

# Notes

Machine Learning by Andrew Ng on Coursera

Sparsh Jain

November 25, 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Supervised Learning . . . . .	7
1.1.1	Regression Problem . . . . .	7
1.1.2	Classification Problems . . . . .	7
1.2	Unsupervised Learning . . . . .	7
<b>I</b>	<b>Supervised Learning</b>	<b>8</b>
<b>2</b>	<b>Linear Regression with One Variable</b>	<b>9</b>
2.1	Notations . . . . .	9
2.2	Supervised Learning . . . . .	9
2.3	Gradient Descent . . . . .	10
2.4	Gradient Descent for Linear Regression . . . . .	10
<b>3</b>	<b>Linear Algebra</b>	<b>11</b>
3.1	Matrix . . . . .	11
3.2	Vector . . . . .	11
3.3	Addition and Scalar Multiplication . . . . .	12
3.4	Matrix Matrix Multiplication . . . . .	12
3.5	Inverse and Transpose . . . . .	12
<b>4</b>	<b>Linear Regression with Multiple Variables</b>	<b>14</b>
4.1	Notations . . . . .	14
4.2	Hypothesis . . . . .	14
4.3	Gradient Descent . . . . .	15
4.3.1	Feature Scaling . . . . .	15
4.3.2	Mean Normalization . . . . .	16
4.3.3	Learning Rate . . . . .	16
4.4	Features and Polynomial Regression . . . . .	16
4.4.1	Features . . . . .	16
4.4.2	Polynomial Regression . . . . .	16

4.5	Normal Equation . . . . .	17
4.5.1	Non Invertibility of $X^T X$ . . . . .	17
<b>5</b>	<b>Octave Tutorial</b>	<b>18</b>
<b>6</b>	<b>Classification</b>	<b>19</b>
6.1	Logistic Regression . . . . .	19
6.2	Decision Boundary . . . . .	20
6.3	Cost Function . . . . .	20
6.4	Advanced Optimization: . . . . .	22
6.5	Multi-Class Classification . . . . .	23
6.5.1	One vs All . . . . .	23
<b>7</b>	<b>Regularization</b>	<b>24</b>
7.1	Problem of Overfitting . . . . .	24
7.2	Addressing Overfitting . . . . .	24
7.3	Cost Function . . . . .	25
7.4	Regularized Linear Regression . . . . .	25
7.5	Regularized Logistic Regression . . . . .	27
<b>8</b>	<b>Neural Networks</b>	<b>28</b>
8.1	Non-Linear Hypothesis . . . . .	28
8.2	Neural Networks . . . . .	28
8.3	Model Representation . . . . .	28
8.4	Notations . . . . .	29
8.5	Forward Propagation . . . . .	30
8.6	Multi-Class Classification . . . . .	31
<b>9</b>	<b>Back Propagation</b>	<b>32</b>
9.1	Notations . . . . .	32
9.2	Cost Function . . . . .	33
9.3	Backpropagation Algorithm . . . . .	33
9.3.1	Gradient Computation . . . . .	33
9.4	Implementation Note . . . . .	35
9.4.1	Unrolling Parameters . . . . .	35
9.4.2	Learning Algorithm . . . . .	36
9.5	Gradient Checking . . . . .	36
9.5.1	Numerical Estimation of gradients . . . . .	36
9.5.2	Implementation Note: . . . . .	38
9.6	Random Initialization . . . . .	38
9.7	Putting it all together . . . . .	39

<b>10 Applying Machine Learning</b>	<b>41</b>
10.1 Deciding what to try next . . . . .	41
10.1.1 Debugging a Learning Algorithm . . . . .	41
10.1.2 Machine Learning Diagnostics . . . . .	41
10.2 Evaluating your Hypothesis . . . . .	42
10.2.1 The Standard Way . . . . .	42
10.2.2 Train/Test Procedure . . . . .	42
10.3 Model Selection . . . . .	42
10.3.1 Evaluating your hypothesis . . . . .	43
10.4 Bias vs Variance . . . . .	44
10.4.1 Effect of regularization . . . . .	44
10.5 Learning Curves . . . . .	45
10.5.1 High Bias . . . . .	45
10.5.2 High Variance . . . . .	46
10.6 Deciding what to try next (Revisited) . . . . .	46
10.6.1 Neural Networks . . . . .	46
<b>11 Machine Learning System Design</b>	<b>47</b>
11.1 Prioritizing what to work on: Spam Classification Example . . . . .	47
11.1.1 Time Management? . . . . .	47
11.2 Error Analysis . . . . .	48
11.2.1 Recommended approach . . . . .	48
11.2.2 Numerical Evaluation . . . . .	48
11.3 Skewed Classes . . . . .	49
11.3.1 Precision/Recall . . . . .	49
11.3.2 Trade-Off . . . . .	50
11.3.3 Working with Large Data . . . . .	50
<b>12 Support Vector Machines</b>	<b>51</b>
12.1 Optimization Objective . . . . .	51
12.1.1 Alternative view of logistic regression . . . . .	51
12.1.2 SVM Hypothesis . . . . .	52
12.2 Large Margin . . . . .	53
12.2.1 Intuition . . . . .	53
12.2.2 Math . . . . .	53
12.3 Kernels . . . . .	54
12.3.1 Non-linear Decision Boundary . . . . .	54
12.3.2 Choosing the Landmarks . . . . .	55
12.3.3 SVM Parameters: . . . . .	56
12.4 Using SVM . . . . .	56
12.4.1 Multi-Class Classification . . . . .	57

12.4.2 Logistic Regression vs SVM . . . . .	57
<b>II Unsupervised Learning</b>	<b>58</b>
<b>13 Clustering</b>	<b>59</b>
13.1 Unsupervised Learning . . . . .	59
13.2 Clustering . . . . .	59
13.2.1 Application . . . . .	59
13.3 $K$ -means . . . . .	59
13.3.1 Algorithm . . . . .	59
13.3.2 $K$ -means for non-separated clusters . . . . .	60
13.3.3 Optimization Objective . . . . .	60
13.3.4 Random Initialization . . . . .	61
13.3.5 Choosing $K$ . . . . .	61
<b>14 Dimensionality Reduction</b>	<b>62</b>
14.1 Motivation . . . . .	62
14.1.1 Data Compression . . . . .	62
14.1.2 Data Visualization . . . . .	62
14.2 Principle Component Analysis . . . . .	62
14.2.1 Problem Formulation . . . . .	62
14.3 Algorithm . . . . .	63
14.3.1 Data Preprocessing . . . . .	63
14.3.2 Procedure . . . . .	63
14.3.3 Reconstruction . . . . .	64
14.4 Choosing the number of principle components . . . . .	64
14.4.1 Algorithm . . . . .	65
14.5 Applying PCA . . . . .	65
14.5.1 Supervised Learning Speedup . . . . .	65
14.5.2 Visualization . . . . .	65
14.5.3 Bad use of PCA: To prevent overfitting . . . . .	66
14.5.4 PCA is sometimes used where it shouldn't be . . . . .	66
<b>15 Anomaly Detection</b>	<b>67</b>
15.1 Example . . . . .	67
15.1.1 Aircraft Engine! . . . . .	67
15.1.2 Fraud Detection . . . . .	67
15.1.3 Manufacturing . . . . .	67
15.1.4 Monitoring computers in a data center . . . . .	67
15.2 Gaussian (Normal) Distribution . . . . .	68
15.2.1 Definition . . . . .	68

15.2.2	Parameter Estimation . . . . .	68
15.3	Algorithm . . . . .	68
15.3.1	Density Estimation . . . . .	68
15.3.2	Anomaly Detection Algorithm . . . . .	69
15.4	Developing and Evaluating an Anomaly Detection System . . . . .	70
15.4.1	The importance of real-number evaluation . . . . .	70
15.4.2	Algorithm Evaluation . . . . .	70
15.5	Anomaly Detection vs Supervised Learning . . . . .	71
15.6	Choosing what features to use . . . . .	71
15.6.1	Transformations . . . . .	71
15.6.2	Error analysis for anomaly detection . . . . .	72
15.6.3	Other techniques: . . . . .	72
15.7	Multivariate Gaussian (Normal) Distribution . . . . .	72
15.8	Anomaly Detection using Multivariate Gaussian Distribution . . . . .	73
15.8.1	Model . . . . .	73
15.8.2	Anomaly Detection . . . . .	73
15.8.3	Relationship to original model . . . . .	74
<b>16</b>	<b>Recommender Systems</b>	<b>75</b>
16.1	Problem Formulation . . . . .	75
16.2	Content-Based Recommendations . . . . .	75
16.2.1	Optimization Objective . . . . .	76
16.2.2	Optimization Algorithm . . . . .	76
16.3	Collaborative Filtering . . . . .	77
16.3.1	Algorithm . . . . .	77
16.4	Vectorization: Low Rank Matrix Vectorization . . . . .	78
16.4.1	Vectorization . . . . .	78
16.4.2	Finding related movies . . . . .	79
16.5	Implementational Detail: Mean Normalization . . . . .	79
<b>17</b>	<b>Large Scale Machine Learning</b>	<b>80</b>
17.1	Learning with Large Datasets . . . . .	80
17.1.1	Problems . . . . .	80
17.1.2	Sanity Check . . . . .	80
17.2	Stochastic Gradient Descent . . . . .	80
17.2.1	Linear Regression . . . . .	81
17.2.2	Stochastic Gradient Descent . . . . .	81
17.3	Mini-Batch Gradient Descent . . . . .	81
17.4	Convergence . . . . .	82
17.4.1	Checking for Convergence . . . . .	82
17.5	Online Learning . . . . .	83

17.5.1 Example . . . . .	83
17.5.2 Algorithm . . . . .	83
17.5.3 Other Examples . . . . .	83
17.6 Map-reduce and data parallelism . . . . .	84
17.6.1 Map-reduce . . . . .	84
17.6.2 Summation over the training set . . . . .	85
17.6.3 Multi-core machines . . . . .	85

# 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*.



# **Part I**

## **Supervised Learning**

# Chapter 2

## Linear Regression with One Variable

### 2.1 Notations

$m$  = number of training examples

$x$ 's = 'input' variables / features

$y$ 's = 'output' variables / 'target' variables

$(x, y)$  = single training example

$(x^{(i)}, y^{(i)}) = i^{th}$  example

### 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

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

where  $\alpha$  = learning rate

### Important!

Simultaneous Update!

$$\begin{aligned} temp_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \forall j \\ \theta_j &:= temp_j \quad \forall j \end{aligned}$$

## 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:**

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

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

### 3.2 Vector

An  $n \times 1$  matrix.

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

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

**Note:** Uppercase for matrices, lowercase for vectors.

### 3.3 Addition and Scalar Multiplication

Add/Subtract (element by element) matrices of same dimension 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

$n$  = number of features

$x^{(i)}$  = input (features) of  $i^{th}$  training example

$x_j^{(i)}$  = value of feature  $j$  of  $i^{th}$  training example

### 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 convinience, define  $x_0 = 1$ . So

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

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

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

## 4.3 Gradient Descent

$$\text{Hypothesis: } h_{\theta}(x) = \theta^T x$$

$$= \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

$$\text{Parameters: } \theta$$

$$= \theta_0, \theta_1, \dots, \theta_n$$

$$\text{Cost Function: } J(\theta) = J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

**Gradient Descent:**

```
repeat{
   $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ 
   $= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$ 
   $= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$ 
}(simultaneously update  $\forall j = 0, 1, \dots, n$ )
```

### 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

$\mu_i$  = average value of  $x_i$

$S_i$  = range (max - min) *or*

=  $\sigma$  (standard deviation)

### 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  $\epsilon$  (say  $10^{-3}$ ) in one iteration.

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

To choose  $\alpha$ , try ..., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

## 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:

$$x_1 = size$$

$$x_2 = size^2$$

$$x_3 = size^3$$

## 4.5 Normal Equation

Solve for  $\theta$  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

$$0 \leq h_{\theta}(x) \leq 1$$

$$h_{\theta}(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad g \text{ is called a } \textit{sigmoid} \text{ function or a } \textit{logistic} \text{ function.}$$

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

### 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  $\theta$

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

## 6.2 Decision Boundary

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

## Non-linear Decision Boundaries

Use same technique as polynomial regression for features.

## 6.3 Cost Function

Training Set :  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$x_0 = 1$$

$$y \in \{0, 1\}$$

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

## How to choose parameter $\theta$ ?

**Linear Regression:**

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

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_{\theta}(x), y) = \frac{1}{2} (h_{\theta}(x) - y)^2$$

**Logistic Regression:**

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & y = 1 \\ -\log(1 - h_{\theta}(x)) & y = 0 \end{cases}$$

$$\text{Cost}(h_{\theta}(x), y) = 0 \text{ if } h_{\theta}(x) = y$$

$$\text{Cost}(h_{\theta}(x), y) \rightarrow \text{inf if } y = 0 \text{ and } h_{\theta}(x) \rightarrow 1$$

$$\text{Cost}(h_{\theta}(x), y) \rightarrow \text{inf if } y = 1 \text{ and } h_{\theta}(x) \rightarrow 0$$

**Note:**  $y = 0$  or  $1$  always.

$$\text{Cost}(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x))$$

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

**To fit parameters  $\theta$ :**

$$\underset{\theta}{\text{minimize}} J(\theta)$$

**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  $\theta_j$

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

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  $\alpha$
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  $\theta$ .

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);
```

## 6.5 Multi-Class Classification

### 6.5.1 One vs All

Build a separate binary classifier  $h_{\theta}^{(i)}(x)$  for each class against all other classes.

$$h_{\theta}^{(i)} = P(y = i|x; \theta) \quad \forall i$$

On a new input  $x$ , to make a prediction, pick the class  $i$  that maximizes  $h_{\theta}^{(i)}(x)$



# Chapter 7

## Regularization

### 7.1 Problem of Overfitting

1. Underfitting (High Bias)
2. Right Fit
3. Overfitting (High Variance)

**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well  $\left( J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h\theta(x^{(i)}) - y^{(i)})^2 \approx 0 \right)$ , but fail to generalize to new examples (predict prices on new examples).

### 7.2 Addressing Overfitting

Options:

1. Reduce the number of features
  - Manually select which features to keep
  - Model selection algorithm (later)
2. Regularization
  - Keep all the features, but reduce the magnitude/values of parameters  $\theta_j$
  - Works well when we have a lot of features, each of which contributes a bit to predicting  $y$

## 7.3 Cost Function

Say our overfitting hypothesis is  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ . Suppose, we penalize and make  $\theta_3, \theta_4$  very small

$$\underset{\theta}{\text{minimize}} \left( \frac{1}{2m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + 1000\theta_3^2 + 1000\theta_4^2 \right)$$

## Regularization

Small values for parameters  $\theta_0, \theta_1, \dots, \theta_n$ .

- *Simpler* hypothesis
- Less prone to overfitting

### Which parameters to penalize?

- Features:  $x_1, x_2, \dots, x_{100}$
- Parameters:  $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

**Note:** The convention, we don't regularize  $\theta_0$  but it doesn't make very much difference.

$\lambda$  here is *regularization parameter*.

What if  $\lambda$  is set too high (say  $10^{10}$ )? Underfitting!

## 7.4 Regularized Linear Regression

**Updated  $J(\theta)$ :**

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$
$$\underset{\theta}{\text{minimize}} J(\theta)$$

**Gradient Descent:**

Repeat{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m \left( (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \quad \forall j \in \{1, 2, \dots, n\}$$

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

}

**Normal Equation:**

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1} \begin{pmatrix} \in \mathbb{R}^{(n+1) \times (n+1)} \end{pmatrix} - X^T y \in \mathbb{R}^{n+1}$$

**Non-Invertibility**Suppose  $m \leq n$ ,

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

If  $\lambda > 0$ ,

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} - X^T y$$

Not a problem!

## 7.5 Regularized Logistic Regression

**Cost Function:**

$$J(\theta) = - \left[ \frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

**Gradient Descent:**

Repeat{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m \left( (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \quad \forall j \in \{1, 2, \dots, n\}$$

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

}

**Advanced Optimization:**

```
function [jVal, gradient] = costFunction(theta)
    jVal = [...code to compute J(theta)...];
    gradient = [...code to compute derivative of J(theta)...];
```

---

Check Lecture7.pdf for more details.

# Chapter 8

## Neural Networks

### 8.1 Non-Linear Hypothesis

If #features is high, hypothesis could have extremely high #terms. Hence the need for non-linear hypothesis.

### 8.2 Neural Networks

- *Origins*: Algorithms that try to mimic brain.
- Widely used in 80s and early 90s; diminished in late 90s.
- Resurgence: State-of-the-art technique for many applications.

### 8.3 Model Representation

#### Neuron Model: Logistic Unit

1. input layer
2. hidden layer / computation layer
3. output layer
4. activation function ( $g()$ )

parameters  $\equiv$  weights

## 8.4 Notations

$a_i^{(j)}$  = *activation* of unit  $i$  in layer  $j$   
 $\Theta^{(j)}$  = matrix of weights controlling  
 function mapping from layer  $j$  to  
 layer  $j + 1$

**Example:**

layer 1 :  $\{x_1, x_2, x_3\}$

layer 2 :  $\{a_1^{(2)}, a_2^{(2)}, a_3^{(2)}\}$

layer 3 :  $\{a_1^{(3)}\}$

$$a_1^{(2)} = g \left( \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right)$$

$$a_2^{(2)} = g \left( \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right)$$

$$a_3^{(2)} = g \left( \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right)$$

$$h_{\Theta}(x) = a_1^{(3)} = g \left( \Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)} \right)$$

**Note:** If network has  $s_j$  units in layer  $j$ , and  $s_{j+1}$  units in layer  $j + 1$ , then

$$\Theta^{(j)} \in \mathbb{R}^{s_{j+1} \times (s_j + 1)}$$

## 8.5 Forward Propagation

$$z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

$$z_2^{(2)} = \Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3$$

$$z_3^{(2)} = \Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3$$

$$a_1^{(2)} = g(z_1^{(2)})$$

$$a_2^{(2)} = g(z_2^{(2)})$$

$$a_3^{(2)} = g(z_3^{(2)})$$

## Vectorized Implementation

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad x_0 = 1$$

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$a^{(1)} = x \quad \in \mathbb{R}^4$$

$$z^{(2)} = \Theta^{(1)} a^{(1)} \quad \in \mathbb{R}^3$$

$$a^{(2)} = g(z^{(2)}) \quad \in \mathbb{R}^3$$

$$\text{Add } a_0^{(2)} = 1 \quad \rightarrow a^{(2)} \in \mathbb{R}^4$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$$

Other network architectures are possible!

### Non-linear classification example: XOR/XNOR

$x_1, x_2$  are binary (0 or 1)

$$\begin{aligned} y &= x_1 \text{ XOR } x_2 & \text{or} \\ &= x_1 \text{ XNOR } x_2 & \equiv \text{NOT}(x_1 \text{ XOR } x_2) \end{aligned}$$

<sup>1</sup> Simple example: AND

$$\begin{aligned} x_0 &= 0 \\ x_1, x_2 &\in \{0, 1\} \\ y &= x_1 \text{ AND } x_2 \\ \Theta_{10}^{(1)} &= -30 \\ \Theta_{11}^{(1)} &= 20 \\ \Theta_{12}^{(1)} &= 20 \\ h_{\Theta}(x) &= g(-30 + 20x_1 + 20x_2) \end{aligned}$$

$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

We can combine AND, OR, NOT, (NOT  $x_1$ ) AND (NOT  $x_2$ ) to get XNOR, XOR, etc.

## 8.6 Multi-Class Classification

Extension of One-vs-All Method!

**Example:** Say 4 classes, so 4 output units. So,  $y \in \mathbb{R}^4$  with 1 in corresponding class and 0 elsewhere.

---

<sup>1</sup>Work out more examples (OR, NOT, (NOT  $x_1$ ) AND (NOT  $x_2$ ))!  
Check Lecture8.pdf for more details.



# Chapter 9

## Back Propagation

### 9.1 Notations

$L$  = total no. of layers in the network

$s_l$  = no. of units (not counting bias unit) in layer  $l$

#### Binary Classification

$$y = 0 \text{ or } 1$$

1 output unit

$$h_{\Theta}(x) \in \mathbb{R}$$

$$s_L = 1$$

$$K = 1 \quad (\text{for simplification})$$

#### Multi-class Classification (K classes)

$$y \in \mathbb{R}^K$$

**Example:**  $K = 4$

$$y \in \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}$$

$K$  output units

$$\begin{aligned} h_{\Theta}(x) &\in \mathbb{R}^K \\ s_L &= K \\ K &\geq 3 \end{aligned}$$

## 9.2 Cost Function

### Logistic Regression

$$J(\theta) = - \left[ \frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

### Neural Network

$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$\begin{aligned} J(\Theta) = & -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K \left( y_k^{(i)} \log \left( h_{\Theta}(x^{(i)}) \right)_k + \left( 1 - y_k^{(i)} \right) \log \left( 1 - \left( h_{\Theta}(x^{(i)}) \right)_k \right) \right) \right] \\ & + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left( \Theta_{ji}^{(l)} \right)^2 \end{aligned}$$

## 9.3 Backpropagation Algorithm

### 9.3.1 Gradient Computation

$$\begin{aligned} J(\Theta) = & -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K \left( y_k^{(i)} \log \left( h_{\Theta}(x^{(i)}) \right)_k + \left( 1 - y_k^{(i)} \right) \log \left( 1 - \left( h_{\Theta}(x^{(i)}) \right)_k \right) \right) \right] \\ & + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left( \Theta_{ji}^{(l)} \right)^2 \end{aligned}$$

$$\min_{\Theta} J(\Theta)$$

**Need code to compute:**

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

**Given one training example**  $(x, y)$ :

Forward Propagation:

$$\begin{aligned}
 a^{(1)} &= x \\
 z^{(2)} &= \Theta^{(1)} a^{(1)} \\
 a^{(2)} &= g(z^{(2)}) && (\text{add } a_0^{(2)}) \\
 z^{(3)} &= \Theta^{(2)} a^{(2)} \\
 a^{(3)} &= g(z^{(3)}) && (\text{add } a_0^{(3)}) \\
 z^{(4)} &= \Theta^{(3)} a^{(3)} \\
 h_{\Theta}(x) &= a^{(4)} = g(z^{(4)})
 \end{aligned}$$

**Gradient Computation:** Back Propagation

Intuition:  $\delta_j^{(l)}$  = “error” of node  $j$  in layer  $l$

For each output unit ( $L = 4$ )

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorize:

$$\begin{aligned}
 \delta^{(4)} &= a^{(4)} - y \\
 \delta^{(3)} &= (\Theta^{(3)})^T \delta^{(4)} \cdot g'(z^{(3)}) \\
 g'(z^{(3)}) &= a^{(3)} \cdot (1 - a^{(3)}) \\
 \delta^{(2)} &= (\Theta^{(2)})^T \delta^{(3)} \cdot g'(z^{(2)}) \\
 g'(z^{(2)}) &= a^{(2)} \cdot (1 - a^{(2)})
 \end{aligned}$$

No  $\delta^{(1)}$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (\text{ignoring } \lambda; \text{ if } \lambda = 0)$$

## Backpropagation Algorithm

Training set  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set  $\Delta_{ij}^{(l)} = 0$  (for all  $l, i, j$ )  $\rightarrow$  accumulators to compute  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

For  $i = 1$  to  $m$

Set  $a^{(1)} = x^{(1)}$

Perform forward propagation to compute  $a^{(l)}$  for  $l = 2, 3, \dots, L$

Using  $y^{(i)}$ , compute  $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$  (Vectorized)

$$D_{ij}^{(l)} := \begin{cases} \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m} \Delta_{ij}^{(l)} & \text{if } j = 0 \end{cases}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

## 9.4 Implementation Note

### 9.4.1 Unrolling Parameters

Advanced optimization functions like ‘fminunc’ assume that inputs ‘theta’, ‘gradient’, etc. are vectors. These are matrices in Neural Networks, so we *unroll* them into vectors.

**Example:** Neural Network with 3 layers (1 input, 1 hidden, 1 output) and  $s_1 = 10$ ,  $s_2 = 10$ ,  $s_3 = 1$ .

$$\begin{aligned} \Theta^{(1)} &\in \mathbb{R}^{10 \times 11}, \Theta^{(2)} \in \mathbb{R}^{10 \times 11}, \Theta^{(3)} \in \mathbb{R}^{(1 \times 11)} \\ D^{(1)} &\in \mathbb{R}^{10 \times 11}, D^{(2)} \in \mathbb{R}^{10 \times 11}, D^{(3)} \in \mathbb{R}^{(1 \times 11)} \end{aligned}$$

**To unroll:**

```
thetaVec = [Theta1(:); Theta2(:); Theta3(:)];  
DVec = [D1(:); D2(:); D3(:)];
```

**To go back to matrices:**

```
Theta1 = reshape(thetaVec(1:110), 10, 11);  
Theta2 = reshape(thetaVec(111:220), 10, 11);  
Theta3 = reshape(thetaVec(221:231), 1, 11);
```

## 9.4.2 Learning Algorithm

Have initial parameters  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$

Unroll to get initialTheta to pass to

```
fminunc(@costFunction, initialTheta, options)
```

```
function [jVal, gradientVec] = costFunction(thetaVec)
```

From thetaVec, get  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$  using `reshape`

Use forward/back propagation to compute  $D^{(1)}, D^{(2)}, D^{(3)}$  and  $J(\Theta)$

Unroll  $D^{(1)}, D^{(2)}, D^{(3)}$  to get gradientVec

## 9.5 Gradient Checking

An unfortunate problem is that it might seem like it's working even when there's a bug in the backpropagation algorithm (implementation). Subtle bugs can hence cause higher level of error and perform worse than a bug-free implementation. Here, we try to make sure that our implementation is 100% correct.

### 9.5.1 Numerical Estimation of gradients

To estimate derivative of  $J(\theta)$ , we calculate  $J(\theta + \epsilon)$  and  $J(\theta - \epsilon)$  and take the slope of the line connecting these two points.

$$J'(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

Use pretty small  $\epsilon$  ( $\approx 10^{-4}$ ).

The above formula is called the two sided difference. Another formula called the one sided difference is

$$J'(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta)}{\epsilon}$$

But the two sided difference usually gives a slightly more accurate approximation.

**Implement:**

$$\text{gradApprox} = (J(\text{theta} + \text{EPSILON}) - J(\text{theta} - \text{EPSILON})) / (2 * \text{EPSILON})$$

**Parameter vector  $\theta$**

$$\begin{aligned} \theta &\in \mathbb{R}^n \\ \theta &= \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \\ \frac{\partial}{\partial \theta_1} J(\theta) &\approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon} \\ \frac{\partial}{\partial \theta_2} J(\theta) &\approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon} \\ &\vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) &\approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon} \end{aligned}$$

**Implement:**

```
for i = 1:n
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*EPSILON);
end;
```

Check that gradApprox  $\approx$  DVec (from backpropagation).

### 9.5.2 Implementation Note:

- Implement backpropagation to compute Dvec (unrolled  $D^{(1)}, D^{(2)}, \dots$ )
- Implement numerical gradient check to compute gradApprox
- Make sure they give similar values
- Turn off gradient checking (much slower) and use backpropagation for learning (much faster)

#### Important:

- Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of `costFunction(...)`) your code will be very slow.

## 9.6 Random Initialization

### Initial value of $\Theta$

For gradient descent and advanced optimization methods, we need initial value of  $\Theta$ .

Consider gradient descent:

Set `initialTheta = zeroes(n, 1)` ?

Works fine for linear/logistic regression, but not when we're training a neural network. Why?

$$\begin{aligned}\Theta_{ij}^{(l)} &= 0 \text{ for all } i, j, l \\ a_1^{(2)} &= a_2^{(2)} \\ \delta_1^{(2)} &= \delta_2^{(2)} \\ \frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) &= \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)\end{aligned}$$

Meaning, after each gradient descent update, parameters corresponding to inputs going into each unit of the next layer are identical. Thus, all the units of the next layer are still computing the same function of the input. The values may change, but will always be the same, preventing the neural network from learning something interesting since it is extremely redundant.

### Random Initialization: Symmetry Breaking

To get around this (*symmetric weights*), we do random initialization. Initialize each  $\Theta_{ij}^{(l)}$  to a random value in  $[-\epsilon, \epsilon]$  (i.e.  $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$ ).

#### Example:

```
Theta1 = rand(10,11)*(2*INIT_EPSILON) - INIT_EPSILON
Theta2 = rand(1, 11)*(2*INIT_EPSILON) - INIT_EPSILON
```

A good choice of  $\epsilon$  is:

$$\epsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$$

Where  $L_{in}$  and  $L_{out}$  are number of units in the layers adjacent to  $\Theta^{(l)}$ .

## 9.7 Putting it all together

Pick a network architecture (connectivity pattern between neurons), like number of layers, number of units in each layer, etc. For example,  $[3 \rightarrow 5 \rightarrow 4]$  vs  $[3 \rightarrow 5 \rightarrow 5 \rightarrow 4]$  vs  $[3 \rightarrow 5 \rightarrow 5 \rightarrow 5 \rightarrow 4]$ .

No. of input units : Dimension of features  $x^{(i)}$

No. of output units : Number of classes

#### Remember:

$y \in \{1, 2, \dots, 10\}$  becomes

$$y \in \left\{ \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}$$

#### For Hidden Layers?

**Reasonable Default:** 1 hidden layer, or if  $> 1$  hidden layer, have same no. of hidden units in every layer (usually, the more the better, just that it may get computationally expensive). Usually, comparable to the number of features.



### Training a neural network:

1. Randomly initialize weights
2. Implement forward propagation to get  $h_{\Theta}(x)$  for any  $x$
3. Implement code to compute cost function  $J(\Theta)$
4. Implement backpropagation to compute partial derivatives  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

```
for i = 1:m
    Perform forward propagation ...
    and backpropagation ...
    using example  $(x^{(i)}, y^{(i)})$ 

    Get activations  $a^{(l)}$  ...
    and delta terms  $\delta^{(l)}$  ...
    for  $l = 2, \dots, L$ 

         $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$ 
         $\vdots$ 
    end
     $\vdots$ 
    compute  $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ 
```

5. (a) Use gradient checking to compare  $J'(\Theta)$  computed using backpropagation vs using numerical estimate.  
(b) Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize  $J(\Theta)$

$J(\Theta)$  in general for a neural network is non-convex, and hence susceptible to local optima. However, it turns out, in practice, this is not usually a huge problem.

---

Check Lecture9.pdf for more details.

# Chapter 10

## Applying Machine Learning

### 10.1 Deciding what to try next

#### 10.1.1 Debugging a Learning Algorithm

Suppose you have implemented regularized linear regression to predict housing prices.

$$J(\theta) = \frac{1}{2m} \left[ \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

However, it makes *unacceptably* large errors in prediction. What to try next?

- Get more training examples (Sometimes doesn't actually help)
- Try smaller set of features (Overfitting?)
- Try additional features (Underfitting?) (Huge project)
- Try adding polynomial features
- Increasing/Decreasing  $\lambda$

Need to understand what would help.

#### 10.1.2 Machine Learning Diagnostics

**Diagnostic:** A test that you can run to gain insight at what is/isn't working with the algorithm, and gain guidance as to how best to improve its performance.

Diagnostics can take time to implement, but doing so can be a very good use of your time as it can save huge time later.

## 10.2 Evaluating your Hypothesis

Just *training error* not a good indicator (Overfitting).

Solution? Plot Hypothesis function? Not feasible with high number of features.

### 10.2.1 The Standard Way

Split the dataset into two portions - *Training Set* and *Test Set* with about 70%-30% ratio (*randomly* if it is in some order).

**Notation:**

Training Examples :  $(x, y)$

Test Examples :  $(x_{test}, y_{test})$

No. of Training Examples :  $m$

No. of Test Examples :  $m_{test}$

### 10.2.2 Train/Test Procedure

1. Learn  $\theta$  from training data (minimize  $J(\theta)$ )
2. Compute test set error ( $J_{test}(\theta)$ ) on test examples
  - In case of classification problem, we can also use misclassification error (Also called 0/1 misclassification error)
  - $$\text{err}(h_{\theta}(x), y) = \begin{cases} 1 & \text{if } h_{\theta}(x) \geq 0.5, y == 0 \\ & \text{or if } h_{\theta}(x) < 0.5, y == 1 \\ 0 & \text{otherwise} \end{cases}$$
  - test error = 
$$\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \text{err}(h_{\theta}(x_{test}^{(i)}), y_{test}^{(i)})$$
  - This gives us the proportion of the test data that was misclassified

## 10.3 Model Selection

Once parameters  $\theta$  were fit to some training set, the error  $J(\theta)$  measured on that data is likely to be lower than the actual generalization error.

**Model Selection:** Let's say we try to choose what degree polynomial to fit to data (linear, quadratic, cubic, ...)

**Notation:**

$d$  = Degree of polynomial

For different  $d$ , we can have the following:

1.  $h_{\theta}(x) = \theta_0 + \theta_1 x$
2.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3$
- $\vdots$
10.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10}$

Choose one model out of the above. We can train them separately, and get different  $\theta^{(k)}$ , and calculate  $J_{test}(\theta)$  for each of them, and choose the one with the least test error.

How well does this model generalize? Report  $J_{test}(\theta)$ .

**Problem:**  $J_{test}(\theta)$  is likely an optimistic estimate of generalization error (our extra parameter  $d$  is fit to *test set*).

### 10.3.1 Evaluating your hypothesis

Divide the data set in three pieces, *training set* (60%), *cross-validation set* (20%), *test set* (20%).

### Notations:

Training Examples :  $(x, y)$

Cross-Validation Examples :  $(x_{cv}, y_{cv})$

Test Examples :  $(x_{test}, y_{test})$

No. of Training Examples :  $m$

No. of Cross-Validation Examples :  $m_{cv}$

No. of Test Examples :  $m_{test}$

Training Error :  $J(\theta)$  or  $J_{train}(\theta)$

Cross Validation Error :  $J_{cv}(\theta)$

Test Error :  $J_{test}(\theta)$

Use *Cross-Validation Set* to select the model. Estimate generalization error for test set.

## 10.4 Bias vs Variance

Plot *error* against *degree* of the polynomial for both, *training* and *validation* data set. Usually, training error tends to decrease. Cross-Validation error (or test error) on the other hand, tends to decrease, and then increase again.

To recognise bias/variance problem:

Bias (Underfit) :  $J_{train}$  will be high

$$J_{cv} \approx J_{train}$$

Variance (Overfit) :  $J_{train}$  will be low

$$J_{cv} \gg J_{train}$$

### 10.4.1 Effect of regularization

Large  $\lambda$  can cause high bias (underfit), and at very small  $\lambda$  causes high variance (overfit). How to automatically choose a good value of  $\lambda$ .

**Choosing  $\lambda$ :**

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2 \\ J_{train}(\theta) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \\ J_{cv}(\theta) &= \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2 \\ J_{test}(\theta) &= \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2 \end{aligned}$$

Have a model  $(h_{\theta}(x), J(\theta))$ , have a bunch of values to try for  $\lambda$  like 0, 0.01, 0.02, 0.04, 0.08, ..., 10 (roughly doubling). Train (minimize  $J(\theta)$ ) for all values. Use  $J_{cv}(\theta)$  to choose a  $\lambda$ . Report generalization error as  $J_{test}(\theta)$ .

**Bias/Variance as a function of  $\lambda$**

Plot  $J_{train}(\theta)$  and  $J_{cv}(\theta)$  as a function of  $\lambda$ . Usually,  $J_{train}(\theta)$  tends to increase while  $J_{cv}(\theta)$  tends to decrease at first but increase again later.

## 10.5 Learning Curves

Useful thing to plot, to sanity check, to improve the performance, to diagnose for bias/variance, ....

Plot  $J_{train}(\theta)$ , and  $J_{cv}(\theta)$  as a function of  $m$ , the number of training examples. Measure  $J_{train}(\theta)$  only on data used to train!. Measure  $J_{cv}(\theta)$  on all cross validation data!

Usually, average training error will increase with  $m$ . Usually, average validation error will decrease with  $m$ .

### 10.5.1 High Bias

**Usually:** Validation error will decrease at first but then will saturate/flatten out soon. Training error will increase at first but then will saturate/flatten out soon, close to validation error.

**Note:** If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

### 10.5.2 High Variance

**Usually:** Training error will increase but still be pretty low. Validation error will decrease but still be pretty high, far from training error.

**Note:** If a learning algorithm is suffering from high variance, getting more training examples is indeed likely to help.

## 10.6 Deciding what to try next (Revisited)

Get more training examples	fixes high variance
Try smaller set of features	fixes high variance
Try additional features	fixes high bias (not always)
Try adding polynomial features	fixes high bias (not always)
Increasing/Decreasing $\lambda$	fixes bias/variance

### 10.6.1 Neural Networks

*Small* neural network, computationally cheaper, but possibly underfitting. *Large* neural networks, computationally more expensive, and possibly overfitting. Use regularization to address overfitting.

Often, large neural network with regularization works better (but may be computationally expensive).

Large network can comprise of a single large hidden layer, or two moderate hidden layers, or three average hidden layers, etc.

---

Check Lecture10.pdf for more details.

# Chapter 11

## Machine Learning System Design

### 11.1 Prioritizing what to work on: Spam Classification Example

Supervised Learning!  $x$  = features of email.  $y$  = spam(1) or not spam(0).

Features  $x$ : Choose 100 words indicative of spam/not spam.

Example: deal, buy, discount, andrew, now, ...

Feature vector:

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} andrew \\ buy \\ deal \\ discount \\ \vdots \\ now \end{bmatrix}$$

0/1 depending on whether a 'word' appears or not in the email.

$$x_j = \begin{cases} 1 & \text{if word } j \text{ appears in email} \\ 0 & \text{otherwise} \end{cases}$$

**Note:** In practice, take most frequently occurring  $n$ (10,000 to 50,000) words in training set, instead of manually picking 100 words.

#### 11.1.1 Time Management?

- Collect lots of data



- E.g. *honeypot* project.
- Develop sophisticated features based on email routing information (from email header).
- Develop sophisticated features for message body, e.g. should *discount* and *discounts* be treated as same word? How about *deal* and *Dealer*? Features about punctuation?
- Develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches.)

## 11.2 Error Analysis

### 11.2.1 Recommended approach

- Start with a simple and quick algorithm, implement and test it on cross-validation data.
- Plot learning curves to decide if more data, more features, etc.
- Error Analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

$m_{cv} = 500$  examples in cross validation set, algorithm misclassifies 100 emails. Manually examine 100 errors, and categorized them based on:

1. What type of email it is (pharma, replica, phishing, ...)
2. What features would have helped (Deliberate misspellings, Unusual email routing, Unusual punctuation)

### 11.2.2 Numerical Evaluation

Have a way of *numerical evaluation* like accuracy/error, to tell how the learning algorithm is doing.

**Example:** Should discount/discounts/discounting be treated as the same word? Can use *stemming* software (E.g. "Porter Stemmer"). It can hurt because the software can mistake universe/university to be the same.

So, error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try and see if it works.

Need numerical evaluation (e.g. cross validation error) of algorithm's performance with and without stemming.

## 11.3 Skewed Classes

**Cancer classification example:** Train logistic regression model  $h_{\theta}(x)$  with 1% error on test set. Good? What if only 0.50% patients in training/test set have cancer? Not so impressive anymore! Even predicting *No Cancer* all the time will get 0.50% error. This is a case of *skewed* classes.

Going from 99.2% to 99.5% accuracy is actually doing something useful? Can't say in case of skewed classes.

### 11.3.1 Precision/Recall

$y = 1$  in presence of rare class that we want to detect.

#### Notations

Notations	Actual Class	Predicted Class
True Positive	1	1
True Negative	0	0
False Positive	0	1
False Negative	1	0

#### Precision

Out of all the predicted *positives*, what fraction is *actually* positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

#### Recall

Out of all the actual *positives*, what fraction is *predicted* positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Now, if we predict no cancer always, then recall = 0. These metrics usually help us evaluate our algorithm even in case of really skewed classes.

### 11.3.2 Trade-Off

Consider same example of cancer, we train logistic regression  $h_{\theta}(x)$  which gives the probability of having cancer.

Suppose we want to predict cancer ( $y = 1$ ) only if we're very sure. One way to do this, is predict  $y = 1$  when  $h_{\theta}(x) \geq 0.7$  instead of 0.5 and  $y = 0$  otherwise. This will give us *higher* precision, but *lower* recall.

Suppose we want to avoid missing too many cases of cancer, we can predict  $y = 1$  when  $h_{\theta}(x) \geq 0.3$ . This will give us *higher* recall, but *lower* precision.

How to choose the threshold?

#### $F_1$ Score (F Score)

We have lost our 'single number' evaluation metric, we now have precision and recall. What are our options?

- Average of precision and recall ( $\frac{P + R}{2}$ )? Not a good idea! (Extremes)
- $F_1$  Score =  $2 \frac{PR}{P + R}$  better choice, commonly used

Use  $F_1$  score on cross-validation set to decide the threshold.

### 11.3.3 Working with Large Data

Consider cases where more data will be helpful.

Say, feature  $x \in \mathbb{R}^{n+1}$  has sufficient information to predict  $y$  accurately. (Counter-examples: predict housing price from only size or area).

A useful test is, given  $x$ , can a human expert confidently predict  $y$ ?

Alongside, Suppose we use a learning algorithm with a large number of parameters (low bias algorithms).

Using a very large training set, it is unlikely to overfit.

---

Check Lecture11.pdf for more details.

# Chapter 12

## Support Vector Machines

Another popular supervised learning algorithm.

### 12.1 Optimization Objective

#### 12.1.1 Alternative view of logistic regression

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

If  $y = 1$ , we want  $h_{\theta}(x) \approx 1$ ,  $\theta^T x \gg 0$

If  $y = 0$ , we want  $h_{\theta}(x) \approx 0$ ,  $\theta^T x \ll 0$

#### Cost of example:

Each example  $(x, y)$  contributes:

$$\begin{aligned} & -(y \log(h_{\theta}(x)) + (1 - y) \log(1 - h_{\theta}(x))) \\ &= -y \log\left(\frac{1}{1 + e^{\theta^T x}}\right) - (1 - y) \log\left(1 - \frac{1}{1 + e^{\theta^T x}}\right) \end{aligned}$$

If  $y = 1$  (want  $\theta^T x \gg 0$ ): we get  $-\log\left(\frac{1}{1 + e^{-z}}\right)$  where  $z = \theta^T x$ . For SVM modify cost function to be flat 0 for  $x > 1$  and a straight line to the left (with negative slope). Denote by  $cost_1(z)$ .

If  $y = 0$  (want  $\theta^T x \ll 0$ ): we get  $-\log\left(1 - \frac{1}{1 + e^{-z}}\right)$  where  $z = \theta^T x$ . For SVM modify cost function to be flat 0 for  $x < -1$  and a straight line to the left (with positive slope). Denote by  $cost_0(z)$ .

### Logistic Regression:

$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m \left( y^{(i)} \left( -\log(h_{\theta}(x^{(i)})) \right) + (1 - y^{(i)}) \left( -\log(1 - h_{\theta}(x^{(i)})) \right) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

### Support Vector Machine:

Modify costs first

$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m \left( y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Get rid of  $\frac{1}{m}$  term. Only a constant, same optimal value!

$$\min_{\theta} \left[ \sum_{i=1}^m \left( y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right) \right] + \frac{\lambda}{2} \sum_{j=1}^n \theta_j^2$$

By convention, instead of optimizing  $A + \lambda B$ , optimize  $CA + B$  where  $C$  is somewhat equivalent of  $\frac{1}{\lambda}$ , again, same optimal value!

$$\min_{\theta} C \left[ \sum_{i=1}^m \left( y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

### 12.1.2 SVM Hypothesis

$$\min_{\theta} C \left[ \sum_{i=1}^m \left( y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

#### Hypothesis:

Unlike logistic regression, SVM does not give probability. It just makes prediction.

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

## 12.2 Large Margin

### 12.2.1 Intuition

If  $y = 1$ , we want  $\theta^T x \geq 1$  (not just  $\geq 0$ )

If  $y = 0$ , we want  $\theta^T x \leq -1$  (not just  $< 0$ )

This builds safety margin factor in between.

### SVM Decision Boundary

If  $C \gg 1$ , say 100,000, we will get

$\min \left( C \times 0 + \frac{1}{2} \sum_{j=1}^n \theta_j^2 \right)$  such that  $\begin{cases} \theta^T x^{(i)} \geq 1 & \text{if } y^{(i)} = 1 \\ \theta^T x^{(i)} \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$  Very interesting decision boundary!

**Linearly Separable Case:** It will choose the decision boundary such that it maximises margin of the SVM.

**Outliers:** If  $C$  is indeed very large it will be affected by outliers easily but if  $C$  is not too large, it will still give a reasonable boundary.

### 12.2.2 Math

#### Vector Inner Product

$u^T v$  (or dot product)

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$\|u\|$  = length of vector  $u$

$$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$$

$p$  = projection of vector  $v$  on vector  $u$

$$u^T v = p \|u\|$$

$$u^T v = u_1 v_1 + u_2 v_2$$

$$u^T v = v^T u$$

## SVM Decision Boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 \text{ s.t. } \begin{cases} \theta^T x^{(i)} \geq 1 & \text{if } y^{(i)} = 1 \\ \theta^T x^{(i)} \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$$

**Simplifications:** Only for the purpose of understanding here!

$$\theta_0 = 0$$

$$n = 2$$

This gives,  $\min_{\theta} \frac{1}{2} (\theta_1^2 + \theta_2^2) = \min_{\theta} \frac{1}{2} \left( \sqrt{\theta_1^2 + \theta_2^2} \right)^2 = \min_{\theta} \frac{1}{2} \|\theta\|^2$   
 $\theta^T x^{(i)} = ?$

$$\begin{aligned} p^{(i)} &= \text{projection of vector } x^{(i)} \text{ on vector } \theta \\ \theta^T x^{(i)} &= p^{(i)} \|\theta\| \\ &= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} \end{aligned}$$

**Optimization Objective:** Modified as follows:

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} \|\theta\|^2 \text{ s.t. } \begin{cases} p^{(i)} \|\theta\| \geq 1 & \text{if } y^{(i)} = 1 \\ p^{(i)} \|\theta\| \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$$

where  $p^{(i)}$  is the projection of  $x^{(i)}$  on vector  $\theta$ .  $p^{(i)}$  will be bigger with large margin helping minimize the  $\frac{1}{2} \|\theta\|^2$ .

## 12.3 Kernels

### 12.3.1 Non-linear Decision Boundary

Predict  $y = 1$  if  $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \geq 0$ .

**New Notation:**  $h\theta = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \dots$  where possibly,  $f_1 = x_1$ ,  $f_2 = x_2$ ,  $f_3 = x_1 x_2$ ,  $f_4 = x_1^2, \dots$

Is there a better/different choice of features  $f_1, f_2, f_3, \dots$ ?

Given  $x$ , choose 3 random landmarks  $l$ s and define new features as:

$$\begin{aligned} f_1 &= \text{similarity}(x, l^{(1)}) &= \exp\left(\frac{-\|x - l^{(1)}\|^2}{2\sigma^2}\right) \\ f_2 &= \text{similarity}(x, l^{(2)}) &= \exp\left(\frac{-\|x - l^{(2)}\|^2}{2\sigma^2}\right) \\ f_3 &= \text{similarity}(x, l^{(3)}) &= \exp\left(\frac{-\|x - l^{(3)}\|^2}{2\sigma^2}\right) \end{aligned}$$

Similarity function is *Kernel* functions. This particular one is *Gaussian* Kernel.

### Kernels and Similarity

If  $x \approx l^{(1)}$ :  $f_1 \approx 1$

If  $x$  is far from  $l^{(1)}$ :  $f_1 \approx 0$

Measures closeness to landmarks.

Setting  $\sigma^2$  low, the value of  $f_1$  falls to 0 very rapidly, and vice-versa.

Predict 1 when  $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$  Say  $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$ , then points close to  $l^{(1)}$  or  $l^{(2)}$  will give 1 creating complex non-linear boundary.

### 12.3.2 Choosing the Landmarks

Where to get landmarks? How many?

One choice would be one landmark per location of each feature. Given  $m$  examples, we have  $m$  landmarks.

Given an example  $x$ , compute  $f_1, f_2, \dots$  using kernels and get the feature vector  $f$  (include  $f_0 = 1$ ). For training examples, at least one feature would be 1.

**Hypothesis:** Given  $x$ , compute  $f \in \mathbb{R}^{m+1}$

Predict 'y=1' if  $\theta^T f \geq 0$

Predict 'y=0' otherwise

### Training:

$$\min_{\theta} C \left[ \sum_{i=1}^m \left( y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) \right) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



$n = m$

Slight modification in last term! Instead of using  $\theta^T \theta$ , use  $\theta^T M \theta$  where  $M$  is a matrix depending on the kernel done for computational expense.

Can use Kernels in other algorithms but tricks (such as  $M$ ) don't go well with other algorithms so it can be slow/expensive.

### 12.3.3 SVM Parameters:

$$C \left( = \frac{1}{\lambda} \right)$$

Large  $C$  : Lower Bias, High Variance

Small  $C$  : Higher Bias, Low Variance

$$\sigma^2$$

Large  $\sigma^2$  : Features  $f_i$  vary more smoothly  
: Higher bias, Low Variance

Small  $\sigma^2$  : Features  $f_i$  vary less smoothly  
: Lower Bias, High Variance

## 12.4 Using SVM

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters  $\theta$ .

**Need to specify:**

- Choice of parameter  $C$
- Choice of kernel (similarity function):
  - Linear Kernel is just another name of no kernel.
  - Gaussian Kernel. Need to choose  $\sigma^2$ .

```
function f = kernel(x1, x2)
    f = exp(- (||x1 - x2||^2) / (2 * sigma^2))
return
```

**Note:** Do perform feature scaling before using the Gaussian Kernel.

- Other choices of kernel

**Note:** Not all similarity functions make valid kernels. (Need to satisfy a technical condition called *Mercer's Theorem* to make sure SVM packages' optimizations run correctly, and do not diverge)

Many off-the-shelf kernels available:

- \* Polynomial Kernel:  $k(x, l) = (x^T l + \text{constant})^{\text{degree}}$  (almost always performs worse than Gaussian Kernel, and not used that much and used only where  $x$  and  $l$  are all strictly positive)
- \* More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

### 12.4.1 Multi-Class Classification

Many SVM packages already have built-in multi-class classification functionality. Otherwise, use one vs all method.

### 12.4.2 Logistic Regression vs SVM

If  $n \geq m$  (say  $n = 10,000$  and  $m = 10$  to  $1000$ ), use logistic regression or SVM without kernel (linear kernel).

If  $n$  is small (say upto  $1000$ ),  $m$  is intermediate (say upto  $10,000$ ), use SVM with Gaussian Kernel.

If  $n$  is small (say upto  $1000$ ),  $m$  is large (say  $50,000+$ ), SVM with Gaussian Kernels may start struggling. Create/Add more features, then use logistic regression or SVM without a kernel.

Neural Network likely to work well for most of these settings, but may be slower to train.

SVM is a convex optimization problem, so no need to worry about local optima.

---

Check Lecture12.pdf for more details.

# **Part II**

## **Unsupervised Learning**

# Chapter 13

## Clustering

### 13.1 Unsupervised Learning

Learning from unlabelled data as opposed to supervised learning. Give data to the algorithm to find some *structure* in our data.

### 13.2 Clustering

Group data in different *clusters*.

#### 13.2.1 Application

- Market Segmentation
- Social Network Analysis
- Organize Computer Clusters
- Astronomical Data Analysis

### 13.3 $K$ -means

#### 13.3.1 Algorithm

**Informally:** Suppose we want to group the data into two clusters.

1. Randomly initialize two cluster centroids.
2. Assign each data point to a cluster centroid based on some *distance* measure.

3. Update cluster centroid to the average of each data point in the same cluster.
4. Go back to step 2 until convergence.

**Formally :**

Input:

- $K$  (number of clusters)
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

*Remark.*  $x^{(i)} \in \mathbb{R}^n$  (Drop  $x_0 = 1$  as convention)

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

```
Repeat {
    for  $i = 1$  to  $m$ 
         $c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid
            closest to  $x^{(i)}$ 
    for  $k = 1$  to  $K$ 
         $\mu_k :=$  average (mean) of points assigned to cluster  $k$ 
}
```

*Remark.* If we get a cluster centroid with zero data points in it, the common thing to do is to entirely eliminate that centroid and get  $K - 1$  clusters. But if you really need  $K$  clusters, you can randomly reinitialize the centroid.

### 13.3.2 $K$ -means for non-separated clusters

**Example:** T-shirt sizing (S/M/L) based on weight and height.

### 13.3.3 Optimization Objective

**Notations**

$c^{(i)}$  = index of cluster  $(1, 2, \dots, K)$  to which example  $x^{(i)}$  is currently assigned

$\mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )

$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned

### Cost Function

Also called *distortion* cost function or *distortion* of the  $K$ -means algorithm.

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

### Optimization Objective

$$\min_{\substack{c^{(1)}, \dots, c^{(m)}, \\ \mu_1, \dots, \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

### 13.3.4 Random Initialization

Many ways, but this one usually works better in most cases. Should have  $K < m$ . Randomly pick  $K$  training examples as cluster centroids.

### Local Optima

On bad days,  $K$ -means can get stuck in local optima. To avoid, we can initialize multiple times. Concretely, repeat about 100 times (usually 50 to 1000 times) and choose the way with least distortion.

For small  $K$  (2 to 10), multiple initialization is likely to give better solution while for larger  $K$ s, it's highly likely that your first solution was pretty good already and not make a huge difference.

### 13.3.5 Choosing $K$

Still the most common thing is to choose the number of clusters by hand. Below are other thoughts.

### Elbow Method

Plot distortion against the number of clusters. Find the *elbow* in the plot. Maybe that is a good number of clusters. Pretty good way if the graph goes down rapidly and then slowly later. But, not used that often coz often the elbow is very often very ambiguous.

Sometimes, we run  $K$ -means to get clusters for some later purpose. Evaluate based on metric for how well it performs for that later purpose.

---

Check Lecture13.pdf for more details.

# Chapter 14

## Dimensionality Reduction

### 14.1 Motivation

#### 14.1.1 Data Compression

For example reduce data from 2D to 1D ( $x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$ ). This also helps to reduce features along with reducing memory requirement. More interestingly, it also helps speed up our learning algorithms.

#### 14.1.2 Data Visualization

Helps us to visualize data in a better way since plotting a visualizing data in smaller dimensions is easier.

### 14.2 Principle Component Analysis

#### 14.2.1 Problem Formulation

*Remark.* Before applying *PCA* perform mean normalization and feature scaling.

The goal of *PCA*, if we want to reduce data from 2D to 1D, is, we're going to find a vector in 2D to project the data minimizing the projection error.

More generally, to reduce from  $n$ -dimensions to  $k$ -dimensions, find  $k$  vectors  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  onto which to project the data, so as to minimize the projection error.

*Remark.* *PCA* is not *Linear Regression*

## 14.3 Algorithm

### 14.3.1 Data Preprocessing

Important! Always perform mean normalization and feature scaling.

### 14.3.2 Procedure

- Find component vectors  $u^{(i)}$ s
- Find reduced data values  $z^{(i)}$ s

Reduce data from  $n$ -dimensions to  $k$ -dimensions.

**Compute *covariance matrix*:**

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

if  $X = \begin{bmatrix} -- & (x^{(1)})^T & -- \\ & \vdots & \\ -- & (x^{(m)})^T & -- \end{bmatrix}$

$$\Sigma = \frac{1}{m} X^T X$$

**Compute *eigenvectors* of matrix  $\Sigma$ :** SVD stands for *Singular Value Decomposition*

$$[U, S, V] = \text{svd}(\Sigma);$$

There's also the `eig` function but `svd` is more numerically stable, though when you apply it on  $\Sigma$ , it'll give you the same values and that's because  $\Sigma$  satisfies *Symmetric Positive Semidefinite*.

*Remark.*  $\Sigma \in \mathbb{R}^{n \times n}$

All we need from `svd` function is the  $U \in \mathbb{R}^{n \times n}$  matrix.

The columns of matrix  $U$  will be exactly vectors  $u^{(i)}$ s we want.

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$



Just take the first  $k$  vectors.

$$\begin{aligned}
 U_{reduce} &= \begin{bmatrix} | & | & \dots & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & \dots & | \end{bmatrix} \in \mathbb{R}^{n \times k} \\
 z &= \begin{bmatrix} | & | & \dots & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & \dots & | \end{bmatrix}^T x \in \mathbb{R}^k \\
 &= \begin{bmatrix} -- & (u^{(1)})^T & -- \\ & \vdots & \\ -- & (u^{(k)})^T & -- \end{bmatrix} x \in \mathbb{R}^k
 \end{aligned}$$

Full procedure in Octave will look like:

```

Σ = (1/m)*X'*X;
[U, S, V] = svd(Σ);
Ureduce = U(:, 1:k);
z = Ureduce'*x;

```

### 14.3.3 Reconstruction

$$\begin{aligned}
 z &= U_{reduce}^T x \\
 x_{approx} &= U_{reduce} z
 \end{aligned}$$

## 14.4 Choosing the number of principle components

$$\text{Average squared projection error: } \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$$

$$\text{Total variation in the data: } \frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$$

Typically, choose  $k$  to be the smallest value so that

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01 \text{ (1\%)}$$

99% of variance is retained

Other popular numbers are 95% or 90% variance retained.

### 14.4.1 Algorithm

One way is to try PCA with  $k = 1, 2, \dots$  and calculate variance retained till you get 99% or more. This procedure is horribly inefficient.

From  $[U, S, V] = \text{svd}(\Sigma)$ ,  $S \in \mathbb{R}^{n \times n}$  is a diagonal matrix. For, a given  $k$ :

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} = 1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}}$$

So, we can test:

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$$

This is much more efficient!

## 14.5 Applying PCA

### 14.5.1 Supervised Learning Speedup

$(x^{(i)}, y^{(i)})$  where say  $x^{(i)} \in \mathbb{R}^{10000}$

1. Extract Inputs: Unlabelled dataset ( $x^{(i)} \in \mathbb{R}^{10000}$ )
2. Apply PCA (say 10x saving) ( $x^{(i)} \rightarrow z^{(i)} \in \mathbb{R}^{1000}$ )
3. New Training Set:  $(z^{(i)}, y^{(i)})$
4. For test example  $x$ , map using same PCA to get  $z$

*Remark.* Mapping  $x^{(i)} \rightarrow z^{(i)}$  should be defined by running PCA only on the *Training Set*. This mapping can be applied as well to the examples in cross-validation and test sets.

### 14.5.2 Visualization

Choose  $k = 2$  or  $k = 3$  coz we know how to plot only in 2 or 3 dimensions.

### 14.5.3 Bad use of PCA: To prevent overfitting

Use  $z^{(i)}$  instead of  $x^{(i)}$  to reduce the number of features to  $k < n$ . Thus, fewer features, less likely to overfit.

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

Bad because it throws away information.

### 14.5.4 PCA is sometimes used where it shouldn't be

Design of ML system:

1. Get training set
2. Run PCA to reduce dimensions (features)
3. Train
4. Test

Good question to ask: How about doing the whole thing without PCA?

Before implementing PCA, first try running whatever you want to do with the original/raw data. Only if that doesn't do what you want, then implement PCA.

# Chapter 15

## Anomaly Detection

### 15.1 Example

#### 15.1.1 Aircraft Engine!

**Features:**

$x_1$  = heat generated  
 $x_2$  = vibration intensity  
...

**Dataset:**  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

**Question:** Is  $x_{test}$  anomalous?

**Answer:** Model  $p(x)$  from dataset, and if  $p(x_{test}) < \epsilon$  then flag anomaly otherwise OK!

#### 15.1.2 Fraud Detection

Identify unusual users by checking  $p(x) < \epsilon$

#### 15.1.3 Manufacturing

Just like Aircraft Engine

#### 15.1.4 Monitoring computers in a data center

Check which system is probably required a review by a system administrator.

## 15.2 Gaussian (Normal) Distribution

### 15.2.1 Definition

Say  $x \in \mathbb{R}$ . If  $x$  is distributed *Gaussian* with mean  $\mu$  and variance  $\sigma^2$  ( $\sigma$  is the standard deviation).

$$x \sim \mathcal{N}(\mu, \sigma^2)$$
$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Bell shaped curve centered at  $\mu$  and width varying by  $\sigma$

### 15.2.2 Parameter Estimation

Given a dataset, estimate  $\mu$  and  $\sigma$  or  $\sigma^2$

**Maximum Likelihood Estimate (MLE):**

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

## 15.3 Algorithm

### 15.3.1 Density Estimation

Training Set:  $\{x^{(1)}, \dots, x^{(m)}\}$

Each example is  $x \in \mathbb{R}$

Model  $p(x)$  as such:

$$p(x) = p(x_1)p(x_2)\dots p(x_n)$$

Assume each feature is distributed Gaussian independently

$$\begin{aligned}x_1 &\sim \mathcal{N}(\mu_1, \sigma_1^2) \\x_2 &\sim \mathcal{N}(\mu_2, \sigma_2^2) \\&\vdots \\x_n &\sim \mathcal{N}(\mu_n, \sigma_n^2) \\p(x) &= p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) \dots p(x_n; \mu_n, \sigma_n^2) \\&= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)\end{aligned}$$

In practice, it works fine even if the features are not really *independent*.

### 15.3.2 Anomaly Detection Algorithm

1. Choose features  $x_i$  that might be indicative of anomaly.
2. Given a training set  $\{x^{(1)}, \dots, x^{(m)}\}$ , fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\begin{aligned}\mu &= \frac{1}{m} \sum_{i=1}^m x^{(i)} \\\sigma^2 &= \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2\end{aligned}$$

3. Given a new example  $x$ , compute  $p(x)$

$$\begin{aligned}p(x) &= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) \\&= \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)\end{aligned}$$

4. Anomaly if  $p(x) < \epsilon$

## 15.4 Developing and Evaluating an Anomaly Detection System

### 15.4.1 The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labelled data, of anomalous and non-anomalous examples. ( $y = 0$  if normal,  $y = 1$  if anomalous).

**Training set:**  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal)

**Cross validation set:**  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

**Test set:**  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Assume we include examples in cross validation set and test set which are known to be anomalous.

### Aircraft Engines Example

Recommended split:

- 10000 good (normal) engines
- 20 - 50 flawed (anomalous) engines
- Training Set: 6000 good engines
- CV: 2000 good engines, 10 anomalous
- Test: 2000 good engines, 10 anomalous

### 15.4.2 Algorithm Evaluation

1. Fit model  $p(x)$  on training set  $\{x^{(1)}, \dots, x^{(m)}\}$
2. On a cross-validation/test example  $x$ , predict

$$y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{otherwise} \end{cases}$$

3. Possible evaluation metrics

- True positive, false positive, false negative, true negative
- Precision/Recall
- $F_1$ -score

4. Can also use cross validation set to choose parameter  $\epsilon$

## 15.5 Anomaly Detection vs Supervised Learning

If we have labelled data, why not use supervised learning?

Anomaly Detection	Supervised Learning
Very small number of positive examples ( $y = 1$ ).	Large number of positive and negative examples.
Large number of negative examples ( $y = 0$ ).	
Many different <i>types</i> of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like.	Enough positive examples for algorithm to get a sense of what positive examples are like.
Future anomalies may look nothing like any of the anomaly examples we've seen so far.	Future positive examples likely to be similar to ones in training set.
Fraud Detection	Email spam classification
Manufacturing (e.g. aircraft engine)	Weather prediction (sunny/rainy/etc.)
Monitoring machines in a data center	Cancer classification
$\vdots$	$\vdots$

## 15.6 Choosing what features to use

### 15.6.1 Transformations

Works fine even if data isn't gaussian, but usually is a good sanity check for a feature.

If not gaussian, play with different kinds of transformations and get something similar to gaussian, like log transformation.



Other transformations:

$$x_1 \leftarrow \log x_1$$

$$x_1 \leftarrow \log x_2 + c$$

$$x_3 \leftarrow \sqrt{x_3} = x_3^{\frac{1}{2}}$$

$$x_4 \leftarrow x_4^{\frac{1}{3}}$$

### 15.6.2 Error analysis for anomaly detection

Want  $p(x)$  large for normal examples  $x$ .

$p(x)$  small for anomalous examples  $x$ .

**Most common problem:**  $p(x)$  is comparable (say, both large) for normal and anomalous examples.

Look at such anomalous examples and try to get a new feature to distinguish.

### 15.6.3 Other techniques:

Choose features that might take on unusually large or small values in the event of an anomaly.

**Example:** Monitoring computers in a data center running servers.

$x_1$  = memory use

$x_2$  = disk access

$x_3$  = CPU load

$x_4$  = network traffic

Possible anomaly = infinite loop, so  $x_5 = \frac{\text{CPU Load}}{\text{network traffic}}$

## 15.7 Multivariate Gaussian (Normal) Distribution

### Example

Monitoring machines in a data center. Consider  $x_1$  = CPU load and  $x_2$  = Memory Use. An outlier in 2D plot would not be triggered anomalous when treating  $x_1$  and  $x_2$ .

$x \in \mathbb{R}^n$ . Don't model  $p(x_1), p(x_2), \dots$ , etc. separately. Model  $p(x)$  all in one go.

**Parameters:**  $\mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$  (covariance matrix)

**Equation:**

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

where  $|\Sigma|$  = determinant of  $\Sigma = \text{det}(\Sigma)$  (in octave)

## 15.8 Anomaly Detection using Multivariate Gaussian Distribution

### 15.8.1 Model

**Parameters:**  $\mu, \Sigma$

**Parameter Fitting:** Given training set  $\{x^{(1)}, \dots, x^{(m)}\}$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$
$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

### 15.8.2 Anomaly Detection

1. Fit model  $p(x)$  by setting  $\mu, \Sigma$
2. Given a new example  $x$ , compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

3. Flag an anomaly if  $p(x) < \epsilon$

### 15.8.3 Relationship to original model

Original model corresponds to multivariate gaussian where the contours of the gaussian are axis aligned. The constraint (mathematically) is

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & 0 & \dots & 0 \\ 0 & 0 & \ddots & & 0 \\ 0 & \dots & 0 & \sigma_{n-1}^2 & 0 \\ 0 & \dots & 0 & 0 & \sigma_n^2 \end{bmatrix}$$

#### Original Model

$$\prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

Manually create features to capture anomalies where  $x_1, x_2$  take unusual combination of values.

Computationally cheaper (scales better to large  $n$ , say 100,000).

OK even if  $m$  (training set size) is small.

#### Multivariate Gaussian

$$\frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

Automatically captures correlations between features.

Computationally more expensive.

Must have  $m > n$ , or else  $\Sigma$  is non-invertible. (Typical rule of thumb:  $m \geq 10n$ )

*Remark.* If  $\Sigma$  is singular (non-invertible), either  $m > n$  fails, or you have redundant (linearly dependent) features.

# Chapter 16

## Recommender Systems

### 16.1 Problem Formulation

**Example:** Predicting movie ratings! Collect movie ratings from users for the movies they have watched, and predict ratings for the movies they haven't watched.

**Notations:**

$n_u$  = no. users

$n_m$  = no. movies

$r(i, j) = 1$  if user  $j$  has rated movie  $i$

$y^{(i,j)}$  = rating given by user  $j$  to movie  $i$   
(defined only if  $r(i, j) = 1$ )

### 16.2 Content-Based Recommendations

Features could be the degree of genre of the movie, say romance or action.

One way could be, for each user  $j$ , learn a parameter  $\theta^{(j)} \in R^{n+1}$ . Predict user  $j$  as rating movie  $i$  with  $(\theta^{(j)})^T x^{(i)}$  stars.

### Notations:

$n$  = no. features

$\theta^{(j)}$  = parameter vector for user  $j$

$x^{(i)}$  = feature vector for movie  $i$

$(\theta^{(j)})^T x^{(i)}$  = predicted rating for user  $j$ , movie  $i$

$m^{(j)}$  = no. movies rated by user  $j$

To learn  $\theta^{(j)}$ :

$$\min_{\theta^{(j)}} \left( \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^n (\theta_k^{(j)})^2 \right)$$

For recommender systems, we make some simplifications:

- Get rid of the constant  $m^{(j)}$

### 16.2.1 Optimization Objective

To learn  $\theta^{(j)}$  (parameter for user  $j$ ):

$$\min_{\theta^{(j)}} J(\theta^{(j)})$$
$$\min_{\theta^{(j)}} \left( \frac{1}{2} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2 \right)$$

To learn  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} J(\theta^{(1)}, \dots, \theta^{(n_u)})$$
$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \left( \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2 \right)$$

### 16.2.2 Optimization Algorithm

Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k = 0)$$

## 16.3 Collaborative Filtering

Feature Learning!

Each user  $j$  just tells us how much they like different types of movies  $\theta^{(j)}$ . If we get these parameters from the users, then it is possible to infer what are the features for each movie from the ratings.

### Optimization Algorithm

Given  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(i)}$ :

$$\min_{x^{(i)}} \left( \frac{1}{2} \sum_{j:r(i,j)=1} \left( \left( \theta^{(j)} \right)^T \left( x^{(i)} \right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n \left( x_k^{(i)} \right)^2 \right)$$

Given  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(i)}} \left( \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left( \left( \theta^{(j)} \right)^T \left( x^{(i)} \right) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n \left( x_k^{(i)} \right)^2 \right)$$

### Collaboration

Guess  $\theta \rightarrow x \rightarrow \theta \rightarrow x \rightarrow \dots$

#### 16.3.1 Algorithm

Minimizing  $x^{(1)}, \dots, x^{(n_m)}$  and  $\theta^{(1)}, \dots, \theta^{(n_u)}$  simultaneously:

$$\begin{aligned} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) &= \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left( \left( \theta^{(j)} \right)^T x^{(i)} - y^{(i,j)} \right)^2 \\ &\quad + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n \left( x_k^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n \left( \theta_k^{(j)} \right)^2 \end{aligned}$$

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$

When learning features this way, we're going to do away with  $x_0 = 1$  and hence, now our  $x \in \mathbb{R}^n$  and  $\theta \in \mathbb{R}^n$ , and the reason is that now that we're learning the features, if there is a need for such a feature, the algorithm can learn it by itself.

### Collaborative Filtering Algorithm

1. Initialize  $x^{(1)}, \dots, x^{(n_m)}$  and  $\theta^{(1)}, \dots, \theta^{(n_u)}$  to small random values (just like neural network).
2. Minimize  $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$  using Gradient Descent (or an advanced optimization algorithm).  
E.g. for every  $j = 1, \dots, n_u, i = 1, \dots, n_m$ :

$$x_k^{(i)} := x_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

3. For a user with parameters  $\theta$  and a movie with (learned) features  $x$ , predict a star rating of  $\theta^T x$ .

## 16.4 Vectorization: Low Rank Matrix Vectorization

### 16.4.1 Vectorization

$$Y = [y^{(i,j)}]$$

$$= [(\theta^{(j)})^T x^{(i)}]$$

$$X = \begin{bmatrix} \text{---} & (x^{(1)})^T & \text{---} \\ \text{---} & (x^{(2)})^T & \text{---} \\ & \vdots & \\ \text{---} & (x^{(n_m)})^T & \text{---} \end{bmatrix}$$

$$\Theta = \begin{bmatrix} \text{---} & (\theta^{(1)})^T & \text{---} \\ \text{---} & (\theta^{(2)})^T & \text{---} \\ & \vdots & \\ \text{---} & (\theta^{(n_u)})^T & \text{---} \end{bmatrix}$$

$$Y = X\Theta^T$$

Name *Low Rank Matrix Vectorization* comes from the fact that  $X\Theta^T$  is a *Low Rank Matrix*.

### 16.4.2 Finding related movies

**Q:** For each product  $i$ , we learn a feature vector  $x^{(i)} \in \mathbb{R}^n$ . How to find movies  $j$  related to movie  $i$ ?

**A:** Small  $\|x^{(i)} - x^{(j)}\| \rightarrow$  movies  $j$  and  $i$  are similar.

5 most similar movies to  $i$ :

Find the 5 movies  $j$  with the smallest  $\|x^{(i)} - x^{(j)}\|$

## 16.5 Implementational Detail: Mean Normalization

Let's consider an example of a user who hasn't rated any movie. So in our optimization objective

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} \frac{1}{2} \sum_{(i,j): r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n \left( x_k^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n \left( \theta_k^{(j)} \right)^2$$

the first term plays no role, and hence we have to minimize  $\frac{\lambda}{2}(\theta_1^2 + \theta_2^2)$  which

will give  $\theta = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$  and so we will predict  $\theta^T x = 0$  for all  $x$ , which will give us no good way to recommend movies to the new user. So, calculate mean rating for each movie and subtract the mean from all ratings (so that each movie has a mean rating of zero).

$$Y = \begin{bmatrix} y^{(1,j)} \\ y^{(2,j)} \\ \vdots \\ y^{(n_m,j)} \end{bmatrix}$$
$$\mu = \begin{bmatrix} \text{mean}(y^{(1)}) \\ \text{mean}(y^{(2)}) \\ \vdots \\ \text{mean}(y^{(n_m)}) \end{bmatrix}$$

$$Y_{new} := Y - \mu$$

Now, learn  $\theta^{(j)}, x^{(i)}$  on this  $Y_{new}$ .

For user  $j$ , on the movie  $i$  predict:  $(\theta^{(j)})^T (x^{(i)} + \mu_i)$

---

Check Lecture16.pdf for more details.



# Chapter 17

## Large Scale Machine Learning

### 17.1 Learning with Large Datasets

Learning Algorithms usually work better when trained on larger datasets.

#### 17.1.1 Problems

Computational Expense! When  $m = 100,000,000$ , each update of a gradient takes much longer.

#### 17.1.2 Sanity Check

A method of sanity check that  $m = 1000$  would do just as well, plot  $J_{train}(\theta)$  and  $J_{cv}(\theta)$  and if it shows high variance, increasing  $m$  makes sense.

### 17.2 Stochastic Gradient Descent

Gradient Descent becomes very computationally expensive for large training sets. Modify to allow us to scale better.

### 17.2.1 Linear Regression

Linear Regression only for demonstration, works well for other problems.

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

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

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

In contrast to Batch Gradient Descent, we get new algorithm with does not have to go through each training example in each iteration.

### 17.2.2 Stochastic Gradient Descent

Rewrite as:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} \left( h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

$$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$$

**Algorithm:** Repeat 1 to 10 times:

```
1 Randomly shuffle dataset;
2
3 repeat {
4     for  $i = 1, \dots, m$  {
5          $\theta_j := \theta_j - \alpha \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$  (for all  $j$ )
6     }
7 }
```

### 17.3 Mini-Batch Gradient Descent

Can sometimes work even faster than Stochastic Gradient Descent!

Batch Gradient Descent : Use all  $m$  examples in each iteration

Stochastic Gradient Descent : Use 1 example in each iteration

Mini-Batch Gradient Descent : Use  $b$  examples in each iteration

where  $b$  = mini-batch size, usually  $\in [2, 100]$

### Algorithm

```
1 Randomly shuffle dataset;
2
3 repeat {
4     for  $i = 1, i + b - 1, i + 2 * b - 1, \dots$  {
5          $\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$ 
6         (for all  $j$ )
7     }
8 }
```

**Q:** Why  $b$  instead of 1?

**A:** Vectorization

**Disadvantage:** New parameter  $b$ .

## 17.4 Convergence

### 17.4.1 Checking for Convergence

- Previously, we could plot the cost function over time! Now?
- During Learning, compute  $cost_{\theta}(x^{(i)}, y^{(i)})$  before updating  $\theta$ .
- Every few (say 1000) iterations, plot cost averaged over the last 1000 examples processed by algorithm.
- Sometimes, a smaller learning rate can give you better optimization because Stochastic Gradient Descent doesn't really converge, rather oscillates near the optimal value, so smaller learning rate means less noisy oscillations.
- Increasing over the number of iterations to plot the cost may give you a smoother plot with the disadvantage of delayed feedback.
- Sometimes, it may look like the cost is not decreasing, but the decreasing trend may be visible by delaying the feedback.
- Increasing is a sign of divergence, lower your  $\alpha$ .

- In most cases of stochastic gradient descent, learning rate  $\alpha$  is typically held constant. But, you can decrease slowly if you want to converge like

$$\alpha = \frac{const1}{iterationNumber + const2}$$

## 17.5 Online Learning

Learn from incoming stream of data.

### 17.5.1 Example

Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ( $y = 1$ ), sometimes not ( $y = 0$ ). We want to optimize the asking price to offer our users.

Features  $x$  capture properties of user, of origin/destination and asking price. We want to learn  $p(y = 1|x;\theta)$  to optimize price.

We can choose, logistic regression?

### 17.5.2 Algorithm

```

1 repeat forever{
2     get (x, y) corresponding to user
3     update  $\theta$  using (x,y):
4          $\theta_j := \theta_j - \alpha(h_{\theta}(x) - y)x_j$ 
5         for all j
6 }
```

*Remark.* Can adapt to changing user preferences.

### 17.5.3 Other Examples

- Product search (learning to search)
  - User searches for "Android phone 1080p camera"
  - Have 100 phones in store. Will return 10 results.
  - $x$  = features of phone, how many words in user query match name of the phone, description of the phone, etc.
  - $y = 1$  if user clicks on link.  $y = 0$  otherwise.
  - Learn  $p(y = 1|x;\theta)$

- Click Through Rate (CTR)
- Use this to show user the 10 phones they're most likely to click on.
- Choosing special offers to show user
- Customized selection of news articles
- Product Recommendations

## 17.6 Map-reduce and data parallelism

Can be used to scale learning algorithm to far larger than possible with even stochastic gradient descent.

### 17.6.1 Map-reduce

#### Batch gradient descent

We are going to assume  $m = 400$  but this is probably applied where  $m = 400,000,000$  and four machines.

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

**Machine 1:** Use  $(x^{(1)}, y^{(1)}), \dots, (x^{(100)}, y^{(100)})$

$$temp_j^{(1)} = \sum_{i=1}^{100} \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

**Machine 2:** Similarly get  $temp_j^{(2)}$

**Machine 3:** Get  $temp_j^{(3)}$

**Machine 4:** Get  $temp_j^{(4)}$

**Master Server:** Combine

$$\theta_j := \theta_j - \alpha \frac{1}{400} \left( temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)} \right) \quad (\text{for all } j)$$

## 17.6.2 Summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set. Hence, can be parallelized easily using map-reduce.

### Example

Logistic Regression

$$J_{train}(\theta) = -\frac{1}{m} \sum_{i=1}^m \left( y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right)$$
$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

## 17.6.3 Multi-core machines

Parallelize to different cores!

*Remark.* Some numerical linear algebra libraries automatically parallelize for you!