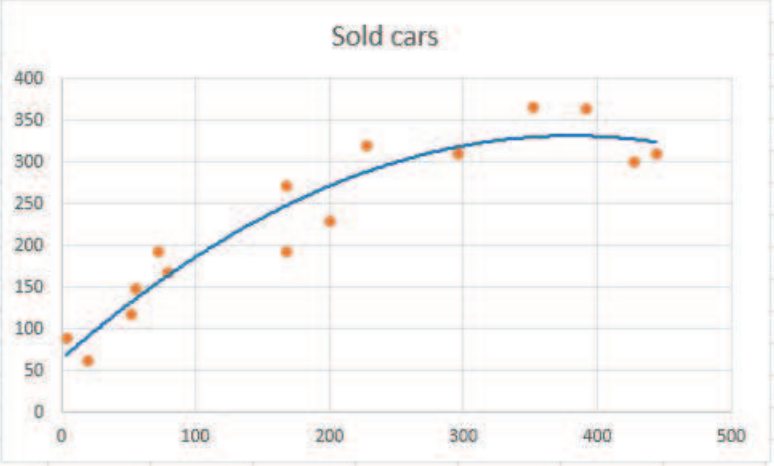
5. LINEAR REGRESSION

Regression is a problem in supervised learning where the output is a real number.
Consider a data set $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.

$x^{(i)}$ = input = independent variable = predictor or explanatory variable $\in \mathbb{R}^d$
$y^{(i)}$ = output = dependent variable = target or response variable $\in \mathbb{R}$

Goal: We want to find a hypothesis $h : \mathbb{R}^d \to \mathbb{R}$ that agrees with the data set D.



https://towardsdatascience.com/how-to-choose-between-a-linear-or-nonlinear-regression-for-your-dataset-e58a568e2a15

A model of *Linear Regression* means that the hypothesis is linear

$$\boxed{h(x; \theta, \theta_0) = \theta^T x + \theta_0} \quad \text{or} \quad \boxed{h(x; \theta, \theta_0) = \theta_1 x_1 + \ldots + \theta_d x_d + \theta_0}$$

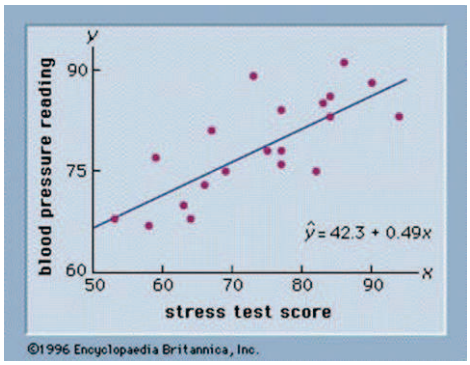
where $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$ are model parameters.

Linear Regression is one of the oldest models and dates to beginning of the 19th century (Gauss, Legendre).

5.1. SIMPLE LINEAR REGRESSION

For simplicity, we will assume $x^{(i)} \in \mathbb{R}$ and in that case the hypothesis we would like to learn is of the form

$$h(x; w, b) = wx + b.$$

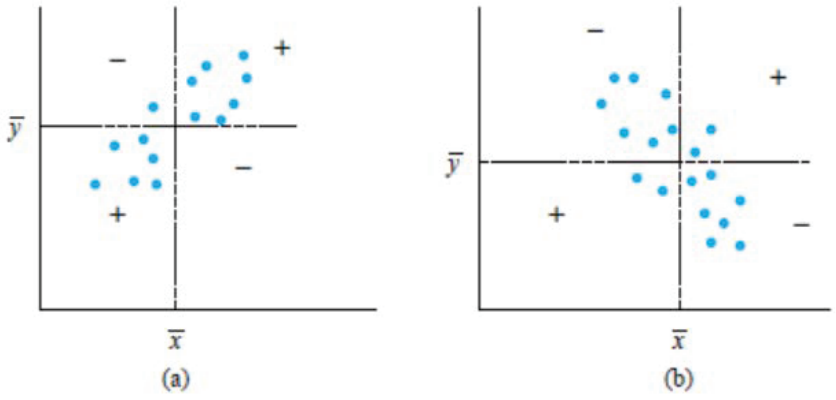


http://abyss.uoregon.edu/ js/glossary/correlation.html

Step 1: *Determine if there is a linear relationship between x and y*

- create a scatter plot
- find sample covariance

$$Cov(x, y) = \frac{1}{n-1} \sum_{i=1}^{n} (x^{(i)} - \bar{x})(y^{(i)} - \bar{y})$$

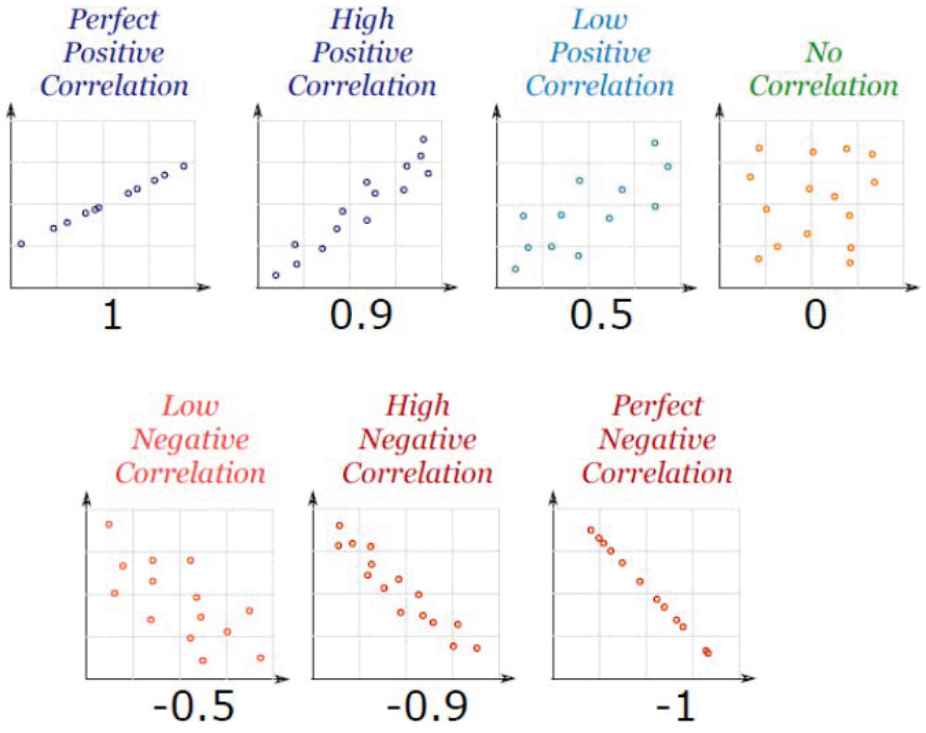


- find sample correlation coefficient

$$r = \frac{Cov(x,y)}{s_x\, s_y} = \frac{\sum(x^{(i)} - \bar{x})(y^{(i)} - \bar{y})}{\sqrt{\sum(x^{(i)} - \bar{x})^2} \cdot \sqrt{\sum(y^{(i)} - \bar{y})^2}}$$

  - r measures the strength of linear relationship between x and y
  - r is independent of the units in which x and y are measured
  - r is between -1 and 1
  - $r = 1$ if all points $(x^{(i)}, y^{(i)})$ lie on a straight line with positive slope, and $r = -1$ if all points $(x^{(i)}, y^{(i)})$ lie on a straight line with negative slope

weak	$-0.5 \leq r \leq 0.5$
moderate	$-0.8 < r < -0.5$ or $0.5 < r < 0.8$
strong	$r \leq -0.8$ or $r \geq 0.8$



https://www.mathsisfun.com/data/correlation.html

Step 2: *Find the hypothesis* $h(x; w, b) = wx + b$

- Given the data set $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, the goal is to find the parameters w and b. We find these parameters by minimizing the loss function which measures the errors we made by predicting $\hat{y}^{(i)} = h(x^{(i)})$ when the true target value is $y^{(i)}$.

  The most common loss functions for regression are:

  - Mean Absolute Error is the mean (or average) of the absolute values of the individual errors and it tells us how big of an error we can expect on average.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}^{(i)} - y^{(i)}|$$

    The main advantage of MAE is that it is in the same unit as the target variable. The disadvantage is that MAE is not a differentiable function.

  - Mean Squared Error is the mean of the individual squared errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2$$

    Note that MSE is differentiable and convex; however, its units do not match that of the original target. The effect of the square term is most apparent with the presence of outliers in the data: while each residual in MAE contributes proportionally to the total error, the error grows quadratically in MSE. This ultimately means that outliers in our data will contribute to much higher total error in the MSE than they would in the MAE and the model will be penalized more for making predictions that differ greatly from the corresponding actual values.

    There is also a probabilistic interpretation of linear regression that justifies using MSE when training data is generated from an underlying linear hypothesis with added Gausian-distributed noise with mean 0 (see [1], pages 11-13).

  - Root Mean Squared Error is the square root of the MSE.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2}$$

    RMSE is differentiable and it is in the same units as the target variable.

- We will consider the MSE loss function

$$L(w, b; D) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^2$$

  Since we are using linear regression model we have

$$\boxed{L(w, b; D) = \frac{1}{n}\sum_{i=1}^{n} (wx^{(i)} + b - y^{(i)})^2}$$

  We will discuss two ways of finding the optimal values w and b for which this loss has a minimum value.

1. Analytical Solution

   Take the first derivatives of L with respect to w and b, set them to 0, and solve for w and b

$$w = \frac{n\sum x^{(i)}y^{(i)} - \left(\sum x^{(i)}\right)\left(\sum y^{(i)}\right)}{n\sum \left(x^{(i)}\right)^2 - \left(\sum x^{(i)}\right)^2}$$

$$b = \bar{y} - w\bar{x}$$

   For details of this derivation in case of multiple linear regression (the case $d \geq 1$), see the APPENDIX at the end of these notes.

2. Gradient Descent

   We initialize w and b and we use the update rule

$$w_{new} = w - \alpha \frac{\partial L}{\partial w}(w, b; D) \tag{1}$$

$$b_{new} = b - \alpha \frac{\partial L}{\partial b}(w, b; D) \tag{2}$$

   We use calculus rules to find derivatives

$$\frac{\partial L}{\partial w}(w, b; D) = \frac{1}{n}\sum_{i=1}^{n} 2\left(wx^{(i)} + b - y^{(i)}\right)x^{(i)} = \frac{2}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)x^{(i)}$$

   and

$$\frac{\partial L}{\partial b}(w, b; D) = \frac{1}{n}\sum_{i=1}^{n} 2\left(wx^{(i)} + b - y^{(i)}\right) = \frac{2}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)$$

   We plug these two derivatives in the above updating rule (1)-(2) and we find w and b that minimize L by

$$\boxed{w_{new} = w - \alpha\frac{2}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})x^{(i)}}$$

$$\boxed{b_{new} = b - \alpha\frac{2}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})}$$

   Note that this updating rule includes computing derivatives on the entire data set D before we take a single step and it is called **batch gradient descent**. A single iteration or update is called an *epoch*.

   If the loss function is convex (as is the case here), batch gradient descent will converge to that one global minimum, but it might take long time if the data set is large and it is also expensive in terms of memory. If the loss function is not convex, we might not even reach the global minimum.

   The following two versions of gradient descent can solve both of these problems.

– **stochastic (or incremental) gradient descent**
In this case the update rule does not wait to use the entire data set, but at each iteration the gradient is calculated by considering the loss at a single random point $(x^{(i)}, y^{(i)})$ which is given by

$$L(w, b; (x^{(i)}, y^{(i)})) = (wx^{(i)} + b - y^{(i)})^2.$$

More precisely, the update rule is

$$w_{new} = w - \alpha \frac{\partial L}{\partial w}(w, b; (x^{(i)}, y^{(i)}))$$
$$b_{new} = b - \alpha \frac{\partial L}{\partial b}(w, b; (x^{(i)}, y^{(i)}))$$

with

$$\frac{\partial L}{\partial w}(w, b; (x^{(i)}, y^{(i)})) = 2\,(\hat{y}^{(i)} - y^{(i)})\,x^{(i)}$$
$$\frac{\partial L}{\partial b}(w, b; (x^{(i)}, y^{(i)})) = 2\,(\hat{y}^{(i)} - y^{(i)})$$

implying

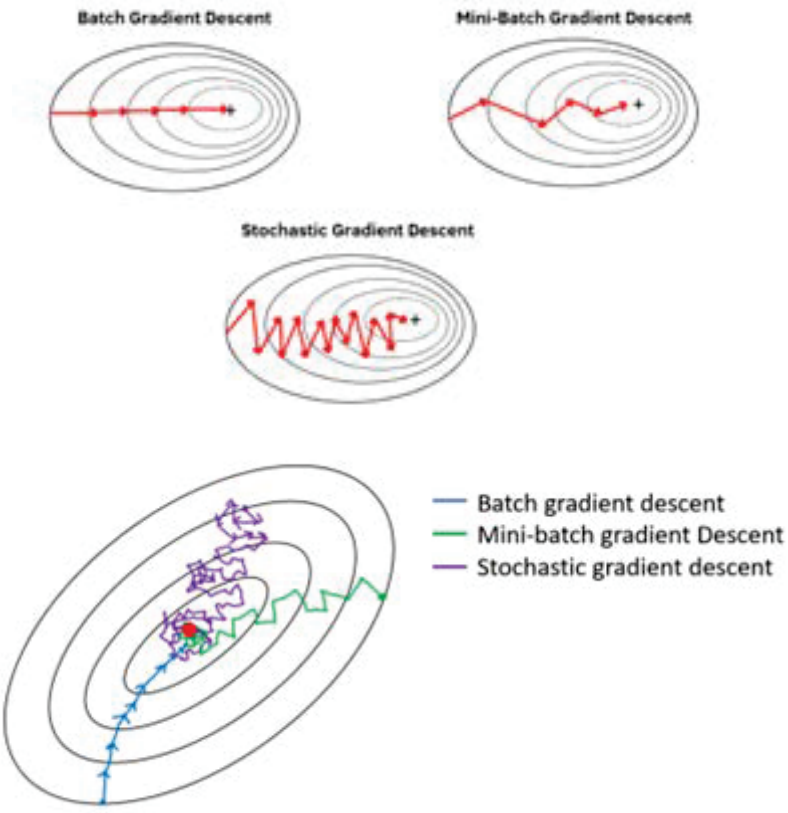
$$\boxed{w_{new} = w - 2\alpha(\hat{y}^{(i)} - y^{(i)})\,x^{(i)}}$$

$$\boxed{b_{new} = b - 2\alpha(\hat{y}^{(i)} - y^{(i)})}$$

Notes:

* The magnitude of the update is proportional to the error $\hat{y}^{(i)} - y^{(i)}$.
* By convention, n of these updates on data points is considered one epoch and usually we just iterate `for i = 1 to n`. However, unlike batch-gradient descent which requires scanning through the entire data set before making a single step, SGD starts making progress immediately.

– **mini-batch gradient descent**
The update rule contains computation of gradients on small random sets of instances called mini-batches.



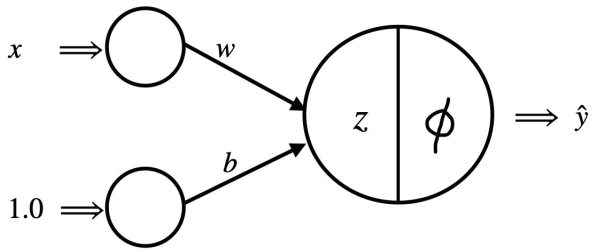
https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

- **Remarks:**

1. The *learning rate* and the *number of iterations* are hyperparameters in the GD method. To find a good learning rate, we can use `grid search`. Regarding the number of iterations, in order not to waste time and memory, set a number of iterations and exit the loop when the gradient becomes sufficiently small. For convex function, the number of iterations to reach the gradient within ε is of order $1/\varepsilon$.
2. In SGD, both the gradient and the loss function decrease on average. They can go up and down even when we are close to the minimum, so the final parameters are good, but not always optimal. Solution to this problem is to gradually reduce the learning rate in time.
   - Theorem: If the loss function is convex and the learning rate $\alpha(t)$ satisfies

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} (\alpha(t))^2 < \infty,$$

     then SGD converges with probability of 1.
3. SGD helps with irregular loss functions since it can jump out of the local minimum.
4. Note that the linear regression model $h(x; w, b) = wx + b$ can be represented using a single neuron with one input x.



   Here, the pre-activation is given by

$$z = wx + b$$

   and the activation function is the identity function

$$\phi(z) = z.$$

   Therefore, $\hat{y} = \phi(z) = z = wx + b$.
5. Scaling and normalization
   Linear regression typically does not require scaling the features, but some other models compare features, which can be meaningless if the features have different scales. However, if the features don't have similar scale, GD might take much longer to converge.
   The following two techniques create features with common ranges.

– *Min-Max Scaling* for the jth feature is defined by

$$\boxed{\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - m_j}{M_j - m_j}}$$

where

$$m_j = \min_i x_j^{(i)} \quad \text{and} \quad M_j = \max_i x_j^{(i)}$$

This scaling creates values within the interval $[0, 1]$ and both end points are achieved.

– *Standard Scaling* for the jth feature is defined by

$$\boxed{\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}}$$

where

$$\mu_j = \sum_i x_j^{(i)} \quad \text{and} \quad \sigma_j = \frac{1}{n}\sum_i (x_j^{(i)} - \mu_j)^2$$

are mean (arithmetic average) and standard deviation across the $j$the feature.

To avoid data leakage, scaling must be done after train-test split. Whatever scaling transformation is applied to the training data must be applied unchanged to the test data as well or the test data will not be sensible inputs for the model.

Summary of algorithms for Linear Regression

algorithm	large n	large d	scaling required	`sklearn`
normal equation	fast	slow	no	N/A
pseudo inverse SVD	fast	slow	no	`LinearRegression`
batch GD	slow	fast	yes	`SGDRegressor`
stochastic GD	fast	fast	yes	`SGDRegressor`
mini-batch GD	fast	fast	yes	`SGDRegressor`

5.2. MULTIPLE LINEAR REGRESSION

In this case the input is multidimensional $x^{(i)} \in \mathbb{R}^d$ with $d > 1$ and the hypothesis is of the form

$$h(x; \theta) = \theta^T x + \theta_0$$

where $\theta \in \mathbb{R}^d$ and θ_0 are parameters determined either by the normal equation or by the gradient descent.

5.3. POLYNOMIAL REGRESSION

In this case the hypothesis has the form of a polynomial of a certain degree. For example, if $x^{(i)} \in \mathbb{R}$ and polynomial is of the third degree, the hypothesis is of the form

$$h(x; \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3.$$

Note that even though this function is a polynomial of degree 3 in variable x, it is a linear function in variables x, x^2, and x^3. Therefore, to perform polynomial regression, we can simply add new (polynomial) features to our data and perform linear regression on the data with those new features.

APPENDIX:

- We show the derivation of optimal parameters in case of multiple linear regression. Assume we have data consisting of n observations described by d explanatory variables and a target variable $\{(x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}, y^{(i)})\}$, $i = 1, \ldots, n$. We are looking for a regression line

$$\theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$

that is the best fit for the data. For convenience, let us introduce $x_0 = 1$ and consider the above line as

$$h(\theta; x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T \cdot x,$$

and the sum of squared errors

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h(\theta; x_i) - y_i)^2 = \frac{1}{n} (X\theta - y)^T (X\theta - y),$$

where $X \in R^{n \times (d+1)}$, $\theta \in R^{d+1}$, and $y \in R^n$. Note that the above expression can be rewritten as

$$\begin{aligned} L(\theta) &= \frac{1}{n} (\theta^T X^T - y^T)(X\theta - y) \\ &= \frac{1}{n} \{\theta^T X^T X \theta - 2\theta^T X^T y + y^T y\}. \end{aligned}$$

We want to find θ that minimizes $L(\theta)$. Note that the gradient of $L(\theta)$ is given by

$$\nabla L(\theta) = \frac{2}{n} X^T X \theta - \frac{2}{n} X^T y,$$

implying that $\nabla L(\theta) = 0$ holds for

$$\frac{2}{n} X^T X \theta - \frac{2}{n} X^T y = 0$$

$$X^T X \theta = X^T y.$$

From here we get

$$\boxed{\theta = (X^T X)^{-1} X^T y} \tag{3}$$

Since $L(\theta)$ is a convex function, the value of θ in (3) is where $L(\theta)$ achieves its minimum. This equation is called the NORMAL EQUATION.

- The computing cost of the inverse of the square matrix $X^T X$, which has size $(d+1) \times (d+1)$, is of order of $(d+1)^3$, and it can be very costly if we have a large number of input variables.

  Since the matrix $X^T X$ might not be invertible, instead pseudo Moore-Penrose inverse is found (this method is based on the singular value decomposition and for a matrix $A = U\Sigma V^T$, its pseudo inverse is $A^+ = V\Sigma^+ U$, where the elements of Σ are inverted and Σ^+ is its transpose). Finding the pseudo-inverse is of order $(d+1)^2$.

  On the other hand, the computational cost of both the normal equation and the pseudo-inverse is linear in n.

- We'll see in future lectures how the problem that the matrix $X^T X$ might not be invertible could be fixed.
- Since computing the optimal coefficients analytically is very costly for a large number of input features, the method of gradient descent is often used instead.
- `Sklearn LinearRegression` is based on the normal equation and pseudo Moore-Penrose inverse.
- `Sklearn SGDRegressor` uses stochastic gradient descent.

Python code: Lecture_5_Linear_Regression.ipynb

Homework 2:

- **Part 1:** Refer to §1 of the code Lecture_5_Linear_Regression.ipynb

  (a) Notice that we used 100 epochs which was waste of time and we could have stopped earlier since after about epoch 55 or so, the loss is not getting lower significantly. Modify that code so that if the percentage change in loss is less than 1%, you exit the iterations.

  (b) The class `MyLinReg` in the above code uses batch gradient descent to find the minimum of the loss function. Modify the original code and use the stochastic gradient descent instead. Iterate over many iterations and see how the RMSE changes. The graph of RMSE for the batch gradient descent is smooth and decreasing as the number of iterations increases. What can you say about the graph of RMSE when the stochastic gradient descent is used?

- **Part 2:** Refer to §3 of the code Lecture_5_Linear_Regression.ipynb

  Try using `sklearn SGDRegressor` class instead of `sklearn LinearRegression`.

  If the input variables are of different scales (here, `TV` and `radio`), scaling those variables typically improves SGD convergence. Read about `sklearn MinMaxScaler` and try to see if using it will give better results.

  https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

- **Part 3:** Import the data file `mtcars.csv`. The goal is to determine two or three continuous numerical variables that can be used to predict mpg (miles per gallon) using multiple linear regression. You can use `sklearn` or custom class; batch GD, SGD, or mini-batch SGD; and scaling.
- **Part 4:** Read about Probabilistic Interpretation of Linear Regression in [1], pages 11-13.

References and Reading Material:

[1] Andrew Ng Lecture Notes on Linear Regression, pages 1-13
[2] Hands-On Machine Learning with Sckit-Learn, Keras & Tensorflow, Geron, pages 111-134