Machine Learning 1, Summer Term 2025
Homework 1
Linear and Logistic Regression. Gradient Descent.

Thomas Wedenig
thomas.wedenig@tugraz.at

Tutor:	Marharyta Papakina, marharyta.papakina@student.tugraz.at
Points to achieve:	25 Points
Deadline:	09.04.2025 23:59 (strict, no late submissions allowed)
Hand-in procedure:	Submit **all Python files and a report (PDF)** to the TeachCenter. Do not rename the Python files. Do not zip them. Do not upload the `data` and `plots` folders.
Plagiarism:	If detected, 0 points on the entire assignment sheet for all parties involved. If this happens twice, we will grade the group with "Ungültig aufgrund von Täuschung"
Course info:	TeachCenter, https://tc.tugraz.at/main/course/view.php?id=1648

Contents

General remarks

Your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed.)
- The quality of your plots (Is everything clearly readable/interpretable? Are axes labeled? …)
- Your submission must run with Python 3.11.5 and the package versions listed in `requirements.txt`.
  - For example, after running `conda activate ml1`, run `pip install -r requirements.txt` to install the correct package versions. If you have not set up your `ml1` environment yet, refer to the notebook of the first session.

Since we run automated tests, it is crucial that you keep the following in mind:

- **Do not add any additional `import` statements** anywhere in the code.
- **Do not modify the function signatures** of the skeleton functions
  - i.e., do not edit the function names and inputs

Scikit-Learn

- In this assignment, we will use an implementation of Logistic Regression from scikit-learn. The documentation for this is available at the scikit-learn website.
- For this class (and all scikit-learn model implementations), calling the `fit` method trains the model, calling the `predict` method with the training or testing data set gives the predictions for that data set (which you can use to calculate the training and testing errors), and calling the `score` method with the training or testing data set calculates the mean accuracy for that data set.

1 Linear Regression [15 points]

1.1 Univariate Linear Regression [6 points]

Tasks:

1. In univariate regression, we only have a single feature $x \in \mathbb{R}$ from which we wish to predict a target $y \in \mathbb{R}$. Assume we are using an linear[1] model $f_{\boldsymbol{\theta}}(x) = b + wx$, with $\boldsymbol{\theta} = (b, w)^\top \in \mathbb{R}^2$, and we have access to a dataset of N many feature-target pairs, i.e., $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$. We use the *mean-squared error* (MSE) as our (univariate) *loss function* $\mathcal{L}_U$:

$$\mathcal{L}_U(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(f_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)} \right)^2 \tag{1}$$

   We wish to find a global minimizer $\boldsymbol{\theta}^*$, i.e.,

$$\boldsymbol{\theta}^* = (b^*, w^*)^\top \in \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}_U(\boldsymbol{\theta}) \tag{2}$$

   Derive a *closed-form analytical solution* for w^* and b^*. (Find the *partial* derivatives of the loss function w.r.t. w and b, set both of them to zero, and express w^* and b^*. Include *all steps* of your derivation in the report.) Your expression for w^* and b^* should include **sums – use the loss function as given in this exercise sheet, without transformations**.

2. You are given data from a smartwatch representing the values for 100 subjects (rows) and 8 different variables of interest (columns): `hours_sleep`, `hours_work`, `average_pulse`, `max_pulse`, `exercise_duration`, `exercise_intensity`, `fitness_level`, `calories_burned`.

   We will now investigate if some of the variables have a linear relationship between them, and will also use the Pearson correlation coefficient to quantify this relationship. The Pearson correlation coefficient r is defined as:

$$r(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{N} (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sqrt{\sum_{i=1}^{N} (x^{(i)} - \bar{x})^2} \sqrt{\sum_{i=1}^{N} (y^{(i)} - \bar{y})^2}} \tag{3}$$

   where $\mathbf{x} = (x^{(1)}, \ldots, x^{(N)})^\top$, $\mathbf{y} = (y^{(1)}, \ldots, y^{(N)})^\top$, $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$ and $\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y^{(i)}$.

   Find 3 pairs of variables where there exists a meaningful linear relations between them, i.e., treat one variable as the feature x, the other one as the target y and compute the best linear fit $f_{\boldsymbol{\theta}^*}(x)$ in the least-squares sense (function `fit_univariate_lin_model`). For each of the 3 pairs, (1) calculate the MSE $\mathcal{L}_U(\boldsymbol{\theta}^*)$ (function `univariate_loss`), (2) calculate the Pearson correlation coefficient (function `pearson_coefficient`), and (3) visualize the data using a scatter plot of the variable pair. In the scatter plot, also plot the linear function $f_{\boldsymbol{\theta}^*}$ that you have found (function `plot_scatterplot_and_line`). Include all plots in your report. For all 3 pairs of variables you chose, also report the *correlation coefficient*, the parameter vector $\boldsymbol{\theta}^*$, and the MSE for this parameter vector $\mathcal{L}_U(\boldsymbol{\theta}^*)$.

3. Find 3 different pairs that are *not* linearly dependent. Repeat the steps as in the previous case (line fitting, correlation coefficient, visualization as a scatter plot with a linear function on top). Include the plots in the report. For all 3 pairs of variables you chose, also report the correlation coefficient, the parameter vector $\boldsymbol{\theta}^*$, and the MSE for this parameter vector $\mathcal{L}_U(\boldsymbol{\theta}^*)$.

4. Briefly comment on the scatter plots. What values can the Pearson correlation coefficient assume? How do you interpret them?

5. In general: Given a dataset $\mathcal{D}$ as defined above, we compute the best linear fit (i.e., we compute $\boldsymbol{\theta}^*$). After inspecting $\boldsymbol{\theta}^*$, we find that the model has a very small, positive slope, i.e., $w^* > 0$ and $w^* \approx 0$. Could it still happen that we observe a Pearson correlation coefficient that is close to 1? Explain your reasoning.

[1] Technically, this is an *affine* function if $b \neq 0$, but we will abuse notation and use "linear" and "affine" interchangebly in this assignment sheet.

1.2 Multiple Linear Regression [6 points]

1. In a *multiple regression* task, we want to predict some *target* $y \in \mathbb{R}$, given some *feature vector* $\mathbf{x} = (x_1, \ldots, x_D)^\top \in \mathbb{R}^D$ with $D \geq 1$. This is an extension to the univariate case in Task 1.1, where we fixed $D = 1$. In this case, our model will be defined as

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = b + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_D \cdot x_D \tag{4}$$

with $\boldsymbol{\theta} = (b, w_1, \ldots, w_D)^\top$. Our dataset again consists of N many feature-target pairs, i.e., $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$. Using this dataset, we want to construct a matrix $\mathbf{X}$ such that we can re-write the minimization objective as

$$\boldsymbol{\theta}^* \in \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}_M(\boldsymbol{\theta}) \qquad \text{with} \quad \mathcal{L}_M(\boldsymbol{\theta}) = \frac{1}{N} \|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2 \tag{5}$$

where $\mathbf{y} = (y^{(1)}, \ldots, y^{(N)})^\top$. Write down the definition of the design matrix $\mathbf{X}$ (i.e., what are the entries $\mathbf{X}_{ij}$) and state the dimensions of $\mathbf{X}$. For your choice of $\mathbf{X}$ the multiple regression loss $\mathcal{L}_M$ should be equivalent to the univariate loss $\mathcal{L}_U$ in Equation 1 if you set $D = 1$. Using matrix calculus, derive an analytical expression for the solution $\boldsymbol{\theta}^*$ by computing the gradient of $\mathcal{L}_M$ w.r.t. $\boldsymbol{\theta}$ and setting it to the zero vector, i.e., $\nabla_{\boldsymbol{\theta}} \mathcal{L}_M(\boldsymbol{\theta}^*) = \mathbf{0}$. Your final expression for $\boldsymbol{\theta}^*$ should involve the Moore-Penrose pseudoinverse. Clearly show all steps of your derivation in your report.

2. Implement the construction of the design matrix in `compute_design_matrix`. Implement the computation for $\boldsymbol{\theta}^*$ in the function `fit_multiple_lin_model` by making use of numpy's `pinv` (read: pseudoinverse) function. Test the functionality of your implementation by calling `task_1` in `main.py` with the `use_linalg_formulation` flag set to `True`: When this is the case, you should not use `fit_univariate_lin_model` to compute $\boldsymbol{\theta}^*$, but instead construct the design matrix $\mathbf{X}$ and use `fit_multiple_lin_model` to compute $\boldsymbol{\theta}^*$. The parameter vector computed in this fashion should be equivalent.

3. We will now use the function you have implemented to perform multiple linear regression on the smartwatch dataset. To this end, select a variable pair that showed a linear relationship in Task 1.1. Keep the target variable y fixed, but add 2 *additional* features to the model (i.e., $D = 3$). Use variables that you believe are meaningful predictors for your target. Compute $\boldsymbol{\theta}_M^* \in \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}_M(\boldsymbol{\theta})$ using `fit_multiple_lin_model`. Report the loss $\mathcal{L}_M(\boldsymbol{\theta}_M^*)$. Write down $f_{\boldsymbol{\theta}_M^*}$ as in Equation 4, but with the actual values of the parameter vector $\boldsymbol{\theta}_M^*$ you have found. Let $\boldsymbol{\theta}_U^* \in \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}_U(\boldsymbol{\theta})$ denote the parameter vector found in the univariate setting in Task 1.1. Compare $\mathcal{L}_M(\boldsymbol{\theta}_M^*)$ with $\mathcal{L}_U(\boldsymbol{\theta}_U^*)$.

4. Mathematically prove or refute the following statement: "No matter how the data looks like (targets and features), if $\boldsymbol{\theta}_M^* \in \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}_M(\boldsymbol{\theta})$ and $\boldsymbol{\theta}_U^* \in \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}_U(\boldsymbol{\theta})$ we will *always* have $\mathcal{L}_M(\boldsymbol{\theta}_M^*) \leq \mathcal{L}_U(\boldsymbol{\theta}_U^*)$. That is, adding more features will never make the fit to the data worse."

1.3 Polynomial Regression [3 points]

We can easily use the setup in Task 1.2 to construct functions that are non-linear in the features (but still linear in $\boldsymbol{\theta}$).

1. As in Task 1.1, we will again consider only a single feature x. That is, the data is again given by $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ with $x^{(i)} \in \mathbb{R}$ and $y^{(i)} \in \mathbb{R}$. We want to define a polynomial model of degree K:

$$f_{\boldsymbol{\theta}}(x) = b + w_1 \cdot x + w_2 \cdot x^2 + \cdots + w_K \cdot x^K \tag{6}$$

   with $\boldsymbol{\theta} = (b, w_1, \ldots, w_K)^\top$. Write down the new design matrix $\mathbf{X}$ such that minimizing the loss $\mathcal{L}_M(\boldsymbol{\theta}) = \frac{1}{N}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|_2^2$ yields a polynomial model $f_{\boldsymbol{\theta}}$ (as defined above) that attains a least-squares fit to the target.

2. Revisiting Task 1.1, choose a variable pair that were highly correlated, but whose model could by improved by a polynomial fit (assess this just by looking at the plots).
   In the function `compute_polynomial_design_matrix`, create the new design matrix that contains the polynomial features up to degree K. Pick a reasonable number for K (with $K > 1$), generate $\mathbf{X}$, and compute the least-squares fit $\boldsymbol{\theta}^*$. Implement `plot_scatterplot_and_polynomial` that, similarly to Task 1.1, plots a scatter plot and the polynomial curve on top. Include the plot in your report. Report which K you chose and the MSE you achieved. Again compare the loss with the univariate loss in Task 1.1. Note: Keep in mind that you may face numerical issues if you pick a K that is too large.

3. Consider the relationship between `duration` (feature), and `calories` (target). Assume we are only given $N = 5$ feature-target data points, i.e., $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{5}$. In `main.py`, this dataset is already constructed for you by slicing out the first 5 elements of the full dataset (`x_small` and `y_small`). We will again perform polynomial regression with a degree K polynomial. What's the smallest $K \in \mathbb{N}$ such that $\mathcal{L}_M(\boldsymbol{\theta}_M^*) = 0$ in this case? Report this value of K and include a plot of the data and the degree K polynomial curve on top. Theoretically justify *why* this particular K can fit this data perfectly. Hint: $\mathcal{L}_M(\boldsymbol{\theta}_M^*) = 0$ if and only if $\mathbf{X}\boldsymbol{\theta}_M^* = \mathbf{y}$. What does this imply regarding the dimensions of $\mathbf{X}$?

2 Logistic Regression [6 points]

For this task we will use 3 different data sets (`X-1-data.npy` should be used with `targets-dataset-1.npy`, and `targets-dataset-2.npy`, `X-2-data.npy` with `targets-dataset-3.npy`), and `sklearn` library.

`X-1-data.npy` contains two features – values for x_1, $x_2 \in \{0, 1, ..., 29\}$. `X-2-data.npy` contains two features – values for $x_1 \in [-2, 2]$, and $x_2 \in [-1.33, -0.14]$. The targets for all data sets are 0 or 1. The goal of this exercise is to train a classifier that predicts either Class 0 or Class 1, as shown in Fig.1A-C.

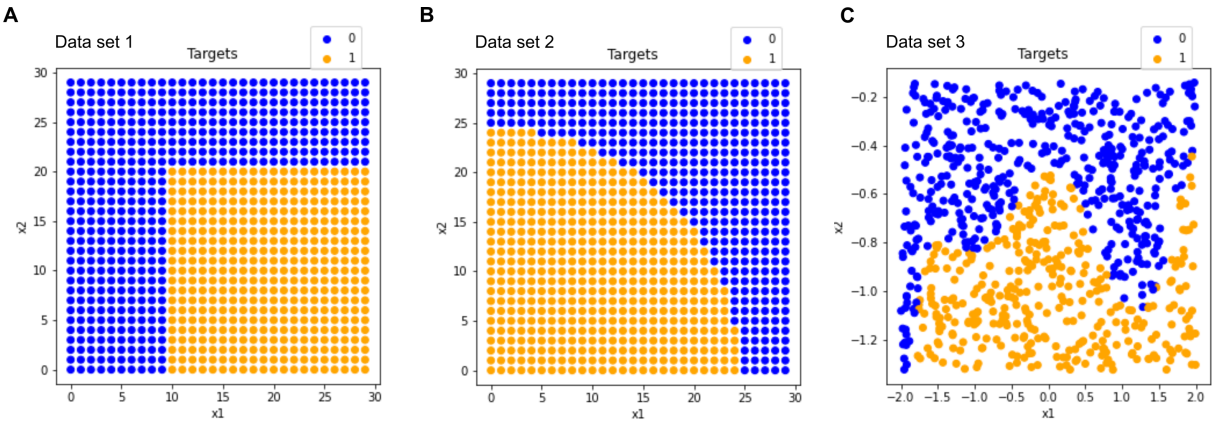

Figure 1: **Targets for: (A) Data set 1; (B) Data set 2; (C) Data set 3.**

Tasks:

1. For each of the three tasks, load the data, then create an appropriate design matrix $\mathbf{X}$. Include any feature that you think is necessary. (There is no need to include a *constant* feature (e.g., a vector consisting of ones), because we will use `LogisticRegression` classifier from the `sklearn` library, for which, by default, the bias term (intercept) is added to the decision function internally.) In the report, for each task, state what design matrix you used, that is, name the features of your design matrices.

2. Split the data set, such that 20% of the data is used for testing. Use the `train_test_split` function (already imported) with the parameter `random_state=0`.

3. The code skeleton then creates a classifier with your parameters (`LogisticRegression` classifier). Fit the model to the data, then calculate accuracy on the train and test set. In addition, using `log_loss` from `sklearn.metrics`, calculate the *cross-entropy* loss on the train and test set. As the cross-entropy loss is a measure of dissimilarity between two distributions, you will first need to calculate the model's output probabilities for each data point in the train and test set.

   Try out different regularization norms ("penalties"). Check the documentation to see what options there are and report your final choice (the one that gives you the best accuracy on the test set). If you are not happy with the final results, and changing the penalty does not help, rethink your design matrices!

   Report the accuracy on the train and test set, the cross-entropy loss on the train and test set, and what penalty you used.

4. For each data set, include 2 plots in the report that you generated using the function `plot_logistic_regression` (one for the train set and one for the test set, respectively). These plots show the decision boundary of the trained classifier and the data points from the respective sets.

5. Report the parameters of your model (including the bias term). Hint: check the classifier's attributes

6. Assume we have trained a logistic regression classifier (binary classes) and are given a test dataset $\mathcal{D}$. Is the following statement correct? "If the classifier predicts the correct class for *all* elements in $\mathcal{D}$ (100% accuracy), then it follows that the cross-entropy loss (w.r.t. $\mathcal{D}$) is 0." Explain your reasoning.

3 Gradient descent [4 points]

Consider the following function $f : \mathbb{R}^2 \to \mathbb{R}$ (called the *Rastrigin* function):

$$f(x, y) = 20 + x^2 + y^2 - 10\left(\cos(2\pi x) + \cos(2\pi y)\right). \tag{7}$$

We want to minimize this function using the *Gradient Descent* algorithm, i.e., we wish to find

$$(x^*, y^*) = \operatorname*{argmin}_{x,y} f(x, y).$$

The global minimum of this function is at the point $(0, 0)$. You will see that it is quite tricky to find the global minimum using gradient descent. However, you should be able to find a good local minimum (x, y) for which $f(x, y) < 1$.

Tasks:

1. In the code, implement the gradient descent algorithm (function `gradient_descent`) with a *decaying learning rate*: In this setting, the learning rate η_t depends on the iteration count t. Specifically, set

   $$\eta_0 \leftarrow \texttt{learning\_rate}, \qquad \eta_t \leftarrow \texttt{lr\_decay} \cdot \eta_{t-1}$$

   where `learning_rate` is the initial learning rate and `lr_decay` $\in (0, 1]$.
2. In the code, implement the Rastrigin function (function `rastrigin`).
3. Using pen and paper, calculate the partial derivatives of f (with respect to x and y), i.e., $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. Implement the expressions you have derived in the code (function `gradient_rastrigin`). Include all steps of your derivation in the report.
4. Choose initial points x_0, y_0 by sampling them from a standard normal distribution, i.e.,

   $$x_0 \sim \mathcal{N}(0, 1),\ y_0 \sim \mathcal{N}(0, 1)$$

   where $\mathcal{N}(0, 1)$ denotes a normal distribution with zero-mean ($\mu = 0$) and unit variance ($\sigma^2 = 1$). To make your code reproducable, we set numpy's random seed to a constant (which will result in the same samples x_0, y_0 in every run of your script). Do not change this random seed (i.e., we will restrict our attention to this one initial point).
5. Write down the definition of the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ for a general (differentiable) function $f(x, y)$. The definition involves a $\lim_{h\to 0}$ operator. We can *approximate* these limits *numerically* by using a small (but finite) value for h instead: In the function `finite_difference_gradient_approx`, implement a *numerical finite difference approximation* to the analytical gradient you have computed in the pen and paper task above. In `task3`, call your functions to compute both the analytical derivatives at (x_0, y_0), as well as the numerical approximations (again at (x_0, y_0)). Report these quantities and briefly comment on the quality of the approximation.
6. Look at the initial plots that are generated. Why is this function challenging to optimize?
7. Choose hyperparameters of the gradient descent algorithm (i.e., number of iterations, learning rate, and learning rate decay) and minimize the Rastrigin function. Report the parameters that you have used. Note that the gradient descent algorithm should have access to the true, analytical gradient. It should *not* use the numerical approximations.
8. The code skeleton already implements `plot_2d_contour` to plot the trajectory of the gradient descent steps on top of the contour plot. Include this plot in your report and briefly comment on it.
9. Generate a plot showing how $f(x_t, y_t)$ evolves over the gradient descent iterations $t \in \{0, \ldots, \texttt{num\_iters}\}$. Include it in the report.
10. Briefly discuss what happens when you set the parameter `lr_decay` to 1.0 (i.e., the learning rate stays the same in all iterations). Why is it very helpful to have a learning rate that decays when optimizing the Rastrigin function?