Machine Learning 1, SS21
Homework 1
Linear and Logistic Regression. Gradient Descent.

Ceca Kraisnikovic

Tutor:	Nico Seddiki, seddiki@student.tugraz.at
Points to achieve:	25 pts
Bonus points:	4* pts
Info hour:	will be announced via TeachCenter
Deadline:	23.04.2021 23:55
Hand-in procedure:	Use the **cover sheet** that you can find in Teach Center. Submit your **python files and a colored report** in Teach Center.
Course info:	TeachCenter

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 visible in the print-out? Are axes labeled? . . .)
- Your submission should run with Python 3.5+.

For this assignment, we will be using 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 [13 points]

1.1 Improving detection of peaks in ECG signal [8 points]

ECG is a method for monitoring and recording heart activity, and an example is shown in Fig.1. A device recording the signal works at a certain sampling rate (frequency), and hence the actual signal, plotted as data points, looks as shown in Fig.1B. Peaks of ECG signal can be used, for example, to measure heart rate (average number of beats in a minute, i.e., number of peaks per minute), or the duration of intervals between two peaks that would, in combination with other measurements, enable other estimates of interest (e.g., blood pressure, heart rate variability). Sampling frequency of a device plays an important role in the quality of the signal. The more often we sample, the more data points we get, and hence, the better quality of the signal, but probably also higher costs (e.g., of the device). **The signal shown in Fig.1 was sampled at a sampling rate of 180 Hz.** The task is to improve the detection of peaks, (both amplitude and the (time) precision of occurrences of peaks) by means of linear regression, or more precisely, by fitting lines through data points around each peak.

Given are the following arrays of data: y – amplitudes of ECG signal, and an array with indices of peaks. You will have to create an array representing x data points (those are evenly spaced data points, sampled at frequency $180Hz$). For example, the first peak has an index 24, for which $(x, y) = (0.133, 1.965)$.

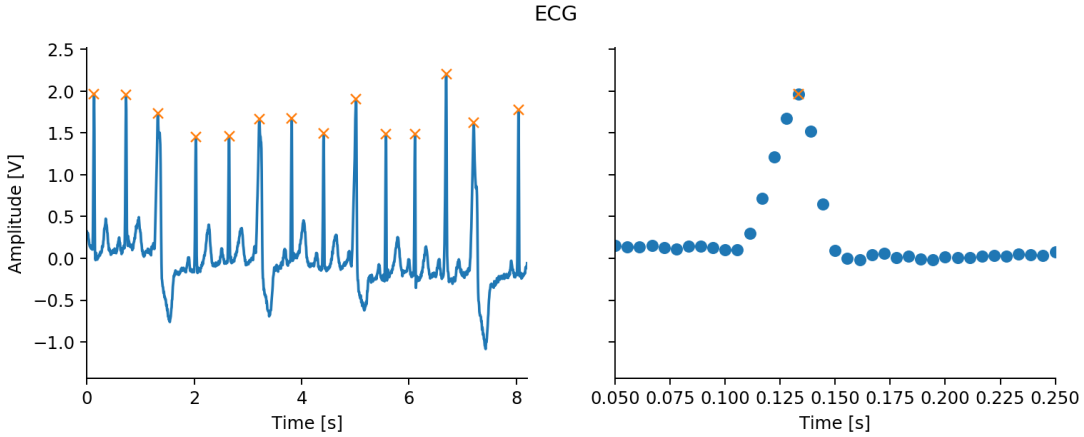

Figure 1: **An example of an ECG signal.** (**A**) ECG signal ($8ms$, blue line), and detected peaks (marked as orange 'x'). (**B**) Scatter plot for the very first peak.

Tasks:

1. Linear function that we want to use is $f(x) = ax + b$. Using the least squares approach, the error function to minimize is then:

$$E(a, b) = \sum_{i=1}^{m} (y_i - f(x_i))^2,$$

where m is the number of data points we use for regression, $(x_i, y_i), i \in 1, ..., m$ are the points that we use for fitting.

(A) Derive expressions for a and b in the form of sums by minimizing the cost function E(a, b).

(B) Derive expressions for a and b using the matrix notation. First, rewrite the cost function as function of θ and use L2-norm for that. Specify how you stack elements in the design matrix X and the vector θ (θ should contain parameters a and b, but in what order?), also specify the dimensions of X, θ and y. Find the gradient of E w.r.t. θ, and do your derivations until you get the expression for the Moore-Penrose pseudoinverse.

HINT: For a function f dependent on vector x, $f(x) = \|Ax - b\|^2$, $\nabla_x f = 2A^T(Ax - b)$.

2. Implement the equations for a and b in the code, either by using the matrix notation or the expressions with sums.

3. Derive the expression and implement the function that finds an intersection point of two lines. Include your derivation in the report (intersection of two lines, e.g., one with parameters a and b, the other one with parameters c and d).

4. Implement the function that finds line coefficients for the line left of the peak, and for the line right of the peak. For each line separately, decide how many points are appropriate to use (for left line, for right line). Also, decide if it is better to include the peak point in the left line, in the right line, in both, or not to include it at all. This function should work for a single peak. Report the number of points used and if the peak was included in the data points for regression lines (for the best performing results).

5. To measure if there is an improvement in the peak, a function in the code is already provided. What point is considered to be an improvement of the peak? Explain both parts of *if*-statement from the code.

6. How does the number of points around the peak that we take for line regression in this case affect the results? What happens (in this task, not in general) with the intersection point if the number of chosen points for regression lines is too high?

7. Report the final score achieved (the percentage of peaks improved).

8. Why is the approach with fitting lines to improve peaks and finding an intersection point, in this case, preferable over, for example, fitting a parabola and finding its peak?

Bonus tasks [3* points]:

- (Python related) Add an **assert** command in the function *fit_line* to check if there are at least two data points given, and an **assert** command in the function *test_fit_line*. Include both lines of the code in the report.

- Implement in the code: in a single figure, plot ECG signal and all peaks, regression lines for each peak and improved peaks. Note that the intersection point of two lines is the improved peak (use a special marker for that, e.g, black circles). In the report, include two figures for time periods $[0.0, 8.2]$ and $[0.05, 0.25]$. This can be easily done by simply setting *xlim* parameter in the plotting function, after everything is plotted.

1.2 Smartwatch data [5 points]

Given is data from from 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*.

Tasks:

1. Find 3 meaningful linear relations between variables: which variable can be predicted by another single one? Fit a line and calculate the correlation coefficient (Pearson coefficient). Visualize the data (chosen *variable 1, variable 2*) by the means of a scatter plot, and plot the best fitting line over it. Include the plots in the report. State also *correlation coefficient, MSE* and θ for all 3 pairs of variables that you chose.

2. What can you say from scatter plots? What from the correlation coefficient (how do you interpret the values of Pearson coefficient)?

3. Find 3 different pairs that are not linearly dependant. Repeat the steps as in the previous case (line fitting, correlation coefficient, scatter plot and a line over it). Include the plots in the report. State also *correlation coefficient, MSE* and θ.

4. Polynomial regression. Choose 2 variables (from step 1) that were highly correlated, but whose model could be improved by polynomial regression. Create a design matrix with a degree n (n is the degree of polynomial and a parameter of the function), and solve the regression problem in the least-squares sense. Calculate the error, and report which n leads to improvement, i.e., a decrease in the error.

5. Multilinear regression. In this case, we want to check if one variable is the outcome of a few other variables. Choose one combination of variables. Create a design matrix, and solve the regression problem in the least-squares sense. Report the model that you get (use *theta* and variable names). For example, if you want to use 3 explanatory variables, it should be in the form $y = ax + by + cz + d$, where y is the dependent variable (outcome variable), x, y, z are independent, explanatory variables, and a, b, c, d are parameters of the model that we were solving for. (Do not spend too much time to find the overall best solution. Report one combination that improves over the model with one explanatory variable.)

2 Logistic Regression [5 points]

For this task we will use *heart_data.npy* and *heart_data_targets.npy*, and *sklearn* library. Our task is to train a classifier that will predict if a patient has a heart disease or not, based on the features given in *heart_data.npy*. Features represent different values, e.g., age, resting blood pressure. Some of them are continuous, some of them discrete, some binary, hence it is always a good idea to normalize the data to zero mean and unit variance, otherwise, the objective function becomes easily influenced by the values with large amplitude.

Tasks:

1. Load the data. Normalize the data using *StandardScaler* from *sklearn* library.

2. Create a classifier (*LogisticRegression* classifier). 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). Fit the data to the model, calculate accuracy on the train and test set. In addition, using *log_loss* from *sklearn.metrics*, calculate loss on the train and test set. (Hint: you will first need to calculate predictions on the train and test set.) Report the accuracy on the train and test set, loss on the train and test set, and what penalty you used.

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

4. When do we use logistic regression?

5. A classifier could predict everything correctly and achieve 100% accuracy, but it can happen that the loss is not zero. Why? (Hint: conclude from the cost function that is used for logistic regression. When is the loss zero?)

3 Gradient descent [7 points]

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

$$f(x, y) = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(cos(2\pi x)+cos(2\pi y))} + e + 20. \quad (1)$$

The global minimum of this function is the point $(0, 0)$, and you should be able to find it by optimizing by the Gradient Descent algorithm.

Tasks:

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

2. In the code, implement the cost function.

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 $\frac{\partial f(x,y)}{\partial x}$).

4. Choose parameters (number of iterations, learning rate for Gradient Descent algorithm, and learning rate decay) and find the minimum of the function. Choose the starting point randomly. Report parameters that you used.
5. Generate a plot showing how the cost changes over iteration. Include it in the report.
6. Why is this function challenging to optimize?
7. Change the parameter *lr_decay* to 1.0, i.e., learning rate stays the same over iteration. Why is, in this case, necessary to have a learning rate that decays?

Bonus task [1* 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 (how can we approximate gradients)?