Compendium of Machine-Learning-Related Knowledge: A Quick Reference

Colorado Reed

January 6, 2012

Contents

1 Bayesian Hypothesis Testing **3**
1.1 Bayes Factor . . . 3

2 Bayesian Probability **3**
2.1 Conjugate Prior . . . 3
2.2 Nonparametric Bayesian Methods . . . 3

3 Basic Classification **3**
3.1 Linear Discriminant Analysis . . . 3
3.1.1 K = 2 Classification . . . 4
3.1.2 K > 2 Classification . . . 4
3.2 Logistic Regression . . . 6
3.2.1 Two Class IRLS . . . 6

4 Deep Learning **7**
4.1 Quick Overview . . . 7
4.2 Current Challenges . . . 7

5 Dimensionality Reduction **7**
5.1 PCA . . . 7
5.1.1 Maximum variance formulation of PCA . . . 8
5.2 Independent Component Analysis . . . 8
5.2.1 ICA Software . . . 9
5.3 Local Linear Embedding . . . 9

6 Distributions **10**
6.1 Dirichlet . . . 10
6.1.1 Introduction . . . 10
6.1.2 Generating Samples from Dirichlet . . . 10
6.2 Gaussian . . . 11

7 Expectation-Maximization **11**
7.1 General Description . . . 12
7.2 Gaussian Mixture EM . . . 13
7.3 Further Reading . . . 15

8 Generalized Linear Model **15**

9 Kernel Methods **16**
9.1 Dual Representation . . . 16

10 Model Selection **16**
10.1 Bias-Variance Decomposition . . . 16

11 Neural Networks **17**
11.1 Multilayer Perceptrons . . . 17
11.2 Back Propagation . . . 18
11.2.1 Rules of Thumb . . . 21

12 Nonparametric Bayesian Methods **21**
12.1 Model Selection . . . 21
12.2 Gaussian Processes . . . 21
12.3 Dirichlet Processes . . . 21
12.3.1 Sampling from DP . . . 22
12.3.2 Applications of DP . . . 23

13 Optimization Techniques **23**
13.1 Gradient Descent . . . 23
13.2 Stochastic Gradient Descent . . . 23

14 Preprocessing **24**
14.1 Data Whitening . . . 24

15 Regression **24**

16 Regularization **24**
16.1 weight decay . . . 24

17 Sampling Methods **24**
17.1 Sequential Monte Carlo Methods [Particle Filters] . . . 24

18 Appendix: Calculus **24**
18.1 Matrix Derivatives . . . 24
18.2 Calculus of variations . . . 25
18.3 Lagrange Multipliers . . . 25

19 Appendix: Definitions **25**

20 Appendix: Distances **25**
20.1 Mahalanobis Distance 25
20.2 Statistical Tests to Compare Distributions 25
20.2.1 Kolmogorov-Smirnoff Test 25

21 Appendix: Information Theory **26**
21.1 Entropy 26

22 Appendix: Pragmatic Tips **26**

1 Bayesian Hypothesis Testing

1.1 Bayes Factor

Assuming a uniform prior is the same as using maximum likelihood.

2 Bayesian Probability

2.1 Conjugate Prior

If the posterior and prior distributions are in the same family they are called conjugate distributions, and the prior is called the *conjugate prior* for the likelihood. So given a fixed likelihood function, we choose a prior such that the posterior has the same algebraic form. A conjugate prior is an algebraic convenience. All members of the exponential family have conjugate priors. Using a conjugate prior allows for an easy online implementation, where the posterior for some data becomes the prior of new data, etc. Ex. if our likelihood is Gaussian, and the mean is known, then we can use an inverse Gamma distribution as the conjugate prior for the variance.

2.2 Nonparametric Bayesian Methods

Bayesian methods with an infinite number of parameters. Examples: ... (Dirichlet Process Mixture Mode: a Bayesian density model)

3 Basic Classification

3.1 Linear Discriminant Analysis

A type of linear classification where the central idea is to project the D-dimensional data onto a lower dimensional manifold for classification.

3.1.1 K = 2 Classification

From (PRML 4.1), given a feature vector $\mathbf{x}$ for each instance, the goal of Fisher's linear discriminant is to project the feature space to a single dimension and linearly separate the classes, where any instance with a projected value $y \leq y_0$ is class 1 and any instance with a projected value $y \geq y_0$ is class 2 where

$$y = \mathbf{w}^\top \mathbf{x} \tag{1}$$

and y_0 is chosen to discriminate the two classes (discussed further below).

We want to maximize the class separation by adjusting $\mathbf{w}$. Fisher's idea was to project $\mathbf{x}$ so as to maximize the between class variance while minimizing the interclass variance. Formally, let $m_k = \mathbf{w}^\top \mathbf{m}_k$, where $\mathbf{m}_k$ is the mean in the D-dimensional space and let $s_k^2 = \sum_{n \epsilon C_k} (y_n - m_k)^2$, which is the inter-class variance for the projected data. The Fisher criterion is:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{2}$$

which can be reformulated with explicit dependence on $\mathbf{w}$ via:

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}} \tag{3}$$

where $\mathbf{S}_B$ is the between class variance, $\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top$, and $\mathbf{S}_W$ is the within class variance, $\mathbf{S}_W = \sum_k \sum_{n \epsilon C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top$. Taking the derivative of $J(\mathbf{w})$ we find that $\mathbf{w}$ is maximized when

$$\mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1) \tag{4}$$

which is the answer we seek since only the direction of $\mathbf{w}$ matters (differences in scale will manifest uniformly for all project data so we will have the same classification as longs as the direction is the same). This same solution can be derived from a specific formulation of the least squares solution (see PRML 4.1.5). The problem is then to choose a y_0 that separates the two classes. There is not a standard way of finding y_0, but given that y is a weighted sum of random variables, from the central limit theorem we expect y to be approximately Gaussian. We can therefore infer Gaussian distributions of the classes in the projected space and use a maximum *a posteriori* estimate of y_0.

3.1.2 K > 2 Classification

The generalization to $K > 2$ is similar to $K = 2$ linear discriminant classification. We now map the D dimensional input feature space to a $D' < D$ dimensional-space via

$$y_k = \mathbf{w}_k^\top \mathbf{x}$$

for $k = 1, \ldots, D'$. For convenience, we define $\mathbf{W} = [w_1^\top | w_2^\top | \ldots | w_{D'}]$ and consequently

$$\mathbf{y} = \mathbf{W}^\top \mathbf{x}. \tag{5}$$

The within class covariance in the original feature space can be generalized from the 2D case as:

$$\mathbf{S}_W = \sum_{k=1}^{K} \sum_{n \epsilon C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^\top \tag{6}$$

and likewise for the between class covariance:

$$\mathbf{S}_B = \sum_{k=1}^{K} N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^\top \tag{7}$$

where in the above equations:

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n \ \epsilon \ C_k} \mathbf{x}_n \tag{8}$$

$$\mathbf{m} = \frac{1}{N} \sum_{k=1}^{K} N_k \mathbf{m}_k \tag{9}$$

where N is the total number of instances and N_k is the number of instances of class k. Likewise we can define the between class and within class covariances in the projected feature space:

$$\mathbf{s}_W = \sum_{k=1}^{K} \sum_{n \epsilon C_k} (\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^\top \tag{10}$$

$$\mathbf{s}_B = \sum_{k=1}^{K} N_k (\mu_k - \mu)(\mu_k - \mu)^\top \tag{11}$$

where

$$\mu_k = \frac{1}{N_k} \sum_{n \ \epsilon \ C_k} \mathbf{y}_n$$

and

$$\mu = \frac{1}{N} \sum_{k=1}^{K} N_k \mu_k$$

.

With these definitions our objective is to minimize the projected within class covariance and maximize the projected between class covariance. From Fukunaga's *Introduction to Statistical Pattern Recognition*, we could define the following scalar cost functions:

$$\mathrm{Tr}\left\{\mathbf{s}_W^{-1} \mathbf{s}_B\right\} \tag{12}$$

or

$$\ln \left|\mathbf{s}_W^{-1} \mathbf{s}_B\right| \tag{13}$$

or

$$\frac{\mathrm{Tr}(\mathbf{s}_B)}{\mathrm{Tr}(\mathbf{s}_W)}. \tag{14}$$

In keeping with PRML and the in-class lectures we select (12), which has an equivalent form of

$$J(\mathbf{W}) = \mathrm{Tr}\left\{(\mathbf{W}\mathbf{S}_W\mathbf{W}^\top)^{-1}(\mathbf{W}\mathbf{S}_B\mathbf{W}^\top)^\top\right\}. \tag{15}$$

In section 10.2 of Fukunaga's *Introduction to Statistical Pattern Recognition*, it is shown that the maximization of (15) corresponds to selecting the $D^{'}$ eigenvectors with the largest eigenvalues to form the column vectors of $\mathbf{W}$. Furthermore it is shown that only $K-1$ eigenvectors have a nonzero eigenvalue. We therefore choose to set $D^{'} = k-1$.

3.2 Logistic Regression

TODO: Discuss the sigmoid and its implications, etc.

3.2.1 Two Class IRLS

Iterative reweighted least squares is a stochastic approach to finding the weight vector for logistic regression. It essentially uses gradient descent to compute the maximum likelihood of $\mathbf{w}$.

Given a feature vector, $\boldsymbol{\phi}$, and target value, $t \,\epsilon\, \{0, 1\}$, for each instance of the input data (in our case, each iris flower), the goal is to learn the weight vector, $\mathbf{w}$, that optimally separates two classes via the logistic sigmoid function:

$$\sigma(\mathbf{w}^\top\boldsymbol{\phi}) = \frac{1}{1+\exp(-\mathbf{w}^\top\boldsymbol{\phi})}. \tag{16}$$

This equation is especially useful when the generative model for the data can be described as a generalized linear model because the posterior probability of class $\mathcal{C}_1$ can be written as a logistic sigmoid acting on the weighted feature vector,

$$P(\mathcal{C}_1|\boldsymbol{\phi}) = \sigma(\mathbf{w}^\top\boldsymbol{\phi}),$$

where in the case of two classes, $P(\mathcal{C}_2|\boldsymbol{\phi}) = 1 - P(\mathcal{C}_1|\boldsymbol{\phi})$ (see 4.3.2 of PRML for further explanation). Given $\mathbf{w}$, we can then probabilistically classify new data.

As described in 4.3.3 PRML. the IRLS is an iterative technique for computing the maximum likelihood estimation $\mathbf{w}$. Specifically, the IRLS uses the Newton-Raphson iterative optimization scheme, which uses local quadratic approximations to the log likelihood function to locate the extrema, in this case the $\mathbf{w}$ that maximizes the likelihood function. The Newton-Raphson update for minimizing some function, $E(\mathbf{w})$, is given via:

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}) \tag{17}$$

where $\mathbf{H}$ is the Hessian matrix of $E(\mathbf{w})$.

The likelihood function of $\mathbf{w}$ given the target calsses, $\mathbf{t} = \{t_n\}_{n=1,\ldots,N}$, can be written as

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n} \tag{18}$$

where in our case $y_n = \sigma(\mathbf{w}^\top \boldsymbol{\phi}_n)$. We then use the negative log-likelihood as our error function,

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}. \tag{19}$$

Taking the gradient and forming the Hessian of $E(\mathbf{w})$ leads to (see 4.3.3 of PRML for a straightforward derivation):

$$\mathbf{w}^{new} = \left(\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^\top \mathbf{R} \mathbf{z} \tag{20}$$

where $\mathbf{\Phi} = [\boldsymbol{\phi}_1 | \boldsymbol{\phi}_2 | \cdots | \boldsymbol{\phi}_n]$, $\mathbf{R}$ is a diagonal matrix with elements $R_{nn} = y_n(1 - y_n)$, and $\mathbf{z} = \mathbf{\Phi}\mathbf{w}^{old} - \mathbf{R}^{-1}(\mathbf{y} - \mathbf{t})$. The algorithm iterates until $\mathbf{w}^{new}$ does not deviate from $\mathbf{w}^{old}$. In the event of an oscillating $\mathbf{w}$, a common approach is to average the $\mathbf{w}$ values over a given number of iterations, e.g. 50.

The implementation of this algorithm is straightforward in Matlab and can be viewed in the attached code. The inversion, $\left(\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi}\right)^{-1}$, is not possible when the y_n probabilities converge as the matrix inside the parentheses becomes singular. Therefore I introduced a small *ridge* term, ϵ, so that the matrix was always invertible, $\left(\mathbf{\Phi}^\top \mathbf{R} \mathbf{\Phi} + \epsilon \mathbb{1}\right)^{-1}$, where $\mathbb{1}$ is the identity matrix. Finally, classification is performed by assigning the label 0 or 1 via $round(\sigma(\mathbf{w}^\top \boldsymbol{\phi}))$, with the optimal $\mathbf{w}$ vector.

4 Deep Learning

4.1 Quick Overview

Unsupervised feature learning and deep learning look to answer the question: is there a better way to find features than to manually extract them from data—can we *learn* features? E.g. can we learn features to represent images better than using pixels. Yes! E.g. string together layers of sparse coding: learns a dictionary of basis functions such that many of the bases will be 0 to represent a given object.

4.2 Current Challenges

1. How do we scale feature learning algorithms?
2. Can we make interpretable features?

5 Dimensionality Reduction

5.1 PCA

PCA is used for dimensionality reduction, lossy data compression, feature extraction and data visualization, AKA *Karhunen-Loeve* transform. Can formulate as either an orthogonal projection of the data onto a lower dimensional linear space (principal subspace)

such that the variance of the projected data is maximized. Or we can formulate it as a linear projection that minimizes the average projection cost (the mean squared distance between the data points and their projections). OR we can treat PCA in a probabilistic sense, whereby we assume a latent variable in the true subspace that generates the observed data.

5.1.1 Maximum variance formulation of PCA

Taken from PRML Chap 12. Consider data $\{x_n\}$ with D dimensions. Want to project onto dimension with M ¡ D while maximizing the variance of the projected data. First, consider M=1. Define the direction of the space using $\mathbf{u}_1$ which WLOG is a unit vector $\mathbf{u}_1^\top \mathbf{u}_1 = 1$. Let $\bar{\mathbf{x}} = \frac{1}{N}\sum_n \mathbf{x}_n$. The variance of the projected data is

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^\top \mathbf{x}_n - \mathbf{u}_1^\top \bar{\mathbf{x}})(\mathbf{u}_1^\top \mathbf{x}_n - \mathbf{u}_1^\top \bar{\mathbf{x}})^\top = \mathbf{u}_1^\top \mathbf{S}\mathbf{u}_1 \tag{21}$$

where $\mathbf{S}$ is the covariance matrix of the data. Now we maximize the projected variance $\mathbf{u}_1^\top \mathbf{S}\mathbf{u}_1$ w/r/t $\mathbf{u}_1$ with the constraint $\mathbf{u}_1^\top \mathbf{u}_1 = 1$ (so it doesn't fly off to infinity). Introducing a Lagrange multiplier we need to maximize

$$\mathbf{u}_1^\top \mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^\top \mathbf{u}_1) \tag{22}$$

Set derivative w/r/t $\mathbf{u}_1$ to 0 and we have:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \tag{23}$$

which shows that the eigenvectors of the covariance matrix are the projections that maximize the variance (and it's trivial to show that the variance is given by the eigenvalues).

5.2 Independent Component Analysis

ICA assumes that observed data are linearly combination of a set of independent sources. The objective of ICA is to demix the observed data into the independent sources. Formally, ICA assumes:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{24}$$

where $\mathbf{x}$ is the observed data, $\mathbf{A}$ is the mixing matrix, and $\mathbf{s}$ is the independent signal. ICA finds a *demixing* matrix that separates the observed data into a set of statistically independent sources. Note that we can't determine the variance of the independent components since both $\mathbf{s}$ and $\mathbf{A}$ are unknown (a scalar multiplier of $\mathbf{s}$ could be cancelled in $\mathbf{A}$).

Nongaussianity is key. Since the observed data is a linear combination of independent sources, which are random variables in our formulation. By the Central Limit Theorem, we expect the observed data to tend towards Gaussian distributions. Therefore in separating the data we look to maximize the nongaussianity of the data. See `http:`

//www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf for a mathematical description.

Kurtosis is the classical measure of nongaussianity (zero for Gaussian), but is susceptible to outliers. In practice, use approximations of negative differential entropy (see link above) (TODO: fill in discussion on relationship to mutual information).

ICA is not commonly used for dimensionality reduction as it is difficult to rank independent components. Furthermore, unlike PCA, ICA does not have a criterion to prioritize the order that ICs are generated by ICA (later generated ICs are not necessarily less informative that earlier generated ICs). Skewness and kurtosis can be used to produce such score for each IC. When the significance of the score is the variance, the IC reduces to PCA.

5.2.1 ICA Software

- http://www.postech.ac.kr/~seungjin/courses/dr/readings/ica.pdf has a list of software in the appendix

5.3 Local Linear Embedding

LLE operates by attempting to reconstruct each data point from a linear combintion of only its k nearest neighbors, where $k \ll N$. The local reconstruction is characterized in terms of a local weight matrix, $\mathbf{W}$:

$$\mathbf{x}^i = \sum_j^N \mathbf{W}_{ij}\mathbf{x}^j, \tag{25}$$

such that $\mathbf{W}_{ij} = 0$ if the jth data point is not a KNN. LLE projects the data to a lower dimensional space by finding $d \ll D$ dimensional datapoints, $\mathbf{y}$ where (25) remains valid:

$$\mathbf{y}^i = \sum_j^N \mathbf{W}_{ij}\mathbf{y}^j. \tag{26}$$

The projection is an eigenvalue problem such that the d projected dimensions sorrespond to the eigenvectors of the matrix

$$\mathbf{M} = (\mathbf{I} - \mathbf{W})(\mathbf{I} - \mathbf{W})^\top$$

see Roweis and Saul 2000 for greater detail. This projection can map nonlinear relationships that PCA cannot; however, the central drawback of this approach is that it does not provide a basis for projection of future data points (the entire analysis must be redone from scratch). Vanderplauss and Connoley (2009) have a technique for alleviating this drawback by only sampling a subset of the training data.

6 Distributions

6.1 Dirichlet

6.1.1 Introduction

The Dirichlet distribution has application in modeling the variability of pmfs. For instance, a bag of dice can represent a bag of pmfs, each dice corresponding to its own pmf (since no dice has a perfectly uniform distribution). Application in modeling the distribution of words in a text document. Given N text documents with vocabulary V, where $|V| = K$, then each document can be represented by a pmf of length K produced by normalizing the empirical frequency of its words. Different Dirichlet distributions can be used to model documents by different authors or documents on different topics[1].

A pmf with k components lies on a $k-1$ dimensional *simplex* (a hyperdimensional generalization of a triangle) as the components must be non-negative and sum to 1, hence a simplex (it's in $k-1$ dimensions as the actual possible combination lie in one less dimension than the space). Each point on the simplex is really a pmf, where it defines a prob for each of the k components. The Dirichlet distribution can be viewed as the probability distribution over the $k-1$ dimensional simplex. That is, it's a distribution over $pmfs$ of length k.

Let $Q = [Q_1, Q_2, \ldots, Q_k]$ be a random pmf (components sum to 1 and are non-zero). Also, suppose $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ with $\alpha_i > 0$ and $\alpha_0 = \sum_i \alpha_i$. The Q has a Dirichlet distribution with parameter α, $Q \sim \text{Dir}(\alpha)$,

$$f(q, \alpha) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} q_i^{\alpha_i - 1} \tag{27}$$

The parameter α controls the probability distribution. If $\{\alpha_i\}_{i=1,\ldots,k} = c$ for some $c > 0$ then the density is symmetric (not constant) about the uniform pmf ($c = 1$ is a uniform, $c > 1$ is peaked in the interior, $c < 1$ has peaks on the edges, usually near the corners). Note that the support of the Dirichlet distribution is open: no component of a pmf drawn for a Dirichlet will ever be zero! When $k = 2$ the Dirichlet reduces to the Beta distribution.

The Dirichlet distribution serves as a conjugate prior to the multinomial distribution (generalizing the cases in which the Beta distribution serves as a conjugate prior for the probability parameter of the binomial distribution). That is if $(X|q)$ dist as $\text{Multinomial}_k(n, q)$ and Q dist as $Dir(\alpha)$ then $(Q|X = x)$ dist as $\text{Dir}(\alpha + x)$ (fairly simple proof, just use the definitions).

6.1.2 Generating Samples from Dirichlet

Say we know α and now wish to obtain realizations from the Dirichlet distribution.

Polya's Urn: put α_i balls of color i for $i = 1, 2, / \ldots, k$ in an urn (not necessarily integer numbers). At each iteration draw one ball randomly from the urn, then place

[1] `https://www.ee.washington.edu/techsite/papers/documents/UWEETR-2010-0006.pdf`

Table 1: Key Notation

$Q \sim \text{Dir}(\alpha)$	random pmf Q coming from a Dirichlet distribution with parameter α
q	pmf, which in this tutorial, will often be a realization of a random pmf $Q \sim \text{Dir}(\alpha)$
q_j	jth component of the pmf q
$q^{(i)}$	ith pmf of a set of L pmfs
k	number of events the pmf q is defined over, so $q = [q_1, q_2, \ldots, q_k]$
α	parameter of the Dirichlet distribution
α_0	$= \sum_{i=1}^{k} \alpha_i$
$m = \alpha/\alpha_0$	normalized parameter vector, mean of the Dirichlet
Δ_k	$(k-1)$-dimensional probability simplex living in $\mathbb{R}^k$
v_i	ith entry of the vector v
v_{-i}	the vector v with the i-th entry removed
$\Gamma(s)$	the gamma function evaluated at s, for $s > 0$
$\Gamma(k, \theta)$	Gamma distribution with parameters k and θ
$\stackrel{D}{=}$	AD=B means random variables A and B have the same distribution

Table 2: **Properties of the Dirichlet Distribution**

Density	$\frac{1}{B(\alpha)} \prod_{j=1}^{d} q_j^{\alpha_j - 1}$
Expectation	$\frac{\alpha_i}{\alpha_0}$
Covariance	For $i \neq j$, $Cov(Q_i, Q_j) = \frac{-\alpha_i \alpha_j}{\alpha_0^2(\alpha_0+1)}$. and for all i, $Cov(Q_i, Q_i) = \frac{\alpha_i(\alpha_0-\alpha_i)}{\alpha_0^2(\alpha_0+1)}$
Mode	$\frac{\alpha-1}{\alpha_0-k}$.
Marginal Distributions	$Q_i \sim \text{Beta}(\alpha_i, \alpha_0 - \alpha_i)$.
Conditional Distribution	$(Q_{-i} \mid Q_i) \sim (1 - Q_i)\,\text{Dir}(\alpha_{-i})$
Aggregation Property	$(Q_1, Q_2, \ldots, Q_i + Q_j, \ldots, Q_k) \sim \text{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_i + \alpha_j, \ldots, \alpha_k)$. In general, if $\{A_1, A_2, \ldots, A_r\}$ is a partition of $\{1, 2, \ldots, k\}$, then $(\sum_{i \in A_1} Q_i, \sum_{i \in A_2} Q_i, \ldots, \sum_{i \in A_r} Q_i) \sim \text{Dir}\left(\sum_{i \in A_1} \alpha_i, \sum_{i \in A_2} \alpha_i, \ldots, \sum_{i \in A_r} \alpha_i\right)$.

Figure 1: Summary of the Dirichlet distribution and notation.

it back into the urn along with a ball of the same color. Iterate more and more times the proportions of balls of each color will converge to a pmf that is a sample from the distribution $\text{Dir}(\alpha)$ (this proof is complicated—come back to it later).

Stick-breaking Iteratively breaking a stick of length 1 into k pieces in such a way that the lengths of the k pieces form a $\text{Dir}(\alpha)$ distribution. Assume we know how to generate random variables from Beta distribution (plug in the two parameters, sample from that distribution), so we address the problem of $k = 3$ and then generalize. The idea is to iteratively draw from a β-distribution with $u_j \sim \beta(\alpha_j, \sum_{i=j+1}^{k} \alpha_i)$ for $1 \leq j \leq k-1$ then assign $q_j = u_j \prod_{i=1}^{j-1}(1 - u_i)$ (where for u_1 we take $q_1 = u_1$. This is like breaking off pieces of a stick for each q_j and using the remaining stick to find the rest of the q_i's. (proof is okay, but it takes some time, will come back to proof eventually, if needed).

Gamma Transformation The most practical computational approach is to use a gamma transformation: draw $z_i \sim \Gamma(\alpha_i, 1)$ and assign $q_i = \frac{z_i}{\sum_j z_j}$. Through change of variables it can be shown that the density corresponding to $\text{Dir}(\alpha)$ distribution is provided via the aforementioned technique. For reference the gamma distribution is:

$$P(x, \kappa, \boldsymbol{\theta}) = x^{\kappa - 1} \frac{e^{-x/\boldsymbol{\theta}}}{\boldsymbol{\theta}^{\kappa} \Gamma(\kappa)}$$

See Section 12.3 for a discussion on the Dirichlet Process.

6.2 Gaussian

$$\mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^{\top} \Sigma_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right\} \tag{28}$$

7 Expectation-Maximization

The *expectation-maximization* (EM) algorithm is an elegant way to find maximum likelihood solutions for models with latent variables or missing values, so long as the missing

values are missing at random (i.e. whether or not the value is missing does not depend on its value).

7.1 General Description

Let $\mathbf{X}$ represent a matrix of the observed variables, $\mathbf{Z}$ represent a matrix of the latent variables, and let $\boldsymbol{\theta}$ represent the parameters of the model. Our goal is to maximize

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

where if $\mathbf{Z}$ is continuous we simply replace the sum by an integral.

We first assume that direct optimization of $p(\mathbf{X}|\boldsymbol{\theta})$ is difficult but that maximization of the complete data likelihood, $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$, is significantly easier. As will be discussed in the various implementations of the EM algorithm, this means we assume that the latent variables are known for maximization purposes.

Now we introduce a distribution $q(\mathbf{Z})$ over the latent variables and note that:

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q||p) \tag{29}$$

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \tag{30}$$

$$\mathrm{KL}(q||p) = -\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\} \tag{31}$$

This decomposition can be proved by plugging in the definitions and using the product rule, $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}) p(\mathbf{X}|\boldsymbol{\theta})$.

$\mathrm{KL}(q||p)$ is the Kullback-Leibler divergence between $q(\mathbf{Z})$ and the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$. The KL-divergence has the property that $KL(q||p) \geq 0$, where $KL(q||p) = 0 \Leftrightarrow q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})$. Therefore, $\mathcal{L}(q, \boldsymbol{\theta}) \leq \ln p(\mathbf{X}|\boldsymbol{\theta})$ ($\mathcal{L}(q, \boldsymbol{\theta})$ is a lower bound for $\ln p(\mathbf{X}|\boldsymbol{\theta})$). The EM algorithm maximizes the likelihood by iteratively optimizing different components of this decomposition.

In the E step, the lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ is maximized with respect to $q(\mathbf{Z})$ while holding $\boldsymbol{\theta}^{old}$, where $\boldsymbol{\theta}^{old}$ is the current value of the parameter vector. The solution of this maximization is to let $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old})$ which makes $\mathrm{KL}(q||p) = 0$. This is the maximization of $\mathcal{L}(q, \boldsymbol{\theta})$ with respect to $q(\mathbf{Z})$ because $\ln p(\mathbf{X}|\boldsymbol{\theta}^{old})$ does not depend on $q(\mathbf{Z})$. This means that changing $q(\mathbf{Z})$ will cause $\mathcal{L}(q, \boldsymbol{\theta}^{old})$ and $\mathrm{KL}(q||p)$ to balance each other (if $\mathcal{L}(q, \boldsymbol{\theta}^{old})$ decreases by 5 then $\mathrm{KL}(q||p)$ increases by 5). As a result, minimizing $\mathrm{KL}(q||p)$ maximizes $\mathcal{L}(q, \boldsymbol{\theta}^{old})$.

In the subsequent M step, $q(\mathbf{Z})$ is held fixed and the lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ is maximized with respect to $\boldsymbol{\theta}$ to give $\boldsymbol{\theta}^{new}$. This maximization increases $\mathcal{L}$, unless we are already at a maximum, which increases the log likelihood. Since we hold $q(\mathbf{Z})$ fixed, it will not equal the new posterior distribution ($p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{new})$ since $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old})$ and will therefore have a non-zero KL divergence, meaning the increases in the log-likelihood function will be greater than the increase in the lower bound. We then return to the E step, etc.

Note that after the E step

$$\mathcal{L}(q, \boldsymbol{\theta}) = \underbrace{\sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}_{Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old})} - \underbrace{\sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}) \ln p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old})}_{\text{constant: entropy of } q} \tag{32}$$

Thus the M Step is maximizing the expectation of the *complete-data log likelihood*. NOTE: we are optimizing with respect to $\boldsymbol{\theta}$, which only appears inside a logarithm. Therefore if the joint distribution $p(\mathbf{Z}, \mathbf{X}|\boldsymbol{\theta})$ is a member of the exponential family, or a product of exponential family members, then the logarithm will cancel the exponential and lead to a simpler M step then directly maximizing the incomplete-data log likelihood function, $p(\mathbf{X}|\boldsymbol{\theta})$. Such is the benefit of the EM algorithm.

The General EM Algorithm Summary:
Given a joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ over observed variable $\mathbf{X}$ and latent variables $\mathbf{Z}$, with parameters $\boldsymbol{\theta}$, the goal is to maximize the likelihood function $p(\mathbf{X}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

1. Initialize the parameters $\boldsymbol{\theta}^{old}$
2. **E Step** Construct $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old})$ (evaluate $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old})$)

   $$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$$

   which is the conditional *expectation* of the complete-data log-likelihood.
3. **M Step** Evaluate $\boldsymbol{\theta}^{new}$ via

   $$\boldsymbol{\theta}^{new} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) \tag{33}$$

4. Check log likelihood and parameter values for convergence, if not converged let $\boldsymbol{\theta}^{old} \leftarrow \boldsymbol{\theta}^{new}$ and return to step 2.

Intractable M-steps still exist even when we know the complete likelihood. To solve this problem we use Generalized Expectation Maximization (GEM) algorithm, which adjusts the parameters until the likelihood increases, not necessarily maximizes. (TODO: expand GEM discussion, EM with a prior over the parameters, and include the stochastic EM approach).

7.2 Gaussian Mixture EM

The goal with the Gaussian mixture model is to maximize the log likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{34}$$

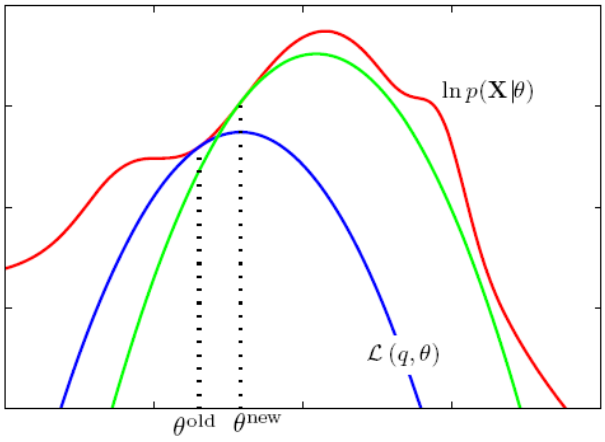

Figure 2: Parameter-space visualization of the EM algorithm, which involves alternately computing a lower bound on the log likelihood for current parameter values and then maximizing the bound to obtain new parameter values.

where the difficulty comes from the sum within the logarithm. On the other hand, if we use the complete data likelihood the maximization becomes much easier

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{n,k}} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{n,k}} \tag{35}$$

so that

$$\ln\, p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \left\{\ln(\pi_k) + \ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right\} \tag{36}$$

which places the logarithm inside the summation, making the maximization much easier.

Now, let's break down the EM algorithm using the summary provided in the previous section. First, we want to determine the expectation of the log likelihood of the complete data:

$$\begin{aligned}
\mathbb{E}_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})] &= \sum_{\mathbf{Z}} \ln(p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})) p(\mathbf{Z}|\boldsymbol{\mu}', \boldsymbol{\Sigma}', \boldsymbol{\pi}') \\
&= \sum_{\mathbf{Z}} \sum_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \left\{\ln(\pi_k) + \ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))\right\} p(\mathbf{Z}|\boldsymbol{\mu}', \boldsymbol{\Sigma}', \boldsymbol{\pi}') \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \left\{\ln(\pi_k) + \ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))\right\} \underbrace{\frac{\mathcal{N}(\mathbf{x}_n'|\boldsymbol{\mu}_k', \boldsymbol{\Sigma}_k')}{\sum_j \boldsymbol{\pi}_j' \mathcal{N}(\mathbf{x}_n'|\boldsymbol{\mu}_j', \boldsymbol{\Sigma}_j')}}_{\gamma(z_{n,k})} \underbrace{\sum_{\mathbf{Z}} z_{n,k}}_{1} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \left\{\ln(\pi_k) + \ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))\right\} \gamma(z_{n,k})
\end{aligned}$$

where $\gamma(z_{n,k})$ is the responsibility, and tilde's denote the "old" parameters. This expectation can be maximized with respect to $\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}$,and π_k while holding the old parameters in $\gamma(z_{n,k})$ constant:

$$\begin{aligned}
\frac{\partial \mathbb{E}_{\mathbf{Z}}}{\partial \mu_k} &= \frac{\partial}{\partial \mu_k}\left(\sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\ln(\pi_k) + \ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}))\right\}\gamma(z_{n,k})\right) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{\partial}{\partial \mu_k}\ln(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}))\gamma(z_{n,k}) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{\partial}{\partial \mu_k}(-\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k) + \text{const})\gamma(z_{n,k}) \\
&= -\frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\frac{\partial}{\partial \mu_k}(\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)\gamma(z_{n,k}) \\
&= -\sum_{n=1}^{N}\sum_{k=1}^{K}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \mathbf{x}_n)\gamma(z_{n,k}) \\
&= \sum_{n=1}^{N}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)\gamma(z_{n,k})
\end{aligned}$$

now setting the derivative to 0 yields

$$\sum_{n=1}^{N}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)\gamma(z_{n,k}) = 0$$

after some rearranging we have

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_{n=1}^{N}\mathbf{x}_n\gamma(z_{n,k})}{\sum_{n=1}^{N}\gamma(z_{n,k})} \tag{37}$$

similar results are obtained for the other parameters.

7.3 Further Reading

1. "Gentle" tutorial on EM for GMM and HMM: [`http://crow.ee.washington.edu/people/bulyko/papers/em.pdf`], different perspective on the EM

8 Generalized Linear Model

In a GLM, each output of a dependent variable, t, is assumed to be generated from a distribution in the exponential family. The expected value of the distribution depends on the independent variables, $\mathbf{x}$ via:

$$E[y] = h(\mathbf{w}^\top \mathbf{x}) \tag{38}$$

where $h(\cdot)$ is known as the *activation function* (the inverse is known as the *link function*). From wikipedia: the GLM consists of 3 components

1. A probability distribution from the exponential family
2. A linear predictor: $\mathbf{w}^\top \mathbf{x}$
3. An activation function h such that $E[y] = h(\mathbf{w}^\top \mathbf{x})$

Then depending on the chosen probability distribution, we can formulate a canonical activation function that naturally describes the mean of the output in terms of the linear predictor. For a Gaussian, the canonical activation function is the identity (in this case we have a general linear model); for a Poisson the canonical activation function is $\exp(\cdot)$. TODO: Fill this section out with deeper relationships with the probability distribution choice: `http://data.princeton.edu/wws509/notes/a2.pdf`.

9 Kernel Methods

A *kernel function* is given by the relationship:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}'), \tag{39}$$

which is an inner-product in feature space. This allows us to build extensions of common algorithms using the **kernel trick**, where the idea is that if we have an algorithm formulated such the $\mathbf{x}$ only enters as a scalar product, then we can reformulate the problem by replacing the scalar product with kernels.

- *linear kernel* (identity): $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- *radial basis function* (homogeneous kernels): $k(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||$

9.1 Dual Representation

Basic idea is to reformulate a given algorithm in terms of kernels (TODO: expand me).

10 Model Selection

10.1 Bias-Variance Decomposition

Formulate a conditional distribution $p(t|\mathbf{x})$ (e.g. via least squares, regularization, or a fully Bayesian approach) which can be combined with a squared loss function for the purpose of making predictions. Then the optimal prediction (minimizes expected loss) is given by the conditional expectation (via calculus of variations to choose the functional):

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x}) \mathrm{d}t \tag{40}$$

Can write the expected squared loss in the form:

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x}) \mathrm{d}x + \int \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t \tag{41}$$

Where t is the given target and $\mathbf{x}$ is the input vector. The second component arises from the noise on the data, and is the minimum achievale value of the expected loss (note this component does not depend on $y(\mathbf{x})$. So in turn, we look to minimize the first component through an optimal choice of $y(\mathbf{x})$.

For any data set D, we can run the learning algorithm and obtain $y(\mathbf{x}; D)$. Different data sets will give different prediction functions and consequently different values for the squared loss. The performance of the learning algorithm is assessed by examining the average performance over the data sets. So we take the expectation of $\{y(\mathbf{x}; D) - h(\mathbf{x})\}^2$ over all D. Adding and subtracting $\mathbb{E}_D[y(\mathbf{x}; D)]$ and performing a bit of algebra yields:

$$\mathbb{E}_D\left[\{y(\mathbf{x}; D) - h(\mathbf{x})\}^2\right] = \underbrace{\{\mathbb{E}_D[y(\mathbf{x}; D)] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_D\left[\{y(\mathbf{x}; D) - \mathbb{E}_D[y(\mathbf{x}; D)]\}^2\right]}_{\text{variance}} \tag{42}$$

This equation is looking to minimize the expected squared difference of the prediction from the real distribution and is composed of two terms. The first term is known as the *squared bias*, the second is the *variance*. We can then integrate over all $\mathbf{x}$ for both the bias and variance terms to obtain the total $(\text{bias})^2$ and variance.

For instance, a high bias system would be a low-degree polynomial that fits the actual data rather poorly.

11 Neural Networks

11.1 Multilayer Perceptrons

The multilayer perceptron (MLP)[2] is a feed-forward neural network that consists of a sequence of "neurons," non-linear activation functions, where the output of each activation function is used as part of a weighted linear superposition for the input of the next activation function. Here, I examine a two-layer[3] MLP, in which the input features are combined through two sequential superpositions of activation functions [Fig. 3]. Each dataset consists of a training and testing sample with two features, and in the case of the training data, a binary classification for each instance.

Given some observed data, $\mathbf{x} = (x_1, x_2, \ldots, x_D)$, we form a two-layer MLP by first constructing M linear combinations of the D input features,

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{43}$$

[2]MLP is a misnomer: the model comprises multiple layers of *logistic regression models*.

[3]I am using the convention provided in PRML that a typical input-hidden-output MLP is two layers (as opposed to three, which is also common).

where (following the notation of PRML 5.1), $j = 1, \ldots, M$ and the superscript indicates the layer of the MLP. The parameter $w_{ji}^{(1)}$ is the weight term, where the bias parameter w_{j0} is absorbed into the summation by taking $x_0 = 1$. The values a_j are referred to as *activations* and are transformed using some differentiable, nonlinear activation function, $h(\cdot)$, yielding

$$z_j = h(a_j). \tag{44}$$

The activation function, $h(\cdot)$, is typically chosen to be a logistic sigmoid, arctangent, or some other sigmoidal-like function. The output from the activation function is referred to as a *hidden unit* and is linearly combined with the output from the activation functions of the otherto form the *output unit activations*

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \tag{45}$$

where z_0 is again set to 1 to embed a bias term. In a two-layer MLP, the output units are transformed using an activation function to return the network outputs, y_k. For instance, in the case of classification the output activation function could be the logistic sigmoid function so that

$$y_k = \sigma(a_k) = \frac{1}{1 + e^{-a_k}}, \tag{46}$$

and in the case of regression, the activation function could be the identity such that $y_k = a_k$. This two-layer MLP can be extended to arbitrarily many layers by increasing the number of hidden-unit layers. As discussed in PRML 5.1, a two-layer network with linear outputs "can approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has sufficiently large number of hidden units." Therefore, in practice it is rare to require an extension beyond a two-layer MLP. The problem of learning an MLP boils down to finding the optimum link-weights and number of hidden nodes—that is, excluding "outside" problems such as feature selection. The number of hidden nodes is a dataset-dependent problem that typically requires testing a range of possible values. The weights of the neural network can be found using an iterative algorithm known as *backpropagation*.

11.2 Back Propagation

The backpropagation algorithm is a two-phase technique for obtaining the gradient with respect to the link-weights, $\nabla E(\mathbf{w})$, for a feed-forward neural network (though the term "backpropagation" is overloaded). With the gradient it is possible to use i.e. batch or stochastic gradient descent to iteratively optimize the weights of the neural net. On a higher level, error backpropagation operates in two phases:

- Forward Phase: The weights of the MLP are fixed, and the input data is propagated through the layers of the MLP from the input to the hidden to the output layer. This phase completes with the computation of an error measure.

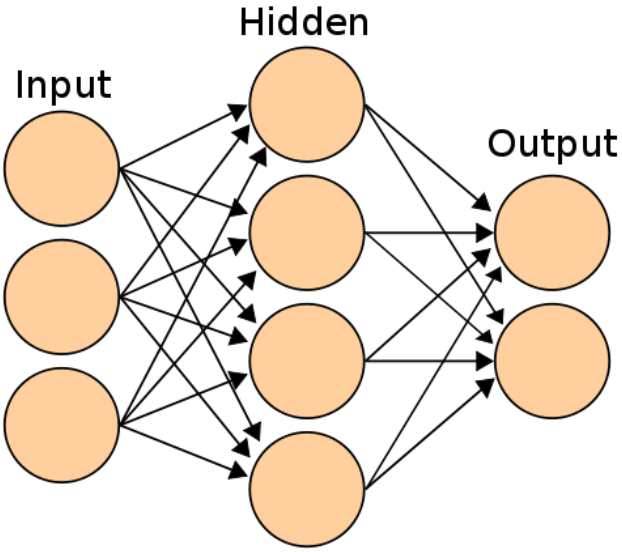

Figure 3: An example two-layer multilayer perceptron with three input nodes, four hidden nodes, and two output nodes; image from wikipedia `http://en.wikipedia.org/wiki/Artificial_neural_network`.

- Backward Phase: The error is propagated through the MLP in the backward direction (output to hidden to input). During this phase, adjustments are applied to the link-weights of the network so as to minimize the error.

In this work, we minimize the sum-of-squares error function,

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} ||y(\mathbf{x_n}, \mathbf{w}) - t_n||^2, \tag{47}$$

using batch gradient descent,

$$E(\mathbf{w}^{\tau+1}) = E(\mathbf{w}^{\tau}) - \eta \nabla E(\mathbf{w}^{\tau}), \tag{48}$$

with learning rate η. To determine the gradient we must find both

$$\frac{\partial E_n}{\partial w_{ji}} \text{ and } \frac{\partial E_n}{\partial w_{kj}} \tag{49}$$

for each of the N input datum, $\mathbf{x}_n$. Here, i represents the input layer, j represents the hidden layer, and k represents the output layer, so w_{ji} is the link-weight from the ith input node to the jth hidden node, while w_{kj} is the link-weight from the jth hidden node to the kth output node.

We use the chain rule to determine these derivatives:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}. \tag{50}$$

From (43),

$$\frac{\partial a_j}{\partial w_{ji}} = z_i, \tag{51}$$

and as shown in 5.2 PRML,

$$\frac{\partial E_n}{\partial a_j} = h^{'}(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k}. \tag{52}$$

Therefore in order to determine $\frac{\partial E_n}{\partial w_{ji}}$ we must first find $\frac{\partial E_n}{\partial w_{kj}}$, hence the name *back*propagation. The final piece to backpropagation is determining $\frac{\partial E_n}{\partial a_k}$. This depends on the output activation function. The sigmoid, our choice of $h(\cdot)$, has the convienient property that

$$\sigma^{'} = \sigma(1 - \sigma),$$

allowing us to easily determine $\frac{\partial E_n}{\partial a_k}$ from (47):

$$\frac{\partial E_n}{\partial a_k} = y_k(y_k - t_k)(1 - y_k). \tag{53}$$

Summarizing both the forward and backward propagation we have the following algorithm for training our MLP (summary customized from 5.32 PRML):

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{54}$$

$$z_j = \tanh(a_j) \tag{55}$$

$$a_k = \sum_{j=0}^{M} wkj^{(2)} z_j \tag{56}$$

$$y_k = \sigma(a_k) \tag{57}$$

in the forward direction and

$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k(y_k - t_k)(1 - y_k) \tag{58}$$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k \tag{59}$$

$$\tag{60}$$

in the reverse direction, allowing us to determine:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \qquad\qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} \delta_k z_j. \tag{61}$$

In batch gradient descent we determine the "total" derivatives via:

$$\frac{\partial E}{\partial w_{ji}^{(1)}} = \sum_n \frac{\partial E_n}{\partial w_{ji}^{(1)}} \qquad\qquad \frac{\partial E}{\partial w_{kj}^{(2)}} = \sum_n \frac{\partial E_n}{\partial w_{kj}^{(2)}}. \tag{62}$$

These equations apply for multiclass classification; however, it's advisable to use the softmax classification and multiclass cross-entropy error function.

11.2.1 Rules of Thumb

- Number of training equations divided by number of training weights should be greater than two (`http://www.groupsrv.com/computers/about642184.html`)?

12 Nonparametric Bayesian Methods

Basically, nonparametric methods do not assume a certain underlying distribution of the data. A histogram is a classic example of a nonparametric method. Kernel density estimator and KNN are other examples. In this section we examine nonparametric Bayesian methods.

12.1 Model Selection

12.2 Gaussian Processes

12.3 Dirichlet Processes

The Direichlet process is a random distribution whose realization are distributions over an arbitrary (possibly infinite) sample space. Given example: suppose we ask people their favorite color, possibly infinite number of colors, and perhaps their response depends on their mood, so we have a Dirichlet distribution over the possible pmfs they could choose that would describe their mood. This Dirichlet process enables us to work with an infinite set of events (colors).

Clearly the set of all probability distributions over an infinite sample space is unmanageable. One solution, that relies on the ability to aggregate components of a Dirichlet distribution and still have a Dirichlet distribution, is to partition the sample space into M subsets of events $\{B_1, B_2, \ldots, B_M\}$ (categorize inputs, such as a color, into certain groups) from the original sample space χ. Then we would have

$$\alpha(\beta_a) = \int_{\chi} \mathbb{1}_{B_i}(x) d\alpha(x)$$

where $\mathbb{1}_{B_i}$ is the indicator function of B_i.

Formal mathematical description of Dirichlet Process: Let χ be the entire space and $\mathscr{B}$ represent a σ-field on χ. The couple $(\chi, \mathscr{B})$ is a measurable space. Let α_0 be a finite non-zero measure (just a measure that is countably additive, not necessarily a probability measure [that sums to one over χ]). Denote all probability distributions on $(\chi, \mathscr{B})$ by $\mathscr{P}$. P is a Dirichlet process with parameter α on $(\chi, \mathscr{B})$ if for any finite measurable partition $\{B_i\}_{i=1}^k$ of χ, the random vector $(P(B_1), \ldots, P(B_k))$ has a Dirichlet distribution with params $(\alpha_0(B_1), \ldots, \alpha_0(B_k))$ (it's a *finite measurable partition* if the B_i's are mutually exclusive and exhaustive). As a result of this restriction, D_α only has support for discrete (atomic) distributions with infitine atoms (continuous distributions like Gaussians have 0

probability). More formally:

$$G \sim \mathrm{DP}(\alpha_0, G_0) \tag{63}$$

where G is a Dirichlet process, G_0 is the base probability measure, α_0 is the positive scaling parameter. G is a random probability measure that has the same support as G_0. If $G \sim \mathrm{DP}(\alpha_0, G_0)$, then with probability one:

$$G = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}$$

where ϕ_k are the independent random variables that are distributed according to G_0 (for instance, they could be distributed according to a Gaussian, δ_{ϕ_k} is an atomx at ϕ_k, β_k is the stick breaking weight which is determined through α_0 (see below for a more thorough description). G is well suited for placing a prior on a mixture component. The basic idea is to associate a mixture component with each atom, then introduce indicator variables to associate each datum with a mixture component.

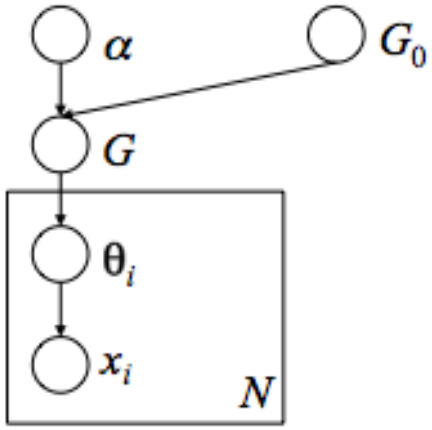

Figure 4: Dirichlet Process Graphical Model

12.3.1 Sampling from DP

Measures from a Dirichlet Process are discrete with probability one. We can sample from the DP with a few different techniques:

Stick-breaking: The stick-breaking construction is based on independent sequences of iid random variables: $\{\pi_k^{'}\}_{k=1,\ldots,\infty}$ and $\{\phi_k\}_{k=1,\ldots,\infty}$ where:

$$\pi_k^{'} | \alpha_0, G_0 \sim \beta(1, \alpha_0)$$

$$\phi_k | \alpha_0, G_0 \sim G_0$$

Then we can define a random measure G as:

$$\pi_k = \pi_k^{'} \prod_{l=1}^{k-1} (1 - \pi_l^{'})$$

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k}$$

where δ_ϕ is a probability meaasure concetrated at ϕ (which can be viewed as the data?).

12.3.2 Applications of DP

DP Mixture Models: The DP is often used as a prior in nonparametric mixture models. The job of the prior is to allow for an infinite number of mixtures, while favoring a small number of groups. This prior can be formulated through the *Chinese Restaurant Process*, whereby the idea is that a Chinese restaurant with an infinite number of tables has customers (data) coming in and choosing tables (clusters). The first customer chooses table 1. The second customer chooses table 1 with probability $\frac{1}{1+\alpha}$ and a new table with $\frac{\alpha}{\alpha+1}$. The n^{th} customer chooses a previously occupied table k with probability $\frac{m_k}{n-1+\alpha}$ and a new table with $\frac{\alpha}{n-1+\alpha}$, where m_k is the number of people at table k. Note that $p(c_1, c_2, \ldots, c_N)$ is exchangable, it doesn't matter the orders in which the customers arrive. With a likelihood function that we can apply to each of the clusters picked out by the CRP we can integrate over the parameters and class assignments (TODO: make this more detailed).

13 Optimization Techniques

13.1 Gradient Descent

For a given function and learning rate:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta_n \nabla F(\mathbf{x}_n) \tag{64}$$

13.2 Stochastic Gradient Descent

Instead of taking the gradient of the entire function, were approximating it by iteratively taking the gradient of individual an individual example. Essentially slowly changes the optimized value. Useful for online applications, where we don't perform batch processing (or when we want to break the data into chunks).
Shown with application to updating weight vectors, $\mathbf{w}$, let the error function (or generally, the optimization function) comprise a sum over data points $E = \sum_n E_n$. After obtaining E_n, we update $\mathbf{w}$ via:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \eta \nabla E_n \tag{65}$$

where η is the learning rate parameter. Using this with sum-of-squares error function is known as *least-mean-squares (LMS) algorithm.*

14 Preprocessing

14.1 Data Whitening

DW is often used in applications such as ICA. DW consists of transforming a given vector $\mathbf{x}$ to a new vector $\tilde{\mathbf{x}}$, such that the components are uncorrelated and their variances equal 1. In other words,

$$\mathbb{E}[\tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top] = \mathbb{I} \quad \text{covariance is identity}$$

Whitening is always possible. A common technique is to use the eigen-value decomposition of the covariance matrix:

$$\mathbb{E}[\tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top] = \mathbf{E}\mathbf{D}^{-\mathbf{1/2}}\mathbf{E} \tag{66}$$

where $\mathbf{E}$ is the orthogonal matrix of eigenvectors of the covariance matrix and $\mathbf{D}$ is the diagonal matrix of eigenvalues. The negative square root is simply the component with negative square root. Whitening is accomplished via:

$$\tilde{\mathbf{x}} = \mathbf{E}\mathbf{D}^{-\mathbf{1/2}}\mathbf{E}\mathbf{x}$$

whitening makes sure the mixing matrix is diagonal in ICA.[4]

15 Regression

TODO

16 Regularization

16.1 weight decay

$$\frac{\lambda}{2}\sum_j |w_j|^q \tag{67}$$

Cases: $q = 1$ is *lasso*, $q = 2$ is *ridge*, see page 145 Bishop (2006) for contours.

17 Sampling Methods

17.1 Sequential Monte Carlo Methods [Particle Filters]

18 Appendix: Calculus

18.1 Matrix Derivatives

$$\frac{\partial}{\partial \mathbf{x}}\left(\mathbf{x}^\top \mathbf{a}\right) = \frac{\partial}{\partial \mathbf{x}}\left(\mathbf{a}^\top \mathbf{x}\right) = \mathbf{a} \tag{68}$$

[4] http://cis.legacy.ics.tkk.fi/aapo/papers/IJCNN99_tutorialweb/node26.html

18.2 Calculus of variations

Goal is to maximize or minimize a *functional*: which maps a set of functions to a real value. CoV can be used to show shortest path between two points is a straight line, Gaussian maximizes entropy, etc.

Euler-Lagrange Equations:

$$\frac{\partial G}{\partial y} - \frac{\partial}{\partial x}\left(\frac{\partial G}{\partial y'}\right) = 0 \tag{69}$$

18.3 Lagrange Multipliers

19 Appendix: Definitions

activation function: Transform a linear function of **w** using a nonlinear function where the output lies in the range (0,1), e.g. sigmoid or tanh.

linking function: Inverse of the activation function.

20 Appendix: Distances

20.1 Mahalanobis Distance

Like a Euclidean distance that takes into account correlations (covariances) of the data set and it is scale invariant. Intuitively, the Mahalanobis distance is the distance of a test point from the centroid of a given dataset normalized by the width of the ellipsoid in the direction of the test point (unlike $\frac{x-\mu}{\sigma}$ which assumes a spherical distribution).

$$D_M(x) = \sqrt{(x-\mu)^\top S^{-1}(x-\mu)} \tag{70}$$

where S is the covariance matrix.

$$D_M(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^\top S^{-1}(\vec{x}-\vec{y})} \tag{71}$$

20.2 Statistical Tests to Compare Distributions

20.2.1 Kolmogorov-Smirnoff Test

Nonparametric test to determine if two datasets differ significantly. Essentially, this test finds the maximum difference between the two CDFs of the distributions. (frequentist approach)

21 Appendix: Information Theory

21.1 Entropy

Entropy, often referred to as *Shannon Entropy*, quantifies the expected value of the information contained in a specific realization of a random variable. Another perspective, is that it measures the average minimum number of bits needed to encode a string of symbols based on the frequency of the symbols:

$$H(X) = -\sum_{i=0}^{N-1} p_i \log_2 p_i \tag{72}$$

22 Appendix: Pragmatic Tips

- Normalize the features of the test a training data using the data mean and standard deviation from the training data (subtract mean and divide by standard deviation) (or scale to [-1,1]: when to use which?)
- Can generate more training data using the original input by e.g. adding white noise, transforming the data slightly, etc.

References

Christopher Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.