EECS 445

Introduction to Machine Learning

Honglak Lee - Fall 2015

Contributors: Max Smith

Latest revision: May 21, 2015

Contents

Abstract

Theory and implementation of state-of-the-art machine learning algorithms for large-scale real-world applications. Topics include supervised learning (regression, classification, kernel methods, neural networks, and regularization) and unsupervised learning (clustering, density estimation, and dimensionality reduction).

1 Readings

1.1 Probability Distributions

Definition 1.1 (Binary Variable)**.** Single variable that can take on either 1, or 0; $x \in \{0, 1\}$. We denote μ ($0 \leq \mu \leq 1$) to be the probability that the random binary variable $x = 1$

$$p(x = 1|\mu) = \mu$$

$$p(x = 0|\mu) = 1 - \mu$$

Definition 1.2 (Bernoulli Distribution)**.** Probability distribution of the binary variable x, where μ is the probability $x = 1$.

$$\text{Bern}(x|\mu) = \mu^x(1 - \mu)^{1-x}$$

The distribution has the following properties:

- $\text{E}(x) = \mu$
- $\text{Var}(x) = \mu(1 - \mu)$
- $\mathcal{D} = \{x_1, \ldots, x_N\} \rightarrow p(\mathcal{D}|\mu) = \Pi_{n=1}^{N} p(x_n|\mu)$
- Maximum likelihood estimator: $\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n = \frac{numOfOnes}{sampleSize}$ (aka. sample mean)

Definition 1.3 (Binomial Distribution)**.** Distribution of m observations of $x = 1$, given a sample size of N.

$$\text{Bin}(m|N, \mu = {}^N_m\mu^m(1 - \mu)^{N-m}$$

- $\text{E}(m) = N\mu$
- $\text{Var}(m) = N\mu(1 - \mu)$

The Beta Distribution

In order to develop a Bayesian treatment for fitting data sets, we will introduce a prior distribution $p(\mu)$.

– **Conjugacy:** when the prior and posterior distributions belong to the same family.

Definition 1.4 (Beta Distribution)**.**

$$\text{Beta}(\mu|a, b) = \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1 - \mu)^{b-1}$$

Where $\Gamma(x)$ is the gamma function. The distribution has the following properties:

- $\text{E}(\mu) = \frac{a}{a+b}$
- $\text{Var}(\mu) = \frac{ab}{(a+b)^2(a+b+1)}$

- conjugacy
- $a \to \infty || b \to \infty \to$ variance
  $to 0$

Conjugacy can be shown by the distribution by the likelihood function (binomial):

$$p(\mu|m, l, a, b) \propto \mu^{m+a-1}(1-\mu)^{l+b-1}$$

Normalized to:

$$p(\mu|m, l, a, b) = \frac{\Gamma(m+a+l+b)}{\Gamma(m+a)\Gamma(l+b)}\mu^{m+a-1}(1-\mu)^{l+b-1}$$

- **Hyperparameters:** parameters that control the distribution of the regular parameters.
- **Sequential Approach:** method of learning where you make use of an observation one at a time, or in small batches, and then discard them before the next observatiosn are used. (Can be shown with a Beta, where observing $x = 1 \to a++, x = 0 \to b++$, then normalizing)
- For a finite data set, the posterior mean for μ always lies between the prior mean and the maximum likelihood estimate.
- A general property of Bayesian learning is when we observe more and more data the uncertainty of the posterior distribution will steadily decrease.
- More information and examples of probability distributions can be found in Appendix B of Bishop's 'Pattern Recognition and Machine Learning.'

1.2 Linear Models for Regression

- **Linear Regression:** $y(\mathbf{x}, \mathbf{w}) = w_0 la + w_1 x_1 + \ldots + w_D x_D$
- Limited on linear function of input variables x_i
- Extend the model with nonlinear functions, where $\phi_j(x)$ are known as basis functions:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$

- w_0 allows for any fixed offset in data, and is known as the **bias parameter**.
- Given a dummy variable $\phi_0(x) = 1$, our model becomes:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) = \mathbf{w}^{\mathbf{T}} \phi(x)$$

- Functions of this form are called **linear models** because the function is linear in weight.

Maximum likelihood and least squares

- Via proof on p. 141-2, the maximum likelihood of the weight matrix is:
$$\mathbf{w_{ML}} = (\phi^{\mathbf{T}}\phi)^{-1}\phi^{\mathbf{T}}\mathbf{t}$$
where: $\phi_{nj} = \phi_j(x_n)$, called the **design matrix**
- This is known as the **normal equations** for the least squares problem.

Theorem 1.1 (Moore-Penrose Pseudo-Inverse)**.** of the matrix ϕ is the quantity:
$$\phi^{\dagger} = (\phi^{\mathbf{T}}\phi)^{-1}\phi^{\mathbf{T}}$$
It is regarded as the generalization of the matrix inverse of nonsquare matrix, because in the case that that the matrix is square we see: $\phi^{\dagger} = \mathbf{phi}^{-1}$

- The bias w_0 compensates for the difference between the averages of the target values and the weighted sum of the average of the basis function values.
- The Geometric interpretation of the least squares solution is an N-dimensional projection onto an M-dimensional subspace.
- Thus in practice direct solutions can lead to numerical issues when $\phi^T\phi$ is close to singular, because it results in large parameters. **Singular value decomposition** is a solution to this as it regularizes the terms.

Sequential Learning

- **Sequential Learning**: data points are considered one at a time, and the model parameters are updated after each such presentation.
  - This is useful for real-time applications, where data continues to arrive

Definition 1.5 (Stochastic Gradient Descent)**.** Application of sequential learning where the model parameters are updated at each additional data point using:
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n$$
Here τ is the iteration number, η is the learning rate, and E_n represents an objective funciton we want to minimize (in this case the sum of errors).
TODO: Pseudocode

Definition 1.6 (Least-Means-Squares (LMS) Algorithm)**.** Stochastic gradient descent where the objective function is the sum-of-squares error function resulting:
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta(t_n - \mathbf{w}^{(\tau)\mathbf{T}}\phi_n)\phi_n$$

- We introduce a regularization term to control over and under fitting.
$$E = E_D(\mathbf{w}) + \lambda E_W\mathbf{w})$$
- A simple example of regularization is given by the sum-of-squares of the weight vector elements:
$$E_W(\mathbf{w}) = 1/2\mathbf{w}^{\mathbf{T}}\mathbf{w}$$
- This regularizer is known as **weight decay** because it encourages weight values to decay towards zero unless supported by the data (stats term: **parameter shrinkage**)
-

2 Stanford Notes

2.1 Linear Regression with One Variable

Model Representation

- Goal is model labelled data (data which we have the correct output for) to a line
- Notation:

  m = number of training examples
  x = input variable/feature
  y = output variable/feature
  (x, y) = one training example
  $(x^{(i)}, y^{(i)})$ = ith training example (parens indicate index)

- We take a training set, input into a learning algorithm, which returns a hypothesis (h) that models the relationship.

$$\begin{tikzcd}
 & \boxed{\text{Training Set}} \arrow[d] & \\
 & \boxed{\text{Learning Algorithm}} \arrow[d] & \\
x \arrow[r] & \boxed{h} \arrow[r] & y
\end{tikzcd}$$

- h maps from x's to y's ($h(x) = y$).
- We need to determine how we want to represent h
- A simple linear model with one variable for h is:

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

  , called **univariate linear regression**.

Cost Function

- Given a hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

  $\theta_i's$ = parameters of the model

- We will now discuss how to choose the parameters of our model
- Idea: choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

- We want to minimize θ_0, θ_1 such that $h(x) - y$ is minimal (reminder: $h(x)$ is the guess at the correct value at y).
- Because we we only are looking to minimize our absolute distance, we square the distance we want to minimize to account for positive and negative differences equally now making our cost function: $(h(x) - y)^2$
- However, we don't want to minimize it for just one example, so we do this for every training example:

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

- To make later math easier, we further refine our formula to be half the average:

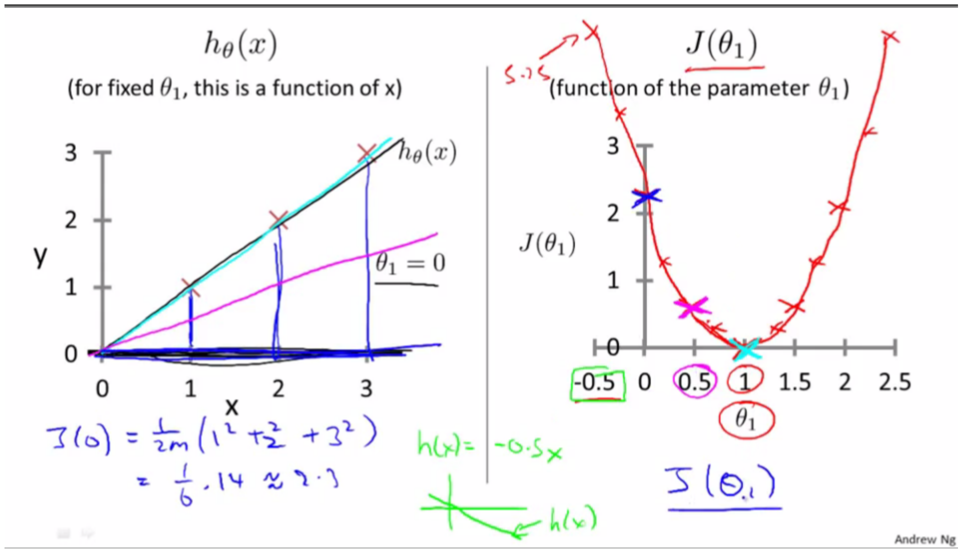
$$\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- This function we created is called our **cost** function, as it measures how expensively incorrect our current model is, which we will denote with J.
- The cost function is dependent on the hypothesis parameters, and our goal is to adjust these parameters to minimize the overrall cost of our model:

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

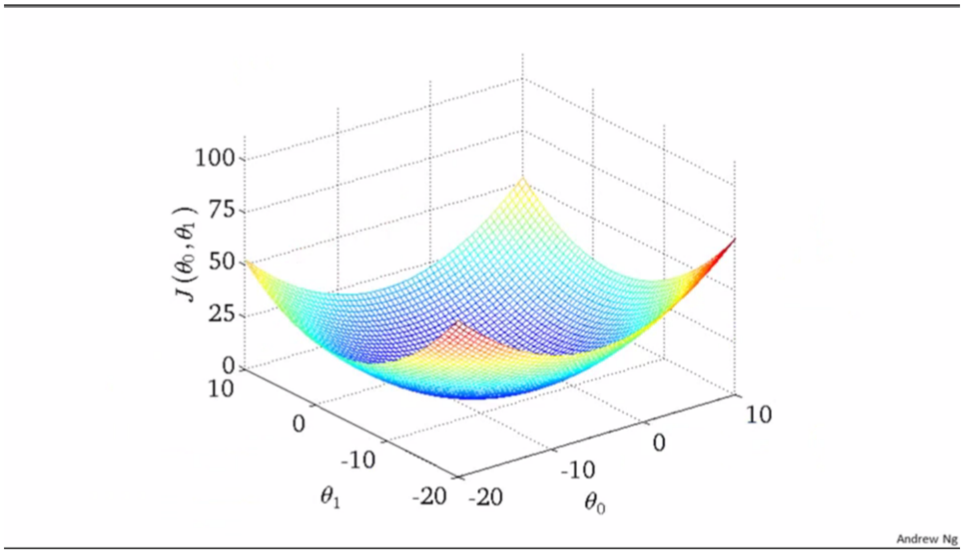
- Now our goal is to minimize J over the variables θ_0, θ_1

Cost Function - Intuition I



This image shows that for varying parameter values, the cost function changes. In this idealistic example there's a global minimum, the goal of minimized cost, that is very easily followed by a hill-climbing style algorithm.

Cost Function - Intuition II



Similarly when you have an additional variable, you want to reach the bottom of this N-dimensional hill (note: not all models will have such a perfect hill).

- The gradient gives the direction of maximumal increase on a surface.
- We will use a negative gradient to find the 'direction' to travel towards the bottom of the hill
- Another common way to represent multidimensional cost functions is through contour plots
-

Gradient Descent

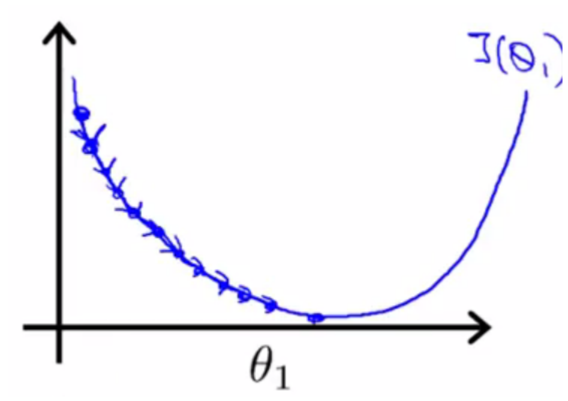
- Given some function $J(\theta_0, \theta_1, \ldots, \theta_n)$ we want to minimize J with respect to $\theta_0, \theta - 1, \ldots, \theta_n$.
- Choose initialize values for the parameters (eg. $\theta_0 = \ldots = \theta_n = 0$)
- Iteratively change $\theta_0, \ldots, \theta_n$ to reduce $J(\theta_0, \ldots, \theta_n)$, until hopefully a minimum is achieved.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

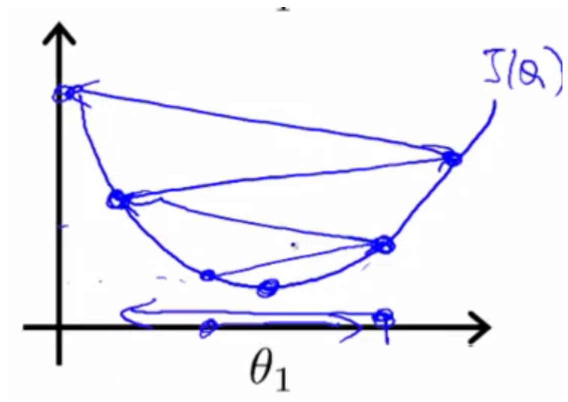
- α is the learning rate, which determines how much change happens in each update
- Ensure simultaneous update, store in temps and then assign.
- An issue with gradient descent is finding local minimums, because you won't be able to find the optimal solution.
-

Gradient Descent Intuition

- To simplify, we will consider the cost funciton with 1 variable ($J(\theta_0)$).
- The negative gradient means that you negative slope, which results in increases with negative slope and dcreases with positive slopes
- If α is too small, gradient descent can be very slow.



- If α is too large, it may fail to converge (not reach minimum).



- If you're already at the local minimum, you will not change your parameters because the gradient is zero.
- You can still converge to a local minimum with a fixed α (learning rate) because as we approach the minimun the gradient descent will automatically take smaller steps.

Gradient Descent for Linear Regression

- Before we continue, we must calculate what the derivative term is:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) &= \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}-y^{(i)}))^2\\ &= \frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x^{(i)}-y^{(i)})^2\end{aligned}$$

– In our linear model we have $j = 0, 1$; therefore, we can simplify our equation further for each case of j:

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)} - y^{(i)}))$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)} - y^{(i)}))x^{(i)}$$

– Linear regression will always have a convex function for it's cost; namely, it is bowl shaped and doesn't have any local min besides the global - always finds best solution.

– **Batch:** each step of gradient descent uses all the training examples

– Gradient descent scales better than normal equations, which is an advanced linear algebra technique that finds the parameters in closed forms - no iterations.

2.2 Linear Regression with Multiple Variables

Multiple Features

– Instead of just one feature (x), we known multiple features ($x_1, \ldots, x_n$). eg. size, number of bedrooms, number of floors, age of home.

– $x_j^{(i)}$: value of feature j in i^{th} training example

– Now our hypothesis must account for multiple features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots \theta_n x_n$$

– Again we define $x_0 = 1$ to simplify future math ($x_0^{(i)} = 1$).

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1},\ \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

– Transposing the θ vector given our assumption for $x_0^{(i)}$ allows us to simplify our hypothesis into:

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

Gradient Descent for Multiple Variables

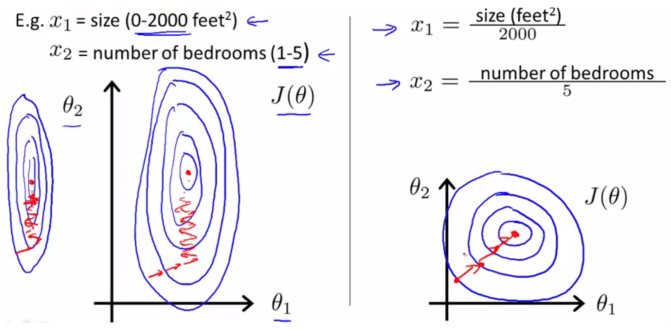
– Repeat until convergence ($j = 0, \ldots, n$):

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

– This is a valid generalization of the previous formula because of our base case $x_0^{(i)} = 1$

Gradient Descent in Practice I - Featuer Scaling

- **Feature scaling:** if features are on similar scales then we converge more quickly
- Your parameters will oscillate along the larger ranged parameter making it's way much slower towards the center (in the case of two variables); whereas, if both axis were equal then you don't have a worst case to fret about



- Typically, we want to scale each feature into approximately a $-1 \leq x_i \leq 1$ range (same order of magnitude).
- **Mean normalization:** replacing x_i with $x_i - \mu_i$ to make features have approximately zero mean (does not apply to $x_0 = 1$).
- Combining mean normalization and feature scaling we assign $x_i := \frac{x_i - \mu_i}{range_i}$

Gradient Descent in Practice II - Learning Rate

- To ensure gradient descent is working correctly, plot the cost function against the number of iterations. It should converge towards 0, decreasing at every iteration.
- The number of iterations required can vary widely for different applications
- You can create an automatic convergence test to ensure appropriately ending of gradient descent by checking if the difference between two iterations ϵ is below a threshold.
- If there is any increase in slope, use a smaller α
- For sufficiently small α, $J(\theta)$ should decrease on every iteration
- But if α is too small, gradient descent an be slow to converge
- To choose α try: $\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$

Features and Polynomial Regression

- Suppose we have a housing price prediction: $h(x) = \theta_0 + \theta_1(frontage) + \theta_2(depth)$
- We can define a new feature $(area) = (frontage)(depth)$, that we can use in a new hypothesis $h(x) = \theta_0 + \theta_1(area)$
- We can map these hypothesis of more complex features into a linear regression problem:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \qquad = \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$

- If your features are like those chosen, then feature scaling is very important
- There are many more choices for modifications to our features (such as: $\sqrt{}$).
- Trying new features can allow you to have a more appropriate model

Normal Equations

- The **normal equation** allows us to solve for θ analytically (without iterations)
- Intuition: $J(\theta) = a\theta^2 + b\theta + c$ In previous calculus classes you would find the minimum by taking the derivative set equal to 0 and solving for θ.
- This can be extended with partial fractions and solving for every $\theta_j \in \theta$.

$$\frac{\partial}{\partial \theta_j} J(\theta) = \ldots = 0 \text{ (for every} j)$$

- We construct a matrix from the features and a vector from the solutions as so (n features, m examples):

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \ldots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \ldots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \ldots x_n^{(m)} & \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

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

- We can then represent the θ by the **normal equation**

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

- X is entitled the **design matrix**
- Normal equation does not perform well with a large n due to the computation $(X^T X)^{-1} \in \ltimes \times \ltimes$ which is typically $\mathcal{O}(n^3)$

2.3 Logistic Regression

Multiple Features

- A potential binary classification solution is to make the binary decision based on a threshold. eg. threshold $= 0.5$:

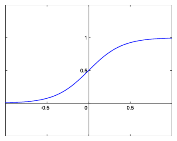
$$\begin{aligned} h(x) &\geq 0.5 && \text{predict } y = 1 \\ h(x) &< 0.5 && \text{predict } y = 0 \end{aligned}$$

- We want a classifier that results in $0 \leq h \leq 1$, as we only have 2 outcomes 0, 1

Hypothesis Representation

– **Sigmoid/Logistic function**:

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



– Note: there are horizontal asymptotes at 0 and 1.

– We will modify our original hypothesis to now be: $h(x) = g(\theta^T x)$, where g is the previously defined sigmoid function

– Interpretation of hypothesis output:

$$h_\theta(x) = \text{ estimated probability that } y = 1 \text{ on input } x$$

– Eg. $h(x) = 0.7 \to 70\%$ chance of tumor being malignant

– $h(x) = p(y = 1|x;\theta)$ is another way of defining this.

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

Decision Boundary

– Suppose we predict $y = 1$ if $h(x) \geq 0.5$. Graphically, we can seee that this is the same as predicting 1 when $\theta^T x \geq 0$

– **Decision Boundary**: region where $h(x)$ is equal to the threshold (the line that seperates 0 predictions vs 1 predictions).

– This concept can be expanded with the higher powered functions, to result in non-linear decision boundaries

$$h(x) = g(\theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$$\text{Predict } y = 1 \text{ if } -1 + x_1^2 + x_2^2 \geq 0$$

Cost Function

– How do we choose/fit the parameters θ?

– We abstract our linear regression cost function to be:

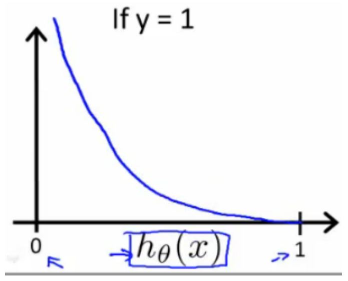
$$j(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(h(x^{(i)} - y^{(i)}))^2 = \frac{1}{m}\sum_{i=1}^{m}\text{cost}(h(x^{(i)}, y)$$

$$\text{cost}(h(x), y) = \frac{1}{2}(h(x) - y)^2$$

– However, this cost function is non-convex for logistic regression, which doesn't allow us to run gradient descent (no garuntee of global minimum reached).

– Let our cost function for logistic regression now be defined as:

$$\text{cost}(h(x), y) = \begin{cases} -\log(h(x)) & y = 1 \\ -\log(1 - h(x)) & y = 0 \end{cases}$$



– This allows us to have no cost when we were correct at our guess, but have increasingly greater cost the more wrong we were.

Simplified Cost Function and Gradient Descent

– We can simplify the cost function to a single formula:

$$\text{cost}(h(x), y) = -y\log(h(x)) - (1 - y)\log(1 - h(x)$$

– This formula works because when $y = 1$ the portion of the equation that is relevant for 0 becomes 0 via $(1 - y)$

– Similarly for $y = 0$.

– Note: $y \in 0, 1$ always.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(h(x^{(i)}), y^{(i)}) \qquad = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h(x^{i}) + (1 - y(i))\log(1 - h(x(i))) \tag{1}$$

– Note the negative sign was pulled out of the summation and brought in front of the summation

– To implement gradient descent we must first fit the parameters θ to the model by minimizing $J(\theta)$

– Then we can make a predition given the new x using: $h(x) = \frac{1}{1+e^{-\theta^T x}}$

– We again minimize our cost function using gradient descent, where:

$$\frac{\partial}{\partial\theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)} - y(i))x_j^{(i)}$$

– To ensure that the learning rate α is set properly, remember to plot the cost function $(J(\theta))$ as a function of number of iterations and make sure $J(\theta)$ is decreasing on every iteration

Advanced Optimization

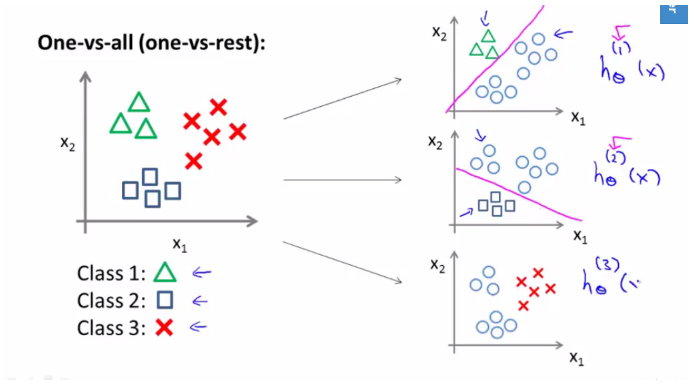
- Optimization algorithm: has the goal of minimizing a cost function $J(\theta)$.
- Given θ we have code that can compute:
  - $J(\theta)$ (to monitor convergence)
  - $\frac{\partial}{\partial\theta_j}J(\theta)$ for $(j = 0, 1, \ldots, n)$
- Gradient descent repeats the following until convergence:

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

- Gradient descent isn't the only optimization algorithm:
  - Conjugate gradient
  - BFGS
  - L-BFGS
- They have the advantages:
  - No need to manually pick α
  - Often faster than gradient descent
- And the disadvantages:
  - More complex
- Don't implement these yourself, use package implementation

Multiclass Classification: One-vs-All

- Example of **multiclass classification**: email foldering/tagging: work, friends, family, hobby, etc.
- In **one-vs-all** we compare each classification against all other possibilities.

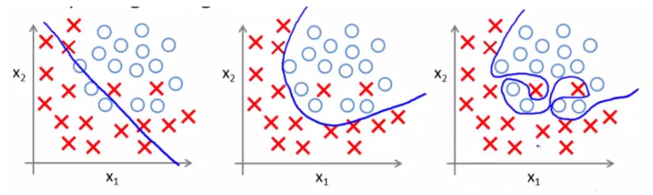


- This gives us a classifier for each class
- Each classifier estimates the probability that the value is that particular class
- This allows us to guess the particular class by taking the representative class of the maximum probability classifier across all classifiers

2.4 Regularization

The Problem of Overfitting

- **Underfitting**: when a model cannot capture the underlying trend of the data (”high bias”)
- **Overfitting**: when a model captures the noise of the data (”high variance”)
- **Generalize**: fails to fit to new examples
- Overfitting’s poor generalization results in low costs not always being correct



- If we have too many features for very little data, overfitting can easily become a big problem.
- Options:
  - Reduce number of features
    - ∗ Manually select which features to keep
    - ∗ Model selection algorithm (later in course)
  - Regularization
    - ∗ Keep all the features, but reduce magnitude/values of parameters θ_j
    - ∗ Works well when we have a lot of features, each of which contributes a bit to predicting y

Cost Function

- Having smaller values for parameters $\theta_0, \theta_1, \dots, \theta_n$
  - “Simpler” hypothesis
  - Less prone to overfitting
- To exemplify lets consider the housing scenario:
  - Features: $x_1, \dots, x_100$
  - Paremeters: $\theta_0, \dots, \theta_100$
  - We don’t know which ones are complex to shrink, so we modify the cost function to shrink every parameter

$$J(\theta) = \frac{1}{2m}(\sum_{i=1}^{m}(h(x^{(i)} - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_j^2)$$

  - We don’t penalize θ_0 by convention, because it’s a constant and makes very little difference
- λ is called the regularization parameter. It controls the trade off of fitting the training set well and keeping the parameters small and simple to prevent overfitting
- If λ is set to be very large we will penalize all the paramters extremely highly resulting which will result in all by the constant being close to zero (fits to a horizontal line). “Underfit”

Regularized Linear Regression

- Using the new regularized linear regression cost function from the previous section we can now update our gradient descent algorithm to encorporate this modification:

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

$$\theta_j := \theta_j - \alpha(\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y(i))x)j^{(i)} + \frac{\lambda}{m}\theta_j)$$

- The θ_j $(j = 1, \ldots, n)$ term can also be written:

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- The term $(1 - \alpha\frac{\lambda}{m})$ is typically < 1, so this results in shrinking θ_j by multiplying by a value < 1, and then performing the same gradient descent function
- We also had the normal equation to solve the same problem, and it can also be updated for regularization:

$$\theta = (X^TX + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \cdots & \\ & & & 1 \end{bmatrix})X^Ty$$

- Suppose $m \leq n$ then (X^TX) will be non-invertible/singular
- Regularization will take care of this flaw so long as $\lambda > 0$

Regularized Logistic Regression

- We can also regularize logistic regression in a similar manner

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

$$\theta_j := \theta_j - \alpha(\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y(i))x)j^{(i)} + \frac{\lambda}{m}\theta_j)$$

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

2.5 Neural Networks: Representation

Non-linear Hypothesis

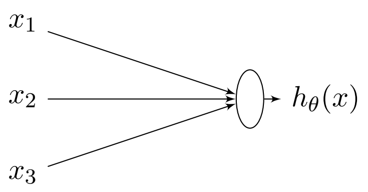
- Suppose you have a housing classification problem with different features $x_1, \ldots, x_100$ and if we were to include all quadratic terms for linear classification there would be an enourmous number of terms (5000 features).
- Using linear classifiers has an extremely bad asymptotic complexity (n is typically large)
- For example computer vision problems are $\mathcal{O}(n^2)$
- Neural networks turn out to be a much better way to solve these style problems

Neurons and the Brain

- Neural networks origins are in algorithms that try to mimic the brain
- **"One learning algorithm hypothesis"**: x cortex can learn whatever is hooked up to it
  - Auditory cortex can learn to see
  - Somatosensory cortex (touch) can learn to see
  - There is one algorithm that can teach anything to do any function

Model Representation I

- Neural networks work by simulating the neurons in the brain
- Has "input wires" dendrite
- Has "output wires" axon
- Neuron is a computational unit that takes in inputs and produces an output
- Neurons communicate with little spikes of electricity through their axons, which another neuron can receiv with its dendrite
- In an artificial neural network we model a neuron as a logistic unit:



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}, h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

  - Here the arrows coming from the x are the input wires
  - The neuron does the computation
  - Finally the output comes out
- The x_0 is called the **bias unit** and is sometimes omitted because it's constant.
- **Activation function**: defines the output of that node given an input or set of inputs
- "weights" are synonomous with parameters of the model
- Neural networks are groups of neurons strung together

  TODO: Neural Network
- **Input layer**: first layer of inputted values

- **Output layer**: the final layer that calculates the output value
- **Hidden layer**: the center layers that don't have known outputs (not input or output layer)
- $a_i^{(j)}$ = "activation" of unit i in layer j
- $\theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$
- If a network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$