Machine Learning 1, SS23
Homework 1
Linear and Logistic Regression. Gradient Descent.

Ceca Kraišniković,
ceca@igi.tugraz.at

Tutor:	Moritz Erlacher, moritz.erlacher@student.tugraz.at
Points to achieve:	25 pts
Bonus points:	3.5* pts
Info hour:	will be announced via TeachCenter
Deadline:	21.04.2023 23:55
Hand-in procedure:	Use the **cover sheet** that you can find in the Teach Center. Submit your **python files and a report** to the Teach Center. Do not zip them. Do not upload the data folder.
Submissions after the deadline:	Each missed day brings a (-5) points penalty.
Plagiarism:	If detected, 0 points for all parties involved. If this happens twice, the course graded as failed.
Course info:	TeachCenter, https://tc.tugraz.at/main/course/view.php?id=1648

Contents

1 Linear Regression – Detection of memristor faults [10 points] 2

2 Logistic Regression [7 points] 3

3 Gradient descent [8 points] 4

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 should run with Python 3.5+.

For this assignment, we will use an implementation of Logistic Regression from scikit-learn. The documentation for this is available at the scikit 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 – Detection of memristor faults [10 points]

Information in the human brain is processed by *neurons* and stored in the form of *synaptic weights*. *Synaptic plasticity* — the ability to increase or decrease these weights — is the underlying mechanism responsible for knowledge-based learning. For the implementation of learning in neuro-inspired computing chips, nano-scale hardware devices, so-called *memristors*, have been proposed and fabricated. A memristor is a resistor with programmable (adjustable) resistance, hence a suitable candidate that mimics biological synapses. However, due to a number of non-idealities (e.g., fabrication, operational constraints, limited endurance), memristors, when programmed, often show deviations from the intended behavior. *Stuck memristors* do not change their resistance regardless of the magnitude of resistance change. Other memristors underestimate or overestimate the magnitude of resistance change, and we refer to such faulty behavior as *concordant faults*. Some memristors even produce resistance change in the opposite direction of the intended one, and we refer to such faults as *discordant faults*. In addition, when being programmed, the achieved resistance states are always noisy due to *programming noise*. An illustration of a memristor with an ideal behavior, a discordant fault, a stuck fault, and a concordant fault can be seen in Fig. 1A, B, C, D, respectively.

To detect memristor faults, one can use a linear regression model, interpret its parameters and finally, classify the faults. In this exercise, we will use two linear regression models, and decide which one works better.

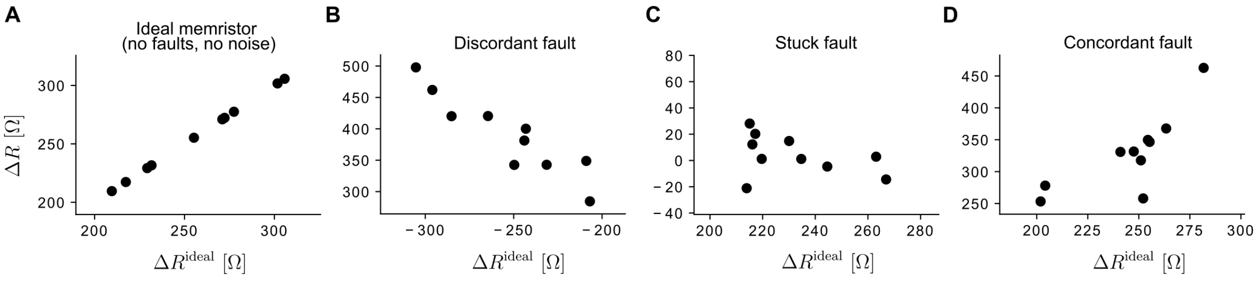

Figure 1: **Examples of memristor faults.** A memristor with (**A**) an ideal behavior (no faults, no noise), (**B**) a discordant fault, (**C**) a stuck fault, (**D**) a concordant fault. Expected resistance changes are given on the $x-$axis ($\Delta R^{\text{ideal}}[\Omega]$), the achieved ones on the $y-$axis ($\Delta R[\Omega]$).

Tasks:

1. **Model 1.** Use a zero-intercept linear regression model, that is, the hypothesis $h_\theta(\Delta R^{\text{ideal}}) = \theta \cdot \Delta R^{\text{ideal}}$, with only one parameter, θ. In this case, the error function to minimize reads as follows:

$$E(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\Delta R_i^{\text{ideal}}) - \Delta R_i)^2,$$

   where m is the number of data points (measurements) we use for regression, more precisely, $(\Delta R_i^{\text{ideal}}, \Delta R_i)$, $i \in 1, ..., m$ are the points used for fitting.

   Derive an expression for θ by minimizing the error function $E(\theta)$, that is, a closed-form analytical solution for θ. (Find the derivative of the error function w.r.t. θ, set it to zero, and express θ. Include all steps of your derivation in the report.) **NOTE:** The derivations must be in the form of sums – use the error function as given in this exercise sheet, without transformations.

2. **Model 2.** Use a linear regression model with intercept, that is, the hypothesis $h_{\theta_0,\theta_1}(\Delta R^{\text{ideal}}) = \theta_0 + \theta_1 \Delta R^{\text{ideal}}$. The error function, in this case, is given by:

$$E(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta_0,\theta_1}(\Delta R_i^{\text{ideal}}) - \Delta R_i)^2,$$

   where m is the number of data points (measurements) we use for regression, and $(\Delta R_i^{\text{ideal}}, \Delta R_i), i \in 1, ..., m$ the points used for fitting.

Derive expressions for θ_0 and θ_1 in the form of sums by minimizing the error function $E(\theta_0, \theta_1)$, that is, a closed-form analytical solution for θ_0 and θ_1. (Find the derivative of the error function w.r.t. θ_0 and θ_1, set them to zero, and express θ_0 and θ_1. Include all steps of your derivation in the report.) **NOTE:** The derivations must be in the form of sums – use the error function as given in this exercise sheet, without transformations.

3. Implement the equation for θ in the code (file *lin_reg_memristors.py*, function *fit_zero_intercept_lin_model*) by using the expressions with sums you derived.

4. Implement the equations for θ_0 and θ_1 in the code (file *lin_reg_memristors.py*, function *fit_lin_model_with_intercept*). **NOTE:** The implementation can be either by using the expressions with sums you derived or the matrix notation (as in the tutorial examples).

5. The data set with measurements for 8 memristors is already loaded (file *main.py*, function *task_1*). It contains a multidimensional array of the size $(8, 10, 2)$ (shape, in the language of Python), representing measurements for 8 memristors; for each memristor there are 10 measurements in the form $(\Delta R_i^{\text{ideal}}, \Delta R_i), i \in 1, ..., 10$, i.e., pairs representing an expected (ideal) resistance change, and an achieved resistance change.

   Implement a function call to fit Model 1 to the data (per memristor), and visualize the data and the fit for each memristor. The code that generates and saves the plot is already implemented.

   Implement a function call to fit Model 2 to the data (per memristor), and visualize the data and the fit for each memristor.

   Include all generated plots in the report. (You are allowed to modify the plots, e.g., make a single plot with all memristors, if you want to do so.)

6. Interpret the parameters of Model 1. Based on the parameter θ, how can we decide if there is a discordant, stuck, or concordant memristor fault? Interpret the parameters of Model 2, and state how would you use them to decide on the type of the memristor fault.

7. Which model would you prefer to use – Model 1 or Model 2? Explain your choice.

8. Finally, implement in the code (using if-statements and thresholds of your choice) how the parameters (of either Model 1 or Model 2, depending on your choice in the previous step) can be used to decide on the memristor fault type. In the report, describe how you did it (or include a screenshot of your code). Report (provide a list with) the memristor id and the fault type assigned to it according to your code output.

Bonus tasks [0.75* point]: (Python-related) Implement test functions with at least two test cases that test the implementation of functions *fit_zero_intercept_lin_model* and *fit_lin_model_with_intercept*. Use Python **assert** command for that. Include the code in the report, or add a screenshot of your code.

2 Logistic Regression [7 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.2A-C.

Tasks:

1. For each of the three tasks, load the data, then create an appropriate design matrix X. Include any feature that you think is necessary. (There is no need to include the zero (dummy) feature, because we will use *LogisticRegression* classifier from the *sklearn* library, for which, by default, the bias term (intercept) is added to the decision function.) In the report, for each task, state what design matrix you used, that is, name the features of your design matrices.

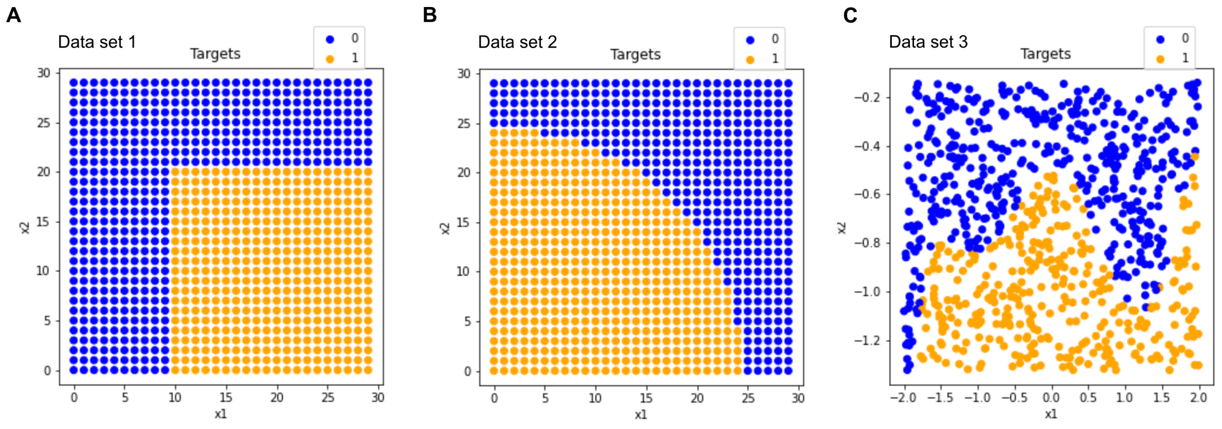

Figure 2: **Targets for: (A) Data set 1; (B) Data set 2; (C) Data set 3.**

2. Split the data set, such that 20% of the data is used for testing. Use the *train_test_split* function (already imported).

3. Create a classifier (*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 loss on the train and test set. (Hint: you will first need to calculate probabilities of predictions on the train and test set.)

   Try out different 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 loss on the train and test set, and what penalty you used.

4. For each data set, include in the report 2 plots that you generated using the function *plot_datapoints* – the first one should show "Predictions on the train set", the second one should be for "Predictions on the test set". (Please include the plots that look similar to those in Fig. 2, with the difference that instead of targets, plotted are the predictions (output of your model)).

5. Report θ^* vector, and also the bias term. Hint: check the Attributes of the classifier.

6. When do we use logistic regression?

7. A classifier could predict everything correctly and achieve 100% accuracy, but it can happen that the loss is not zero. Why?

3 Gradient descent [8 points]

The following function (called the Eggholder function), should be optimized using Gradient Descent algorithm:

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}. \tag{1}$$

The global minimum of this function (on the search domain $-512 \le x, y \le 512$) is the point $(512, 404.2319)$, for which $f(512, 404.2319) = -959.6407$.

Tasks:

1. In the code, implement the gradient descent solver.

2. In the code, implement the cost function (Eq. 1).

3. Derive the expression to calculate partial derivatives (with respect to x and y) and implement it in the code. Include your derivation in the report (derivation for $\frac{\partial f(x,y)}{\partial x}$ and $\frac{\partial f(x,y)}{\partial y}$). Note: $\frac{d}{dx}|u| = \frac{u}{|u|}\frac{du}{dx}$.

4. The search domain for this function is: $-512 \leq x, y \leq 512$. Choose the starting point randomly. (If you get a cost lower than the global minimum reported here, it can happen when your GD solver finds a point that is outside of the search bounds, and it is OK to leave it so, just state that in the report.) Try out different parameters (number of iterations, learning rate) for the gradient descent algorithm, and try to find the minimum of the function. Generate 3 plots showing how the cost changes over iteration (one with slow convergence, one with smooth (and moderate speed) convergence, and one with fast convergence). Include them in the report, and for each plot write which learning rate was used.

5. Why is this function challenging to optimize? Was it (always) possible to find the global minimum?

6. Is the *absolute* function differentiable in all points? What is the meaning of the derivative for the absolute function that we used above, more precisely, of the term $\frac{u}{|u|}$?

7. In the report, specify points for which the gradient of the Eggholder function might be problematic. How can we computationally overcome the problem? Implement it in the code, and describe in the report your approach. In the code, evaluate the gradient function with 2 *problematic* points (x, y). In the report, include also the points that you tested.

Bonus tasks [2.75* point]:

- Gradient Descent algorithm needs a cost function to be specified, and gradients of the cost function with respect to all of its variables. If we would like to have a generic Gradient Descent algorithm, i.e., an algorithm that works for any cost function without defining gradients of it w.r.t. all variables, what could we use? How could we make the function *gradients* in a general form, i.e., how can we approximate gradients? [0.25* points]

- Implement such a generic solver. Use it to check if your gradients are implemented correctly. [2.5* points]