Introduction to Machine Learning

Programming Assignment 1

Christian Wiskott, 01505394.

November 16, 2020

Abstract

All tasks were completed and uploaded to Gitlab. The files require no input parameters and can simply be run with a standard IDE or from the command-line. The print statements produce the relevant results. Some operations that take extensive run-times were commented out (*1.5, 1.6, 5.1*).

Task: 1: Least Squares Linear Regression

Subtask 1: Data Preparation

The dataset "California Housing", after been split into the training- (70%) and the test set (30%), consists of 14447 training- and 6192 test-samples using 8 features.

Subtask 2: Regression using scikit-learn

After fitting a linear regression model (without bias) with the training data, the MSE on the training set amounted to 0.60 and 0.618 on the test set. This, of course varies slightly with newly drawn samples from the data-set.

Subtask 3: Implement Linear Regression on Your Own I

For the manual prediction, the closed form solution of the weight vector given by $w_* = [X^T X]^{-1} X^T Y$ was computed. The resulting weight vector is given by: $[0.522, 0.016, -0.195, 1.004, 8.893 \cdot 10^{-6}, -0.0043, -0.0583, -0.0135]$ and resulted in the same exact MSEs compared in the previous subtask.

Subtask 4: Implement Least Squares Linear Regression on Your Own II

To approximate the true solution, the gradient descent method was employed using a reformulation of the original, to the vectorized problem to utilize the efficiency of numpy matrix-vector multiplications. The resulting MSEs for 1 + 1000k gradient steps, where $k = (0,\ 100)$ are shown in the upper left in fig. 1.

The results show that the test- and training MSEs after the 100k iterations, being 2.75 and 2.6 respectively, don't converge to the analytical solutions. Hence, in this case this method doesn't provide the expected results.

Subtask 5: Implement Least Squares Linear Regression on Your Own III

With the bold driver adaptive learning rate, the GD-method leads to better MSEs (~ 1.2), despite being terminated after 30k iterations due to the excessive run-times caused by the cost-evaluations done every iteration. Despite the efforts that went into trying to increase performance with the help of matrix-vector operations, no significant speed-up could be achieved.

Subtask 6: Implement Least Squares Linear Regression on Your Own IV

Using the *MinMaxScalar* of sklearn that transforms the features to a range between (0,1) via the equation 1 (with min=0, max=1)[1], the GD and the GD using adaptive learning rate were applied to the normalized data. The second row in fig. 1 shows the corresponding results. The naive GD-method doesn't respond well to the changes, since the method doesn't provide improved convergence behaviour. The MSEs appear to remain at a constant value of ~ 5.65, independent of the number of iterations. The reason for this problem couldn't be identified and was reported by multiple other students as well.

The bold driver however seemed to react well to the normalization. The results converged nicely to the analytical solutions but similarly, the computation was terminated after 30k iterations due to excessive run-time.

$$\begin{aligned} X_{std} &= (X - X.min(axis = 0))/(X.max(axis = 0) - X.min(axis = 0)) \\ X_{scaled} &= X_{std} * (max - min) + min \end{aligned} \tag{1}$$

Subtask 7: Least Squares Linear Regression Using Higher-order Features

Table 1 showcases the MSEs of the data-set for the given degrees of polynomial features. Although the training-MSEs decrease with higher degrees, the increasing

[1] taken from `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

	degree 2	degree 3	degree 4
Train-MSE	0.415	0.340	0.280
Test-MSE	1.253	2.71×10^4	2.01×10^9

Table 1: Training- and test MSEs using polynomial feature transformations of degrees (2,4). At higher degrees the over-fitting becomes very apparent.

test MSEs paint the typical picture of over-fitting. No feature transformation attains a better performance than the linear model.

Task: 2: Non-linear Regression and Classification

Subtask 1: Toy Data Regression

In order to achieve the target-MSE of 0.01 on the test data, the non-linear feature transformation acc. to eq. 2 has been chosen to approximate the cone-shaped data visible in fig. 2. This transformation was implemented by adding a column of the transformed features to the feature matrix X as well as a column of 1's to incorporate the offset-terms.

$$\sqrt{x_1^2 + x_2^2} \tag{2}$$

To further increase the accuracy of the model, the data was standardized i.e. centered around 0 with a unit standard deviation via the StandardScalar[2] of sklearn.

The best performance that could be achieved was a training-MSE of 0.0167 and a test-MSE of 0.0178 using the above mentioned feature transformation. Neither adding newly transformed features nor tweaking parameters yielded any improvements w.r.t. the test-MSE.

Using polynomial feature transformation the best results were achieved using degree=6, which resulted in a training-MSE of 0.091 and a test-MSE of 0.182.

Subtask 2: Toy Data Classification

In order to achieve the 95% classification accuracy, the feature transformation acc. to eq. 3 was used, which fits the circle-shaped data nicely.

$$x_1^2 + x_2^2 \tag{3}$$

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

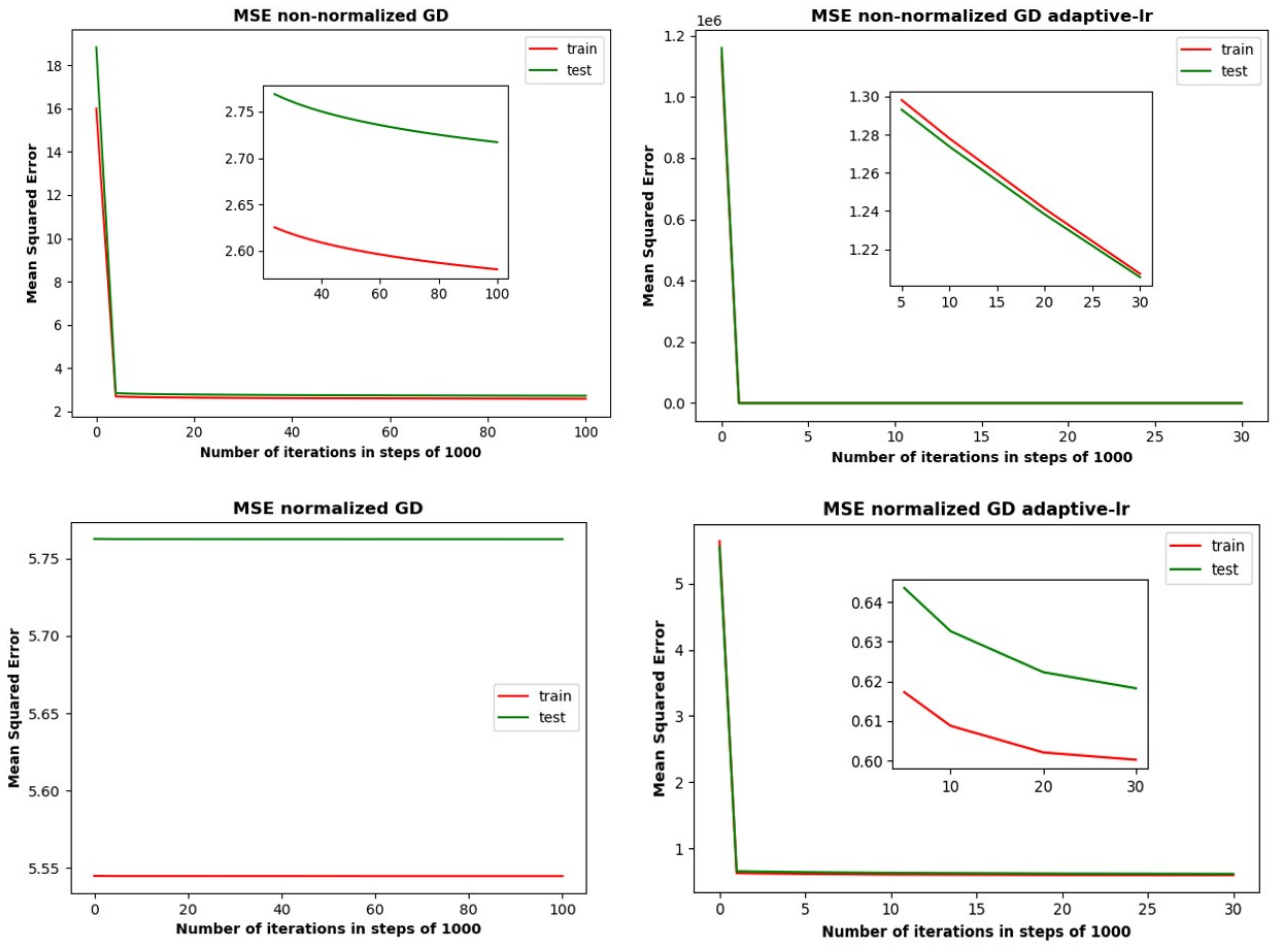

Figure 1: Results of GD- and GD with adaptive learning rate (bold driver) using non-normalized and normalized features for both methods. It's apparent that the bold driver using normalized features lead to the best performance and converged nicely to the analytical solution.

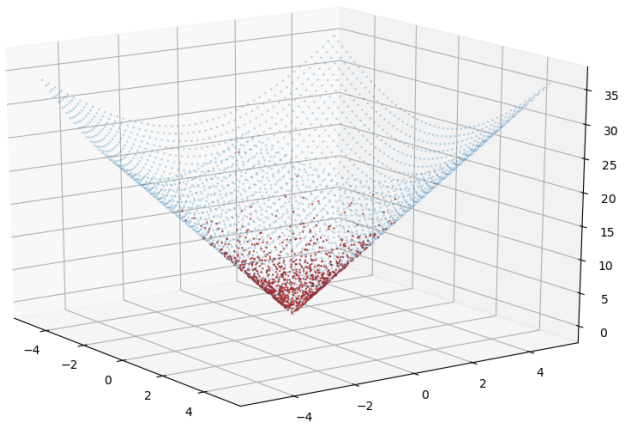

Figure 2: 3-d plot of the features in red and the used model in blue. Visual inspection showcases the accuracy of the regression-model.

The transformation of subtask 1 (see eq. 2) was added as well via additional columns, which further improved the accuracy of the model. The SMV was fitted with the transformed data and resulted in exactly 95% classification accuracy, which can be seen by stuyding the computed confusion matrix, given by:

$$\begin{bmatrix} 358 & 14 & 0 & 0 \\ 14 & 485 & 1 & 0 \\ 0 & 9 & 105 & 0 \\ 0 & 0 & 12 & 2 \end{bmatrix} \tag{4}$$

which presents the prediction accuracy of the model via *True Positives* & *True Negatives* (sum of all diagonal entries = 950, thus 95% accuracy) and *False Positives* & *False Negatives* (sum of all off-diagonal entries = 50) with the feature matrix X containing 1000 samples.

Task: 3: Estimating Generalization Errors

Subtask 1: Baseline

After applying the prescribed train/test split and normalizing the data using the *MinMaxScalar*, the linear regression model was fitted to the data and resulted in

polynomial degree	1	2	3	4	5
(1) Training MSEs					
MSE	0.515	0.383	0.317	0.255	0.184
(2) 50/50 Split					
MSE	0.516	65.16	1.45×10^7	4.45×10^{14}	1.24×10^{20}
(3) Cross-Validation					
MSE	0.518	15.16	1.41×10^7	4.53×10^{14}	4.75×10^{19}

Table 2: Results of the different methods for choosing the optimal poynomial degree. (1) the highest degree results in the smallest MSE, (2) the linear model is the best perfoming model, (3) returns also the linear model as the best one.

a training MSE of 0.515 and a test MSE of 0.532. Due to the unorthodox split, the results hugely depend on the randomly selected samples in the training/test set, which can lead to large fluctuations of the MSEs. Using the polynomial features of degree 2, the train MSE was determined as 0.383 and the test MSE as 3.218, hence an over-fitted model, which already shows that the linear model approximates the data the best.

Subtask 2: Implement Cross-validation

Using the manual 10-fold cross-validation, the MSE of the linear estimator was estimated as 0.518, which is already a quite good approximation of the true value.

Subtask 3: Model Selection

To predict the performance of the polynomial feature transformations, three methods were used:

1. The MSEs on the training data for all polynomial transformations are shown in table 2. As the degree increases, the model becomes better trained, hence the highest degree=5 performs the best on the training data.

2. After splitting the training data 50/50 into a new training and a validation set, the MSEs on this validation set for all degrees are shown in table 2. The linear model provides the lowest error.

3. Performing the 10-fold cross-validation on the training set resulted in the linear model being the optimal model, as its MSE was the lowest.

Forward Selection			Backward Selection		
Features	MSE-Train	MSE-Test	Features	MSE-Train	MSE-Test
1	0.209	0.185	13	0.0	0.037
2	0.072	0.055	12	0.0	0.037
3	0.048	0.092	11	0.0	0.037
4	0.032	0.055	10	0.0	0.037
5	0.008	0.055	9	0.008	0.037
6	0.016	0.055	8	0.008	0.037
7	0.016	0.0	7	0.008	0.055
8	0.008	0.0	6	0.008	0.055
9	0.008	0.037	5	0.016	0.055
10	0.0	0.037	4	0.040	0.074
11	0.0	0.037	3	0.072	0.111
12	0.0	0.037	2	0.112	0.129
13	0.0	0.037	1	0.346	0.481

Table 4: MSEs for each number of features for forward and backward greedy selection. The optimal number of features w.r.t. the test MSE for forward selection is 7 and for backward selection it's 3 features.

Evaluating the test set with the respectivly determined optimal degrees, shows that the methods ***(2)*** & ***(3)*** are good ways of choosing the optimal degree, since the test MSE for the linear model is the lowest of all degrees. The optimal degree determined by method ***(1)*** provides the lowest training MSE, but the highest test MSE ($\sim 9.42 \times 10^{12}$). This is a classical sign of over-fitting as the model is too complex and thus perfoms poorly.

The advantage of methods ***(2)*** & ***(3)*** are, that they give a more "unbiased" approximation of the true MSE by using a or several independent validation sets, thus resulting in a more realistic test MSE.

Task: 4: Feature Selection

Subtask 1: Forward Greedy Feature Selection

For forward greedy selection the subset of features that resulted in the least possible test MSE contained the following seven features in the following order: 6 (flavanoids), 0 (alcohol), 2 (ash), 3 (alcalinity of ash), 12 (proline), 8 (proanthocyanins), 9 (color intensity).

Subtask 2: Backward Greedy Feature Selection

For backward greedy selection the subset of features that resulted in the least possible test MSE contained the following 13 features in the following order: 0 (alcohol), 1 (malic acid), 2 (ash), 3 (alcalinity of ash), 4 (magnesium), 5 (total phenols), 6 (flavanoids), 7 (nonflavanoid phenols), 8 (proanthocyanins), 9 (color intensity), 10 (hue), 11 (od280/od3159, 12 (proline).

Subtask 3: Feature Importance

By constraining the selection process to only selecting four features, the forward selection resulted in these: 6 (flavanoids), 0 (alcohol), 2 (ash), 3 (alcalinity of ash). The backward selection resulted in: 1 (malic acid), 10 (hue), 11 (od280/od3159), 12 (proline).

Task: 5: Kernelized SVM

Subtask 1: SVMs With Different Kernels

First the SVMs (*sklearn.smv.SVC*) were instantiated using the different kernels ['linear', 'poly', 'rbf', 'sigmoid'] with the default parameters. The resulting classification accuracies are reported in table 5, with the best classifier w.r.t. the test-accuracy being the SMV using the linear kernel.

The selection of the optimal hyperparameters for each kernel, was realized via cross-validation using the GridSearchCV-function[3]. This function takes a dictionary containing all parameters that shall be tested and returns the set with the lowest k-fold CV-score. Table 5 showcases the results of the model validation including the corresponding classification accuracies, which lead to the conclusion, that the linear-kernel SVM, with a 10% increase in test-performance, gained the most from using the obtained hyperparameters, while the sigmoid- and the rbf-kernel essentially remained unchanged. Since there are no relevant hyperparameters for the linear-kernel SVM, it was omitted in the examination.

Subtask 2: Develop Your Own Kernel

Equation 5 shows the custom kernel that has been constructed as a modified version of the standard linear kernel by exploiting the rule, that any existing valid kernel multiplied by a positive constant is again a valid kernel (1). The constant $c = 8.1$ has been empirically identified as being the near-optimal value and resulted in a

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV.score`

	linear	poly	rbf	sigmoid
\w default parameters				
train-accuracy	96.1%	99.5%	98.6%	89.4%
test-accuracy	75.9%	69.9%	74.5%	71.3%
\w optimized parameters				
train-accuracy	-	99.9%	98.7%	89.4%
test-accuracy	-	78.2%	74.5%	71.3%
hyperparameters				
coef0	-	7	-	0
gamma	-	-	1	1
degree	-	2	-	-

Table 5: Results of classification accuracies for the selected kernels, before and after model selection with the chosen hyperparameters. The polynomial kernel with the selected hyperparameters proved to be the most accurate with a test accuracy of 78.2%.

\w custom kernel	
train-accuracy	99.9%
test-accuracy	78.3%

Table 6: Results of classification accuracies for the custom kernel. Provided a 0.1% improved test accuracy compared to the best kernel from table 5.

marginally improved test-accuracy compared to the kernels from subtask 1 (see table 6).

$$k(x, x') = 8.1 \cdot x^T x' \tag{5}$$

In the code the kernel is explicitly written as $k(x, x') = xx'^T$, which appears to be the outer- instead of the inner product, which wouldn't be symmetric and hence would not be a valid kernel. But since the vectors x and x' are row-vectors of shape $(1, n)$ (with n being the number of features) the product transforms into the inner product and hence the linear kernel, which is of course a valid kernel.

To further prove that this is a valid kernel, the second necessary condition was checked i.e. the semi pos. definiteness of the Gram-matrix k_{ij}. Thus, the eigenvalues of the Gram-matrix were computed and showed that all were non-negative, hence the custom kernel is a valid kernel.

Conclusion

The implementation of efficiently computing the cost during the bold driver GD, in task 1 posed a problem, although 30k iterations were enough to see the convergence to the true values. Another problem was the strange behaviour of the GD on the normalized data that couldn't be explained.

Task 2 went very well, although it took some time to figure out the best feature transformations for the given data.

Task 3 nicely showed the advantage of model-selection techniques and the downsides of over-fitting a model. The unorthodox split lead to large fluctuations in the results, which the CV handled better than other methods.

The chosen training/test sets also had a large impact on the results of task 4 and sometimes showed questionably looking results where the corresponding MSEs had become exactly 0 when using a large number of features, which seemed unrealistic.

Task 5 went well, but the sparse format of the data made it hard to implement anything other than linear kernels. But since the requested improvement, could be achieved, albeit a minor one, the task was successfully solved.

Overall the assignment gave a good overview of the techniques used in regression and classification analysis and the important effects of normalization, over-fitting and model-selection on the results.

References

[1] Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag 2006.