One of the common methods to analyze a dataset is to perform linear regression. Linear regression, a.k.a. **least squares fitting**, aims to fit a hyperplane to the data points (where the predicted target value for a data point lies in this hyperplane), such that the sum of squares of the differences between the real and the predicted target values over all the data points is minimized. In this lab session, we will work with the linear regression tasks, including univariate linear regression, multivariate linear regression and polynomial linear regression, and a regularization technique for linear regression.

1 Univariate Linear Regression

1.1 Short Theory

Univariate linear regression is a linear regression model with *a single explanatory variable*. In other words, univariate linear regression fits a straight line through the set of n data points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the data points and the fitted line) as small as possible. This model is among the simplest ones in machine learning and statistics.

Suppose there are n training samples (data points) $\underline{x}_i = (1, x_{i1})^T$ and y_i, where $i = 1, 2, \ldots, n$. These samples are assumed to be instantiations of the input random vector $\underline{X} = (1, X_1)^T$ and the output random variable Y, respectively. The function that describes $\underline{x}_i$ and y_i is:

$$y_i = \underline{\theta}^T \underline{x}_i + \epsilon_i = \theta_0 + \theta_1 x_1 + \epsilon_i,$$

where $\underline{\theta} = (\theta_0, \theta_1)^T$ is the parameters' vector and ϵ_i is the error for a tuple $(\underline{x}_i, y_i)$. The goal is to find the equation of the straight line:

$$Y = \theta_0 + \theta_1 X_1$$

that would provide a "best" fit for the data points. Here the "best" will be understood as in the least-squares approach: a line that minimizes the sum of squared residuals of the linear regression model. In other words, θ_0 (the

y-intercept) and θ_1 (the slope) solve the following minimization problem:

$$[\hat{\theta}_0, \hat{\theta}_1] = \arg\min_{\theta_0,\theta_1} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_{i1})^2 \tag{1}$$

By using either calculus (the geometry of inner product spaces) or simply expanding to get a quadratic expression in θ_0 and θ_1, it can be shown that the values of θ_0 and θ_1 that minimize the expression above are

$$\hat{\theta}_1 = \frac{\sum_{i=1}^{n} (x_{1i} - \bar{x_1})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_{1i} - \bar{x_1})^2} = \frac{\mathrm{Cov}[\mathbf{x}_1, \mathbf{y}]}{\mathrm{Var}[\mathbf{x}_1]}, \tag{2}$$

$$\hat{\theta}_0 = \bar{\mathbf{y}} - \hat{\theta}_1 \bar{\mathbf{x}}_1, \tag{3}$$

where $\mathbf{x}_1 = (x_{11}, \ldots, x_{i1}, \ldots, x_{n1})$ and $\mathbf{y} = (y_{11}, \ldots, y_{i1}, \ldots, y_{n1})$ are the row vectors (of dimension n) that contain all sample values of the variables X_1 and Y, respectively. The horizontal bar over a quantity indicates the sample-average of that quantity. A more thorough description of the univariate linear regression theory can be found in any of the following sources:

- The lecture notes and slides.
- Section 3.1, Chapter 3 from the book "Pattern recognition and machine learning" of Bishop and Nasrabadi.
- `https://en.wikipedia.org/wiki/Simple_linear_regression`
- `http://mathworld.wolfram.com/LeastSquaresFitting.html`

1.2 Python Implementation

Often mathematical models of experimental or physiological systems are developed to form the basis of a measurement technique. For example, it is not possible to measure cardiac output directly, but we may be able to infer it from analysis of a mathematical model of respiratory gas exchange. For such models we are often solving an inverse problem of parameter estimation (the cardiac output), where the parameter of interest is embedded (somewhere) within the mathematical model. In these circumstances, it is often advisable to introduce simulated experimental error into the system to test the robustness of the recovery procedure. The following is one of

the simplest possible examples, where the parameters form part of a linear model.

The following activity will lead you through generating some experimental data adding artificial noise and then performing least squares estimation to try to elucidate the underlying model parameters.

1. Generate and plot the data (values in $\mathbf{x}_1$ and $\mathbf{y}$) for a simple straight line of the form $y = \theta_0 + \theta_1 x_1$ where $\theta_0 = 1$ and $\theta_1 = 2$ are constants and $x_1 \in [0, 1]$. You should calculate the value of y at 21 evenly spaced points between 0 and 1.

2. Generate random errors from a normal distribution, with zero mean $\mu = 0$ and standard deviation $\sigma = 0.1$, using Numpy's built-in commands, and add these to each of the y coordinates in step 1. Plot these values as points on the same graph as in step 1. What happens if μ is non-zero?

3. In this step, you will calculate the parameters of the simple linear regression model using the formulas (2), (3). Then, using this model to fit the data generated in step 2, and plot the regression line on the same plot.

2 Multivariate Linear Regression

In this exercise, you will investigate multivariate linear regression using the normal equation, as introduced in Lecture 2.

2.1 Short Theory

Given a set of samples $x_i \in \mathbb{R}^m$, and their corresponding scores $y_i \in \mathbb{R}^1$, with $1 \leq i \leq n$, we want to predict the score y_i from a sample x_i. Each values in x_i represent a feature of the sample, e.g. intelligence score or extroversion score.

In linear regression, we form a hypothesis that y is a linear function of the features. Particularly,

$$y_i = \beta_0 + \beta^1 * x_i^1 + \beta^2 * x_i^2 + ... + \beta^m * x_i^m, \tag{4}$$

where β^i with $0 \leq \beta \leq m$ are the parameters.

From all the samples and scores, we form two matrices, $X \in \mathbb{R}^{n \times (m+1)}$ by stacking all samples into a big matrix. We also add 1 as the last feature of each sample, to account for the intercept term. Analogously we form a matrix $Y \in \mathbb{R}^{n \times 1}$ by stacking all scores together.

The parameter β can be found using the **normal equation**:

$$\beta = \left(X^T X\right)^{-1} X^T Y \tag{5}$$

After estimating β, the score predictions can be calculate by:

$$\mathtt{pred} = X\beta \tag{6}$$

2.2 Python Exercise

This exercise is contained in the file *multivairate_linear_reg.ipynb*. You need to proceed through this file and add some missing code sections.

Step 1: Load data

The *load_dat* funciton is provided for you. This function open a .dat file, read all samples line by line, form and return a big matrix. In order to load the data used in this exercise, you need to call `load_dat` function to load X from `ex1x.dat`, and Y from `ex1y.dat`

Step 2: Preprocess data

For solving linear regression using the **normal equation**, no special data preprocessing is needed. We only need to put an additional 1 to all rows of X (i.e. all the samples) to account for the intercept term. The code for this step is provided for you.

Step 3: Fit the model

Use the **normal equation** to estimate the parameters β. You need to:

- Implement the *pseudo_inverse* function which calculates the pseudo-inverse of a given matrix

- Implement the *sse* function, which calculates the sum of square errors (SSE) between a prediction vector and a reference vector. This function is used to evaluate the model.
- Call the *pseudo_inverse* function you implemented to estimate the parameters β.

Note: you can use the following Numpy functions in your implementation:

- *numpy.matmul*: Calculate multiplication of two matrices
- *numpy.linalg.inv*: Calculate the inverse of a given matrix

You may need to check the online references of these functions for more details.

Step 4: Evaluate the model

After estimating β, you are ready to make predictions and evaluate the model as well as the hypothesis.

- Calculate the predictions according to Eq. (6).
- Calculate the sum of square error of your prediction w.r.t the original scores by calling the equation you implemented in step 3.

Question: What do you think about the prediction, the error and the hypothesis?

Extra step: Fitting on synthetic data

After step 4, you have finished implementing the multivariate linear regression algorithm. Let's apply it to a synthetic data generated with a linear function to see if we can fit the well the data using the algorithm.

The new scores Ys are synthetic scores, generated using a linear function of the samples' features. Applying steps 3 and 4 to X and Ys and evaluate the model/results.

3 Polynomial Regression

In this exercise, you will investigate polynomial regression using least square fitting algorithm as introduced in Lecture 3.

3.1 Short Theory

In the previous exercise, we worked on multivariate linear regression using least square algorithm. We made a hypothesis that our target function is linear with respective to all the variables. This hypothesis does not work well in case we have datasets in which the target function is non-linear with respect to its variables. Polynomials of order higher than 1 are among the common non-linear functions.

Suppose our data has two variables, X_1 and X_2 and our hypothesis function $\hat{y}$ by a polynomial of order 3 on X_1 and order 2 on X_2 and has a multiplicative term X_1X_2, we can first write Y in an explicit form:

$$\hat{y} = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^3 + \beta_4 X_2 + \beta_5 X_2^2 + \beta_6 X_1 X_2. \tag{7}$$

If we consider X_1^2, X_1^3, X_2^2 and X_1X_2 as new variables, our hypothesis function $\hat{y}$ becomes a linear funciton of six variables. The parameter of the hypothesis function has 7 dimensions, including the intercept term: $\beta = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6]^T$. Appying multivariate linear regression algorithm, we can solve for β which is optimal in least square sense.

By generalizing this technique to datasets with higher number of variables and polynomials of higher orders, we can fit solve a number of real regression problems.

3.2 Python Exercise

(Open the file *polynomial_regression.ipynb*) In this exercise, we will work with a real dataset of house prices in Boston. This dataset contains 506 samples, each with 13 features, and the target values in range $5 - 50$ (thounsands dollars). For your reference, the meaning of each feature in this dataset is shown in Fig. 1.

```
- CRIM     per capita crime rate by town
- ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS    proportion of non-retail business acres per town
- CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX      nitric oxides concentration (parts per 10 million)
- RM       average number of rooms per dwelling
- AGE      proportion of owner-occupied units built prior to 1940
- DIS      weighted distances to five Boston employment centres
- RAD      index of accessibility to radial highways
- TAX      full-value property-tax rate per $10,000
- PTRATIO  pupil-teacher ratio by town
- B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT    % lower status of the population
```

Figure 1: Meaning of features in the Boston house dataset

Step 1: Load data

As this dataset is included in sklearn, we simply call the built-in funciton *load_boston* in sklearn to load the house features and prices, denoted as X and y, respectivelly.

Step 2: Split training and testing data

In order to fairly evaluate a machine learning algorithm, a seperated testing set is required. In this step, we split the original dataset into non-overlapping training and testing set. Only the training set will be used during parameter fitting. For convenience, a splitting of ratio 80/20 has been implemented for you.

Step 3: Make hypothesis and create virtual features

Let's hypothesize that the target function is linear with regard to all 13 variables, and contains terms which are:

- second degree of the first variable
- second degrees of eighth variable
- third and second degrees of the eleventh variable

These terms can be considered as virtual features of the data. Generate these virtual features and concatenate to X.

Note:

- Don't forget to add another variable of value 1 to account for the intercept term in the polynomial.
- You can check the *stack* or *hstack* functions of Numpy for concatenating matrices.

In the end, X should have 18 variables including the virtual features and the variable for the intercept term.

Step 4: Fit the model

Use the normal equation to estimate the parameters β. For convenience, the function to calculate the pseudo-inverse has been given. You will need to call the pseudo inverse function to estimate the parameters β

Step 5: Make predictions and evaluate

After estimating the parameter β, we can make predictions on the testing set. You will need to:

- Make predictions from X_{te} using the estimated parameteres β.
- Implement two evaluation functions, namely *Mean Square Error (MSE)* and *Mean Absolute Error (MAE)* and call them to evaluate the learned model.

4 Regularized Regression

In this exercise, you will investigate regularized regression as introduced in Lecture 3.

4.1 Short Theory

If we have a model with a lot of parameters and we allow the parameters to grow arbitrarily large, the model can "overfits" the training data, i.e. brings a very low training error but high testing error. To mitigate this problem, regularization techniques are often employed.

In case of linear regression, we can regularize the model by adding a penalty term on the square values of β. The influence of this penalty term is controlled by a parameter λ.

With X the input features, y the target values, λ the regularization parameter, the close form solution for the regularized linear regression,

$$\beta = \left(X^T X + \lambda * I\right)^{-1} X^T y \tag{8}$$

Similar to the un-regularized linear regression algorith, in this case, the predicted values $\hat{y}$ is calculated by:

$$\hat{y} = X\beta \tag{9}$$

4.2 Python Exercise

Continue with the next blocks in the the file *polynomial_regression.ipynb*.

Step 1: Fitting the model

You will need to:

- Implement the function to calculated the pseudo inverse with regularized term taken into account.
- Use the function you implement to estimate the parameter $\beta_{regularized}$

Step 2: Make prediction and evaluate

After estimating the parameter β, we can make predictions on the testing set. You will need to:

- Make predictions from X_{te} using the estimated parameteres $\beta\_regularized$.
- Call the *Mean Square Error (MSE)* and *Mean Absolute Error (MAE)* functions to evaluate the learned model.