Machine Learning notes

LWang

January 10, 2017

Contents

List of Figures

This notes are mostly based on Andrew Ng's Machine Learning course on Coursera

1 Day 1

1.1 Introduction

Machine learning is the field of study that gives computers the ability to learn without being explicitly progarmmed.

1.1.1 Supervised Learning

Supervised learning: Given a data set as the input, we have already know what the correct output should look like, alternatively, we have the idea the relationship between the input and the output. Supervised learning problems can be categorized into two kinds. One is "Regression", the other one is "Classification".

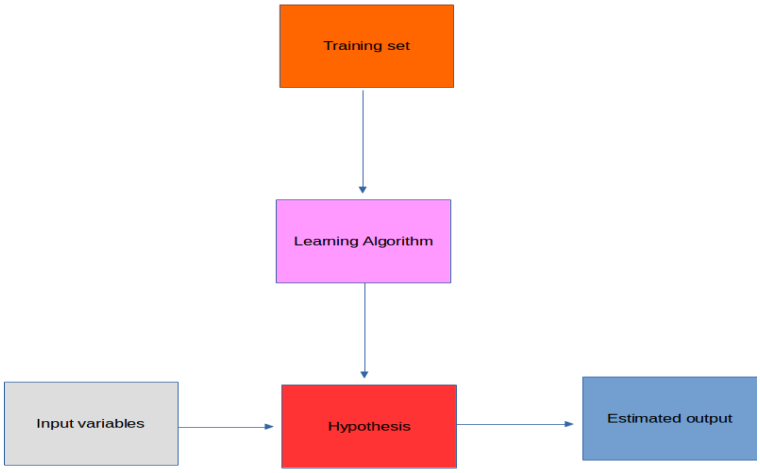


Figure 1: Supervised Learning

- Regression: The output predicted result is continuous, in other words, we map input variables to continuous outpust.
- Classification: The output predicted result is discrete. we map input variables to discrete categories.

Examples:
(a)Regression: Given the price of rent in Beijing, we have to estimate the annual living expenditure.
(b)Classification: Given an email, we have to pridict it is spam or normal.

1.1.2 Unsupervised learning

Unsupervised learning: We can derive structure from data where we don't necessarily know the effect of the variables. It allows us to approach or address problems with little or no information

what our results should look like. With unsupervised learning there is no feedback on the prediction results. Examples:

- Clustering: Take a collection of 1,000,000 different genes, and find a way to automatically group these genes into groups that are somehow similar or related by different variables, such as lifespan,location,roles, and so on.
- Non–clustering: The "Cocktail Party Algorithm", allows you find structure in a chaotic enviroment.(i.e.identifying individual voices and music from a mesh of sounds at a cocktail party).

1.2 Model and Cost Function

1.2.1 Model

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

input and output:

$$(x, y)$$

1.2.2 Cost function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta^{(i)} - y^{(i)})^2 \tag{1.2.1}$$

We measure the accuracy of our hypothesis function by employing cost function,It takes the average of the square error. The goal is to minimize the cost function, it is also called "Square error function"

1.2.3 Gradient Descent

The gradient decscet algorithm is : repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\alpha}{\alpha \theta_j} J(\theta_0, \theta_1) \tag{1.2.2}$$

where j=0,1 represent the feature index numbe, α is the learning rate, m is the size of training set.
When the sign of the derivative at that point is positive, the value of the parameter decreases, when the sign is negative, the value of the parameter increases. However, regardless the sign of $\frac{\alpha}{\alpha \theta_j} J(\theta_0, \theta_1)$, θ_0 and θ_1 will converge to their minimum value eventually. To ensure a reasonable computing time, the choice of parameter α is significant, if α is too small, gradient descent can be slow, if α is large, gradient descent can overshoot the minimum, causing failure to converge.

2 Day 2

2.1 Multiple Features

To start with, let's clarify some notations by defining

$$\begin{aligned} x_j^{(i)} &= \text{value of feature j in the } i^{th} \text{ training example} \\ m &= \text{the number of training examples}, i = 1, 2 \dots m \\ n &= \text{ the number of features}, j = 0, 1, 2 \dots n \end{aligned}$$

For any example, the hypothesis is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \ldots + \theta_n x_n \tag{2.1.1}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

where x_0=1.
For simplicity, we rewrite the above equation in matrix notation as

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x \tag{2.1.2}$$

For m training examples, the variable matrix X is of size $m \times n$, with each example as a row, and the features as columns.

$$\mathbf{X} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2))} & \dots & x_j^{(2)} & \dots & x_n^{(2)} \\ x_0^{(i)} & x_1^{(i)} & x_2^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_0^{m} & x_1^{(m)} & x_2^{(m)} & \dots & x_j^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

we can caculate the hypothesis as a column vector of size $(m \times 1)$ with:

$$h_\theta(X) = X\theta \tag{2.1.3}$$

Now the cost function for multiple variables is

$$J(\theta_0, \theta_1 \dots \theta_n) = \frac{1}{2m} \sum_i^m (h_\theta^{(i)} - y^{(i)})^2 \tag{2.1.4}$$

Gradient descent

$$\theta_j = \theta_j - \alpha \frac{\alpha}{\alpha \theta_j} J(\theta_0, \theta_1 \dots \theta_n) \qquad \text{simultaneously update for every j =0,1 } \dots \text{ n} \tag{2.1.5}$$

To be more clear, the algorithm runs like this:
For $n \geq 1$

$$\begin{aligned} \theta_0 &= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta^{(i)} - y^{(i)}) x_0^{(i)} \\ \theta_1 &= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta^{(i)} - y^{(i)}) x_1^{(i)} \\ &\vdots \\ \theta_n &= \theta_n - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta^{(i)} - y^{(i)}) x_n^{(i)} \end{aligned}$$

repeat until convergence.

2.2 Feature Scaling

Idea:all the features are on same scale, get each feature into approximately a $-1 \leq x \leq 1$ methods: dividing the input values by the range.

2.3 Mean normalization

implementing both techniques, the new variable is

$$x_i = \frac{x_i - \mu_i}{x_{max} - x_{min}} \tag{2.3.1}$$

2.4 Comparison between gradient descent and normal equation

$$\text{Normal equation} \qquad \theta = (x^T x)^{-1} x^T y \tag{2.4.1}$$

3 Week 3

3.1 Notes on Classification and Logistic regression

One method to pproach classification problems is to employ linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as 0. However, this method doesn't work well because classification is not actually a linear function. Alternatively, we use Logistic Function(Sigmoid Function)

$$h_\theta(x) = g(\theta^T x) \tag{3.1.1}$$

$$z = \theta^T x \tag{3.1.2}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.1.3}$$

The function g(z), maps any real number to the (0,1) interval, making it useful for transforming an arbitrary–values function into a function better suited for classification. Next, we translate the output of the hypothesis function as follows:

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) \leq 0.5 \rightarrow y = 0$$

Since

$$z \geq 0 \rightarrow g(z) \geq 0.5$$

and we have eq(3.1.2), we are confidently to say

$$\theta^T x \geq 0 \rightarrow y = 1 \tag{3.1.4}$$

$$\theta^T x \leq 0 \rightarrow y = 0 \tag{3.1.5}$$

$$\tag{3.1.6}$$

The decision boundary is the line that separate the area where y=0 and where y=1. It is created by our hypothesis function.

3.2 Cost function for Classification

$$J_{(\theta)} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{Cost}(h_\theta(x^{(i)}), y^{(i)}) \tag{3.2.1}$$

$$\mathbf{Cost}(h_\theta(x), y) = -\log(h_\theta(x)), \quad \textbf{if} \quad y = 1 \tag{3.2.2}$$

$$\mathbf{Cost}(h_\theta(x), y) = -\ log(1 - h_\theta(x)), \quad \textbf{if} \quad y = 0 \tag{3.2.3}$$

$$\tag{3.2.4}$$

Alternatively, we can re-express as the follows:

$$\mathbf{Cost}(h_\theta(x), y) = 0, \quad \textbf{if} \quad h_\theta(x) = y$$
$$\mathbf{Cost}(h_\theta(x), y) \rightarrow \infty, \quad \textbf{if} \quad y = 0 \quad \textbf{and} \quad h_\theta(x) \rightarrow 1$$
$$\mathbf{Cost}(h_\theta(x), y) \rightarrow \infty, \quad \textbf{if} \quad y = 1 \quad \textbf{and} \quad h_\theta(x) \rightarrow 0$$

We can fully write our entire cost function as follows:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)})] \tag{3.2.5}$$

3.3 Advanced Optimization

Optimization algorithms:
– Gradient descent
– Conjugate gradient
– BFGS
– L-BFGS
Advantages: – No need to manually pick α
– Often faster than gradient descent
Disadvantages: More complex

3.4 Multiclass Classification

$$h_\theta^{(i)}(x) = P(y=i|x,\theta) \qquad (i=1,2,3) \tag{3.4.1}$$

The idea is to train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$. On a new input x, to make a prediction, pick the class i that maximizes

$$\max h_\theta^{(i)}(x)$$

To be more clear, we are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

3.5 Softmax Regression

In this part, we use an example to demonstrate using softmax regression for multi–class classification problems.

MNIST is a dataset consists of handwritten digits, each image has been labeled with number (0,1,2,3,4,5,6,7,8,9) agrees with its appearence. In this tutorial , we build up a model by traing the dataset and use this model to predict what digits are they for a test dataset. The MINIST data consists of three parts: 55,000 data points of training data,(minist.train), 10,000 points of test data, and 5,000 points of cross validation data. The reason for doing this is to make sure our training model can be generalized to fit new dataset.

Each image has dimension 28× 28 , each pixel has value between (0,1), 0 represents black, 1 presents white, and we flat this 2D matrix into a vector of 28× 28 =784 numbers. now, the train dataset has dimension 55,000× 784 , same for test dataset of dimension 10,000× 784 , and cross validation data points of 5,000× 784. Consequently, the train labels is a 55,000× 10 matrix, for example, if the label is 4 for an image, then it would ba a vector of [0,0,0,1,0,0,0,0,0,0], the fouth number is 1, and zero for the rest.

Since we have to choose one label from ten to assign test images, and we have labeled train datasets, this is a supervised multi–class classification problem, Softmax regression is the right way to go, because it gives us a list of probabilites between 0 and 1 that add up to 1.

The steps of softmax:

- step 1, computing evidentce for each class k given an input example vector x_i , $z_k^{(i)} = \Theta_k^T x^{(i)}$, for (k = 1,...,K), Θ has dimension $(n+1)\times K$, and x has dimension $(n+1)\times m$.
- step 2, $\text{softmax}_k = \frac{\exp(z_k)}{\sum_{j=1}^{K}\exp(z_j)}$, it can be interpreted as a specific probabilitie of K cases, and $y_k^{(i)} = \text{softmax}(\Theta_k^T x_i)$.

To be more clear, our hypothesis function $h_\theta(x^{(i)})$ becomes

$$\begin{aligned} h_\theta(x^{(i)}) &= \begin{bmatrix} p(y^{(i)}=1|x^{(i)},\Theta) \\ p(y^{(i)}=2|x^{(i)},\Theta) \\ \vdots \\ p(y^{(i)}=K|x^{(i)},\Theta) \end{bmatrix} \\ &= \frac{1}{\sum_{k=1}^{K} e^{\Theta_k^T x^{(i)}}} \begin{bmatrix} e^{\Theta_1^T x^{(i)}} \\ e^{\Theta_2^T x^{(i)}} \\ \vdots \\ e^{\Theta_K^T x^{(i)}} \end{bmatrix} \\ &= \frac{1}{\sum_{k=1}^{K} e^{z_k^{(i)}}} \begin{bmatrix} e^{z_1^{(i)}} \\ e^{z_2^{(i)}} \\ \vdots \\ e^{z_K^{(i)}} \end{bmatrix} \end{aligned}$$

- compute cost function

$$\begin{aligned} J(\Theta) &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\Big[(y^{(i)}==k)\log h_\theta(x^{(i)})\Big] \\ &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\Big[(y^{(i)}==k)\log p(y^{(i)}=k|x^{(i)},\Theta)\Big] \end{aligned} \tag{3.5.1}$$

In fact eq(3.5.1) is usually called cross entropy cost function, actually, it is just a two class logistic cost function.

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\Big] \tag{3.5.2}$$

- Compute gradient

$$\begin{aligned} \frac{\partial J(\Theta)}{\partial \Theta_j} &= \frac{\partial J(\Theta)}{\partial z_j}\frac{\partial z_j}{\partial \Theta_j} \\ &= -\frac{1}{m}\sum_{i=1}^{m}\Big(1-p(y^{(i)}=j|x^{(i)},\Theta)\Big)x^{(i)} \\ &= \frac{1}{m}\sum_{i=1}^{m}\Big(p(y^{(i)}=j|x^{(i)},\Theta)-1\Big)x^{(i)} \end{aligned} \tag{3.5.3}$$

4 Week 5

4.1 Cost function

Let's first clarify a few notations that we will need to use:

- L=total number of layers in the network
- s_l=number of units(not counting bias unit)in layer l
- K=number of output units/classes

We denote $h_\theta(x)_k$ as being a hypothesis that results in the k^{th} output. Recall that the cost function for regularized logistic regression was:

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

It is going to be a little bit complicated for neural networks:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\Big[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k)+(1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\Big]+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}(\Theta_{j,i}^{(l)})^2$$

Note:

- In neural networks, all the outputs are summed to the cost function.
- The i in the triple sum does not refer to training example i.

4.2 Overfitting problem

Fitting problems:
– Underfit, high bias: The model doesnot fit the data set very well (the data structure is not captured by the model), the fitting is unsuccessful.
– Overfit, high variance: If we have too many features to describe, the learned hypothesis may fit the training data set very well ($J\theta = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta^{(i)}(x)-y^{(i)}) \approx 0$), but fail to make accurate predictions on new examples.
Ways to address overfitting:
1. Reduce number of features
– Manually select which features to keep
– Model selection algorithm
2. Regularization
– Keep all the features, but reduce mangitude/values of parameters θ_j
– Works well when we have a lot of features, each of which contributes a bit to predicting y

4.3 Cost Function

If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their cost. For example, we have a data set, and we make the following hypothesis,

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

If we want to eliminate the influence of $\theta_3 x^3$ and $\theta_4 x^4$. Without actually getting rid of these features or changing the form of out hypothesis, we can instead modify our cost function.

$$\mathbf{min}_\theta\{\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\ldots\theta_3^2 + 1000\ldots\theta_4^2\}$$

The two extra termshe at the end to inflate the cost of θ_3 and θ_4. Therefore, in order to minimize the cost function to close to zero, the values of θ_3 and θ_4 will have to be reduced. This in turn will greatly reduce the values of $\theta_3 x^3$ and $\theta_4 x^4$ in our hypothesis function.

In general, we regularize all of our theta parameters in a single summation as:

$$\mathbf{min}_\theta\frac{1}{2m}\Big[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\Big]$$

where λ is called the regularization paremeter. It determines how much the costs of our theta parameters are inflated. The cost function with extra summation will allow us to smooth the output of our hypothesis function to reduce overfitting.

4.4 Regularized Linear Regression

Gradient Descent Repeat:

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

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

Normal equation

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$$

where

$$L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

L has dimension of $(n+1) \times (n+1)$, if m≤n, then $X^T X$ is non–invertible. However, the added term $\lambda \cdot L$, then $X^T X + \lambda \cdot L$ becomes invertible.

4.5 Regularized Logistic Regression

Cost function

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2 \tag{4.5.1}$$

5 Neural Networks

5.1 Model Representation

We use neural networksa to represent a hypothesis function, neurons are basically computational units that take inputs(dendrites) as electrical inputs that are channeled to outputs(axons).

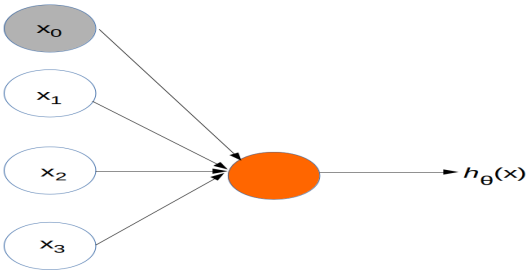
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

where $x_0 = 1$, it is a bias unit.

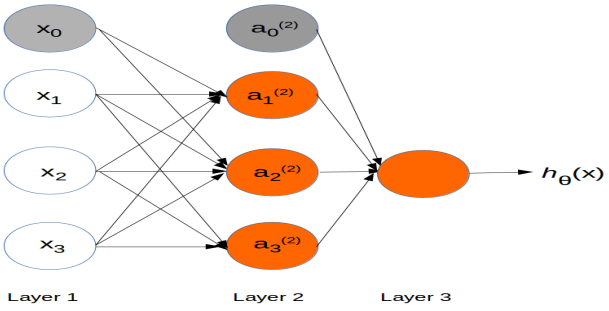
In Figure 2(a), there are only one input layer and one output, in Figure 2(b), there are one input layer(Layer 1), one hidden layer(Layer 2) and one output layer(Layer 3), both Layer 1 and Layer 2 have 3 units, where Layer 3 has only one unit.
$a_i^{(j)}$: "activation" of unit i in layer j.
θ^j: matrix of weights controlling function mapping from layer j to layer j+1
$h_\theta(x) = \frac{1}{1+e^{(-\theta^T X)}}$

$$a_1^{(2)} = g(\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$$
$$a_2^{(2)} = g(\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$$
$$a_3^{(2)} = g(\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$$
$$h_\theta(x) = a^3 = g(\theta_{10}^2 a_0^{(2)} + \theta_{11}^2 a_1^{(2)} + \theta_{12}^2 a_2^{(2)} + \theta_{13}^2 a_3^{(2)})$$

where $a_0^{(2)} = 1$ is an added term.
If network has s_j units in layer j, s_{j+1} units in layer j+1, then $\theta_{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.



(a) Neuron Model 1



(b) Neuron Model 2

Figure 2: Neural Network

5.2 Multi–class classification

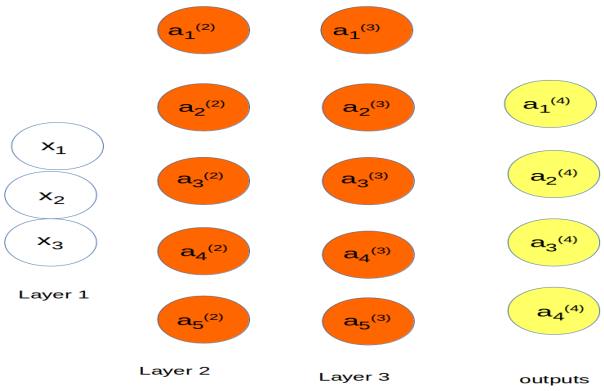


Figure 3: Multiple outputs

$h_\theta(x) \in \mathbb{R}^4$ .More specifically,

$$h_\theta(x) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \textbf{ or } \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \textbf{ or } \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \textbf{ or } \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

5.3 Backpropagation Algorithm

A network forward propgates activation to produce an output and it backward propgates error to determine weight parameters. The goal of Backpropagation Algorithm in neural network is to minimize cost function(the sum squared error between the network's output values and the given target values), $\min_\theta J(\Theta)$, more specifically, we want to compute $\frac{\partial}{\partial \Theta_{j,i}^{(l)}} J(\Theta)$

The algorithm runs as follows:

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

set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j), we obtain a matrix full of zeros.

For training example t= 1 to m:

1. Set $a^{(1)} := X$, add x_0
2. Perform forward propgation to compute $a^{(l)}$, for l=2,3...L

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

3.Using $y^{(t)}$, compute error values $\delta^{(L)} = a^{(L)} - y^{(t)}$.

4.Compute error values for each layer, $\delta^{(L-1)}, \delta^{(L-2)} \dots \delta^{(2)}$, where

$$\delta^{(l)} = \left((\Theta^{(l)})^T \delta^{(l+1)}\right) * g'(z^{(l)})$$

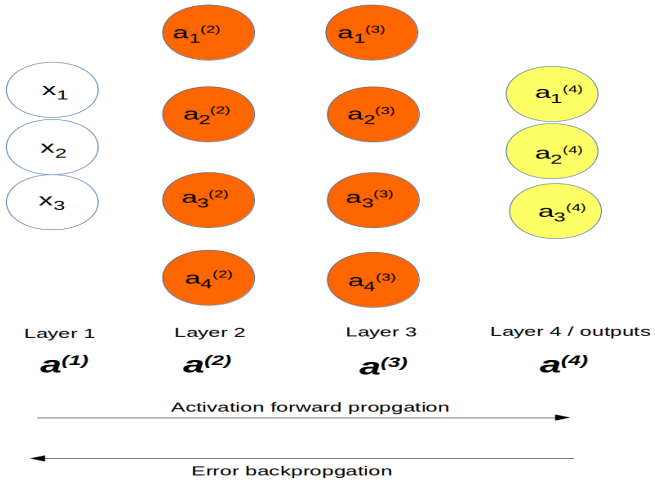


Figure 4: 4 layers neural networks

The g–prime derivative term can also be written out as :

$$g'(z^{(l)} = a^{(l)} .* (1 - a^{(l)})$$

5.Update term.$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$
$D_{i,j}^{(l)} = \frac{1}{m}(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}), \quad \text{if} \quad j \neq 0$
$D_{i,j}^{(l)} = \frac{1}{m}(\Delta_{i,j}^{(l)}, \quad \text{if} \quad j = 0$
where $\frac{\partial}{\partial\Theta_{j,i}^{(l)}}J(\Theta) = D_{(i,j)}^{(l)}$

More details about Backpropgation: notations:

- L: layers of neural network
- s_l : number of units in the lth layer(no bias term counted)
- $z_i^{(l)}$: the ith input in the lth layer(i=1,2,... s_l)
- $a_i^{(l)}$: the ith output in the lth layer(i=0,1,...,s_l)
- $\Theta_{ij}^{(l)}$: parameter (j=0,1,...,s_l, i= 1,2,...,s_{l+1})
- $h_\Theta(x)$: the output of the neural network
- $J(\Theta)$: the cost function
- $\delta_i^{(l)}$: the computation error in the ith unit in the lth layer(i=1,2,...,s_l)

The goal is to compute $\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$ for all i, j, l.

$$\frac{\partial J(\Theta)}{\partial\Theta_{ij}^{(l)}} = \underbrace{\frac{\partial J(\Theta)}{\partial a_i^{(l+1)}}}_{\text{part 1}} \cdot \underbrace{\frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}}}_{\text{part 2}} \cdot \underbrace{\frac{\partial z_i^{(l+1)}}{\partial\Theta_{ij}^{(l)}}}_{\text{part 3}} \qquad \text{(using chain rule)} \tag{5.3.1}$$

$$\begin{aligned}
\text{part 1} = \frac{\partial J(\Theta)}{\partial a_i^{(l+1)}} &= \sum_{j=1}^{s_{l+2}} \frac{\partial J}{\partial z_j^{(l+2)}} \frac{\partial z_j^{(l+2)}}{\partial a_i^{(l+1)}} \\
&= \sum_{j=1}^{s_{l+2}} \delta_j^{(l+2)} \frac{\partial}{\partial a_i^{(l+1)}} \Big(\sum_{k=1}^{s_l+1} \Theta_{jk}^{(l+1)} \cdot a_k^{(l+1)} \Big) \\
&= \sum_{j=1}^{s_{l+2}} \delta_j^{(l+2)} \Theta_{ji}^{(l+1)} \qquad (i = 1, 2, \ldots, s_{l+2}, \qquad \text{no bias term})
\end{aligned} \tag{5.3.2}$$

$$\text{part 2} = \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} = \frac{\partial}{\partial z_i^{(l+1)}} g(z_i^{(l+1)})$$
$$= g(z_i^{(l+1)})\big(1 - g(z_i^{(l+1)})\big)$$
$$= a_i^{(l+1)}\big(1 - a_i^{(l+1)}\big) \qquad (i = 1, 2, \ldots, s_{l+1}, \qquad \text{no bias term}) \tag{5.3.3}$$

$$\text{part 3} = \frac{\partial z_i^{(l+1)}}{\partial \Theta_{ij}^{(l)}} = \frac{\partial}{\partial \Theta_{ij}^{(l)}} \big(\sum_{k=0}^{s_l} \Theta_{ik} a_k^{(l)}\big)$$
$$= a_j^{(l)} \qquad (j = 0, 1, 2, \ldots, s_{l+1}, \qquad \text{bias term included}) \tag{5.3.4}$$

now, let's combine part 1 , 2 and 3,

$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \sum_{k=1}^{s_{l+2}} \delta_k^{(l+2)} \Theta_{ki}^{(l+1)} a_i^{(l+1)} \big(1 - a_i^{(l+1)}\big) a_j^{(l)} \tag{5.3.5}$$

since $\delta_i^{(l)} = \frac{\partial}{\partial z_i^{(l)}} J(\Theta)$, we will have

$$\delta_i^{(l+1)} = \frac{\partial}{\partial z_i^{(l+1)}} J(\Theta)$$

which also equals to

$$\delta_i^{(l+1)} = \text{part 1} \cdot \text{part 2}$$
$$\delta_i^{(l+1)} = \sum_{k=1}^{s_{l+2}} \delta_k^{(l+2)} \Theta_{ki}^{(l+1)} a_i^{(l+1)} \big(1 - a_i^{(l+1)}\big)$$
$$\delta^{(l+1)} = (\Theta^{(l+1)})^T \delta^{(l+2)} a^{(l+1)} \big(1 - a^{(l+1)}\big) \tag{5.3.6}$$

Rewrite eq(6.2.5), the gradient can be neatly expressed as

$$\frac{\partial J(\Theta)}{\partial \Theta^{(l)}} = \delta^{(l+1)} \big(a^l\big)^T \tag{5.3.7}$$

6 Week 6

Evaluating your hypothesis.
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions, what should you try next?

- Getting more training examples
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing or decreasing λ

A hypothesis may have a low error for the training examples but still be inaccurate(overfitting problems). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a training set and a test set. Typically, the training set consists of 70% of the data the test set is the remaining 30% .

The new procedure using these two sets is then:
1. Learn Θ and minimize $J_{\text{train}}(\Theta)$ using the rraining set
2. Compute the test set error $J_{\text{test}}(\Theta)$.
The test error:
1. For linear regression: $J_{\text{test}}(\Theta) = \frac{1}{2m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} (h_\Theta(x_{\text{test}}^{(i)}) - y_{\text{test}}^{(i)})^2$
2.For classification: $\text{err}(h_\Theta(x), y)$= 1 or 0
One way to breakdown our dataset into the three sets is

- Training set: 60%
- Cross validation set: 20%
- Test set: 20%

Now we compute three types of error using three different sets using the following method:
1. Optimizing the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree **d** with the least error using the vross validation set.
3. Estimate the generalization error using the test set with $J_{\text{test}}(\Theta^{(\mathbf{d})})$, (**d** = theta from polynomial with lower error)

Note: the degree of the polynomial **d** has not been trained using the test set.

6.1 Bias VS Variance

We need to find out the relationship between the degree of the polynomial **d** and the underfitting or overfitting of our hypothesis.

First of all, we need to distinguish whether bias or variance is the problem contributing to bad predictions.

Second, High bias is underfitting and high variance is overfitting. Ideally, we need to find a golden mean between these two.

The training error will tend to decrease as we increase the degree **d** of the polynomial. At the same time, the cross validation error will tend to decrease as we increase **d** up to a point, and then it will increase as **d** increases, forming a convex curve.

High bias(under fitting): both $J_{\text{train}}(\Theta)$ and $J_{\text{cv}}(\Theta)$ will be high. $J_{\text{train}}(\Theta) \approx J_{\text{cv}}(\Theta)$

High variance(overfitting): $J_{\text{train}}(\Theta)$ will be low and $J_{\text{cv}}(\Theta)$ will be much greater than $J_{\text{train}}(\Theta)$.

6.2 Regularization and Bias/Variance

The regurization term, λ helps us to make the fitting "just right", in order to choose the model and λ, we need to :

1. Create a list of lambdas.
2. Create a set of models with different degrees or any other features.
3. Iterate through the lambdas and for each λ go through all the models to learn some Θ, using $J_{(\text{train})}(\Theta)$.
4. Compute the train error using the learned Θ on the $J_{(\text{train})}(\Theta)$ without regurization term or $lambda = 0$.
5. Compute the cross validation error using the learned Θ on the $J_{(\text{CV})}(\Theta)$ without regurization term or $\lambda = 0$.
6. Select the best combo that produces the lowest error on the cross validation set.
7. Applying the selected combo Θ and λ on $J_{(\text{test})}(\Theta)$ to see if it has a good generalization of the problem.

6.3 Learning curve

Train error and Cross validation error VS training set size. Some approaches to tackle pratical problems.

Fixes high variance:

- getting more training examples
- Trying smaller sets of features
- Increasing λ

Fixes high bias:

- Adding features
- Adding polynomial features
- Decreasing λ

The typical three steps to solve machine learning problems:

- Start with a simple algorithm, implement it quickly, and test it early on your cross vallidation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.

Accuracy=(true positives + true negatives)/total examples
precision= (true positives)/(true positives + false positives)
recall = (true positives)/(true positives + false negatives)
F_1 score = (2*precision*recall)/(precision+recall)

7 Week 7

Support Vextor Machines(SVMs) and Optimization objective
The two key ideas of support vector machines are

- The maximum margin solution for a linear classifier.
- The "kernel trick"; a method of expanding up from a linear classifier to a non–linear one in an efficient manner.

7.1 Decesion boundary

Alternative view of logistic regression

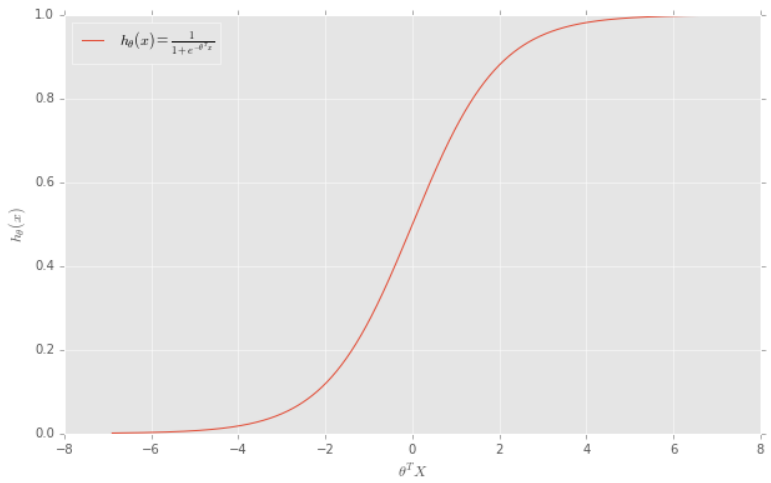


Figure 5: Logistic regression

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

One example of cost function

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

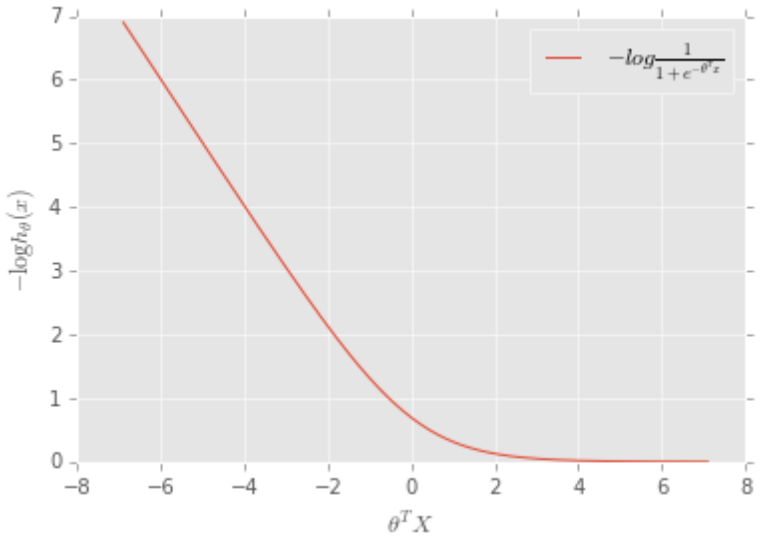


Figure 6: Cost function

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

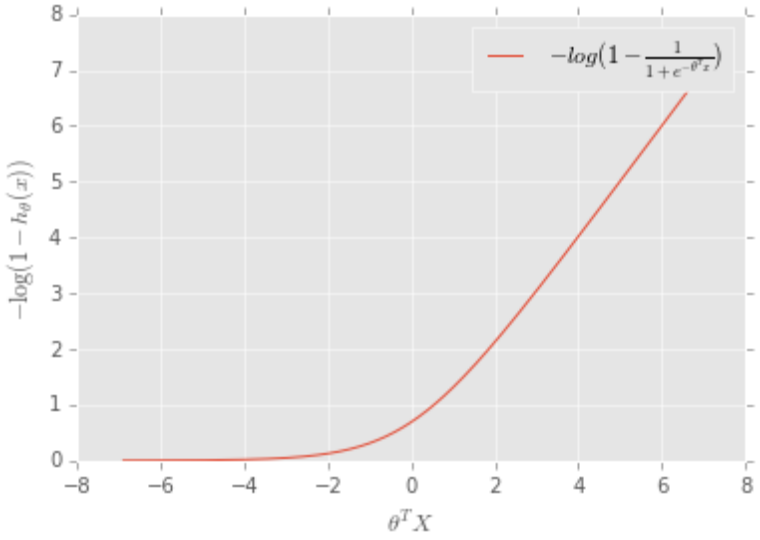


Figure 7: Cost function

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

Running the perceptron learning algorithm on the training data, with the decision boundary ($\theta^T x = 0$) we will obtain θ that would classify the training examples correctly. However, there is a whole version space of weight vectors that yield to the same classification of the training points. The SVMs algorithm chooses a particular weight vector, that gives rise to the "maximum margin" of separation. The goal of Logistic regression is to minimize the cost function

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

Rewrite the above equation in SVM (Support vector machine) hypothesis

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

7.2 Widest street approach

In binary classification problems, we want to label a dataset as positive(+1) or negative examples(-1), and suppose we have built a logistic regression hypothesis and run the perceptron algorithm to linearly separate the data, one can obtain a bunch of parameters that can do the job(e.g. Fig 9), but there exists a particular solution that seperates the positive and negative examples as wide as possible, we call this is the widest street approach, and the decesion boundary is the median line of the two "streets", where variables lie on the two lines.

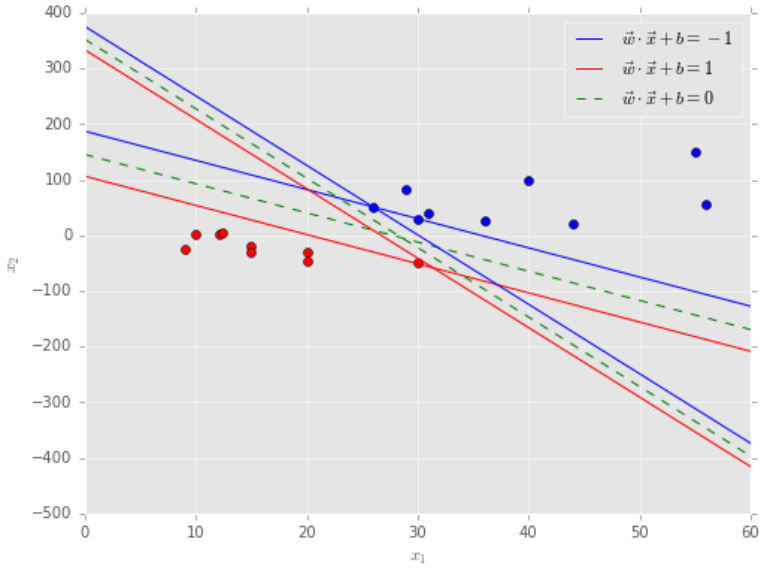


Figure 8: Decesion boundary

Let's denote w and intercept b as our parameters, and the vector that perpendicular to the median line of the "streets" is $\vec{w}$.
The decesion rule works like this: For a unknown vector $\vec{u}$, if

$$\vec{w}\cdot\vec{u}+b\geq 0 \tag{7.2.1}$$

the we say this new example $\vec{u}$ belongs to the positive group. For the existed positive examples, we have

$$\vec{w}\cdot\vec{x_+}+b\geq 1 \tag{7.2.2}$$

while the negative examples have the property

$$\vec{w}\cdot\vec{x_-}+b\leq -1 \tag{7.2.3}$$

To be mathematically convenient, we introduce a new variable y, such that y=1 if positive examples, and $y=-1$ for negative examples.

Therefore, we can combine eq(7.2.2) and eq(7.2.3) as one equation, this is for example vectors.

$$y_i(\vec{w}\cdot\vec{x_i}+b)\geq 1 \tag{7.2.4}$$

7.3 Optimization

The goal is to find the parameters w and b to maximaize the width of the street.

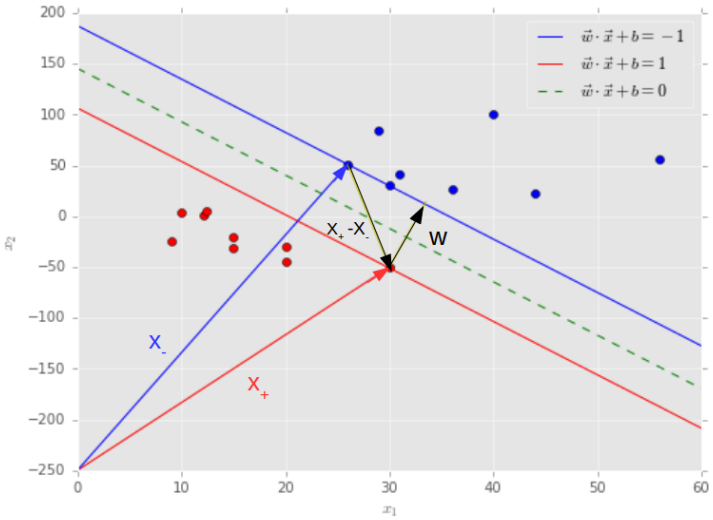


Figure 9: Width of street

The width can be computed as $(x_+ - x_-)\frac{\vec{w}}{\|\vec{w}\|} = \frac{2}{\|\vec{w}\|}$ (using eq(7.2.4)). Now the question transforms to minimize $\|\vec{w}\|$, for the sake of mathematical convenience, it is equivalent to minimize $\frac{1}{2}\|\vec{w}\|^2$.

Since we have constraints eq(7.2.3) and the goal of minimizing $\frac{1}{2}\|\vec{w}\|^2$, we now employ Lagrange multiplier.

$$L = \frac{1}{2}\|\vec{w}\|^2 - \sum_i \alpha_i \big[y_i(\vec{w}\cdot\vec{x_i} + b) - 1\big] \tag{7.3.1}$$

The two partial derivatives:

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_i \alpha_i y_i \vec{x_i} = 0 \tag{7.3.2}$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \tag{7.3.3}$$

then we have

$$\vec{w} = \sum_i \alpha_i y_i \vec{x_i} \tag{7.3.4}$$

$$\sum_i \alpha_i y_i = 0 \tag{7.3.5}$$

Next we subsitute eq(7.3.4) back into to eq(8.3.1),

$$L = \frac{1}{2}\sum_i \alpha_i y_i \vec{x_i} \sum_j \alpha_j y_j \vec{x_j} - \sum_i \alpha_i \big[y_i (\sum_j \alpha_j y_j \vec{x_j} \vec{x_i})\big] - \sum_i \alpha_i y_i b + \sum_i \alpha_i$$

then using eq(7.3.5)

$$\begin{aligned} L &= \sum_i \alpha_i - \frac{1}{2}\sum_i \alpha_i y_i \vec{x_i} \sum_j \alpha_j y_j \vec{x_j} \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x_i} \vec{x_j} \end{aligned} \tag{7.3.6}$$

It turns out that the optimization depends on the dot product of sample vectors. And of course, we can rewrite the decesion boundary eq(7.2.2) as

$$\sum_i \alpha_i y_i \vec{x_i} \vec{u} + b \geq 0 \tag{7.3.7}$$

For an unknown vector $\vec{u}$, as long as e1(8.4.1) holds, this example can be labeled as positive.

7.4 Kernels

The Kernel trick is used to transform non–linear space into linear space. For linear case, we tend to come up a function like

$$\begin{aligned} \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \geq c \rightarrow class1 \\ \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \leq c \rightarrow class2 \end{aligned}$$

In the case of non–linear separation, one can have hypothesis function of

$$\theta_0 + \theta_1 x_1^2 + \theta_2 x_2^3 \ldots$$

To make the computation more easier, we can introduce new variables like $f_1 = x_1^2$, this can provide linearly separable space. More generally, we use a kernel function totransfer variables into other space which is linear separable,

$$K(x_i, x_j) = \Phi(x_i)\Phi(x_j) \tag{7.4.1}$$

SVM parameters:

- C:$\frac{1}{\lambda}$
  - Large C: lower bias, high variance
  - Small C: Higher bias, low variance
- σ^2
  - Large σ^2,fearures vary more smoothly Higher bias, lower variance
  - small σ^2,fearures vary less smoothly lower bias, higher variance

8 Unsupervised learning

Unlike Supervised learning, whose training set is $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$,in Unsupervised learning, the training set is $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$. Unsupervised learning has application in social network analysis, Astronomical data analysis.

8.1 K–means algorithm

Some notation to clearify
K: number of clusters
Training set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, There is no x_0 here.
The k–means algorithm runs as follows:

- Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$
- Assign each data point to its closest cluster center.
  For i =1 to m, compute cost function $J(c^1, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_i^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$
  $c^{(i)}$: index(from 1 to K) of cluster centroid closest to $x^{(i)}$ (cluster assignment step)
  $\mu_{c^{(i)}}$: cluster centroid of cluster to which example $x^{(i)}$ has been assigned.
- update each cluster center μ_k by computing the mean of all points currently assigned to it, $\mu_k = \frac{1}{m_k}\sum_{i=1}^{m} x_{ik}$.
- Repeat step 1, 2 and 3 until converged, the choice of K depends on $\min J(c^{(1,\ldots,c^{(m)})}, \mu_1, \ldots, \mu_K)$.

8.2 EM algorithm

Consider the following experiment with coin A has probability of θ_A being head, and coin B has probability θ_B flipping head , we pick one coin at a time, and flip it m times, in total, we have chosed n coins, in other words, we have n and have fliped $n \times m$ times of coin.

Complete information : Suppose we write down which coin we have picked every time, we would have the complete likelihood function $p(x, z|\theta)$. where x is a vector of m flips in one sample, z is the label of coins. For mathemetical conveninet, we would like to compute $\log p(x, z|\theta)$, where

$$\log p(x, z|\theta) = \log \prod_{i=1}^{n} p(x_i, z_i|\theta) = \sum_{i=1}^{n} \log p(x_i, z_i|\theta) \tag{8.2.1}$$

the way we solve the parameters is the unsurprising Maximumizing Likelihood function(MLE).

$$\theta := \underset{\theta}{\operatorname{argmax}} \log p(x, z|\theta) \tag{8.2.2}$$

Incomplete information(missing information), we didn't record which coin we have picked, onlying knowing the flipping results. Models with hidden variables are known as latent variable models(LVM), in general, there are K laten variables(z_k, k=1,2,. . . ,K), and m visible variables(x_i, i=1,2,. . . ,m). The incomplete likelihood function is $p(x|\theta)$

$$\begin{aligned}
\log p(x|\theta) &= \log \prod_{i=1}^{n} p(x_i|\theta) \\
&= \sum_{i=1}^{n} \log p(x_i|\theta) \\
&= \sum_{i=1}^{n} \log \sum_{k=1}^{K} p(x_i, z_i = k|\theta)
\end{aligned}$$

This likelihood function is not easy to solve since we now have unkown variable z_i and unknown parameter θ, hence, **EM algorithm ** comes to play. First of all , we introduce the distribution of hidden variable $Q_i(z_i)$.

$$\begin{aligned}
\sum_{i=1}^{n} \log \sum_{k=1}^{K} p(x_i, z_i = k|\theta) &= \sum_{i=1}^{n} \log \sum_{k=1}^{K} Q_i(z_i) \frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)} \\
&= \sum_{i=1}^{n} \log E\big[\frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)}\big] \qquad (8.2.3)
\end{aligned}$$

Again , it is not so elegant, however, we have Jasen inequality to handle this complexity. Recall $f(E[x]) \geq E[f(x)]$ holds for convex functions,and the equale sign exists when $f(x)$ is a constant, now we plug this into the above equation $x \to \frac{p(x_i, z_i=k|\theta)}{Q_i(z_i)}$,

$$\begin{aligned}\sum_{i=1}^{n} \log \sum_{k=1}^{K} p(x_i, z_i = k|\theta) &= \sum_{i=1}^{n} \log E\big[\frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)}\big] \\ &\geq \sum_{i=1}^{n} E\big[\log \frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)}\big] \\ &\geq \sum_{i=1}^{n} \sum_{k=1}^{K} Q_i(z_i) \log \frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)} \end{aligned} \tag{8.2.4}$$

now the goal is to find a solution that the equation has "=" holds.

$$\log p(x|\theta) \geq \sum_{i=1}^{n} \sum_{k=1}^{K} Q_i(z_i) \log \frac{p(x_i, z_i = k|\theta)}{Q_i(z_i)}$$

Remember that we say the equal sign "=" holds if $f(x) = C$.
Define $\frac{p(x_i, z_i=k|\theta)}{Q_i(z_i)} = C$, we know the fact that the summation of the distribution of z_i must be 1, $\sum\limits_{k=1}^{K} Q_i(z_i = k) = 1$, it is not difficult to get $\sum\limits_{k=1}^{K} p(x_i, z_i = k|\theta) = C$.
now, let's see the compact solution of $Q_i(z_i)$

$$\begin{aligned} Q_i(z_i) &= \frac{p(x_i, z_i = k|\theta)}{C} \\ &= \frac{p(x_i, z_i = k|\theta)}{\sum\limits_{k=1}^{K} p(x_i, z_i = k|\theta)} \\ &= \frac{p(x_i|z_i = k, \theta)p(z_i|k, \theta)}{\sum\limits_{k=1}^{K} p(x_i, z_i = k|\theta)} \\ &= \frac{p(x_i|z_i = k, \theta)p(z_i|k, \theta)}{p(x_i|\theta)} \\ &= p(z_i|x_i, \theta) \end{aligned} \tag{8.2.5}$$

With this in mind, we now explain **EM algorithm**

- 1.Initial parameter θ
- 2. E step, compute $Q_i(z_i) := p(z_i|x_i, \theta)$
- 3. M step, $\theta := \underset{\theta}{\text{argmax}} \sum\limits_{i=1}^{n} \sum\limits_{k=1}^{K} Q_i(z_i) \log \frac{p(x_i, z_i=k|\theta)}{Q_i(z_i)}$
- 4. Repeat step 2 and 3 until converged.

In practice, Gaussian mixture model is widely used, and parameters $\theta = \mu, \Sigma$, the prior distribution of k is denoted as $\pi_k = p(z_i = k|\mu_k, \Sigma_k)$

$$p(\vec{x}|\mu, \Sigma) = \prod_{i=1}^{m} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|z_i = k, \mu_k, \Sigma_k) \tag{8.2.6}$$

For mathemetical convenience, we would like to compute $\text{argmax}_\theta \log[p(\vec{x}|\theta)]$

$$\log[p(\vec{x}|\mu, \Sigma)] = \sum_{i=1}^{m} \log \big[\sum_{k=1}^{K} p(z_i = k|\mu_k, \Sigma_k)\mathcal{N}(x_i|z_i = k, \mu_k, \Sigma_k)\big] \tag{8.2.7}$$

also, we define $\gamma_{ik} = p(z_i = k|x_i, \mu_k, \Sigma_k)$, the posterior distribution that point i belongs cluster k, it is knowns as the reponsibility of cluster k for point i. According to Bayes' rule

$$\begin{aligned}\gamma_{ik} = p(z_i = k|x_i, \mu_k, \Sigma_k) &= \frac{p(z_i = k|\mu_k, \Sigma_k)p(x_i|z_i = k, \mu_k, \Sigma_k)}{p(x_i|\mu_k, \Sigma_k)} \\ &= \frac{p(z_i = k|\mu_k, \Sigma_k)p(x_i|z_i = k, \mu_k, \Sigma_k)}{\sum_{k'=1}^{K} p(z_i = k'|\mu'_k, \Sigma'_k)p(x_i|z_i = k', \mu'_k, \Sigma'_k)}\end{aligned} \tag{8.2.8}$$

The EM algorithm for Gaussian Mixture Models work as follows:

1. Initialize the different parameters π, μ and Σ.
2. E step, Compute the responsibilities $\gamma_{ik} = \frac{\pi_k \mathcal{N}(x_i|z_i=k,\mu_k,\Sigma_k)}{\sum_{k'=1}^{K} \pi'_k \mathcal{N}(x_i|z_i=k',\mu'_k,\Sigma'_k)}$
3. M step, set partial derivative eq(8.2.7) wrt $\mu_k$andΣ_k.

$$\begin{aligned}\frac{\partial \log[p(\vec{x}|\mu, \Sigma)]}{\partial \mu_k} &= \sum_{i=1}^{m} \frac{\pi_k \mathcal{N}(x_i|z_i = k, \mu_k, \Sigma_k)}{\sum_{k'=1}^{K} \pi'_k \mathcal{N}(x_i|z_i = k', \mu'_k, \Sigma'_k)} \Sigma_k^{-1}(x_i - \mu_k) \\ &= \sum_{i=1}^{m} \gamma_{ik} \Sigma_k^{-1}(x_i - \mu_k) \\ &= 0\end{aligned}$$

there, we obtain

$$\mu_k = \frac{\sum_{i=1}^{m} \gamma_{ik} x_i}{\sum_{i=1}^{m} \gamma_{ik}} = \frac{1}{m_k} \sum_{i=1}^{m} \gamma_{ik} x_i$$

Similarly,

$$\begin{aligned}\frac{\partial \log[p(\vec{x}|\mu, \Sigma)]}{\partial \Sigma_k} &= \sum_{i=1}^{m} \gamma_{ik} \Big[\exp[(x_i - \mu_k)\Sigma^{-1}(x_i - \mu_k)^T] + \Sigma_k \exp[(x_i - \mu_k)\Sigma^{-1}(x_i - \mu_k)^T](-\Sigma_k^{-2}) \Big] \\ &= \sum_{i=1}^{m} \gamma_{ik} \Big[1 - (x_i - \mu_k)(x_i - \mu_k)^T \Sigma^{-1} \Big] \exp[(x_i - \mu_k)\Sigma^{-1}(x_i - \mu_k)^T] \\ &= 0\end{aligned}$$

Since $\exp[(x_i - \mu_k)\Sigma^{-1}(x_i - \mu_k)^T] \neq 0$, the solution of μ_k is

$$\Sigma_k = \frac{1}{m_k} \sum_{i=1}^{m} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T \tag{8.2.9}$$

Next, we introduce Largange multiplier to solve π_k. the constrains on π_k is $\sum_{k=1}^{K} \pi_k = 1$

$$\log p(x|\mu, \Sigma) + \lambda(\sum_{k=1}^{K} \pi_k - 1)$$

maximizing the above wrt π_k

$$\frac{\partial \log[p(\vec{x}|\mu, \Sigma)]}{\partial \pi_k} = \sum_{i=1}^{m} \frac{\mathcal{N}(x_i|z_i = k, \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|z_i = k, \mu_k, \Sigma_k)} + \lambda = 0$$

Multipling both part with π_k and summing over k, we have $\lambda = -N$, again multipling both sides with π_k yields

$$0 = \sum_{i=1}^{m} \gamma_{ik} - \pi_k m$$

we have

$$\pi_k = \frac{m_k}{m} \tag{8.2.10}$$

4. Repeat step 2 and 3 until convergence.

8.3 Dimensionality reduction

Motivations: Data compression, data visualization

8.3.1 Principle Component Analysis(PCA)

Steps:

- Preprocessing (feature scaling/mean normalizing) $\mu_j = \frac{1}{m}\sum_i^m x_j^{(i)}$, where $x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$
- Compute covariance matrix $A = \frac{1}{m}\sum_i^m (x^{(i)})(x^{(i)})^T$, [U,S,V]= svd(A),
- new feature $z^{(i)} = U[:, 1:K]^T x^{(i)}$
- Choose k to be smallest value that $\frac{\frac{1}{m}\sum_i^m \|x(i) - x_{\text{approx}}^{(i)}\|^2}{\frac{1}{m}\sum_i^m \|x^{(i)}\|^2} \leq 0.1$ or pick the smallest value of k for which $\frac{\sum_i^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$, where 99% of variance is retained.

9 Anomaly detection

We have dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, and we would like to know is x_{test} anomalous?

$$\begin{aligned} p(\vec{x}) &= p(x_1, \mu_1, \sigma_1^2) p(x_2, \mu_2, \sigma_2^2) \ldots p(x_n, \mu_n, \sigma_n^2) \\ &= \prod_{j=1}^{n} p(x_j, \mu_j, \sigma_j^2) \end{aligned} \tag{9.0.1}$$

Anomaly detection algorithm

- Choose fearures x_i that you think might be indicative of anomalous examples.
- Compute $\mu_j = \frac{1}{m}\sum_i^m x_j^{(i)}$, $\sigma_j^2 = \frac{1}{m}\sum_i^m (x_j^{(i)} - \mu_j)^2$
- Given new example $\vec{x}_{\text{new}}$, compute $p(\vec{x}_{\text{new}})$
- Anomaly if $p(\vec{x}_{\text{new}}) < \epsilon$

Algorithm evaluation

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

$$\begin{cases} y = 1 & \text{anomaly} \quad \text{if p(x)} \leq \epsilon \\ y = 0 & \text{normal} \quad \text{if p(x)} \geq \epsilon \end{cases}$$

- F_1 score,
  $F_1 = \frac{2*\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}}$,
  $\text{Precision} = \frac{\text{True positive}}{\text{True positive}+\text{false positive}}$,
  $\text{Recall} = \frac{\text{True positive}}{\text{True positive + false negative}}$,
  We can use cross validation set to choose ϵ which has the biggest F_1 score.

Comparison between Anomaly detection and Supervised learning

Anomaly detection	Supervised learning
Very small number of positive examples (y=1) (0 20 is common)	Large number of positive and negative examples
Large number of negative examples(y=0)	
Many different "types" of anomalies Hard for any algotithm to learn from positive examples what the anomalies look like	Enough positive examples for algoritm to get a sense of what, positive examples are like
Future anomalies may look nothing like any of the anomalous examples we've seen fo far	Future positive examples likely to be similar to ones in training set

Potential applications:

Anomaly dectection	Supervised learning
Fraud detection	Email spam classification
Manufacturing	Weather prediction
Monitoring machines in a data center	Cancer classification

Here are some tricks to use in Anomaly detection. 1.For non–gaussian features, transformation tricks: $x \to \log(x)$
$x \to \log(x + c)$
$x \to \sqrt{x}$
$x \to x^{(1/n)}$

2. Adding new features $\frac{x_i}{x_j}$

Anomaly detection with multivariate Gaussian

Methods:
1.Fit model p(x) by setting

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

2.Given a new example x, compute

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

and we flag it anomaly if $p(x) < \epsilon$.

Original model	Multivariate Gaussian
$p(\vec{x}) = p(x_1, \mu_1, \sigma_1^2)p(x_2, \mu_2, \sigma_2^2)\dots p(x_n, \mu_n, \sigma_n^2)$	$p(x, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\|\Sigma\|^{\frac{1}{2}}} \exp\big(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\big)$
Manually create features to capture nonmalies where x_1, x_2 take unusual combinations of values	Automatically captures correlations between features
Computationally cheaper	Computationally more expensive
Works even if m(the training size) is small	Must have $m > n$ or Σ is non–invertible

Recommender system
$y^{(i,j)}$, rating by user j on movie i.
$r(i, j) = 1$, if user j has rated movie i(0 otherwise).
$x^{(i)}$, feature vector for movie i
$\theta(j)$, parameter vector for user j.
For user j, movie i, predicted rating: $(\theta^{(j)})^T(x^{(i)})$.

Collaborative filtering Algorithm

- Initialize $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ and $\theta^{(1)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values.

- Compute cost function

$$J(x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}, \theta^{(1)}, \theta^{(1)}, \ldots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \big((\theta^{(j)})^T x^{(i)} - y^{(i,j)}\big)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2 \quad (9.0.2)$$

- Using gradient descent, we can optimize $x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}$ and $\theta^{(1)}, \theta^{(1)}, \ldots, \theta^{(n_u)}$ simultaneously.

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

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

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

10 Week 10

10.1 Gradient Descent with large datasets

Plot $J_{\text{train}}(\theta)$ and $J_{\text{CV}}(\theta)$ vs m (the number of examples)

Let's recall the algorithm about Linear regression with gradient descent:

Hypothesis : $h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$ and with cost function: $J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Repeat $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ (for every j=0,1,. . . , n)

- batch gradient descent : using all m examples in each iteration

  $J_{\text{train}}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$. Repeat $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$, m is the number of examples

- stochastic gradient descent:using 1 example in each iteration.

  cost(θ,$x^{(i)}$,$y^{(i)}$)= $\frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$, $J_{\text{train}}(\theta) = \frac{1}{2} \sum_{i=1}^{m} \text{cost}(\theta, x^{(i)}, y^{(i)}$.

  1:Randomly shuffle dataset.

  2. Repeat i=1,2,. . . ,m . $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$(for every j=0,1,. . . , n)

- mini–batch gardient descent: use b examples in each iteration Repeat i:=i+10.

  $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$(for every j=0,1,. . . , n),

Photo OCR pipeline Image→ Text detection → Character segmentation → Character recognition

11 Applications

to be continued