Report for Milestone 1 in CS-233: Introduction to Machine Learning

Candidates: 385347, 375887, 385346

21.04.2024

1 Introduction

This is a report for Milestone 1 in CS-233 Introduction to machine learning at EPFL. It focuses on the teams' methodology, choice of hyperparameters and the different results from our implementations for regression and classification.

2 Method

The different models are trained and tested on data from The Stanford Dogs dataset. Our data preparation consists of normalization, adding a bias term and creating a validation set. The normalization and adding bias term uses the given functions in utils.py, while splitting the training data into 70/30 yields our validation set. This set is normalized and implemented while experimenting with different hyperparameters in the logistic regression model, which will be discussed later in the report.

2.1 Methods for Linear Regression

The method implemented in the linear model uses a regularizer (ridge regression) with an adjustable lambda parameter. Since the group wanted to create a model that can be used on large datasets, we chose not to use the expensive analytical formula presented in lectures:

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_F)^{-1}\mathbf{X}^T\mathbf{y} \tag{1}$$

Hence the optimal weights are found using the gradient descent algorithm, such that the performance of our model additionally relies on the learning rate and number of iterations.

$$\nabla_w J(w) = -\frac{2}{N}\sum_{i=1}^{N}(y_i - x_i^T w)x_i + 2\lambda w \tag{2}$$

The process of finding the optimal hyperparameters consisted of exploring different combinations and measuring the accuracy of the training labels after being applied to the model. In addition the group has discussed implementing feature expansion. But with the impression that feature expansion is tedious to implement without using a kernel, and kernels not being included in the syllabus for MS1, the group chose not to.

2.2 Methods for Logistic Regression

The argument of the analytical solution being expensive for the linear model, also holds for the logistic regression model. Hence the implementation of the gradient descent algorithm follows, while using the softmax function:

$$\hat{y}^{(k)}(\mathbf{x}) = \frac{exp(\mathbf{w}_k^T\mathbf{x})}{\sum_{j=1}^{C} exp(\mathbf{w}_j^T\mathbf{x})} \tag{3}$$

We've also decided to add a regularizer to the error, and therefore use the same hyperparameters as for the linear model. For this model we chose a different approach by plotting the performance with the different combinations of lambda, learning rate and iterations (appendix), using the validation set presented in the introduction. The plots are generated by using "$--plot\_hyperparameters$" in the command for logistic regression.

2.3 Methods KNN

The K-Nearest Neighbors (KNN) algorithm represents an intuitive approach applicable to both classification and regression tasks. In classification, the algorithm takes an unlabeled data point and identifies the K nearest labeled data points within the training set using the *Euclidean distance*:

$$d(\mathbf{x_i}, \mathbf{x}) = \sqrt{\sum_{d=1}^{D}(x_i^{(d)} - x^{(d)})^2} \tag{4}$$

It then will assign the majority class for the new data point, in case of .
In linear regression, instead of forecasting a categorical label, the algorithm predicts a continuous value. It is doing so by computing the average of the target values of the K nearest neighbors and assigning this calculated value to the new data point.

3 Results

3.1 Results for Linear Regression

During hyperparameter tuning, it was observed that the model's performance was particularly sensitive to the lambda value when combined with a large learning rate (e.g. lr = 1), often leading to divergence. Conversely, with a smaller learning rate (e.g. lr =0.0001), changes in lambda had minimal impact on performance. The combination of lr = 0.01 and lambda = 1 resulted in both training and test loss reaching 0.046 (MSE), which, based on our experiments, suggests this could be near the optimal setting for the model. This had a runtime for training of 0.023 seconds and a runtime for predictions of $2.30 * 10^{-5}$ seconds.

3.2 Results for Logistic Regression

Our results (appendix) indicates that using the highest learning rate (lr = 0.1) leads to the model's underperformance, suggesting that too much regularization might cause underfitting. The model's accuracy improves when the learning rate is adjusted to 0.01 and 0.005. However, a very low learning rate (lr = 0.001 or lr = 0.0001) has a negative effect on performance, pointing to the importance of finding a balanced learning rate. The poor performance for a high lambda value of 100 and 10 particularly telling, with a maximum accuracy of 47.62% and 85.49%, supporting the idea that less regularization is preferable. An optimal setting, based on our data, involves a low lambda with a learning rate near 0.025, extending beyond 200 iterations. With these settings, specifically lambda = 0.1 and lr = 0.005 at 200 iterations, the model achieves accuracy of 87.917% with an F1-score of 0.874 on the training set, and an accuracy of 86.239% with an F1-score of 0.850 on the test set. This had a runtime for training of 0.178 seconds and a runtime for predictions of $2.00 * 10^{-4}$ seconds.

3.3 Results for KNN

For the classification task: our results accuracy varied between 81.346% and 88.073% depending on our selected k (appendix). The algorithm yielded the best results with k=8, k=16, k=17 and worst with k = 2. We learned in class that when k is lowest the fitting of the algorithm requires the most resources yet we saw in our project that this led to over-fitting and much poorer performance.

For the regression task: our results' loss varied between 0.009 and under 0.0050 depending on our selected k. We received our worse results with k = 1 and better results as we increased our k. It seems that we reach a plateau of loss 0.0050 at k = 6.

For both classification and regression, the fitting task had a long runtime of around 18 seconds and the prediction task had a runtime of about 2 seconds. We noticed the runtime doesn't increase dramatically for larger k's, we believe it is a result of the fact that even for a lower k, the heavy calculation of measuring it's distance from all other points is always necessary. Unlike other models, KNN requires no prior fitting and bases each prediction on the same kind of intricate calculation, which is the reason why our runtime for prediction was so high in comparison to the other models.

Because the run-time doesn't increase dramtically with k, when considering which k factor we should use to analyze different data, we mainly focus on the model's performance.

4 Discussion

4.1 Cross-Validation

In evaluating model performance, data partitioning is crucial. Our dataset uses a 70-30 split for training and testing, but this split's impact on performance isn't fixed; it can vary. Cross-validation allows us to explore these variations. The *Leave-One-Out* method offers the best performance by not wasting any data, but it's expensive for large datasets like ours. Alternatively, *10-Fold Cross-Validation* strikes a balance between reliability and computational cost, making it a viable option. To optimize performance, experimenting with different partitions while considering computational costs is essential, and could be a way to improve our model.

4.2 Results

Comparing to the benchmarks in the project description, each method shows reasonably good performance, as reflected in the result table. Improving performance beyond cross-validation involves optimizing data preparation. Normalizing data before fitting can enhance analysis stability, while adding regularization to linear and logistic regression prevents overfitting. Through group discussions, a suitable mix of data preparation methods has been determined.

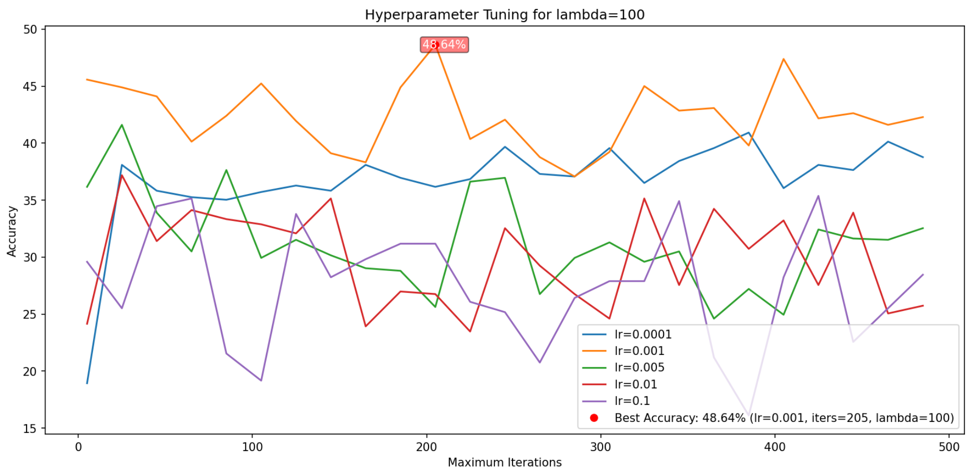
5 Conclusion

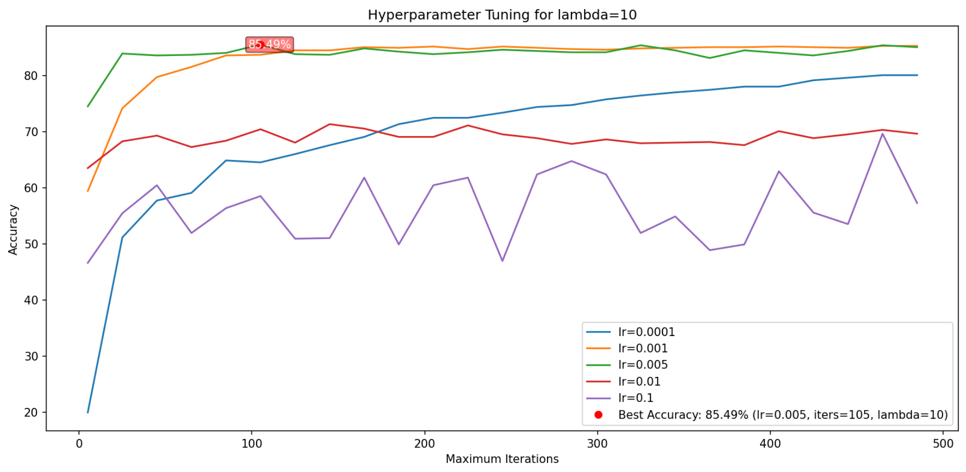
In our project, we underscored the importance of hyperparameter tuning and data preparation, in the implementation of linear regression, logistic regression, and the KNN algorithm. This has substantially improved our understanding of the models' implementations and their hyperparameter sensitivities. Moreover, it highlighted the critical role of cross-validation in determining optimal hyperparameters.

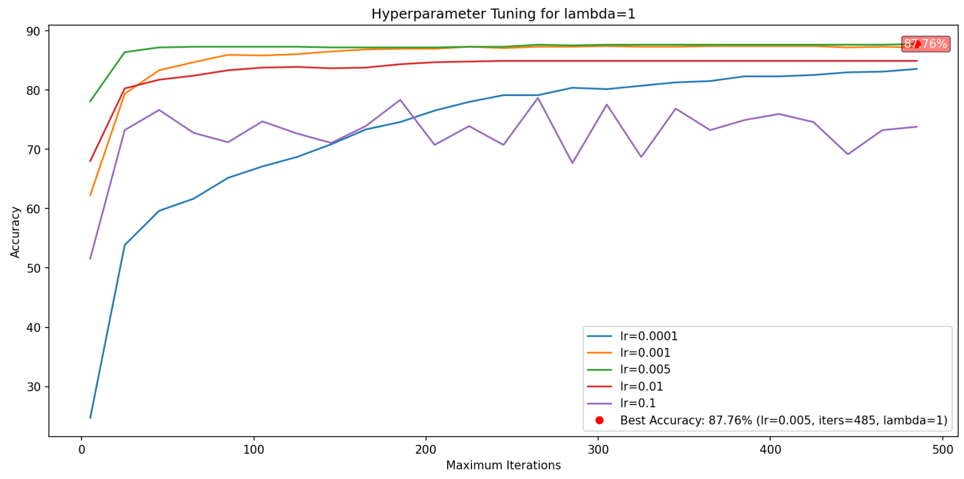
6 Appendix

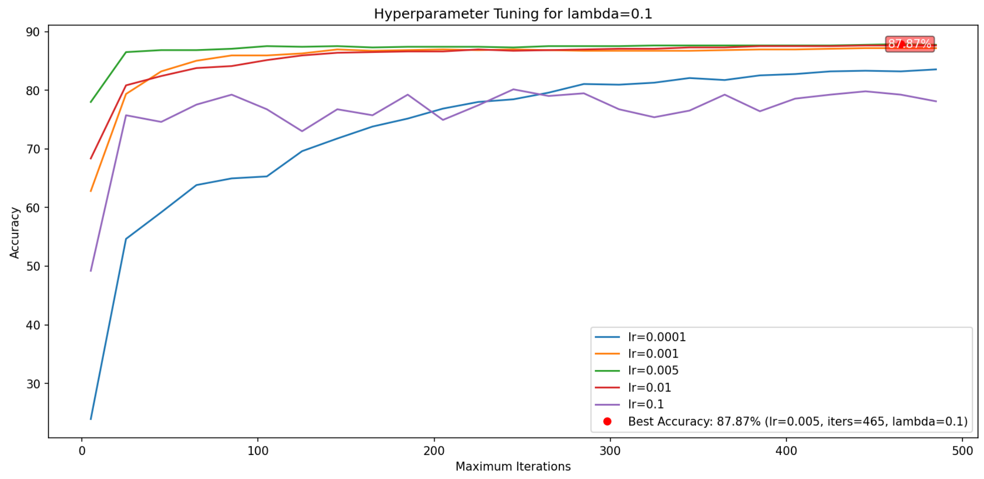
Result table:

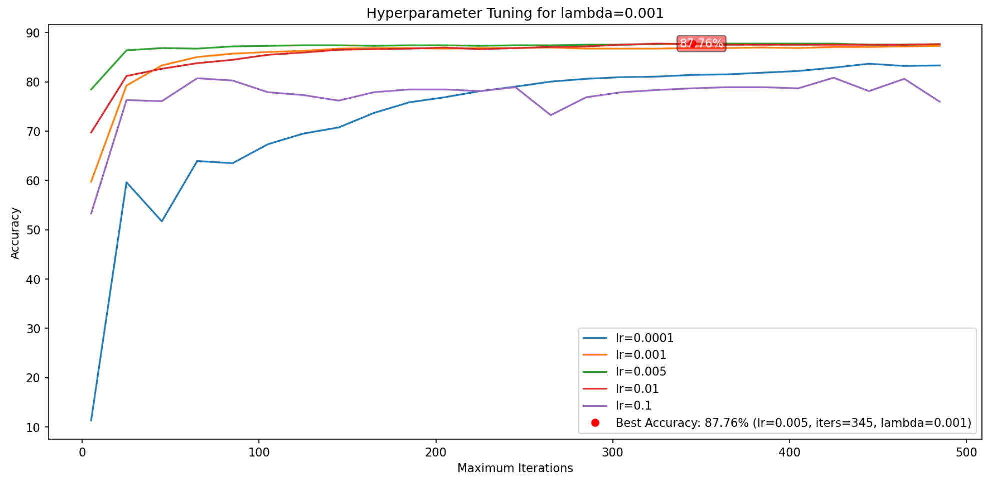
Model	**Linear Regression**	**Logistic Regression**	**KNN**
Parameter	lr $= 0.01$, $\lambda = 1$	lr $= 0.005$, $\lambda = 0.1$, Iter. $= 200$	K $= 8$, K$=$ 16, K$=$17
Loss (MSE)	0.046	-	0.005
Benchmark loss (MSE)	0.02	-	0.01
Accuracy	-	87.92%	81.346% - 88.073%
Benchmark Accuracy	-	80.04%	81%
Runtime (Training & Prediction)	0.023 s & 2.30×10^{-5} s	0.178 s & 2.00×10^{-4} s	$\sim$ 18 s & $\sim$ 2s

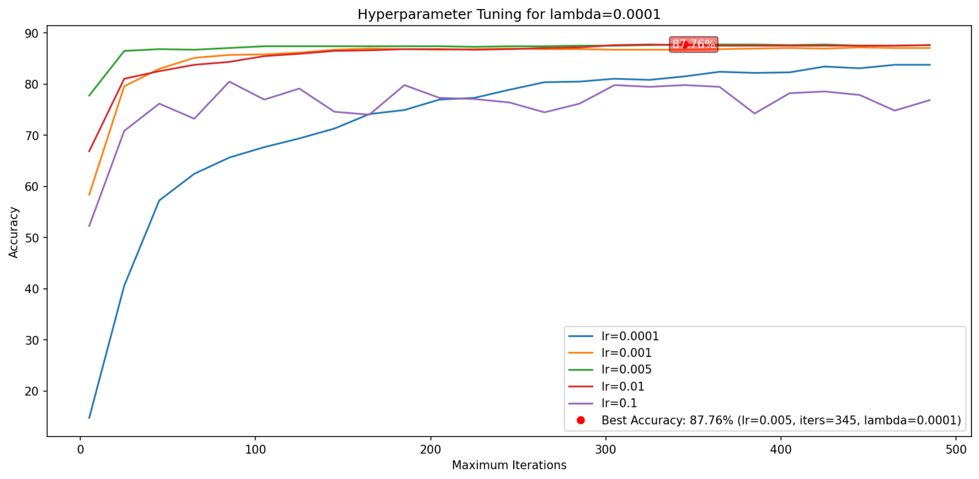




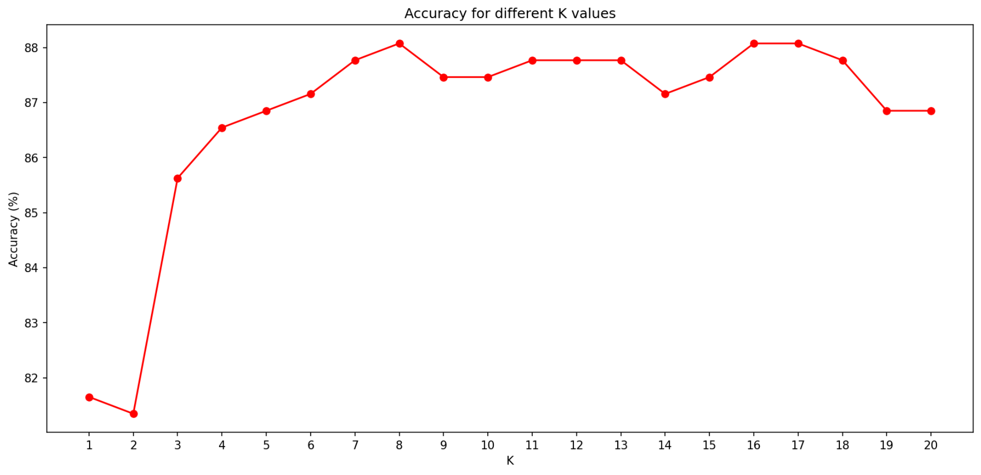

Hyperparameter Tuning for lambda=1
87.76%
Accuracy
Maximum Iterations
lr=0.0001
lr=0.001
lr=0.005
lr=0.01
lr=0.1
Best Accuracy: 87.76% (lr=0.005, iters=485, lambda=1)


Hyperparameter Tuning for lambda=0.1
87.87%
Accuracy
Maximum Iterations
lr=0.0001
lr=0.001
lr=0.005
lr=0.01
lr=0.1
Best Accuracy: 87.87% (lr=0.005, iters=465, lambda=0.1)

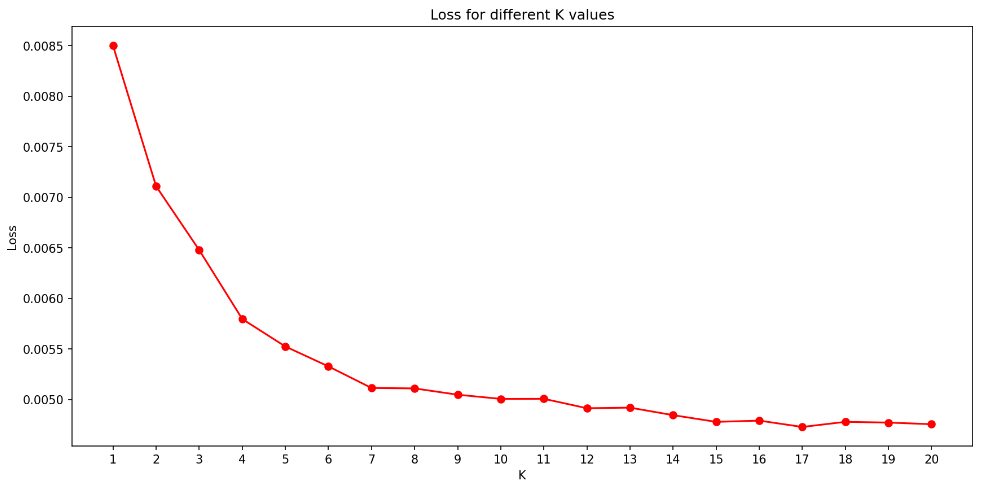

Hyperparameter Tuning for lambda=0.001
87.76%
Accuracy
Maximum Iterations
lr=0.0001
lr=0.001
lr=0.005
lr=0.01
lr=0.1
Best Accuracy: 87.76% (lr=0.005, iters=345, lambda=0.001)



Hyperparameter Tuning for lambda=0.0001
Accuracy
Maximum Iterations
87.76%
lr=0.0001
lr=0.001
lr=0.005
lr=0.01
lr=0.1
Best Accuracy: 87.76% (lr=0.005, iters=345, lambda=0.0001)



Accuracy for different K values
Accuracy (%)
K



Loss for different K values
Loss
K