Problem 1.1 (2.5 points)

Consider the following 1D polynomial regression scheme of degree m, which provides a prediction $\hat{y}_{(i)}$ for each scalar input $x_{(i)}$ according to:

$$\hat{y}_{(i)} = w_0 + w_1 x_{(i)} + w_2 x_{(i)}^2 + \cdots + w_m x_{(i)}^m. \tag{1.1.1}$$

We can fit this regressor to a given training dataset $T_{train} = \{x_{(i)}, y_{(i)}\}_{i=1}^n$, by minimizing the loss $J_{\text{LS}}(\mathbf{w})$ via the corresponding least squares problem:

$$\begin{bmatrix} \hat{y}_{(1)} \\ \hat{y}_{(2)} \\ \hat{y}_{(3)} \\ \vdots \\ \hat{y}_{(n)} \end{bmatrix} = \begin{bmatrix} 1 & x_{(1)} & x_{(1)}^2 & \cdots & x_{(1)}^m \\ 1 & x_{(2)} & x_{(2)}^2 & \cdots & x_{(2)}^m \\ 1 & x_{(3)} & x_{(3)}^2 & \cdots & x_{(3)}^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{(n)} & x_{(n)}^2 & \cdots & x_{(n)}^m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \tag{1.1.2}$$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}}\ J_{\text{LS}}(\mathbf{w}) = \underset{\mathbf{w}}{\text{argmin}}\ \|X\mathbf{w} - \mathbf{y}\|^2. \tag{1.1.3}$$

While least squares provides a weight configuration $\hat{\mathbf{w}}$ which minimizes the error on T_{train}, it lacks a dedicated scheme to control the *generalization* to unseen data.

Ridge regression is an extension to least squares which introduces an additional regularization parameter $\lambda \geq 0$. As such, the loss is given by:

$$J_{\text{Ridge}}(\mathbf{w}) = \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|^2. \tag{1.1.4}$$

In the following we will examine the role this *regularization* parameters plays in addressing *over-&underfitting*, by evaluating how least squares and ridge regression perform on the test dataset $T_{test} = \{x_{(i)}, y_{(i)}\}_{i=n+1}^N$.

$\boxed{1.1.1}$ Download the files *regression_train.csv* and *regression_test.csv* from Tuwel and use *pandas*[1] to extract the data. Generate a scatterplot of T_{train} and T_{test} combined and highlight each dataset by using a distinct color.

$\boxed{1.1.2}$ Implement the code to fit the weight vector $\hat{\mathbf{w}}$ to the training dataset T_{train}. I.e., minimize eq. 1.1.3 for an arbitrary degree m. Use the trained regressor and calculate the MSE[2] of your predictions over T_{test} with $m = [0,1,\ldots,10]$. Provide a stem plot showing the values over m. Which degree m leads to the smallest error?
Hint: Use the numpy function pinv for numerically stable matrix inversions.

$\boxed{1.1.3}$ Additonaly compute the predictions $\hat{y}$ of the regressor for inputs in the interval $x \in [0,2]$ for all the configurations of m from task 1.1.2. Depict the predictions as a line plot ontop of the scatter plots from 1.1.1. What do you observe for high and low values of m?

[1]https://pandas.pydata.org/
[2]Mean Squared Error

$\boxed{1.1.4}$ Show that ridge regression has the following analytical solution:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J_{\text{Ridge}}(\mathbf{w}) = \left(X^T X + I\lambda\right)^{-1} X^T \mathbf{y}. \tag{1.1.5}$$

Hint: For that, use the matrix calculus rules:

$$\begin{aligned} \frac{\partial\big((X\mathbf{w}-\mathbf{y})^T(X\mathbf{w}-\mathbf{y})\big)}{\partial \mathbf{w}} &= -2X^T\mathbf{y} + 2X^T X\mathbf{w} \\ \frac{\partial\big(\lambda \mathbf{w}^T\mathbf{w}\big)}{\partial \mathbf{w}} &= 2\lambda\mathbf{w} \end{aligned} \tag{1.1.6}$$

$\boxed{1.1.5}$ Extent the code from 1.1.2 to ridge regression, by implementing the analytical solution given in equation 1.1.5. *Hint: Note, that ridge regression is equivalent to least squares for $\lambda = 0$.*

$\boxed{1.1.6}$ Fit a ridge regressor with $m = 5$ to T_{train} and use the trained model to calculate the MSE on T_{test}. Plot the MSE over λ for $\lambda = [0, 0.1, 1, 5, 10, 100, 500, 1000]$. Which value of λ leads to the smallest error?

$\boxed{1.1.7}$ Compute the predictions $\hat{y}$ of the trained ridge regressor for inputs in the interval $x \in$ [0,2] for all the configurations of λ from task 1.1.6. Depict the predictions as a line plot ontop of the scatter plots from 1.1.1. What do you observe for high and low values of λ?

Problem 1.2 (2.5 points)

In the following we will evaluate the use of linear least squares for a 2D classification problem. As such, we are given datasets $T_{data} = \{\mathbf{x}_{(i)}, y_{(i)}\}_{i=1}^{N}$ consisting of inputs $\mathbf{x} = [x_0, x_1]^T$ and their respective labels $y \in \{-1, 1\}$. For each input $\mathbf{x}_{(i)}$, the classifier generates a soft-label prediction $\tilde{y}_{(i)}$ based on:

$$\tilde{y} = w_0 x_0 + w_1 x_1 + w_2. \tag{1.2.1}$$

Following a least squares approach the model can be trained by minimizing the corresponding loss function

$$J_{\text{LS}}(\mathbf{w}) = \|X\mathbf{w} - \mathbf{y}\|^2 = \|\tilde{\mathbf{y}} - \mathbf{y}\|^2 \tag{1.2.2}$$

and obtaining the respective weight estimate

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}}\, J_{\text{LS}}(\mathbf{w}). \tag{1.2.3}$$

Subsequently we use the following rule to transform the soft-label output $\tilde{y}_{(i)}$ into a hard-label $\hat{y}_{(i)}$:

$$\hat{y}_{(i)} = \begin{cases} 1, & \text{if } \tilde{y}_{(i)} \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{1.2.4}$$

We can then examine the performance of the trained classifier by comparing the predictions $\hat{y}_{(i)}$ with the given labels $y_{(i)}$.

$\boxed{1.2.1}$ Download the files *data_blob_train.csv* and *data_blob_test.csv* from TUWEL and generate a scatterplot of each the training & test dataset. Highlight the respective labels of the samples by color.

$\boxed{1.2.2}$ Fit the classifier to the training data, i.e., obtain the least squares solution for $\hat{\mathbf{w}}$ with X and $\mathbf{y}$ from the training dataset. Provide the accuracy of your trained classifier for T_{train} and T_{test}. *Hint: The accuracy is defined as the percentage of correctly classified samples.*

$\boxed{1.2.3}$ Generate a 2D heatmap of the soft-label output $\tilde{y}$ of your trained classifier in the interval $x_0 \in [-3,3]$ and $x_1 \in [-3,3]$.

$\boxed{1.2.4}$ Find an expression for the decision surface, i.e., the line separating the two classes $\hat{y} = 1$ and $\hat{y} = -1$. Plot this line ontop of the scatter plots from 1.2.1. *Hint: The decision surface is a function of x_0 and $\mathbf{w}$.*

$\boxed{1.2.5}$ Calculate the least squares error of your trained classifier for the samples s_1 & s_2. I.e., obtain the loss $J_{\text{LS}}(\hat{\mathbf{w}})$ between the given label y and the soft-label $\tilde{y}$ for each of the two samples. Which of the samples contributes the higher loss to the optimization?

$$s_1 = \{[-10,10]^T, 1\} \qquad s_2 = \{[0,1.5]^T, -1\} \tag{1.2.5}$$

Now assume, that the hard-labels $\hat{y}$ are used for the loss calculation and compare the results. I.e., again calculate $J_{\mathrm{LS}}(\hat{\mathbf{w}})$ for both samples, but replace $\tilde{y}$ with $\hat{y}$. Why is the optimization of the soft-labels via least squares not ideal?

$\boxed{1.2.6}$ Download the file *data_moon_train.csv* and *data_moon_test.csv* from TUWEL and repeat tasks 1.2.1-1.2.4. Is a linear classifier an appropriate choice for this dataset?

Problem 1.3 (2.5 points)

The california housing dataset was derived from the 1990 US census and consists of the features shown in table 1.3.1. As such, each entry includes data for a distinct californian district with the median house value in 100.000 USD as the target variable.

Input: $\mathbf{x}$							Target: y
HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal

Table 1.3.1: California Housing Data

In the following you will develop a regressor operating on this real world dataset. I.e., you will train a model that can predict the median house price for a given configuration of features. For this we will use the Python machine learning library *scikit-learn*. The dataset is then available via:

from sklearn.datasets import fetch_ california_ housing

Note, that you might want to set the following optional parameters:

as_ frame=True, download_ if_ missing=False

$\boxed{1.3.1}$ Download the california housing dataset and collect the inputs and targets in a *pandas* dataframe. Examine the output of *.describe*() as a sanity check. Additionally print the output of *.corr*(). Which feature has the highest correlation with the target value?

$\boxed{1.3.2}$ Provide a scatter plot of longitude and latitude and highlight the median house value in color. What do you see? Additionally provide separate histograms of latitude and longitude. At which values are the respective maxima?

$\boxed{1.3.3}$ Use the pandas function $.sample(frac = 1, random\_state = 1)$ to shuffle the dataset and split it into training and test data. Use the first 15.000 elements of the shuffled dataset as T_{train} and the remaining as T_{test}.

$\boxed{1.3.4}$ Fit a ridge regressor[3] with $alpha = 1$ to the training dataset T_{train}. Extract the weight vector and the intercept of your trained regressor and provide a list of the features sorted by their weights. Which features have a positive/negative impact on the house price? *Hint: The coefficients and the intercept of the ridge regressor can be accessed via .coef_ and intercept_ .*

$\boxed{1.3.5}$ Compute the MAE[4] in USD between the predictions $\hat{y}$ of your model and the targets y for T_{train} and T_{test} separately. Additionally provide histograms of the error $(\hat{y}_i - y_i)$ for both datasets — is your regressor *biased*?

[3]from sklearn.linear_model import Ridge
[4]Mean Absolute Error

$\boxed{1.3.6}$ Generate a scatter plot of your predictions $\hat{y}$ and the respective ground truth values y for the samples in T_{test}. What would the plot look like for a model without any error? *Hint: Either use scatter() from matplotlib or the function jointplot() from the seaborn library.*

$\boxed{1.3.7}$ Fit a random forest regressor[5] to the training dataset and repeat tasks 1.3.5 and 1.3.6. Does the non-linear regressor outperform the linear one?

$\boxed{1.3.8}$ Finally, perform 10-fold cross-valdidation on you random forest regressor from task 1.3.7 over the whole dataset. I.e., concatenate T_{train} and T_{test}, run *cross_val_score*[6] and calculate *mean* and *standard deviation* of the reported score values.

[5] sklearn.ensemble.RandomForestRegressor(n_estimators=50)

[6] sklearn.model_selection.cross_val_score