Minimzing MSE for Linear Models

$f(x, \theta)$ is our model of y given x.

Linear model: $f(x, \theta) = \theta^T x$

In Linear Regression $f(x, \theta) = \theta^T x$

Linear models are linear in the parameters but can be non-linear in the inputs x_j
Non-linear models are not linear in both senses

$$f(x, \theta) = \sum g_k(x, \theta_k)$$

Our initial goal is to minimize the mean squared error

$$MSE(\theta) = \frac{1}{N} \sum (y_i - f(x_i, \theta))^2$$

If $f(x, \theta)$ is linear, then minimizing is solving a set of p linear equations, with p as number of parameters. If not linear, it is a much more complex problem and gradient methods have to be used.

Minimizing MSE with Linear Model

Let $X_D = (x_1^T, ..., x_N^T)$, an $N \times d$ matrix and $Y_D = (y_1, ..., y_N)$, an $N \times 1$ matrix. Then the following is true

$$MSE(\theta) = \sum_{i=1}^{N} (y_i - \theta^T x_i)^2 = (Y_D - X_D\theta)^T (Y_D - X_D\theta)$$

It is minimized when

$$X_D^T X_D \theta = X_D^T Y$$

This is a set of d simultaneous equations.

$$\hat{\theta}_{MSE} = (X_D^T X_D)^{-1} X_D^T Y_D$$

Complexity of solving equations or finding matrix inverse is $O(Nd^2 + d^3)$

Minimizing MSE in general

$MSE(\theta)$ is concave so you can use gradient descent for linear and non-linear models:

$$\theta^{new} = \theta^{current} - stepSize \cdot \nabla(MSE)$$

Stochastic Gradient Descent can also be used, faster but noisier

$$\nabla(MSE) = \frac{\partial MSE}{\partial \theta_j}$$

Probabilistic Interpretation of Regression

$p(y|x)$: for fixed x, there is variation in y
2 types of variation:
- Measurement noise
- Unobserved Variables
2 sources of variability:
$p(y|x)$: variability in y given x
$p(x)$: distribution of input data in the space
We have a joint distribution: $p(x, y) = p(y|x)p(x)$ and we learn $p(y|x)$

Modeling Framework

$y_x = E[y|x] + e$ where
y_x: what we observe
$E[y|x]$: what we try to learn with $f(x, \theta)$
e: unpredictable error term

Simple Model:

$p(y|x) = N(f(x, \theta), \sigma^2)$ where $f(x, \theta) = \theta^T x$

Conditional Likelihood for Regression

$$L(\theta) = \prod p(y_i|x_i, \theta)$$

As an example, we can use a Gaussian model for $p(y|x, \theta)$.
This will give us

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

This equation comes from the fact that $E[y|x] = f(x, \theta)$. After doing some algebra

$$logL(\theta) \propto -MSE(\theta)$$

Thus maximizing log likelihood is the same as minimizing MSE. This gives us a useful framework to go beyond.

Bayesian View of Regression

Posterior Density on θ is:

$$p(\theta|D_x, D_y) \propto p(D_x, D_y|\theta)p(\theta) = p(D_y|D_x, \theta)p(D_x|\theta)p(\theta)$$

$$p(\theta|D_x, D_y) \propto p(D_y|D_x, \theta)p(\theta)$$

This is because we do not model $p(D_x|\theta)$

Gaussian Error Model

$$L(\theta) = p(D_y|D_x, \theta) \propto \prod exp(-\frac{1}{2}(\frac{y_i - f(x;\theta)}{\sigma})^2)$$

We model the y_i as CI given x_i and θ.

$$p(y_i|x_i) = N(f(x_i, \theta), \sigma^2)$$

Common to have independent priors on the θ_j

$$p(\theta) = \prod p(\theta_j)$$

As an example

$$p(\theta_j) = N(0, \sigma_0^2)$$

Eventually we have

$$-log(p(\theta|D_x, D_y)) \propto \sum (y_i - f(x_i;\theta))^2 + \lambda \sum \theta_j^2$$

where $\lambda = \frac{\sigma^2}{\sigma_0^2}$
So the negative of our log likelihood is the squared error plus λ times the sum of the weights squared.
In non-Bayesian setting, λ may be fit using cross-validation on our data to determine our best value.
In Bayesian setting, we would instead compute λ using our prior.
Could use Laplacian Prior, which is proportional to $e^{-|\theta_j|}$. It will push the weights toward zero.
Another prior is the spike and slab, consists of a mixture of delta function and a Gaussian, both centered at 0

Properties of Minimizing MSE

Because of this fact:

$$MSE(\theta) = \int \int (y - f(x;\theta))^2 p(x, y)\, dx\, dy$$

It eventually (**TODO: make sure you can derive this. 2-19 notes**) holds that the optimal value is when $f(x;\theta) = E[y_x]$.

We are limited because
Bias: $f(x, \theta)$ might not be able to exactly approximate $E[y_x]$
Variance: Even if so, only have finite data to learn $E[y_x]$
Tradeoff exists between complex model with low bias but high variance and
simple model with high bias and low variance

Bias-Variance Tradeoff

TODO: DERIVE THIS FROM 2-19 NOTES Eventually we have $MSE_x = \sigma_{yx}^2 + Bias^2 + Variance$
This leads to the fundamental bias-variance tradeoff

- simple models with few parameters have high bias, but low variance
- complex models with many parameters have low bias, but high variance

Logistic Regression Classifier

Regression but $y_i \in 0, 1$ and $p(y|x)$ is the logistic function. Good explanation of Logistic Regression:
http://www.stat.cmu.edu/ cshalizi/350/lectures/26/lecture-26.pdf
We use the log function so that changing an input variable multiplies the probability by a fixed amount. To get an unbounded range, we use the logistic transformation, $log(\frac{p}{1-p})$. We make this a linear function. This gives us the logistic regression equation.

$$p(c = 1|x) = \frac{1}{1 + exp(-(\theta^T x + \theta_0))}$$

Defines linear decision boundary, since we set $p(c = 1|x) = 0.5$ and solve the resulting linear equation for x. Can write as log-odds

$$log(\frac{p(c = 1|x)}{p(c = 0|x)})$$

Let $\theta = (\theta_1, ..., \theta_d)$ and $x = (x_1, ..., x_d)$. Assume the dth entry is always 1 because this accounts for the bias. Conditional Likelihood is

$$L(\theta) = \prod p(c_i|x_i, \theta)$$

Let $f(x, \theta) = p(c_i = 1|x_i, \theta)$ then

$$L(\theta) = \prod f(x_i; \theta)^{c_i}(1 - f(x_i; \theta))^{1-c_i}$$

$$l(theta) = \sum c_i log(f(x_i; \theta)) + (1 - c_i) log(1 - f(x_i; \theta))$$

This is the log-loss function. No closed form solution to $\hat{\theta}_{ML}$.

Newton-Raphson

Basic Iteration of newton-raphson:

$$\theta^{new} = \theta^{current} + H^{-1} \nabla l(\theta)$$

where H is the Hessian defined by

$$H_i j = \frac{\partial^2}{\partial i \partial j} l(\theta)$$

$\nabla l(\theta)$ is the gradient vector
Gradient can be written as $X^T(f - c)$ where $f_i = f(x_i, \theta)$
Hessian be written as $-X^T V X$ where V is diagonal with $v_i i = f_i(1 - f_i)$
We end up with $\theta^{new} = \theta^{old} + (X^T V X)^{-1} X^T (f - c)$
It takes $O(d^3)$ operations to invert the Hessian matrix

Gradient Descent

With this, we have

$$\theta^{new} = \theta^{current} + \epsilon \nabla l(\theta)$$

It has complexity $O(dN)$

Stocastic Gradient Descent

We often use Stochastic Gradient which in the extreme case each update is based on a single data point

$$\theta^{new} = \theta^{current} + \epsilon (f_i - c_i) x_i$$

Parameters are updated after each visit, so there are N updates per iteration. Can converge before looking at all the data.

Multi-class Logistic Regression

$$p(c = k|x) = \frac{exp(\theta_k^T x)}{\sum exp(\theta_k^T x)}$$

We have a weight vector for each class.

Logistic Functions and Feed-Forward Neural Networks

Example of feed forward neural network
Diagram with x_1, x_2, x_3 each connected to $h_1(x), h_2(x)$ which are connected to $g(x)$
h_j are known as hidden units, are non linear functions of $w_j^T x$. Often function used is logistic function.
$g(h_1(x), ..., h_H(x)) = g(h)$ is a linear or nonlinear function of $w^T h(x)$
For learning, minimize $\sum (y_i - g(x_i; \theta))^2$ as function of θ (could use log-loss for binary y_i)
Backpropagation can be considered as an effective way to do gradient calculation.

Generative Approach to Classification

We model $p(x|c)$ instead of $p(c|x)$. Models are generative when we model the distribution of both x's and c's.

$$L(\theta) = \prod p(x_i|c_i, \theta)p(c_i)$$

Key Points:

- learn how x_i values are distributed for each class, so θ_k is set of parameters for class k model.
- learn p(c=k)
- possibly decomposes into k optimization problems
- it is optimal if distributional assumptions are correct
- predict using bayes rule

  $$p(c = k|x, \theta) \propto p(x|c = k, \theta_k)p(c = k)$$

  or we could be Bayesian and average over the θ values.

Weaknesses of Gaussian model for each class:

- sensitive to Gaussian assumption
- scales poorly as d increases. for high dimensions, we can assume covariance matrices are diagonal.

Naive Bayes Model:
If x_i are binary vectors, we can use a Naive Bayes model for each class. Parameters are $\theta_k = \{\theta_{k1}, ..., \theta_{kd}\}$ where θ_{kj} is Bernoulli probability that $x_{ij} = 1$
*FIRST ORDER MARKOV MODEL EXAMPLE WILL NOT BE ON THE FINAL**

Discriminant Functions

To make a decision about mostly class, we can compute $argmax_k\, p(c = k|x)$
Using Bayes rule, this is $argmax_k\, p(x|c = k)p(c = k)$
This is equal to $argmax_k\, log(p(x|c = k)) + log(p(c = k))$
All of these can be used as discriminant functions $g_k(x)$

For 2 class case, decision boundary is when $g(x) = g_1(x) - g_2(x) = 0$

For Multivariate Gaussian classifier, $g_k(x) = logp(x|c_k) + logp(c_k)$
and it holds that $p(x|c_k) = N(\mu_k, \Sigma_k)$, thus g_k can be derived easily.

Finite Mixture Models Definition

Definition is

$$p(x) = \sum p(x, z) = \sum p_k(x|\theta_k, z = k)p(z = k)$$

where $\sum p(z = k) = 1$
The z variable is hidden and the densities can be a mixture

Learning Mixture Models

Given data $D = x_i$ and form of each component $p_k(x|\theta_k)$.
Pseudo-code:
for i = 1:N

- sample component $p(z = k)$ for i-th data point, call it k*
- sample x_i from $k*$. $p_k(x|\theta_k, z = k*)$

Let α_k be the component weight, then log likelihood is

$$l(\theta) = \sum log(p(x_i|\theta)) = \sum log \sum p_k(x_i|\theta_k)\alpha_k$$

The EM algorithm

We assume that

$$p(x|\Theta) = \sum \alpha_k p_k(x|z_k, \theta_k)$$

$p_k(x|z_k, \theta_k)$ are mixture components
z is a vector of binary indicator variables

$\alpha_k = p(z_k)$ is the probability that a randomly chosen x was generated by component k.
Can compute membership weights as follows:

$$w_{ik} = p(z_{ik} = 1|x_i, \Theta) = \frac{p(x_i|z_k, \theta_k)\alpha_k}{\sum p_m(x_i|z_m, \theta_m)\alpha_m}$$

This is just from direction application of Baye's rule
Each distribution is a multivariate Gaussian for our case.
E-step: compute the membership weights using above formula and α_k values.
Note that $\sum w_{ik} = 1$
M-step: Calculate new parameter values. Let $N_k = \sum w_{ik}$ which is effectively the number of components assigned a particular weight. Then

$$\alpha_k^{new} = \frac{N_k}{N}$$

$$\mu_k^{new} = \frac{1}{N_k} \sum w_{ik}.x_i$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum w_{ik}.(x_i - \mu_k^{new})(x_i - \mu_k^{new})^t$$

Convergence is defined using the log-likelihood and seeing when it stops changing significantly. Note that

$$l(\theta) = \sum_{i=1}^{N} (log(\sum_{k=1}^{K} \alpha_k p_k(x_i|z_k, \theta_k)))$$

k-Means clustering

Here is the procedure:
1. Randomly select K mean vectors
2. Assign each of the n data vectors to cluster corresponding to whichever mean it is closest to, using Euclidean distance
3. Compute its new mean as the mean of the data vectors that were assigned to this cluster in Step 2
4. Check for convergence: see if in step 2, any vectors changed cluster assignments
One can reduce Gaussian mixture to k-means by doing the following:
1. Fix all the covariances for the K components to be identity matrix and not update them during the M-step
2. during the E-step, assign membership probability of 1 for component it is most likely to belong to and 0 for all other memberships.

Notes about EM

- EM converges to a local minimum of the likelihood or log-likelihood. We are essentially doing a search in θ space.
- similar to gradient ascent but more popular because step size is chosen automatically
EM respects parameter constraints

The rate at which EM converges is a function of how much missing info is in the problem. Two Gaussians close together has a lot of missing info, but if they are far apart, the uncertainty is lower.
EM can be mixed with other methods

Variations of EM

Online EM: in the E and M step only use a subset of points. Useful when N is large. After many noisy steps, it eventually converges
Generalized EM: In the M-step, instead of maximizing $l(\theta)$, we just move uphill. Useful when M step cannot be computed in closed form.
Approximate E steps: in some models, computing the likelihood in the E step so we can replace the E step with an approximation, such as Monte Carlo sampling
Semi-supervised learning: some x_i have labels and some do not. This is handled nicely in EM since we can fix our weights for the x_i that have values.

Singular Solutions in EM

To avoid it, common solution is to set lower limit on σ_k^2.
Another method is to put priors $p(\theta)$ on parameters and extend EM to compute MAP estimates instead of ML in the M step. Gives us a weighted average of our usual estimate

$$\Sigma_{MAP,k} = \alpha\Sigma_k^{ML} + (1-\alpha)\Sigma_0$$

The prior Σ_0 is usually chosen to be diagonal.

Kernal Density Estimation

Idea is that at each data point, we put a Gaussian. We fix the covariance matrix. Otherwise, the variance would converge to zero.

$$p(x) = \frac{1}{n}\sum N(x; x_i, \Sigma)$$

If the covariance is large, there will be a good smooth curve. Otherwise, there will be many peaks and valleys.

Hidden Markov Models

At each time t, z_t produces a new data point x_t and then transitions to a new state z_{t+1}
Two assumptions:
observations x_t are CI of all other variables given z_t so observation at time t depends only on the current

state
the z_t values form a first order Markov Chain.
Equation from graphical model is the following

$$p(x_{1...T}, z_{1...T}) = \prod p(x_t|z_t)p(z_t|z_{t-1})$$

Two sets of parameters:
1. Transition Matrix that is K by K. $a_ij = p(z_t = j|z_{t-1} = i)$
2. K emission distributions/densities $p(x|z = j)$

Computing Likelihood of HMMs

By Law of Total Probability,

$$\alpha_T(j) = p(z_T = j) = \sum p(z_T = j, z_{T-1} = i, x_{1...T})$$

By graphical model structure

$$\alpha_T(j) = \sum p(x_T|z_T = j)p(z_T = j|z_{T-1} = i, X_{1...T-1})$$

Given $\alpha_{T-1}(i)$ we can compute $\alpha_T(j)$ in time $O(K^2 + Kf(d))$.
First term is due to face that we need to compute all i,j pairs
function f includes complexity of computing the likelihood of our data point

DERIVATION OF FORWARD-BACKWARD ALGORITHM FOR HMM WILL NOT BE ON THE FINAL

FINAL MATERIAL ENDS HERE. EM FOR HMM WILL NOT BE ON FINAL