Machine Learning
Lecture 2

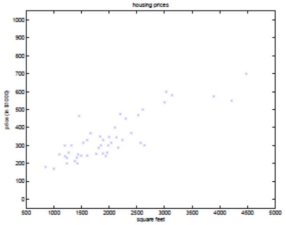
Linear Regression

Machine Learning for house hunting

- Suppose we have a dataset giving the living areas and prices of some houses :

Living area (feet^2)	Price(1000$s)
2014	400
1600	330
2400	369
1416	232
3000	540
...	
2005	?
3200	?
1280	?

We can plot this data set:



- How can we learn to predict the prices of other houses, as a function of the size of their living areas?

The learning problem

- $x^{(i)}$ denotes the "input" variables/features
- $y^{(i)}$ denotes the "output" or target variable that we are trying to predict (price)
- A pair ($x^{(i)}$, $y^{(i)}$) is called a training example
- A list of m training examples $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$—is called a training set.
- **X** denote the space of input values, and **Y** the space of output values. In this example, **X** = **Y** = **R**.
- Our goal:

  Given a training set, learn a function h : **X** —>**Y** so that h(x) is a "good" predictor for the corresponding value of y.

  For historical reasons, this function h is called a hypothesis.

A slightly richer dataset

- If you want to find the most reasonably priced house satisfying your needs: square-ft, # of bedroom, distance to work place...

Living area (feet^2)	# bedrooms	Price(1000$s)
2014	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
...		
2005	3	?
3200	4	?
1280	2	?

The learning problem

- Features:
  - Living area, #bedroom, distance to work place …
  - Denote as x = $[x_1, x_2, ..., x_n]^T$
- Target:
  - Price
  - Denoted as y
- Training set:

$$X=\begin{bmatrix} --(x^{(1)})^T-- \\ --(x^{(2)})^T-- \\ \vdots \\ --(x^{(m)})^T-- \end{bmatrix}=\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad Y=\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

m: #examples/samples
n: #features

Linear Regression

- Assume that Y (target) is a linear function of X (features):

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

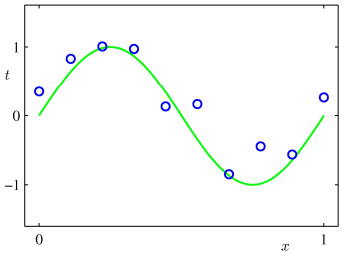
Here, the θ_i 's are the **parameters** (also called **weights**) parameterizing the space of linear functions mapping from $\mathcal{X}$ to $\mathcal{Y}$ When there is no risk of confusion, we will drop the subscript θ in $h_\theta(x)$, and write it more simply as $h(x)$. To simplify our notation, we also introduce the convention of letting $x_0 = 1$ (this is the **intercept term**), so that

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T \underline{x}$$
$$\underline{\emptyset(x)}$$

Pre-processing of features or feature extraction

Linear Basis Function Models (1)

Example: Polynomial Curve Fitting



$$y(x,\theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_M x^M = \sum_{j=0}^{M} \theta_j x^j$$

The Least Mean Square (LMS) method

- The Cost Function:

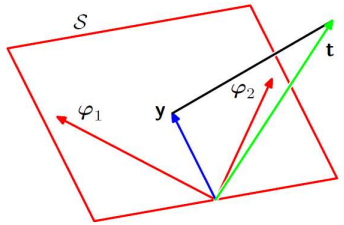
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- Consider a gradient descent algorithm:

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

Geometry of least squares

Geometrical interpretation of the least-squares solution, in an N-dimensional space whose axes are the values of $t_1, \ldots, t_N$. The least-squares regression function is obtained by finding the orthogonal projection of the data vector **t** onto the subspace spanned by the basis functions $\phi_j(\mathbf{x})$ in which each basis function is viewed as a vector $\boldsymbol{\varphi}_j$ of length N with elements $\phi_j(\mathbf{x}_n)$.



The Least Mean Square (LMS) method

- For a single training example, this gives the update rule:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

- This is known as the LMS update rule, or the Widrow-Hoff learning rule
- If the training set has more than one example

Repeat until convergence {

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

Batch gradient descent

Stochastic gradient descent

- The above results were obtained with batch gradient descent. There is an alternative to batch gradient descent that also works very well. Consider the following algorithm:

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

}

- When the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

The normal equations

- Given a training set, define the design matrix X to be the m-by-n matrix that contains the training examples' input values in its rows:

$$X=\begin{bmatrix} --(x^{(1)})^T-- \\ --(x^{(2)})^T-- \\ \vdots \\ --(x^{(m)})^T-- \end{bmatrix}=\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad Y=\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

m: #samples
n: #features

The normal equations

- Write the cost function in matrix form:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) \\ &= \frac{1}{2}\nabla_\theta \left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta \left(\operatorname{tr}\theta^T X^T X\theta - 2\operatorname{tr}\vec{y}^T X\theta\right) \\ &= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right) \\ &= X^T X\theta - X^T\vec{y} \end{aligned}$$

The normal equations

- To minimize J, we set its derivatives to zero, and obtain the normal equations:

$$X^T X\theta = X^T\vec{y}$$

- Thus, the value of that minimizes J is given in closed form by the equation:

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

Comments on the normal equation

- In most situations of practical interest, the number of data points N is larger than the dimensionality k of the input space and the matrix **X** is of full column rank. If this condition holds, then it is easy to verify that X^TX is necessarily invertible.
- The assumption that X^TX is invertible implies that it is positive definite, thus at the critical point we have found is a minimum.
- What if **X** has less than full column rank → Regularization

Regularized least squares

The total error function:

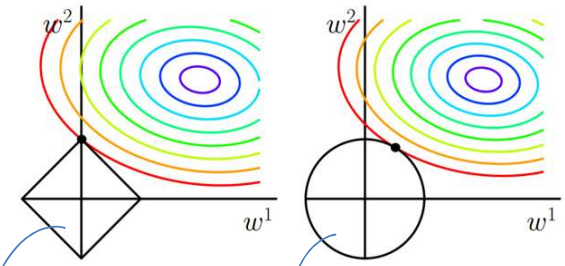
$$\frac{1}{2}\sum_{n=1}^{N}(t_n - \boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \frac{\lambda}{2}\boldsymbol{w}^\top\boldsymbol{w}$$

$$\boldsymbol{w} = (\lambda\mathbf{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}$$

Regularization has the advantage of limiting the model complexity (the appropriate number of basis functions).
This is replaced with the problem of finding a suitable value of the regularization coefficient .

Regularized Least Squares

- Lasso tends to generate sparser solutions than a quadratic regularizer.



(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes."

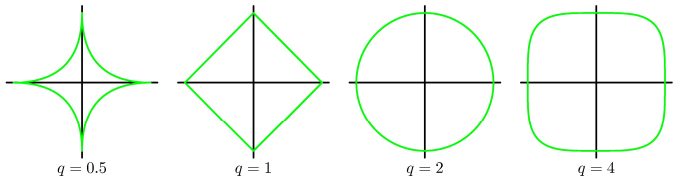
$||w_1| + |w_2| \leqslant t$

$w_1^2 + w_2^2 \leqslant t$

Regularized Least Squares

- With a more general regularizer, we have

$$\frac{1}{2}\sum_{i=1}^{m}\{h_\theta(x^{(i)}) - y^{(i)}\}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}|\theta_j|^q$$



Lasso Quadratic

Direct and Iterative methods

- Direct methods: we can achieve the solution in a single step by solving the normal equation
  - Using Gaussian elimination or QR decomposition, we converge in a finite number of steps
  - It can be infeasible when data are streaming in in real time, or of very large amount
- Iterative methods: stochastic or steepest gradient
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate α

Probabilistic Interpretation of LMS

- Let us assume that the target variables and the inputs are related via the equation

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

Where $\epsilon^{(i)}$ is an error term of unmodeled effects or random noise, and distributed i.i.d.

- Now assume that $\epsilon^{(i)}$ follows a Gaussian $N(0,\sigma)$, then we have:

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

Probabilistic Interpretation of LMS

- When we wish to explicitly view this as a function of θ, we will instead call it the likelihood function:

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta)$$

- By independence assumption:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}) \end{aligned}$$

Probabilistic Interpretation of LMS

- Hence the log-likelihood is:

$$\begin{aligned} \iota(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}) \\ &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}) \\ &= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \times \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2 \end{aligned}$$

- Do you recognize the last term? maximizing $\iota(\theta)$ gives the same answer as minimizing $J(\theta)$
- Thus under independence assumption, LMS is equivalent to MLE of θ!

Locally weighted linear regression (LWR)

The algorithm:

- Instead of minimizing $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$
- Now we fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$
- Where do $w^{(i)}$'s come from $w^{(i)} = exp(-\frac{(x^{(i)} - x)^2}{2\tau^2})$
- Where **x** is the query point for which we'd like to know its corresponding **y**
- Essentially we put higher weights on (errors on) training examples that are close to the query point (than those that are further away from the query)

Parametric vs. non parametric

- Locally weighted linear regression is the first example we are running into of a **non-parametric** algorithm.
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm because it has a fixed, finite number of parameters (θ), which are fit to the data;
- Once we've fit the θ and stored them away, we no longer need to keep the training data around to make future predictions.
- In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around.
- The term "non-parametric" (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis grows linearly with the size of the training set.

Classification and Logistic Regression

Supervised Learning

- **Regression**

Market information up to time t → Share Price "$ 24.50" — Continuous Labels — Regression

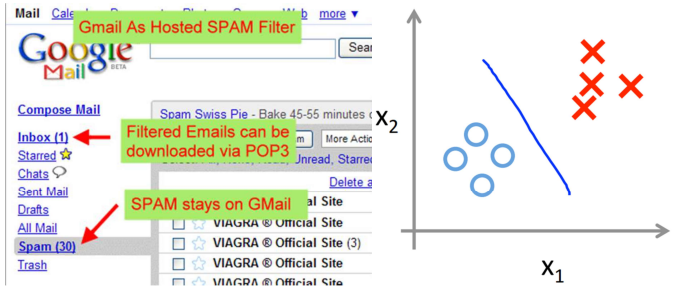
- **Classification**

Feature Space $\mathcal{X}$ → **Label** Space $\mathcal{Y}$

Words in a document → "Sports" "News" "Science" … — Discrete Labels — Classification

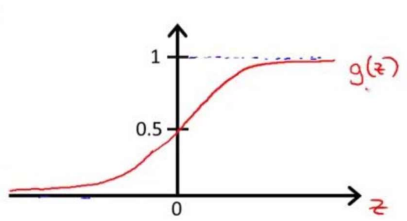
Introduction

- Logistic Regression is a classification model, although it is called "regression"
- It is a binary classification model
- It is a linear classification model



The logistic function

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



Model Description

- Hypothesis

$$P(y=1|x;\theta) = h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$
$$P(y=0|x;\theta) = 1 - h_\theta(x)$$

- Compact Form

$$P(y|x;\theta) = (h_\theta(x))^y (1-h_\theta(x))^{1-y}$$

- Parameters θ

Maximum Likelihood Estimation

- (Conditional) Likelihood

$$\begin{aligned} L(\theta) &= p(\vec{y}|X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1-h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

- Log-likelihood

$$\begin{aligned} \iota(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})) \end{aligned}$$

Cross-Entropy

Unconstraint Optimization Methods

- Problem

$$argmax_\theta \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))$$
$$where \quad h(x) = \frac{1}{1+exp^{-\theta^T x}}$$

- Optimization Methods
  - Gradient Descent
  - Stochastic Gradient Descent
  - Newton Method
  - Quasi-Newton Method
  - Conjugate Gradient
  - …

Gradient Ascent

- Property of sigmoid function:

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})}(1-\frac{1}{(1+e^{-z})}) \\ &= g(z)(1-g(z)) \end{aligned}$$

Gradient Ascent

- Gradient

$$\begin{aligned} \frac{\partial L(\theta)}{\partial \theta_j} &= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{1}{1-h_\theta(x^{(i)})})\frac{\partial}{\partial \theta_j}h_\theta(x^{(i)}) \\ &= \sum_{i=1}^{m}(y^{(i)}\frac{1}{h_\theta(x^{(i)})} - (1-y^{(i)})\frac{1}{1-h_\theta(x(i))})h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))\frac{\partial}{\partial \theta_j}\theta^T x^{(i)} \\ &= \sum_{i=1}^{m}(y^{(i)}(1-h_\theta(x^{(i)})) - (1-y^{(i)})h_\theta(x^{(i)}))x_j \\ &= \sum_{i=1}^{m}(y - h_\theta(x^{(i)}))x_j \end{aligned}$$

- Batch Gradient Ascent Method

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Stochastic Gradient Ascent

- Randomly choose a training sample (x, y)
- Compute gradient

$$(y - h_\theta(x))x_j$$

- Updating weights

$$\theta_j := \theta_j + \alpha(y - h_\theta(x))x_j$$

- Repeat …

Batch Gradient Ascent -- batch updating

Stochastic Gradient Ascent—online updating

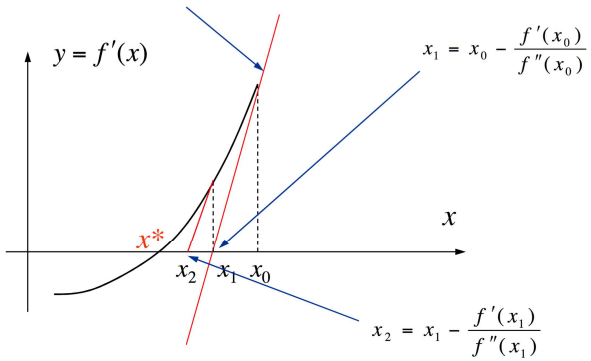
The Newton's method

- Finding a zero of a function

$$\theta^{t+1} := \theta^t - \frac{f(\theta^t)}{f'(\theta^t)}$$

Illustration of Newton's Method

- Tangent line: $y = f'(x_0) + f''(x_0)(x - x_0)$



Newton's Method

- Problem

$$\arg\min f(x) \iff solve: \nabla f(x) = 0$$

- Second-order Taylor expansion

$$\phi(x) = f(x^{(k)}) + \nabla f(x^{(k)})(x - x^{(k)}) + \frac{1}{2}\nabla^2 f(x^{(k)})(x - x^{(k)})^2 \approx f(x)$$

$$\nabla\phi(x) = 0 \Rightarrow x = x^{(k)} - \nabla^2 f(x^{(k)})^{-1}\nabla f(x^{(k)})$$

- Newton's method (also called Newton-Raphson method)

$$x^{(k+1)} = x^{(k)} - \nabla^2 f(x^{(k)})^{-1}\nabla f(x^{(k)})$$

The Newton-Raphson method

- In LR the θ is vector-valued, thus we need the following generalization:

$$\theta := \theta - H^{-1}\nabla_\theta \iota(\theta)$$

Here, $\nabla_\theta \iota(\theta)$ is, as usual, the vector of partial derivatives of $\iota(\theta)$ with respect to the θ_i's; and H is an n-by-n matrix (actually, n + 1-by-n + 1, assuming that we include the intercept term) called the Hessian, whose entries are given by

$$H_{ij} = \frac{\partial^2 \iota(\theta)}{\partial\theta_i \partial\theta_j}$$

Newton's Method for Logistic Regression

- Problem

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

- Gradient and Hessian Matrix

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

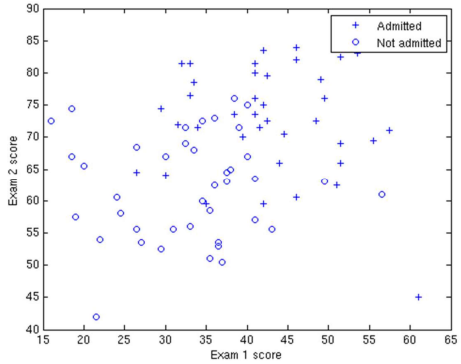
$$H = \frac{1}{m}\sum_{i=1}^{m} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T$$

- Weight updating using Newton's method

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla J(\theta^{(t)})$$

An Example of Logistic Regression

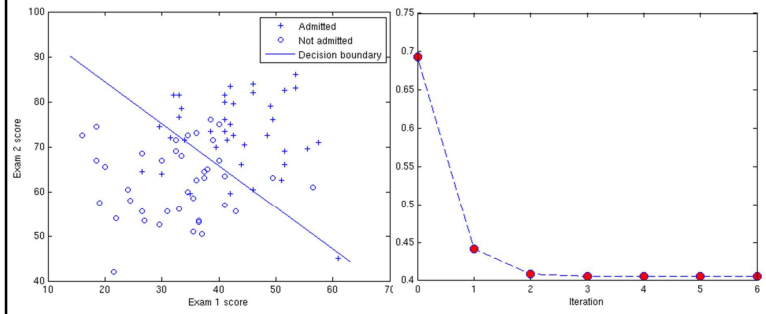
- Training data set



http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course =DeepLearning&doc=exercises/ex4/ex4.

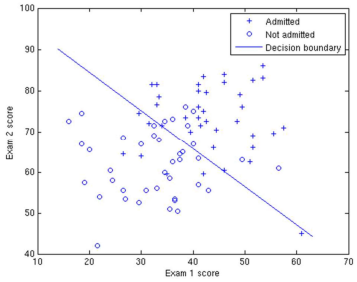
An Example (using Newton's method)

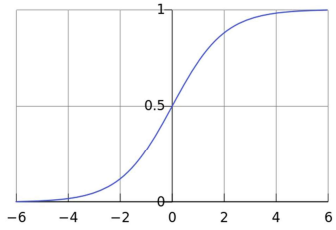
- Hyper-plane
- Cost-functions



A Linear Classification Model

- Logistic regression has a linear decision boundary (hyperplane)
- But with a nonlinear activation function (Sigmoid function) to model the posterior probability





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

Case study: predicting gene expression

The genetic picture

causal SNPs

CGTTTCACTGTACAATTT

a univariate phenotype:

i.e., the expression intensity of a gene

Association Mapping as Regression

	Phenotype (BMI)	Genotype
Individual 1	2.5	..C..... T..C.....T... ..C..... A..C.....T...
Individual 2	4.8	..G.....A..G.....A... ..C..... T..C.....T...
⋮		
Individual N	4.7	..G.....T..C.....T... ..G.....T..G.....T...

Benign SNPs Causal SNP

Association Mapping as Regression

	Phenotype (BMI)	Genotype
Individual 1	2.5	..0..... 1..0.....0...
Individual 2	4.8	..1 .....1..1.....1...
⋮		
Individual N	4.7	..2 .....2..1.....0...

$$y_i = \sum_{j=1}^{J} x_{ij}\beta_j$$

SNPs with large $|\beta_j|$ are relevant

Linear discriminant analysis

- One way to view a linear classification model is in terms of dimensionality reduction.
  - Consider first the case of two classes, and suppose we take the n dimensional input vector x and project it down to one dimension
    $$y = \mathbf{w}^T\mathbf{x}$$
  - We can place a threshold on y and classify y as class C_1/C_2
- In general, the projection onto one dimension leads to a considerable loss of information, and classes that are well separated in the original n-dimensional space may become strongly overlapping in one dimension
- However, by adjusting the components of the weight vector **w**, we can select a projection that maximizes the class separation

Linear discriminant analysis

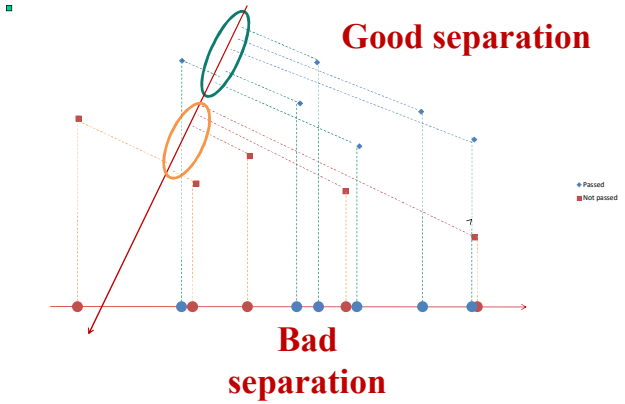
- Linear discriminant analysis is another way of finding a linear transformation that reduces the number of dimensions
- Unlike PCA or ICA it uses labeled data: it is a supervised technique, but designed for classification
- Often used for dimensionality reduction prior to classification, but can be used as a classification technique itself
- When used for dimensionality reduction, it yields (k-1) dimensions for a k-class classification problem

48

Purpose

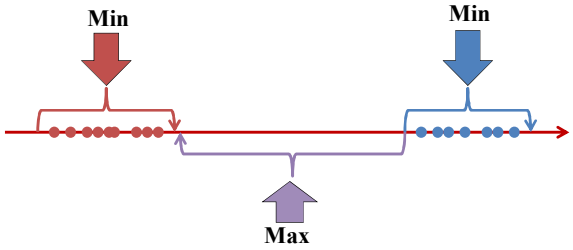
- Discriminant Analysis classifies objects in two or more groups according to linear combination of features
- Dimensionality reduction
  - Which set of features can best determine group membership of the object?
- Classification
  - What is the classification rule or model to best separate those groups?

Method (1)



Method (2)

- Maximize the between-class scatter
  - Difference of mean values (m1-m2)
- Minimize the within-class scatter
  - Covariance



Fisher's linear discriminant analysis

- Let us now consider Fisher's LDA projection for dimensionality reduction, considering the two-class case first
- We seek a projection vector a that can be used to compute scalar projections $y = \mathbf{a}^T\mathbf{x}$ for input vectors x
- This vector is obtained by computing the means of each class, μ_1 and μ_2, and then computing two special matrices
- The between-class scatter matrix is calculated as

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T$$

(note the use of the outer product of two vectors here, which gives a matrix)

- The within-class scatter matrix is

$$\mathbf{S}_W = \sum_{i:c_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T + \sum_{i:c_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T$$

Fisher's LDA: the solution vector

- The solution vector a for FLDA is found by maximizing the "Rayleigh quotient"

$$J(\mathbf{a}) = \frac{\mathbf{a}^{\mathrm{T}}\mathbf{S}_B\mathbf{a}}{\mathbf{a}^{\mathrm{T}}\mathbf{S}_W\mathbf{a}}$$

- This leads to the solution

$$\mathbf{a} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

Multi-class FLDA

- FLDA can be generalized to multi-class problems
- Aim: projection **A** so that $y = \mathbf{A}^{\mathrm{T}}\mathbf{x}$ yields points that are close when they are in the same class relative to the overall spread
- To do this, first compute both the means for each class and the global mean, and then compute the scatter matrices

$$\mathbf{S}_B = \sum_{c=1}^{C} n_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^{\mathrm{T}} \qquad \mathbf{S}_W = \sum_{j=1}^{C}\sum_{i:c_i=j}(\mathbf{x}_i - \boldsymbol{\mu}_{c=j})(\mathbf{x}_i - \boldsymbol{\mu}_{c=j})^{\mathrm{T}}$$

- Then, find the projection matrix A that maximizes $J(\mathbf{A}) = \dfrac{|A^{\mathrm{T}}\mathbf{S}_B A|}{|A^{\mathrm{T}}\mathbf{S}_W A|}$

  Determinants are analogs of variances computed in multiple dimensions, along the principal directions of the scatter matrices, and multiplied together
- Solutions for finding **A** are based on solving a "generalized eigenvalue problem" for each column of the matrix **A**.

From Two-Class to Multiple-Class

Model:

- Binary: One-vs-One
- Binary: One-vs-Rest
- Binary: ECOC

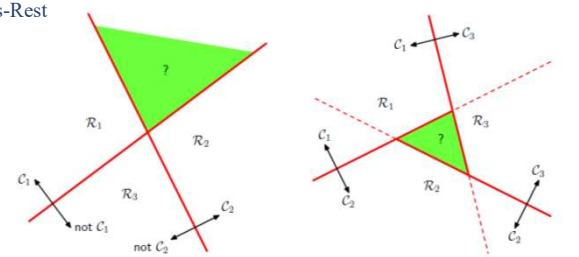


Figure 4.2 Attempting to construct a K class discriminant from a set of two class discriminants leads to ambiguous regions, shown in green. On the left is an example involving the use of two discriminants designed to distinguish points in class C_k from points not in class C_k. On the right is an example involving three discriminant functions each of which is used to separate a pair of classes C_k and C_j.

From Two-Class to Multiple-Class

- Decompose a n-class problem into several two-classes problems:
- One-vs-Rest: Use n - 1 binary classifiers each of which solves a two-class problem of separating points in a particular class from points not in that class.
- Pros: small storing cost and short test time
- Cons: long training time, imbalanced Samples.
- One-vs-One: Use (n - 1)n/2 binary classifiers, one for every possible pairs of classes. Each point is classified according to a majority vote amongst the discriminant functions.
- Pros: short training time
- Cons: Too many classifiers; large storing cost and long test time

A case study

Classification (Discrimination, Supervised Learning) Using Microarray Data

Gene expression data

mRNA samples

Genes	Normal sample1	Normal sample2	Normal sample3	Cancer sample4	Cancer sample5	...
1	0.46	0.30	0.80	1.51	0.90	...
2	-0.10	0.49	0.24	0.06	0.46	...
3	0.15	0.74	0.04	0.10	0.20	...
4	-0.45	-1.03	-0.79	-0.56	-0.32	...
5	-0.06	1.06	1.35	1.09	-1.09	...

Gene expression level of gene *i* in mRNA sample *j*

Tumor Classification Using Gene Expression Data

- Three main types of statistical problems associated with the microarray data:
  - Identification of "marker" genes that characterize the different tumor classes (feature or variable selection).
  - Identification of new/unknown tumor classes using gene expression profiles (unsupervised learning – clustering)
  - Classification of sample into known classes (supervised learning – classification)

Classification

Y	Normal sample1	Normal sample2	Normal sample3	Cancer sample4	Cancer sample5	...	unknown =Y_new New sample
1	0.46	0.30	0.80	1.51	0.90	...	0.34
2	-0.10	0.49	0.24	0.06	0.46	...	0.43
3	0.15	0.74	0.04	0.10	0.20	...	-0.23
4	-0.45	-1.03	-0.79	-0.56	-0.32	...	-0.91
5	-0.06	1.06	1.35	1.09	-1.09	...	1.23
	X						X_new

Each object (e.g. arrays or columns)associated with a class label (or response) $Y \in \{1, 2, \ldots, K\}$ and a feature vector (vector of predictor variables) of G measurements: $X = (X_1, \ldots, X_G)$

Aim: predict Y_new from X_new.

Classification Methods

- Logistic Regression
- Fisher Linear Discriminant Analysis
- Nearest Neighbor Classification