

Coursera-Stanford-ML-Notes

Quentin Truong

20 June 2017 - ? July 2017

Contents

1	Week 1: Introduction	2
1.1	Overview	2
2	Week 2: Linear Regression with Multiple Variables	3
2.1	Overview	3
2.2	Notation	3
2.3	Gradient Descent	3
2.4	Normal Equation	3
3	Week 3: Logistic Regression	4
3.1	Overview	4
3.2	Logistic Regression Hypothesis Function	4
3.3	Logistic Regression Cost Function	4
3.4	Proof of Logistic Regression Cost Function Derivative	5
3.5	Regularization	6
4	Week 4: Artificial Neural Networks Representation	7
4.1	Overview	7
4.2	ANN Notation	7
4.3	ANN Equations	7
4.4	Sample Three Layer System	8
5	Week 5: Artificial Neural Network Learning	9
5.1	Notation	9
5.2	Cost Function	9
5.3	Backpropagation Algorithm	9
5.4	Backpropagation Derivation - Base Case	10
5.5	Backpropagation Derivation - Recursive Case	10
5.6	Backpropagation Intuition	11
5.7	Gradient Checking	11
5.8	Random Initialization	11
6	Week 6: Advice for Machine Learning	12
6.1	Overview	12
6.2	Evaluating Hypothesis	12
6.3	Bias, Variance, Learning Curves	12
6.4	Error Analysis	13
6.5	Precision and Recall	13
6.6	Large Data Sets	13

7	Week 7: Support Vector Machines	14
7.1	Overview	14
7.2	Large Margin Classification	14
7.3	Math behind SVM	14
7.4	Kernels	14
8	Week 8: Unsupervised Learning	15
8.1	Overview	15
8.2	K-Means Notation	15
8.3	K-Means Distortion Function/Optimization Objective	15
8.4	K-Means Algorithm	15
8.5	Dimensionality Reduction: Principal Component Analysis	15
8.6	PCA Notation	15
8.7	PCA Equations	15
8.8	PCA Algorithm	16
8.9	PCA K	16
9	Week 9: Anomaly Detection	17
9.1	Overview	17
9.2	Gaussian (Normal) Distribution Equations	17
9.3	Anomaly Detection Algorithm	17
9.4	Multivariate Gaussian Distribution Equations	17
9.5	Recommender Systems Notation	17
9.6	Recommender Systems Algorithm	18
9.7	Collaborative Filtering Equations	18
9.8	Collaborative Filtering Algorithm	18

1 Week 1: Introduction

1.1 Overview

- Machine Learning: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”
- Supervised Learning: know what our correct output looks like
 - Regression: want continuous output
 - Classification: want discrete output
- Unsupervised Learning: little or no idea what our results should look like
 - Clustering: find groups according to similarity in various variables
 - Nonclustering: find structure in chaos

2 Week 2: Linear Regression with Multiple Variables

2.1 Overview

- Use linear regression for continuous output
- Choose gradient descent if many features (million+) because the inverse matrix required for the normal equation can become expensive to compute
- Normal equation will directly compute theta
- Normalize features if using gradient descent

2.2 Notation

$m = \text{number of samples}$

$n = \text{number of feature}$

$x = (n \times 1)$

$X = (m \times n)$

$X_j = (m \times 1)$

$\theta = (n \times 1)$

$\theta_j = (1 \times 1)$

2.3 Gradient Descent

Hypothesis Function	$h_{\theta}(x) = \theta^T \times x$
Vectorized Hypothesis Function	$h_{\theta}(X) = X \cdot \theta$
Linear Regression Cost Function	$J(\theta) = \frac{1}{2m} \sum (h_{\theta}(X) - y)^2$
Derivative of Linear Regression CF wrt θ_j	$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum (h_{\theta}(X) - y) \cdot X_j$
Change in θ_j	$\begin{aligned} \theta_j &= \theta_j - \alpha \frac{\partial}{\partial \theta_j} \\ &= \theta_j - \alpha \frac{1}{m} \sum (h_{\theta}(X) - y) \cdot X_j \end{aligned}$
Vectorized Change in θ	$\theta = \theta - \alpha \frac{1}{m} X^T (X \cdot \theta - y)$

2.4 Normal Equation

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

3 Week 3: Logistic Regression

3.1 Overview

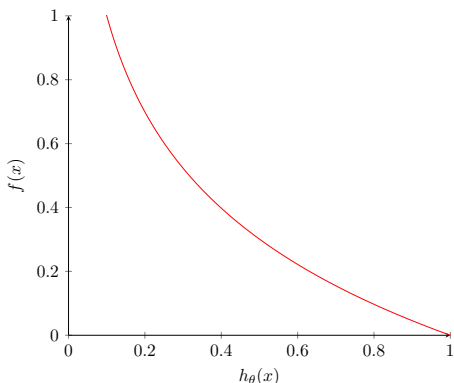
- Use logistic regression for discrete output (classification)
 - $h_{\theta}(x) = (y = 1|x; \theta)$; gives probability that the output is 1 given x
 - Sigmoid/Logistic function maps any real number to $(0, 1)$
 - Logarithm turns sum into product, allowing easier differentiation without altering search space
- For multi-class classification, use one-vs-all
 - Pick class i that maximizes $h_{\theta}^i(x)$
- Overfitting is when learned hypothesis fits training data well but fails to generalize; underfitting is when doesn't fit training data
- Address overfitting by reducing number of features, model selection, and regularization
 - Regularization results in simpler hypothesis and less overfitting
 - Extremely large λ will result in underfitting and gradient descent will fail to converge
 - Do not regularize λ_0
- Use other prewritten optimization algorithms (conjugate gradient, BFGS, L-BFGS) because they are faster

3.2 Logistic Regression Hypothesis Function

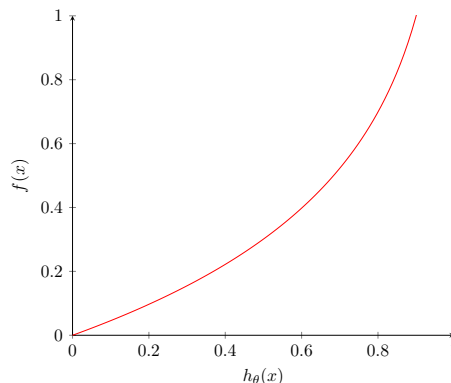
Sigmoid/Logistic Function	$g(z) = \frac{1}{1 + e^{-z}}$
Hypothesis Function	$h_{\theta}(x) = g(\theta^T x)$ $= \frac{1}{1 + e^{-\theta^T x}}$

3.3 Logistic Regression Cost Function

$$\begin{aligned} \text{Cost}(h_{\theta}(x), y) &= \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases} \\ &= -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x)) \\ J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^i), y^i) \\ J(\theta) &= \frac{-1}{m} \sum_{i=1}^m [y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))] \end{aligned}$$



(a) if $y=1$



(b) if $y=0$

3.4 Proof of Logistic Regression Cost Function Derivative

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

Credit¹

¹<https://math.stackexchange.com/questions/477207/derivative-of-cost-function-for-logistic-regression>

3.5 Regularization

Regularizing Term	$\lambda \sum_{j=1}^n \theta_j^2$
Regularized Linear Regression CF	$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2 + \lambda \sum_{j=1}^n \theta_j^2$
Regularized Logistic Regression CF	$J(\theta) = \frac{-1}{m} \sum_{i=1}^m [y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$
Regularized GD (Lin/Log Regression)	$\begin{cases} \theta_0 = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) x_0^i \right] \\ \theta_j = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) x_j^i + \frac{\lambda}{m} \theta_j \right] \quad (j=1,2,\dots,n) \end{cases}$
Regularized Normal Equation	$\theta = (X^T X + \lambda \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{n+1,n+1})^{-1} X^T y$

4 Week 4: Artificial Neural Networks Representation

4.1 Overview

- Neural networks allow for non-linear classification in situations with many features
 - Necessary b/c 100 features at 3rd level polynomials generates 170k features, which quickly becomes intractable
 - “One learning algorithm” hypothesis; you can see with your tongue : brain learns using one algorithm, not thousands of different programs
 - Can have multiple hidden layers
 - Can have multiple outputs (one-vs-all for multi-class classification)
 - If network has s_j units in layer j and s_{j+1} units in layer $j + 1$, then Θ^j will be of dimension $s_{j+1} \times s_j + 1$
 - The +1 comes from the addition in $\Theta^{(j)}$ of the bias node, x_0 and $\Theta_0^{(j)}$
- Forward Propagation is used to predict based on learned parameters
- Bias node gives each node a trainable constant value
 - Allows bias weight to shift the activation curve left/right
 - Other weights affect steepness
- Fun fact: For image recognition, particular order of pixels does not matter for ANN (but does for humans), you just need to keep the convention the same

4.2 ANN Notation

- $g(x)$: sigmoid function
- $\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j+1$; each layer gets own Θ^j
- $\Theta_{j,0}, \dots, \Theta_{j,n}$ weights corresponding to the inputs a_0, \dots, a_n going into z_j
- $w_{0,j}, \dots, w_{n,j}$ weights corresponding to the inputs a_0, \dots, a_n going into z_j
- $z_k^{(j)}$: encompasses parameters inside of g function
- $a_i^{(j)}$: “activation” of unit i in layer j

4.3 ANN Equations

$$\begin{aligned}z_k^{(j)} &= \Theta_{k,0}^{(j-1)} x_0 + \Theta_{k,1}^{(j-1)} x_1 + \dots + \Theta_{k,n}^{(j-1)} x_n \\&= \Theta_{k,0}^{(j-1)} a_0^{(j-1)} + \Theta_{k,1}^{(j-1)} a_1^{(j-1)} + \dots + \Theta_{k,n}^{(j-1)} a_n^{(j-1)} \\z^{(j)} &= \Theta^{(j-1)} a^{(j-1)} \\a^{(j)} &= g(z^{(j)})\end{aligned}$$

4.4 Sample Three Layer System

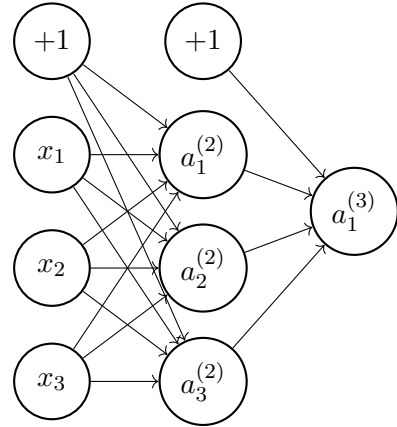
$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_{\Theta}(x)$$

$$a_1^{(2)} = g(\Theta_{1,0}^{(1)}x_0 + \Theta_{1,1}^{(1)}x_1 + \Theta_{1,2}^{(1)}x_2 + \Theta_{1,3}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{2,0}^{(1)}x_0 + \Theta_{2,1}^{(1)}x_1 + \Theta_{2,2}^{(1)}x_2 + \Theta_{2,3}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{3,0}^{(1)}x_0 + \Theta_{3,1}^{(1)}x_1 + \Theta_{3,2}^{(1)}x_2 + \Theta_{3,3}^{(1)}x_3)$$

$$\begin{aligned} h_{\Theta}(x) &= g(\Theta_{1,0}^{(2)}a_0^{(2)} + \Theta_{1,1}^{(2)}a_1^{(2)} + \Theta_{1,2}^{(2)}a_2^{(2)} + \Theta_{1,3}^{(2)}a_3^{(2)}) \\ &= g(z^3) \\ &= a_1^{(3)} \end{aligned}$$



5 Week 5: Artificial Neural Network Learning

5.1 Notation

- 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
- $h_{\Theta}(x)_k$: hypothesis that results in the k th output
- δ_k : Error signal

5.2 Cost Function

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

Picking training example

Picking output node

Picking layer

Picking node

Picking Θ

$$\sum_{i=1}^m \sum_{k=1}^K \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}}$$

5.3 Backpropagation Algorithm

1. Set $a(1) := x(t)$
2. Perform forward propagation to compute $a(l)$ for $l = 2, 3, \dots, L$
3. Using $y^{(t)}$, compute $\delta^L = a^{(L)} - y^{(t)}$
4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \cdot * a^{(l)} \cdot * (1 - a^{(l)})$
5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
6. $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$ where $\begin{cases} D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} & (j = 0) \\ D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right) & (j \neq 0) \end{cases}$

5.4 Backpropagation Derivation - Base Case

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}} &= \frac{\partial C}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} && \text{Cost } C \text{ varies wrt input accumulator } z_j, z_j \text{ varies wrt } w_{ij} \text{ (Chain Rule)} \\
\frac{\partial C}{\partial z_j} &= \frac{\partial}{\partial z_j} (y_j - a_j)^2 && \text{Cost defined as } (y_j - a_j)^2 \\
&= \frac{\partial}{\partial z_j} (y_j - g(z_j))^2 \\
&= -2(y_j - g(z_j))g'(z_j) \\
g'(z_j) &= \frac{d}{dz_j} \frac{1}{1 + e^{-z_j}} && \text{Derivative of logistic function} \\
&= \frac{1}{1 + e^{-z_j}} \frac{e^{-z_j}}{1 + e^{-z_j}} \\
&= g(z_j)(1 - g(z_j)) \\
\frac{\partial C}{\partial z_j} &= -2(y_j - g(z_j))g(z_j)(1 - g(z_j)) \\
&= -2(y_j - a_j)a_j(1 - a_j) \\
\frac{\partial z_j}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_q a_q w_{q,j} && \text{Definition of } z_j \text{ as sum of previous node's inputs and their weights} \\
&= a_i \\
\frac{\partial C}{\partial w_{ij}} &= -2(y_j - a_j)a_j(1 - a_j)(a_i) \\
&= -\Delta_j a_i
\end{aligned}$$

5.5 Backpropagation Derivation - Recursive Case

$$\begin{aligned}
\frac{\partial C}{\partial w_{i,j}} &= \sum_k \left(\frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{i,j}} \right) && C \text{ depends on } z_k, z_k \text{ depends on } a_j, a_j \text{ depends on } z_j, z_j \text{ depends on } w_{i,j} \\
\frac{\partial C}{\partial z_k} &= -\Delta_k \\
\frac{\partial z_k}{\partial a_j} &= \frac{\partial}{\partial a_j} \sum_s a_s w_{s,k} \\
&= w_{j,k} \\
\frac{\partial a_j}{\partial z_j} &= \frac{\partial}{\partial z_j} g(z_j) \\
&= g(z_j)(1 - g(z_j)) \\
&= a_j(1 - a_j) \\
\frac{\partial z_j}{\partial w_{i,j}} &= \frac{\partial}{\partial w_{i,j}} \sum_q a_q w_{q,j} \\
&= a_i \\
\frac{\partial C}{\partial w_{i,j}} &= \sum_k (-\Delta_k w_{j,k} a_j(1 - a_j) a_i) \\
&= \sum_k (-\Delta_k w_{j,k}) a_j(1 - a_j) a_i \\
&= -\Delta_j a_i
\end{aligned}$$

5.6 Backpropagation Intuition

δ_k is the error signal from the output

$$\delta_k = (a_k - t_k)g'_k(z_k)$$

so, error wrt output weights is

$$\frac{\partial E}{\partial w_{j,k}} = \delta_k a_j$$

similarly, error wrt internal weights is

$$\frac{\partial E}{\partial w_{i,j}} = \delta_j a_i$$

the error is determined by following layer's error, so it may also be understood as

$$\frac{\partial E}{\partial w_{i,j}} = g'_j(z_j) \sum_k (\delta_k w_{j,k}) a_i$$

5.7 Gradient Checking

- Gradient checking ensures that backpropagation is actually working
 - Turn off gradient checking once backpropagation is verified to work
- Approximate the derivative of cost function using slope
 - Pick $\epsilon = 10^{-4}$
 - Check all Θ_j

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

5.8 Random Initialization

- Need Symmetry Breaking
 - All hidden units would receive the same signal and the same updates
 - Results in finding only one feature, redundantly copied many times over
- Fixed by randomly initializing each $\Theta_{ij}^{(l)}$ to a value in $[-\epsilon, \epsilon]$
 - A good choice of ϵ init is $\epsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$, where $L_{in} = s_l$ and $L_{out} = s_{l+1}$, the number of units in the layers adjacent to $\Theta^{(l)}$

²<https://pdfs.semanticscholar.org/69c2/814c7f8ca16aa836fd32e3c4975f8208e63f.pdf>

³<https://stats.stackexchange.com/questions/94387/how-to-derive-errors-in-neural-network-with-the-backpropagation-algorithm>

⁴<https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/>

6 Week 6: Advice for Machine Learning

6.1 Overview

- Training set: 60%
- Cross validation set: 20%
- Test set: 20%
- Randomize your data and plot learning curves to determine high/low variance

6.2 Evaluating Hypothesis

Linear regression	$J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\Theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$
Classification	$err(h_{\Theta}(x), y) = \begin{cases} 1 & \text{if } h_{\Theta}(x) \geq 0.5 \text{ and } y = 0 \\ 1 & \text{if } h_{\Theta}(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases}$
Test Error	$= \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_{\Theta}(x_{test}^{(i)}), y_{test}^{(i)})$

6.3 Bias, Variance, Learning Curves

- High variance is when overfitting data
 - Get more training examples, try a smaller set of features, increase lambda
- High bias is when underfitting data
 - Add features, add polynomial features, decrease lambda
- Small neural networks are more prone to underfitting and are cheaper
- Large neural networks are more prone to overfitting and are more expensive

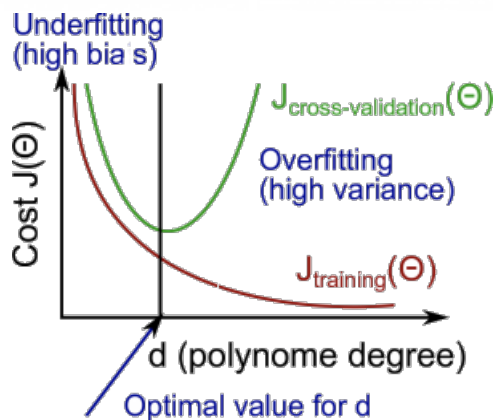
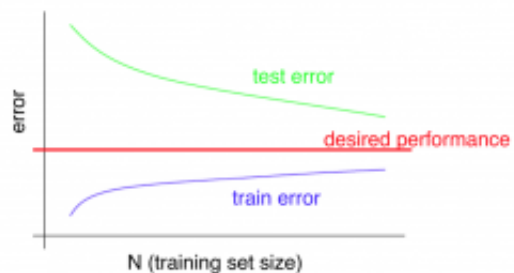
More on Bias vs. Variance

Typical learning curve for high bias (at fixed model complexity):



More on Bias vs. Variance

Typical learning curve for high variance (at fixed model complexity):



6.4 Error Analysis

- Difficult to know which feature is actually most helpful
- Start with simple algorithm and test it on cross validation data
- Look at learning curves to decide where to focus
- Manually look at errors in cross validation data to spot trends
- Iterate towards improvement according to a single numerical value of error

6.5 Precision and Recall

- Better metric for skewed classes, where there is only a very small chance of $y = 1$
- Precision = $\frac{\text{true positives}}{\text{predicted positives}}$
- Recall = $\frac{\text{true positives}}{\text{actual positives}}$
- Tradeoff between precision and recall by changing threshold
 - Predict 1 if $h_{\theta}(x) \geq \text{threshold}$
- F_1 score = $2 \frac{PR}{P+R}$
 - Maximize F_1 score for picking threshold

6.6 Large Data Sets

- Use large data (many training examples) if a human can confidently predict the results from the feature set
- Learning algorithms with many features/hidden units may need more data

7 Week 7: Support Vector Machines

7.1 Overview

- Change cost function and lambda, but otherwise is basically logistic regression
- Creates larger margins
- Use prewritten SVM's because they have computational tricks to make them efficient enough to work
- Use feature scaling
- Large C: lower bias, higher variance
- Large σ : higher bias, lower variance, varies more smoothly
- If n is small and m is intermediate, use SVM w/ gaussian kernel
- If n is small and m is large, add features and use logistic regression or SVM w/o kernel

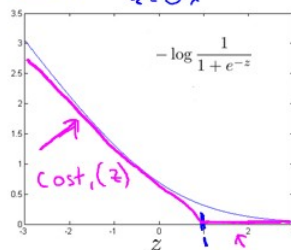
7.2 Large Margin Classification

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

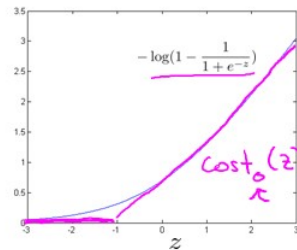
7.3 Math behind SVM

- $\theta^T x^{(i)} = p^{(i)} |\theta|$
- Try to maximize $p^{(i)}$ so that $|\theta|$ can be minimized

If $y = 1$ (want $\theta^T x \gg 0$):
 $z = \theta^T x$



If $y = 0$ (want $\theta^T x \ll 0$):



7.4 Kernels

- Compute features according to proximity to landmarks $l^{(i)}$
 - Pick landmarks based on training data
 - Gaussian Kernel: $f_1 = \text{similarity}(x, l^{(1)}) = \exp(-\frac{|x-l^{(1)}|^2}{2\sigma^2})$
- Other kernels exist, like linear or polynomial; all must satisfy Mercer's Theorem

8 Week 8: Unsupervised Learning

8.1 Overview

- Unsupervised learning will find structure in data
- K-Means is a way to cluster
 - Pick number of clusters K by hand or application
 - K-Means can get stuck in local optima, especially for small numbers of clusters
 - Run algorithm many times, pick cluster with lowest cost

8.2 K-Means Notation

- K : number of clusters
- $x^{(1)}$: training examples $\in \mathbb{R}$
- $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$: training set
- μ_k : average of points assigned to cluster k
- $c^{(i)}$: index of cluster centroid closest to $x^{(i)}$

8.3 K-Means Distortion Function/Optimization Objective

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

8.4 K-Means Algorithm

1. Randomly initialized K cluster centroids according to K training examples
2. $c^{(i)} :=$ index of cluster centroid closest to $x^{(i)}$
3. $\mu_k :=$ average of points assigned to cluster centroid k

8.5 Dimensionality Reduction: Principal Component Analysis

- Useful for data compression, visualizations, and speeding up computation
- PCA projects data onto lower dimensional surface
 - Preprocess data with feature scaling and mean normalization
 - Try to maintain at least 85% variance, preferably 95%+
- Don't use PCA to address overfitting

8.6 PCA Notation

- Covariance matrix: Σ
- Number of principal components: k
- $[U, S, V] = \text{svd}(\Sigma)$
 - Matrix of eigenvectors: U
 - $U_{\text{reduce}} = U(:, 1:k)$
- New inputs: $z = U_{\text{reduce}}^\top * x$

8.7 PCA Equations

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

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

$$\text{Total variance} = \frac{1}{m} \sum_{i=1}^m |x^{(i)}|^2$$

8.8 PCA Algorithm

1. Mean normalize and feature scaling
2. Compute covariance matrix Σ
3. Compute eigenvectors of Σ using singular value decomposition
4. Compute z

8.9 PCA K

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

alternatively, use S from $\text{svd}(\Sigma)$

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

9 Week 9: Anomaly Detection

9.1 Overview

- Use Anomaly detection over Supervised learning when there are a very few positive examples, there are many different types of anomalies, or when future anomalies may not look like current examples
- Gaussian Distribution
 - Positive μ shifts bellcurve right
 - Small σ increases height of bellcurve
 - For non-gaussian features, do something like $x_1 \leftarrow \log(x_1)$
 - Can also create your own features; $x_3 = \frac{x_1}{x_2}$
 - Choose ϵ by maximizing F_1
- Multivariate Gaussian will automatically capture correlations between features
 - Is more computationally expensive bc Σ^{-1}
 - Should have $m > 10n$ to verify invertibility
- For recommender systems, mean normalization can help recommend items for new users

9.2 Gaussian (Normal) Distribution Equations

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j; \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$
$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$
$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

9.3 Anomaly Detection Algorithm

1. Choose features x_j indicative of anomalous examples
2. Fit parameters μ and σ
3. Compute $p(x)$; anomaly if $p(x) < \epsilon$

9.4 Multivariate Gaussian Distribution Equations

$$p(x; \mu; \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right)$$
$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$
$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^\top$$

9.5 Recommender Systems Notation

- n_u : number of users
- n_m : number of movies
- $r(i, j) = 1$ if user j has rated movie i
- $y^{(i,j)}$: rating given by user j to movie i
- $\theta^{(j)}$: parameter vector for user j
- $x^{(i)}$: feature vector for movie i
- $m^{(j)}$: number of movies rated by user j

9.6 Recommender Systems Algorithm

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

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

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

9.7 Collaborative Filtering Equations

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

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

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

9.8 Collaborative Filtering Algorithm

1. Initialize $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$ to small random values
2. Minimize $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$
3. Predict rating using $\theta^\top x$