Introduction to Machine Learning Report

Erik Hübner[1], **Ruri Osmon**[2], **Richard Petre**[3]

[1]*EPFL (sciper 341025)*
[2]*EPFL (sciper 347405)*
[3]*EPFL (sciper 355627)*

1. Introduction

In this project for the Introduction to Machine Learning course, our aim is to explore and implement fundamental machine learning algorithms and apply them to a real-world dataset. The primary dataset used in this study is a subset of the Stanford Dogs dataset, which consists of images representing various dog breeds from around the world. As part of our data preprocessing efforts, we normalized the input data for both regression and classification tasks to ensure uniformity in scale and to mitigate algorithm bias. This was achieved by standardizing each feature to have a mean of zero and a standard deviation of one. Additionally, we augmented our dataset by adding a bias term to each instance, enhancing the model's ability to adjust the intercept term effectively. The overarching objectives of the project are twofold: firstly, to accurately identify the breed of a dog from its image; and secondly, to precisely locate the center point of the dog within the image. Through the course of this project, we will develop, test, and refine methods such as linear regression, logistic regression, and k-nearest neighbors (kNN) to address these classification and regression challenges. By applying these techniques, we not only aim to achieve high accuracy in our predictions but also gain deeper insights into the practical applications and challenges of machine learning algorithms in image-based tasks.

2. Methods

2.1. Regression Task

In this section, we explore the regression task of our machine learning project, where our objective is to determine the center points of dogs within images.

2.1.1. Linear Regression

In this project, we wanted to use regularised linear regression with a λ as a hyper-parameter. We used the closed form solution to get W, the matrix of the model parameters, from the training data.
With X being the training data parameters matrix and Y training data results matrix and λ set by default to 1.

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

2.1.2. kNN

This method, like the linear regression, has a hyper-parameter k which is used in determining how many nearest neighbors do we need to take in account. During the training period, we save the training data and training labels in order to use them in the prediction of the test data, and predict the labels of the training data even if it is not useful.
For the prediction, we calculate the distance between a point in the input test data and the input training data, using the euclidean distance. Then using these distances, we take the k-nearest neighbors and we calculate their weight using the following formula:

$$\text{weight} = \frac{1}{\text{eucliean\_distances\_nearest}} \tag{2}$$

Using these weights, we can find the label of the point using the following formula

$$\hat{\mathbf{y}} = \frac{1}{\sum_{i=1}^{k} w_{NNi}} \sum_{i=1}^{k} w_{NNi} \mathbf{y}_{NNi} \tag{3}$$

which is implemented using the method `average` in the `numpy` library. We have also added the option to use a K-fold method to get the best k for the training data. By default the it starts at 1 and tries a k every 4 k until 40 and splits the data into 5 folds. We found that a good value for k is 20 so we set it by default.

2.2. Classification Task

Following the regression task, we address the classification task, which involves identifying the breed of the dog from its image. This part of the project utilizes logistic regression and k-nearest neighbors (KNN) to categorize each image into one of the several dog breeds present in our dataset.

2.2.1. Logistic Regression

The first method we will use for training is multi-class logistic regression. In this approach, we combine gradient descent with the softmax function (see equation (4)) to optimize our weights. This method relies heavily on two critical hyperparameters: the learning rate and the maximum number of iterations. The learning rate is pivotal as it dictates the step size at each iteration, affecting both the convergence speed and the stability of the training process. An appropriately set learning rate ensures efficient convergence to a good solution without overshooting or getting stuck in local minima. The maximum number of iterations sets a cap on how many times the training algorithm will cycle through the entire dataset, which helps prevent overfitting by halting the training when additional iterations no longer significantly reduce the loss. Determining the optimal values for these parameters is essential for effective learning and is typically achieved through experimentation with techniques like grid search, random search, or adaptive learning methods. We put the most optimized parameters as default in the python file.

$$y^{(k)}(\mathbf{x}_i) = P(y_i = k | \mathbf{x}_i, \mathbf{W}) = \frac{\exp \mathbf{x}_i^\top \mathbf{w}_{(k)}}{\sum_j^C \exp \mathbf{x}_i^\top \mathbf{w}_{(j)}} \tag{4}$$

2.2.2. kNN

As described previously this method has a hyper parameter k which is the amount of neighbours it needs to consider. For classification the data saved when training is the same as for regression. Normally, the method used for classification is the majority vote of the k nearest neighbours with the smallest average distance of the labels for a tiebreak, but while experimenting we found that by making it a weighted vote like in kNN regression the test loss decreased slightly. The method implemented was similar to the regression's one but used `bincount` instead. As in the regression, there is the option to use the K-fold cross validation to get the best k for the training data and it uses the same default parameters.

3. Results

3.1. Regression Task

In linear regression, we obtained a train loss of 0.6% and a test loss of 0.5% based on mean square error. Changing the λ has no observable effect if it is below 50. In kNN regression, the hyperparameter k also has very little impact on the loss, going from 0.5% at 10 to 0.6% and even when it takes its maximum value it stays consistent.
The most accurate parameters are $k = 228$ and $\lambda < 50$ both with losses of 0.5%

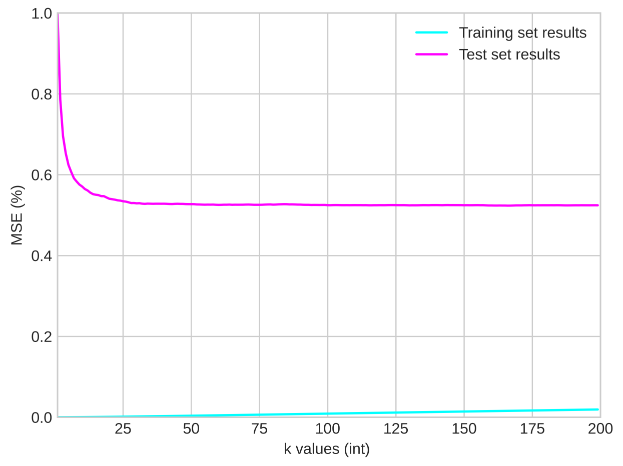

Figure 1. kNN Regression Task: MSE depending on k hyperparameter

3.2. Classification Task

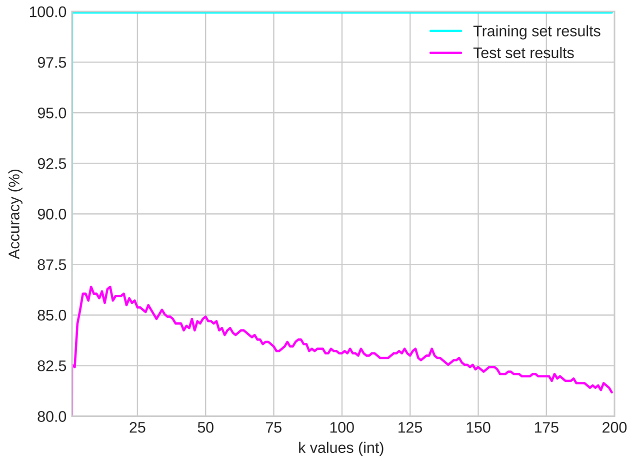

Figure 2. kNN: classification precision on validation set depending on the k parameter

K-fold says $k = 12$ is the best with an accuracy of 85% on the test set. To optimize our logistic regression model, we initially set the learning rate at 1 and focused on identifying the optimal number of iterations for gradient descent. After determining this parameter, we fixed it and fine-tuned the learning rate.

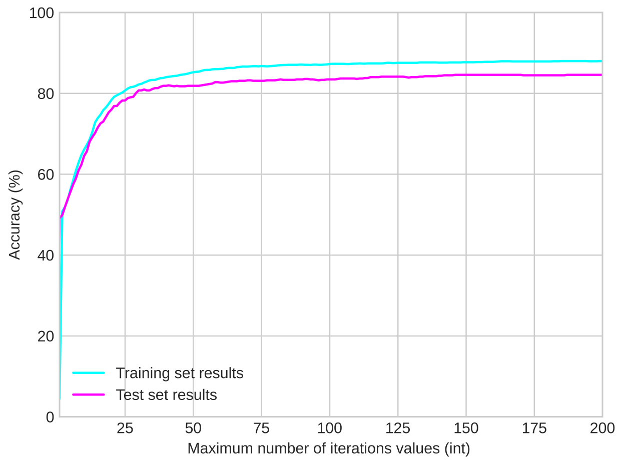

Figure 3. Linear Regression: precision depending on the maximum number of iterations

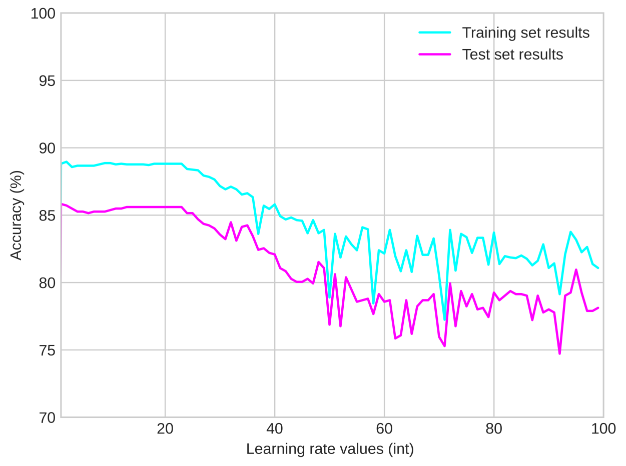

Figure 4. Linear Regression: precision depending on learning rate

Method	Linear Regression	kNN Regression	Logistic Regression	kNN Classification
Fit	0.001	0.160	0.050	0.119
Pred	0.000	0.226	0.001	0.050
K-fold Fit	N\A	6.731	N\A	4.729

Table 1. *Runtimes of all methods with default parameters except $k = 10$*

4. Conclusion

4.1. Regression Task

Our regression methods have similar accuracies. An increase of λ has no visible impact on runtime and only a negative impact on the accuracy. On the other hand an increase in k will impact the runtime and the optimal k for the training data seems to be 228 but the loss difference between $k = 36$ and $k = 228$ is less than 0.01%, and the errors of the training set prediction grows with k so we settled on a compromise at $k = 50$ for better performance. As the kNN is much longer and the only benefit is a small decrease in error when predicting training data, the linear regression is the better method for this data set.

4.2. Classification Task

After fine tuning the parameters in the logistic regression we achieved an accuracy of 89% on the training set and an 86% accuracy on the test set. With the kNN classification we achieved a near 100% accuracy on the training set which was expected and an 86% on the test set which is better than the logistic regression. The kNN performs much more accurately on the training set however the two methods have approximately the same results on the test set.

4.3. General

Overall when using kNN, the training data has an almost 100% accuracy and it has a similar accuracy for the test set compared to the other methods. This is because we added weights depending on the inverse euclidean distance, without them it the accuracy on the training data is similar to the accuracy of test data. However, it has a much longer run time as it must compute the distances for each sample and fitting only saves the training data to be reused later. It therefore has larger impact in storage and performance compared to the others.
In conclusion, the weighted kNN approach significantly enhances model accuracy on the training data and maintains competitive performance on the test set, though at the cost of increased computational and storage demands.