent for each with maximum 6000 iterations. The optimal learning rate was around 10^{-3} taking into account both train and test set performance.



[Figure 2: Accuracy and F1 for Logistic Regression]

To test the k-nearest neighbors (KNN) algorithm for classification and regression, we varied the parameter k from 1 to 100 and visualized the metrics. Figure 3 shows that as K is increased, the training losses increase for both tasks, hence for training the optimal K value is 0. For the test accuracy, as k increases, the accuracy initially improves, reaching a peak value. However, beyond certain k values, the accuracy starts to decrease. We observed this trend in both tasks: for breed identification, the optimal k value was

found to be 11, while for center locating, it was 30. These values represent the points where the test performance is maximized for their respective tasks.



[Figure 3: Loss for kNN]

4. Discussion and Conclusion

In conclusion, Table 1 shows the training performance of our final model with optimal hyperparameters.

	Metric	Linear Regression	Logistic Regression	KNN
Breed	Acc.	-	87.146% (lr = 1e-3)	99.966% (K=1)
	F1	-	0.863500 (lr = 1e-3)	0.999620 (K=1)
Center	MSE	0.005 (λ=0)	-	0.000 (K=1)

[Table 1: training performance]

	Metric	Linear Regression	Logistic Regression	KNN
Breed	Acc.	-	85.769% (lr = 1e-3)	88.379% (K=11)
	F1	-	0.846251 (lr = 1e-3)	0.870884 (K=11)
Center	MSE	0.005 (λ=0)	-	0.005 (K=30)

[Table 2: test performance]

For the breed identifying task, kNN outscored logistic regression in both accuracy and F1-score metrics. This may be because of the training data having some inherent nonlinearities which kNN can capture, and logistic regression cannot, since the latter is a linear method.

For the center locating task, KNN also showed more success than linear regression in terms of training performance, which is expected since setting the hyperparameter k to 1 essentially is equivalent to capturing the closest data point without considering the influence of surrounding data, whereas linear regression attempts to model the relationship between features and target variables by fitting. Considering the test performance however, the MSE errors for both methods were approximately the same, which is due to the fact that kNN might overfit the training data and may not generalize well to unseen test data despite choosing the optimal value for the hyperparameter k. This leads to similar test errors as linear regression, which assumes a linear relationship between features and target variables.

Despite better performance in terms of evaluation metrics, kNN is also the most computationally expensive model out of the three since it requires the computation of the distances to all training samples for all the test samples. Still, due to not having parameters to optimize during training and having found the optimal value for k, it exhibited the best performance for both tasks.

In terms of fine-tuning our models, for logistic regression, it was observed that normalizing the data as well as the multi-class cross entropy loss provided almost a 7% increase in accuracy. This was probably because by doing so, the "landscape" (the loss function) on which the gradient descent was also normalized, providing better convergence properties.

During hyperparameter search for logistic regression, the border values for the learning rate were chosen empirically, as larger learning rates cause the gradient descent to diverge/explode, and smaller ones just learn too slowly. This can be seen from the figure 2 as well, as towards both sides of the plots, the performance drops significantly. Iterations were decided to not be searched as the change in weights were thresholded with an epsilon of 10^{-4}, and the maximum number of 6000 iterations were never reached because of this. Also from figure 2, it was observed that train and test accuracies are very similar, which shows that the algorithm generalizes well and does not overfit the training data.

We can also analyze the pattern of loss curves in linear regression from figure 1. Lambda is a hyperparameter that indicates regularization to prevent overfitting. Therefore, it is natural for the test loss to increase as lambda increases, indicating a reduction in overfitting. In addition, since both training and test losses show a similar trend, we can analyze as follows. Except when the size of the training dataset is smaller than the input dimensionality, the likelihood of overfitting in linear regression is very low. Therefore, due to the sufficiently large training set, the algorithm generalizes well so even when lambda is low, the test loss remains low. Moreover, as lambda increases, reducing the complexity of the model, overall accuracy may decrease. This decrease in model complexity can increase both training and test losses.

For kNN, the performance was enhanced by first normalizing the training and test samples by z-score standardization. Later for the test samples and training samples separately, we experimented with different values of the parameter k. For both tasks, we were able to detect the optimal values for k, hence obtaining the best performance with the lowest loss values. The optimal value of k is essential to prevent overfitting or underfitting, which are common concepts in machine learning. To improve kNN performance even more in regression, instead of taking the mean value, the weighted average of the k nearest neighbors' target values can be computed, which is inversely proportional to the distances. This will enable a better performance by taking into account the probabilities of the closer points to the sample, hence reducing the error.

Overall, the methods were all successful with the training data as well as the test data. It was observed that kNN beats linear and logistic regression in center locating and breed identifying tasks respectively, possibly since it can capture nonlinearities in the dataset, and the other two cannot.