Problem 2.1 (25 %)

In the following, we will examine two variants of gradient descent that vary in regards to how they traverse the training data set $T_{train} = \{\mathbf{x}_{(i)}, y_{(i)}\}_{i=0}^{N-1}$ troughout the iterations $k = [1, \dots, K]$.

Batch-Gradient-Descent: In BGD the loss is evaluated over the complete training set, before the weights are updated. As such, the update equation is given by (2.1.1), where $L(\mathbf{w}, \hat{y}_{(i)}, y_{(i)})$ denotes the sample-wise loss function.

$$\hat{\mathbf{w}}^{(k)} \leftarrow \hat{\mathbf{w}}^{(k-1)} - \alpha \cdot \nabla_{\mathbf{w}} J(\hat{\mathbf{w}}^{(k-1)}) \qquad J(\mathbf{w}) = \frac{1}{N} \sum_{i=0}^{N-1} L(\mathbf{w}, \hat{y}_{(i)}, y_{(i)}) \tag{2.1.1}$$

Stochastic-Gradient-Descent: Here, the gradients are evaluated on a per-sample basis. Correspondingly, the weights are updated immediately. See Eq. (2.1.2) — where $i = (k-1) \mod N$.

$$\hat{\mathbf{w}}^{(k)} \leftarrow \hat{\mathbf{w}}^{(k-1)} - \alpha \cdot \nabla_{\mathbf{w}} L(\hat{\mathbf{w}}^{(k-1)}, \hat{y}_{(i)}, y_{(i)}) \tag{2.1.2}$$

For the following task, consider a single neuron with sigmoid activation in a binary classification setup with classes $\mathcal{C}_0, \mathcal{C}_1 = \mathcal{C}_0^{\complement}$. Given the input vector $\mathbf{x} = [x_0, x_1]^T$, the output for sample s is computed through a sigmoid function $\sigma(\cdot)$ as:

$$\hat{y} = \sigma\big(\mathbf{w}^T\mathbf{x}\big) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}. \tag{2.1.3}$$

Moreover, we select binary-crossentropy with l_2 regularization (2.1.4) as the loss function, where the targets $y_{(i)}$ are given by the class probabilities according to $y_{(i)} = P\{s_{(i)} \in \mathcal{C}_1\}$.

$$L(\mathbf{w}, \hat{y}_{(i)}, y_{(i)}) = -\big(y_{(i)} \cdot \log(\hat{y}_{(i)}) + (1 - y_{(i)}) \cdot \log(1 - \hat{y}_{(i)})\big) + \lambda \|\mathbf{w}\|^2 \tag{2.1.4}$$

$\boxed{2.1.1}$ Download the dataset *data.csv* and generate a surface plot of the loss function $J(\mathbf{w})$, with $L(\mathbf{w})$ from Eq. (2.1.4). Briefly compare the plots for $\lambda = 0$ and $\lambda = 10^{-2}$. *Hint: Consider using logarithmic scale.*

$\boxed{2.1.2}$ Derive an analytical expression for the gradient $\nabla_{\mathbf{w}} L(\mathbf{w}, \hat{y}_{(i)}, y_{(i)})$ of the loss function for binary-crossentropy given in (2.1.4). *Hint: Note, that the derivative of the sigmoid function is given by $\sigma(\cdot)' = \sigma(\cdot)(1 - \sigma(\cdot))$.*

$\boxed{2.1.3}$ Use the result from Task 2.1.2 to implement BGD for the regressor from (2.1.4) and fit it to *data.csv*. Plot $J(\mathbf{w})$ over the iterations $k = [1, \dots, K]$. Additionally provide a 2D scatter plot showcasing the history of the weight estimates $\hat{\mathbf{w}}^{(k)}$ throughout the optimization process. *Hint: Use the following configuration: $K = 1000$, $\alpha = 10^{-1}$, $\lambda = 10^{-2}$, $\hat{\mathbf{w}}^{(0)} = [4, -7]^T$.*

$\boxed{2.1.4}$ Repeat the Task 2.1.3 for SGD using the same configuration. What do you observe when comparing the plots for BGD and SGD? Further, specify the number of *epochs* —

denoting the a full iteration over the training dataset — required for BGD and SGD in this example. Which gradient descent variant would you choose for a large training dataset? Explain your answer.

$\boxed{2.1.5}$ Deep learning frameworks such as *tensorflow* allow for efficient implementation of gradient-based machine learning algorithms, by providing an *automatic differentiation* feature. Study the tutorial in `https://www.tensorflow.org/guide/autodiff` and adapt your code to use the *tf.GradientTape* API. Visualize the optimization process and validate that the results are equivalent to the the manual implementation from Task 2.1.2.

Problem 2.2 (25 %)

SVCs[1] are obtained by adjusting the loss function of the perceptron to include a buffer zone between the separating hyperplane $\mathbf{x}_{(i)}^T\mathbf{w} + b = 0$ and the symmetric translated versions of itself given by $\mathbf{x}_{(i)}^T\mathbf{w} + b = \pm 1$. This leads to the following formulation of the classification objective for a given dataset $T = \{\mathbf{x}_{(i)}, y_{(i)}\}_{i=1}^N$:

$$\begin{aligned} \mathbf{x}_{(i)}^T\mathbf{w} + b &\geq 1 \qquad && \text{if } y_{(i)} = 1 \\ \mathbf{x}_{(i)}^T\mathbf{w} + b &\leq -1 \qquad && \text{if } y_{(i)} = -1. \end{aligned} \tag{2.2.1}$$

This criterion is reflected in the SVC loss function:

$$J(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N} \max\big(0{,}1 - y_{(i)}\big(\mathbf{x}_{(i)}^T\mathbf{w} + b\big)\big) + \lambda\|\mathbf{w}\|^2 \tag{2.2.2}$$

where $\lambda \geq 0$ is a hyperparameter. The term $\lambda\|\mathbf{w}\|^2$ in Eq. (2.2.2) encodes the objective, that the margin around the hyperplane should be maximized.

SVCs are of particular interest because they offer an efficient extension to a non-linear regime. By using the kernel trick, input samples can be mapped to a higher dimension $d > d'$, in which the data is again linearly separable.

$$\phi : \mathbb{R}^d \to \mathbb{R}^{d'}, \quad \mathbf{x}_{(i)} \mapsto \phi(\mathbf{x}_{(i)}) \tag{2.2.3}$$

$\boxed{2.2.1}$ Visualize the SVC loss function from (2.2.2) for a single sample over the soft label predictions $\hat{y} = \mathbf{x}^T\mathbf{w} + b$ for the two cases $y = \pm 1$. How does the SVC objective differ from the Perceptron introduced in Problem 1.4?

$\boxed{2.2.2}$ Download the file *blobs.csv* from TUWEL and generate a scatter plot that highlights the two classes in color. Fit a SVC[2] with a linear kernel and $C = 1$ to the dataset and extract the weights $\mathbf{w}$ and the bias b from the trained model.

$\boxed{2.2.3}$ Plot the separating hyperplane as well as the two equidistant translations running through the support vectors ontop of the scatter plot from Task 2.2.2. Calculate the distance of the margin of the separating hyperplane. *Hint: The margin distance is a function of* $\mathbf{w}$ *and can be derived from the geometry of the buffer area bounded by* $\mathbf{x}_{(i)}^T\mathbf{w} + b = \pm 1$.

$\boxed{2.2.4}$ Repeat Tasks 2.2.2 and 2.2.3 for the dataset *circles.csv*. Is this dataset lineraly separable in the given space?

[1]Support Vector Classifiers

[2]sklearn.svm.SVC(C=1, kernel='linear')

$\boxed{2.2.5}$ Find a transformation ϕ so that the *circles.csv* dataset is linearly separable in the new space $\mathbf{x}'$.

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3, \quad \phi(\mathbf{x}) = \mathbf{x}' = [x_0, x_1, x_2']^T \tag{2.2.4}$$

Repeat the training of the SVC on the transfomed dataset $\mathbf{x}'$ using a linear kernel and $C = 100$. Provide a 2D scatterplot of the transformed data, showcasing only x_0 and x_2'. Additionly, obtain the hyperplane and the two equidistant translations of your trained model and plot them on top of the scatterplot. *Hint: Set $x_1 = 0$ for the visual representation of the hyperplane.*

$\boxed{2.2.6}$ Now fit a SVC with a polynomial kernel[3] of degree 2 to the original 2D *circles.csv* dataset and plot a heatmap of the predicted labels $\hat{y}$ for the interval $x_0 \in [-2, 2]$ and $x_1 \in [-2, 2]$. Is the separating hyperplane a linear function of $\mathbf{x}$?

$\boxed{2.2.7}$ In the dual formulation of the SVC loss function, the input data $\mathbf{x}$ only enters in the form of the inner product $\langle \mathbf{x}_{(i)}, \mathbf{x}_{(j)} \rangle$ between two samples i and j. As such, the transformation $\phi(\mathbf{x})$ is implicitly defined via the kernel function k:

$$k\big(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}\big) = \big\langle \phi\big(\mathbf{x}_{(i)}\big), \phi\big(\mathbf{x}_{(j)}\big) \big\rangle. \tag{2.2.5}$$

The polynomial kernel of degree 2 used in Task 2.2.6 is for instance given by:

$$k_{\text{poly},d=2}\big(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}\big) = \big(\mathbf{x}_{(i)}^\top \mathbf{x}_{(j)} + 1\big)^2. \tag{2.2.6}$$

Find an expression for the transformation $\mathbf{x}' = \phi(\mathbf{x})$ that is implicitly applied to $\mathbf{x} = [x_0, x_1]^T$ via the polynomial kernel in Equation (2.2.6). What is the dimension d' after the transformation?

[3] sklearn.svm.SVC(C=1, kernel="poly", degree=2)

Problem 2.3 (25 %)

Let us revisit *Ridge Regression* from Exercise 1.1, with the loss function given by:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \frac{1}{2}\lambda\|\mathbf{w}\|^2. \tag{2.3.1}$$

Here, we used the pseudoinverse to compute the weight vector $\hat{\mathbf{w}}$ which allows us to obtain predictions for new data samples $\mathbf{X}'$ according to:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{d\times d}\right)^{-1}\mathbf{X}^T\mathbf{y} \qquad \hat{\mathbf{y}} = \mathbf{X}'\hat{\mathbf{w}} = \mathbf{X}'\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{d\times d}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.3.2}$$

In the following we will derive an adapted version of *Ridge Regression*, namely *Kernel Ridge*, which utilizes the Kernel trick to obtain a non-linear variant of the regression scheme. Equivalently to the SVM, we apply the transformation $\mathbf{x}_{(i)} \to \boldsymbol{\phi}_{(i)} = \boldsymbol{\phi}\left(\mathbf{x}_{(i)}\right)$, which maps the input from $\mathcal{R}^d$ to $\mathcal{R}^{d'}$.

The essence of the Kernel trick is to not explicitly compute the transformed inputs $\boldsymbol{\phi}_{(i)}$, but to derive a formulation where the mapping $\boldsymbol{\phi}(\cdot)$ is implicitly defined via the kernel $k\left(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}\right) = \left\langle \phi\left(\mathbf{x}_{(i)}\right), \phi\left(\mathbf{x}_{(j)}\right)\right\rangle$.

$\boxed{2.3.1}$ Show that the inverse in (2.3.2) can equivalently be applied in the data domain with dimension n instead of the feature domain with dimension d. For that, prove that the following matrix identity holds true:

$$\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{d\times d}\right)^{-1}\mathbf{X}^T = \mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_{n\times n}\right)^{-1}. \tag{2.3.3}$$

$\boxed{2.3.2}$ Now, use the above matrix identity to obtain a variant of Equation 2.3.2, where the input vectors $\mathbf{x}_{(i)}$ in $\mathbf{X}$ only enter via the inner product. Further, implement the corresponding estimator in Python for arbitrary degrees m of the polynomial kernel:

$$k\left(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}\right) = \left(\mathbf{x}_{(i)}^\top\mathbf{x}_{(j)} + 1\right)^m. \tag{2.3.4}$$

$\boxed{2.3.3}$ Fit your regressor to the 1D dataset from Exercise 1.1 using $m \in [1, 2, 3, 10]$ and plot the predictions in the intervall $x \in [0, 2]$. Also make sure to analyse the influence of different regularization parameters.

$\boxed{2.3.4}$ Repeat Task 2.3.3 with the RBF kernel given by $k\left(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}\right) = e^{-\frac{1}{2l_s^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2}$. Briefly discuss your interpretation of the role of the kernel parameter l_s.

$\boxed{2.3.5}$ Derive an analytical expression of the mapping $\mathbf{x}_i \to \boldsymbol{\phi}_i = \boldsymbol{\phi}(\mathbf{x}_i)$ implied by the RBF kernel for the scalar case $d = 1$. What is the dimension d' after the transformation? *Hint: Use the series expansion of the exponential function* $e^x = \sum_{i=0}^{\infty} \frac{x^n}{n!}$.

Problem 2.4 (25 %)

Most machine learning algorithms are based on the assumption that the dataset at hand is balanced. I.e., that the targets y of the samples are equally distributed. In the following, we will examine how common evaluation metrics such as *accuracy* are misleading, whenever this assumption is not fulfilled.

$\boxed{2.4.1}$ Download the files *imbalanced_ train.csv* and *imbalanced_ test.csv* from TUWEL and extract T_{train} and T_{test} using *pandas*. Addionally, evaluate the class distribution by plotting histograms of y for each the training and test dataset.

$\boxed{2.4.2}$ Fit a support vector classifier[4] with an RBF kernel and a regularization parameter of $C = 0.01$ to T_{train}. What is the accuracy of your model on T_{test}?

$\boxed{2.4.3}$ Calculate the confusion matrix — see Table 2.4.1 — of your classifier on T_{test}. If possible, provide the values for *Precision*, *Recall* and *F1-Score*. What do you observe?
Hint: The Precision is given by $\frac{TP}{TP+FP}$*, while the Recall is defined as* $\frac{TP}{TP+FN}$*. We denote the harmonic mean of both measures as the F1-Score.*

	$\hat{y} = 0$	$\hat{y} = 1$
$y = 0$	True Negatives (TN)	False Positives (FP)
$y = 1$	False Negatives (FN)	True Positives (TP)

Table 2.4.1: Confusion Matrix

$\boxed{2.4.4}$ *Resampling* is a commonly used technique to tackle class imbalances in the training dataset. In *undersampling* we randomlly select samples (without replacement) from the majority class until we obtain a balanced problem. Implement *undersampling* for T_{train} and repeat Tasks 2.4.2 and 2.4.3.
Hint: Note, that we train on all samples from the minority class in undersampling.

$\boxed{2.4.5}$ *Oversampling* refers to a scheme, where we sample (with replacement) from the minority class until the training dataset is equally split. Implement *oversampling* for T_{train} and again repeat Tasks 2.4.2 and 2.4.3. Which of the two *resampling* variants achieves the highest *F1-Score*?
Hint: Note, that we train on all samples from the majority class in oversampling.

[4]sklearn.svm.SVC()