EECS 445

Introduction to Machine Learning

Honglak Lee - Fall 2015

Contributors: Max Smith

Latest revision: May 11, 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.

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

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

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:

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

- 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

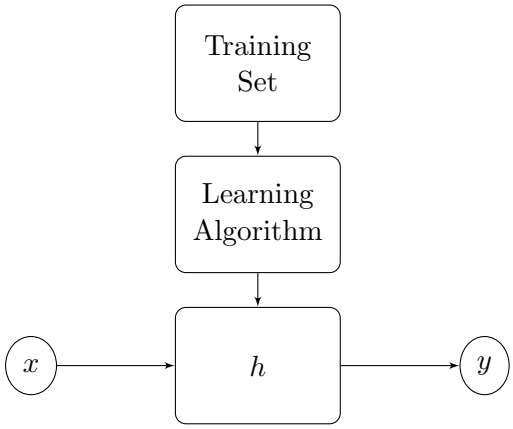
  x = input variable/feature

  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.



- 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:

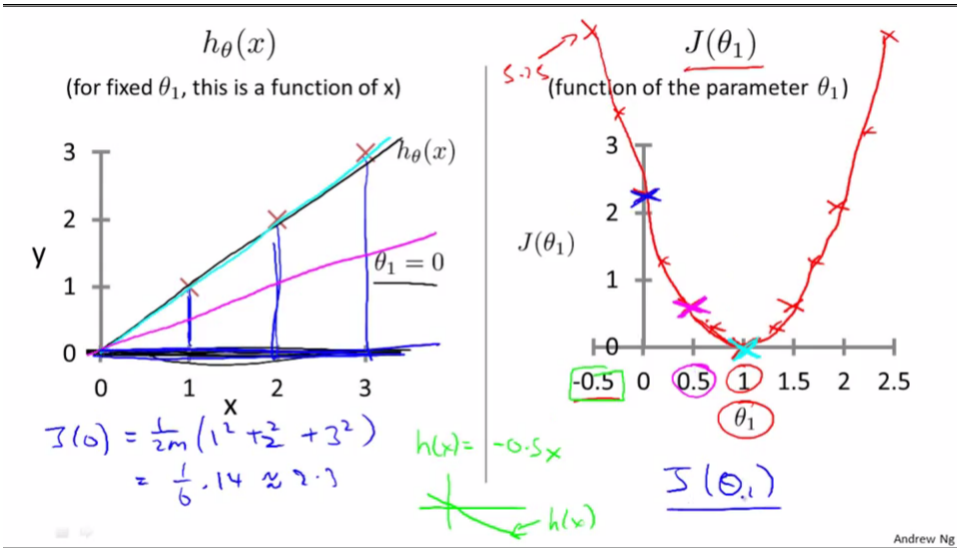
$$\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- 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:

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

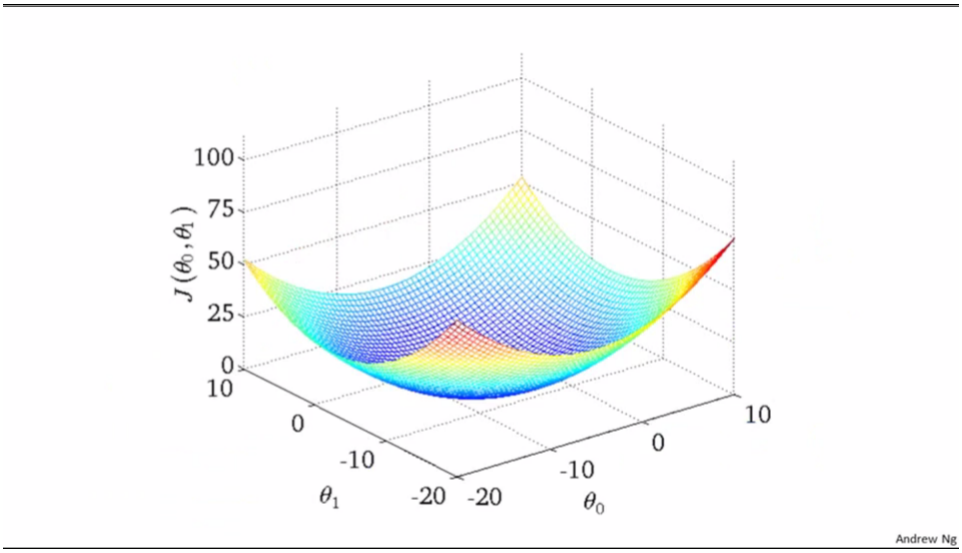
- Now our goal is to minimize J over the variables θ_0, θ_1

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

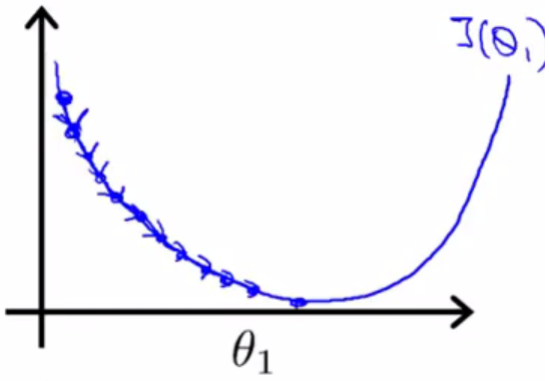
- Given some function $J(\theta_0, \theta_1, \ldots, \theta_n)$ we want to minimize J with respect to $\theta_0, \theta - 1, \ldots, \theta_n$.
- Choose initialize values for the parameters (eg. $\theta_0 = \ldots = \theta_n = 0$)
- Iteratively change $\theta_0, \ldots, \theta_n$ to reduce $J(\theta_0,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

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

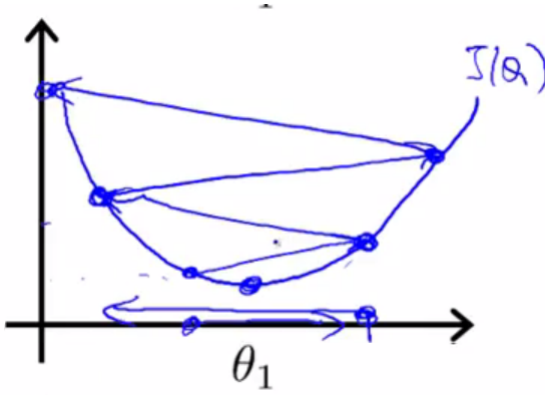
- α is the learning rate, which determines how much change happens in each update
- Ensure simultaneous update, store in temps and then assign.
- An issue with gradient descent is finding local minimums, because you won't be able to find the optimal solution.
-

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.
-