Notes

Machine Learning by Andrew Ng on Coursera

Sparsh Jain

September 25, 2020

Contents

1 Introduction **3**
- 1.1 Supervised Learning . . . 3
  - 1.1.1 Regression Problem . . . 3
  - 1.1.2 Classification Problems . . . 3
- 1.2 Unsupervised Learning . . . 3

2 Linear Regression with One Variable **4**
- 2.1 Notations . . . 4
- 2.2 Supervised Learning . . . 4
- 2.3 Gradient Descent . . . 5
- 2.4 Gradient Descent for Linear Regression . . . 5

3 Linear Algebra **6**
- 3.1 Matrix . . . 6
- 3.2 Vector . . . 6
- 3.3 Addition and Scalar Multiplication . . . 7
- 3.4 Matrix Matrix Multiplication . . . 7
- 3.5 Inverse and Transpose . . . 7

4 Linear Regression with Multiple Variables **9**
- 4.1 Notations . . . 9
- 4.2 Hypothesis . . . 9
- 4.3 Gradient Descent . . . 10
  - 4.3.1 Feature Scaling . . . 10
  - 4.3.2 Mean Normalization . . . 11
  - 4.3.3 Learning Rate . . . 11
- 4.4 Features and Polynomial Regression . . . 11
  - 4.4.1 Features . . . 11
  - 4.4.2 Polynomial Regression . . . 11
- 4.5 Normal Equation . . . 12
  - 4.5.1 Non Invertibility of $X^T X$ . . . 12

5 Octave Tutorial **13**

6 Classification **14**
6.1 Logistic Regression 14
6.2 Decision Boundary 15
6.3 Cost Function 15
6.4 Advanced Optimization: 17

Appendices **19**

Chapter 1

Introduction

Machine learning (task, experience, performance) can be classified into *Supervised* and *Unsupervised* learning.

1.1 Supervised Learning

Supervised learning can be basically classified into *Regression* and *Classification* problems.

1.1.1 Regression Problem

Regression problems work loosely on continuous range of outputs.

1.1.2 Classification Problems

Classification problems work loosely on discrete range of outputs.

1.2 Unsupervised Learning

An example is *Clustering Problem*.

Check Lecture1.pdf for more details.

Chapter 2

Linear Regression with One Variable

2.1 Notations

$$\begin{aligned}
m &= \text{number of training examples} \\
x\text{'s} &= \text{`input' variables / features} \\
y\text{'s} &= \text{`output' variables / `target' variables} \\
(x, y) &= \text{single training example} \\
(x^{(i)}, y^{(i)}) &= i^{th} \text{ example}
\end{aligned}$$

2.2 Supervised Learning

We have a data set (*Training Set*).

Training Set $\rightarrow$ Learning Algorithm $\rightarrow$ h (*hypothesis*, a function $X \rightarrow Y$)

To Represent h

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Cost

$$\underset{\theta_0,\ \theta_1}{\text{minimize}} \frac{1}{2m} \sum_1^m (h_\theta(x) - y)^2$$

Cost Function

Squared Error Cost Function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{1}^{m} (h_\theta(x) - y)^2$$

$$\underset{\theta_0,\ \theta_1}{\text{minimize}}\, J(\theta_0, \theta_1)$$

2.3 Gradient Descent

Finds local optimum:

1. Start with some value
2. Get closer to optimum

Algorithm

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\ \forall j$$

where α = learning rate

Important!

Simultaneous Update!

$$temp_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\ \forall j$$
$$\theta_j := temp_j\ \forall j$$

2.4 Gradient Descent for Linear Regression

Cost function for linear regression is convex!

Batch Gradient Descent: Each step of gradient descent uses all training examples.

Check Lecture2.pdf for more details.

Chapter 3

Linear Algebra

3.1 Matrix

Rectangular array of numbers:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Dimension of the matrix: #rows x #cols (2 x 3)

Elements of the matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$A_{ij} = \text{“}i, j \text{ entry” in the } i^{th} \text{ row, } j^{th} \text{ col}$$

3.2 Vector

An $n \times 1$ matrix.

$$y = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$y_i = i^{th} \text{ element}$$

Note: Uppercase for matrices, lowercase for vectors.

3.3 Addition and Scalar Multiplication

Add/Subtract (element by element) matrices of same dimention only!

Multiply/Divide (all elements) a matrix by scalar!

3.4 Matrix Matrix Multiplication

$m \times n$ matrix multiplied by $n \times o$ matrix gives a $m \times o$ matrix.

Properties

1. Matrix Multiplication is *not* Commutative.
2. Matrix Multiplication is Associative.
3. *Identity Matrix (I):* 1's along diagonal, 0's everywhere else in an $n \times n$ matrix. $AI = IA = A$.

3.5 Inverse and Transpose

Inverse

Only square ($n \times n$) matrices *may* have an inverse.

$$AA^{-1} = A^{-1}A = I$$

.

Matrices that don't have an inverse are *singular* or *degenerate* matrices.

Transpose

Let A be an $m \times n$ matrix and let $B = A^T$, then

$$B_{ij} = A_{ji}$$

Example:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$B = A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Check Lecture3.pdf for more details.

Chapter 4

Linear Regression with Multiple Variables

4.1 Notations

$$\begin{aligned} n &= \text{number of features} \\ x^{(i)} &= \text{input (features) of } i^{th} \text{ training example} \\ x_j^{(i)} &= \text{value of feature } j \text{ of } i^{th} \text{ training example} \end{aligned}$$

4.2 Hypothesis

Previously:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Now:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convinienice, define $x_0 = 1$. So

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \theta^T x$$

4.3 Gradient Descent

$$\begin{aligned} \text{Hypothesis} : h_\theta(x) = \theta^T x \qquad & = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \\ \text{Parameters} : \theta \qquad & = \theta_0, \theta_1, \ldots, \theta_n \\ \text{Cost Function} : J(\theta) = J(\theta_0, \theta_1, \ldots, \theta_n) \qquad & = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \end{aligned}$$

Gradient Descent :

$$\begin{aligned} & \text{Repeat}\{ \\ \theta_j & := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\ & = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \ldots, \theta_n) \\ & = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}) \\ & \}(\text{simultaneously update } \forall j = 0, 1, \ldots, n) \end{aligned}$$

4.3.1 Feature Scaling

Idea: Make sure features are on a similar scale.

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

4.3.2 Mean Normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

General Rule

$$x_i \leftarrow \frac{x_i - \mu_i}{S_i}$$

where

$$\begin{aligned} \mu_i &= \text{average value of } x_i \\ S_i &= \text{range (max - min)} \qquad\qquad\qquad\qquad \textit{or} \\ &= \sigma (\text{standard deviation}) \end{aligned}$$

4.3.3 Learning Rate

$J(\theta)$ should decrease after every iteration. #iterations vary a lot.

Example *Automatic Convergence Test:* Declare convergence if $J(\theta)$ decreases by less than ϵ (say 10^{-3}) in one iteration.

If $J(\theta)$ increases, use smaller α. Too small α means slow convergence.

To choose α, try $\ldots,\ 0.001,\ 0.003,\ 0.01,\ 0.03,\ 0.1,\ 0.3,\ 1,\ \ldots$

4.4 Features and Polynomial Regression

4.4.1 Features

Get an insight in your problem and choose better features (may even combine/separate features).

Ex: size = length $\rightarrow$ breadth.

4.4.2 Polynomial Regression

Ex:

$$\begin{aligned} x_1 &= size \\ x_2 &= size^2 \\ x_3 &= size^3 \end{aligned}$$

4.5 Normal Equation

Solve for θ analytically!

$$
\begin{aligned}
x^{(i)} &= \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} && \in \mathbb{R}^{n+1} \\
X &= \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} && \in \mathbb{R}^{m\times(n+1)} \\
&= \begin{bmatrix} x_0 & x_1 & \dots & x_n \end{bmatrix} && \in \mathbb{R}^{m\times(n+1)} \\
y &= \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} && \in \mathbb{R}^{m} \\
\theta &= (X^T X)^{-1} X^T y
\end{aligned}
$$

Inverse of a matrix grows as $O(n^3)$, use wisely.

4.5.1 Non Invertibility of $X^T X$

Use 'pinv' function in Octave (pseudo-inverse) instead of 'inv' function (inverse).

If $X^T X$ is non-invertible, common causes are

1. Redundant features (linearly dependent)
2. Too many features ($m \leq n$). In this case, delete some features or use *regularization*

Check Lecture4.pdf for more details.

Chapter 5

Octave Tutorial

Check Lecture5.pdf for more details.

Chapter 6

Classification

Classify into categories (binary or multiple).

6.1 Logistic Regression

$$
\begin{aligned}
0 \leq h_\theta(x) &\leq 1 \\
h_\theta(x) &= g(\theta^T x) \\
g(z) &= \frac{1}{1+e^{-z}} \qquad g \text{ is called a } \textit{sigmoid} \text{ function or a } \textit{logistic} \text{ function.} \\
h_\theta(x) &= \frac{1}{1+e^{\theta^T x}}
\end{aligned}
$$

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that $y = 1$ on input x
$h_\theta(x) = P(y = 1|x;\theta)$ = probability that $y = 1$, given x, parameterized by θ

$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1$$

6.2 Decision Boundary

$$
\begin{aligned}
\text{Predict: } y = 1 \qquad & \text{if } h_\theta(x) \geq 0.5 \\
& (\theta^T x \geq 0) \\
\text{Predict: } y = 0 \qquad & \text{if } h_\theta(x) < 0.5 \\
& (\theta^T x < 0) \\
\theta^T x = 0 \qquad & \text{is the } \textit{decision boundary.}
\end{aligned}
$$

Non-linear Decision Boundaries

Use same technique as polynomial regression for features.

6.3 Cost Function

$$
\begin{aligned}
\text{Training Set : } & \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ./dots, (x^{(m)}, y^{(m)})\} \\
x = & \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \\
x_0 = & \ 1 \\
y \in & \ \{0, 1\} \\
h_\theta(x) = & \frac{1}{1 + e^{-\theta^T x}}
\end{aligned}
$$

How to choose parameter θ?

Linear Regression:

$$
\begin{aligned}
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2 \\
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\
\mathrm{Cost}(h_\theta(x), y) &= \frac{1}{2}(h_\theta(x) - y)^2
\end{aligned}
$$

Logistic Regression:

$$
\begin{aligned}
&\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & y = 1 \\ -\log(1 - h_\theta(x)) & y = 0 \end{cases} \\
&\mathrm{Cost}(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y \\
&\mathrm{Cost}(h_\theta(x), y) \to \inf \ \text{ if } y = 0 \text{ and } h_\theta(x) \to 1 \\
&\mathrm{Cost}(h_\theta(x), y) \to \inf \ \text{ if } y = 1 \text{ and } h_\theta(x) \to 0
\end{aligned}
$$

Note: $y = 0$ or 1 always.

$$
\begin{aligned}
\mathrm{Cost}(h_\theta(x), y) &= -y\log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x)) \\
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}\left(h_\theta(x^{(i)}), y^{(i)}\right) \\
&= -\frac{1}{m}\left[\sum_{i=1}^{m}\left(y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right)\right]
\end{aligned}
$$

To fit parameters θ:

$$\underset{\theta}{\mathrm{minimize}}\, J(\theta)$$

To make a prediction given a new x:

$$\text{Output } h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Gradient Descent:

Simultaneously update all θ_j

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Plug in the derivative

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Don't forget feature scaling!

6.4 Advanced Optimization:

Something better than gradient descent:

1. Conjugate Gradient
2. BFGS
3. L-BFGS

Advantages:

1. No need to manually pick α
2. Often faster than gradient descent

Disadvantages:

1. More complex

Use libraries! Beware of bad implementations!

How to use: We first need to provide a function that evaluates the following two functions for a given input value of θ.

1. $J(\theta)$
2. $\frac{\partial}{\partial \theta_j} J(\theta)$

```
% We can write a single function that can return both of these:
function [jVal, gradient] = costFunction(theta)
        jVal = [...code to compute J(theta)...];
        gradient = [...code to compute derivative of J(theta)...];
end
```

Then we can use octave's *fminunc()* optimization algorithm along with the *optimset()* function that creates an object containing the options we want to send to *fminunc()*.

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
        [optTheta, functionVal, exitFlag] = fminunc(@costFunction,
                initialTheta, options);
```

Check Lecture6.pdf for more details.

Appendices