Math 156 Note

October 24, 2017

Eric's office hours: Thursday MS 3965 11AM OH: thursday 2-3PM (Just this week) dataset: $\{(x_1, t_1), (x_2, t_2), ..., (x_n, t_n)\}$ with polynomial model:

$$y(x, w) = \sum_{m=0}^{M} w_m x^m$$

probabilitis model: let

$$\begin{aligned} D &= \{(x_1, t_1), (x_2, t_2), ..., (x_n, t_n)\} \\ X &= (x_1, x_2, x_3, ..., x_n) \\ W &= (w_0, w_1, w_2, w_3, ..., w_n) \end{aligned}$$

$P(D|w)$: the probability that the dataset was generated by $y(X, W)$. If there is no noise, then $P(D|w) > 0$ only if $y(x_n, w) = t_n$ for every n. Assume that there was noise in the measurements. $y(x_n, w) + U_n = t_n$ where U_n is a random variable representing noise. (in this class, Gaussian) Suppose that $U_n \sim N(\mu, \sigma)$, and that we havefor a gaussian with expected value μ and variance σ^2

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$$

Assume that U_n is sampled from the gaussian with 0 as expected value . then: $y(x_n, w) - t_n = U_n$ shows how far the cost function deviates from the standard normal.

$$p((x_n, t_n)|W) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\{-\frac{(y(x_n, w) - t_n)^2}{2\sigma^2}\}$$

Assume the noise at each point is independent.then we obtain the expression:

$$\begin{aligned} P(D|W) &= \prod_{i=1}^{N} P((x_i, t_i)|W) \\ &= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} exp\{-(y(x_n, w) - t_n)^2/2\sigma^2\} \end{aligned}$$

$$\begin{aligned} argmax_W P(D|w) &= argmax_W log(P(D|W)) \\ &= argmax_W (\sum_{n=1}^{N} \frac{1}{2} log(\frac{1}{2\pi\sigma^2})) - (y(x_n, w) - t_n)^2/2\sigma^2) \\ &= argmax_W \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 \end{aligned}$$

Avoiding overfitting with prior assumptions. Introduce **prior probability** $P(w)$ as the probability of parameter choice before we get the actual data. In polynomial model, the size of $y'(x, w)$ depends on the absolute values of w's

$$P(W) = \frac{1}{C(\sigma, M)} exp\{-(w_1^2 + ... + w_M^2)/2\sigma^2\}$$

Instead of finding $argmax_W P(D|W)$ we are trying to find $argmax_W P(D|W)p(W)$, or $P(D, W)$. or $P(w|D)p(D)$ by Bayes theorem. Or equivalently:

$$Posterior = \frac{Likelihood \times Prior}{dataset}$$
$$P(W|D) = \frac{P(D|W)P(W)}{P(D)}$$

1 Modeling data distributions

grayscale image of 32 x 32, each pixel is $\in \mathbb{R}$, so the whole image is in $\mathbb{R}^{1024}$.
Two datasets, one all cats and the other one all dogs. Build two models, one from cat called P_{cat} and the other P_{dog}. Choose which one is cat or dog based on: if $P_{cat} > P_{dog}$, then cat. If $P_{dog} > P_{cat}$ then dog. How do we build the probability distributions?
start with a Gaussian model in m dimensions:

$$N(\mu, \Sigma) = \frac{exp\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\}}{(2\pi)^{-m/2}|\Sigma|^{1/2}}$$

Σ is the covariance matrix of dimension m x m.
$\sum_{ij}$ gives how x_i and x_j vary around the mean μ.
$|\Sigma| = det(\Sigma)$ and $cov(x, y) = E[(x - E(x))(y - E(y))]$
Also, covariance matrix is also positive definite since $\Sigma = AA^\top$. and all of its eigenvalues > 0.

$$exp\{-(x-\mu)^\top A^\top A(x-\mu)\} = exp\{-||A(x-\mu)||^2\}$$

The problem is to find

$$argmax_{\mu,\Sigma} P(D|\mu, \Sigma)$$

where $D = \{x_1, x_2, x_3, ..., x_N\}$, each independent. so that, we need to maximize the likelihood function:

$$P(D|\mu, \Sigma) = \prod_{n=1}^{N} P(x_n|\mu, \Sigma)$$

take logarithm to maximize, we end up need to minimize the function:

$$\sum_{n=1}^{N} \frac{1}{2}(x_n - \mu)^\top \Sigma^{-1}(x-\mu) + log(-|\Sigma|^{-1/2})$$

take gradient in respect to μ, then

$$\begin{aligned}
0 &= \sum_{n=1}^{N} \frac{-1}{2}\Sigma^{-1}(x_n-\mu) - \frac{1}{2}(x_n-\mu)^\top\Sigma^{-1} \\
0 &= -\sum_{n=1}^{N}\Sigma^{-1}(x_n-\mu) \\
0 &= -\sum_{n=1}^{N}(x_n-\mu) \\
\mu_{ML} &= \frac{1}{N}\sum_{n=1}^{N}x_n
\end{aligned}$$

To see how we arrive at this conclusion: let $\Sigma^{-1} = B^\top B$ then

$$\begin{aligned}
&(x-\mu)^\top\Sigma^{-1}(x-\mu) = (x-\mu)^\top B^\top B(x_n-\mu) = ||B(x-\mu)||^2 \\
&\nabla_\mu ||B(x-\mu)||^2 = -2B^\top B(x-\mu) = -2\Sigma^{-1}(x-\mu)
\end{aligned}$$

again, to see how the last step is derived, we have (suppose all gradient are in respect to μ), with preliminary that $(a, b) = a^\top b$:

$$\begin{aligned}
\nabla||B(x-\mu)||^2 &= \nabla(B(x-\mu), B(x-\mu)) \\
&= (\nabla(B(x-\mu)), B(x-\mu)) + (B(x-\mu), \nabla(B(x-\mu))) \\
&= (B(\nabla(x-\mu)), B(x-\mu)) + (B(x-\mu), B(\nabla(x-\mu)) \\
&= (-B, B(x-\mu)) + (B(x-\mu), -B) \\
&= -B^\top B(x-\mu) + (x-\mu)^\top B^\top(-B) \\
&= -2B^\top B(x-\mu) = -2\Sigma^{-1}(x-\mu)
\end{aligned}$$

Interpretation of the modeling: suppose that the dataset has two unknown parameters: $(\mu_{true}, \Sigma_{true})$, and that after the modeling from a Gaussian noise linear regression, we obtained sample mean:

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}x_n \tag{1}$$

and maximum likelihood covariance:

$$S = \frac{1}{N-1}\sum_{n=1}^{N}(x_n-\mu_{ML})(x_n-\mu_{ML})^\top = \frac{1}{N-1}\Sigma_{true} \tag{2}$$

sample mean will be the same as the true sample mean. However, covariance will not the same as the sample mean.

2 PCA and Dimensionality Reduction

let $D = \{x_1, ..., x_N\} \in \mathbb{R}^d$, where d is a very large number. we need to project the data into lower space. defiine:

$$\begin{aligned}
&(a, b) \text{ as inner product of vector a and b.}\\
&\sum_{i=1}^{N}(x, u_i)u_i = U^\top U x\\
&P : \mathbb{R}^d \to \mathbb{R}^m, \text{ where } m < d\\
&rank(P) = m, Im(P) = V, dim(V) = m\\
&V^\perp = \{w \in \mathbb{R}^d : (w, v) = 0 \text{ for } v \in V\}
\end{aligned}$$

A decomposition of vector space $\mathbb{R}^d$ into V and $V^\perp$ means that $\forall x \in \mathbb{R}^d$, we have $x = v_1 + v_2$, where $v_1 \in V$ and $v_2 \in V^\perp$ and the decomposition is unique.

$$\begin{aligned}
&P(x) = P(v_1 + v_2) = v_1\\
&P^2(x) = P(P(x)) = P(P(v_1 + v_2)) = p(v_1) = v_1 \Longrightarrow P(x) = P^2(x)
\end{aligned}$$

if $\{y_1, ..., y_m\}$ is an orthonormal basis for V, then

$$P(x) = \sum_{j=1}^{m}(x, y_j)y_j$$

Goal: given a dimension $m < d$ find a projection P which minimizes the error $\sum_{n=1}^{N} ||x_n - Px_n||^2$. The problem is thus formulated as optimization problem:

$$\underset{P}{\text{argmin}} \sum_{n=1}^{N} ||(I - P)(x_n - \bar{x})||^2$$

Assuming a matrix A has n column matrices, then define **Frobenius norm** as the matrix norm:

$$||A||_F^2 = tr(AA^\top) = \sum_{ij} a_{ij}^2 = \sum_{i=1}^{N} ||a_i||^2$$

then we have

$$\begin{aligned}
\sum_{n=1}^{N} ||(I - P)(x_n - \bar{x})^2||_F &= \sum_{n=1}^{N} tr[(I - P)(x_n - \bar{x})(x_n - \bar{x})^\top (I - P)^\top]\\
&= tr\{(I - P)[\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})](I - P)^\top\}
\end{aligned}$$

since optimization does not care about constance terms, combining the formula with equation (2) and we have that

$$\underset{P}{\text{argmin}} \text{ tr}\{(I - P)[\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})](I - P)^\top\} = \underset{P}{\text{argmin}} \text{ tr}[(I - P)S(I - P)^\top]$$

where P is a projection and dim(Im(P)) = m, and S is a positive definite matrix. so that $S = UDU^\top$, with D a diagonal matrix and U an orthogonal matrix. let $Q = (I - P)D$, rewrite the problem as:

$$\underset{P}{\text{argmin}}\ \text{tr}[(I-P)UDU^\top(I-P)^\top] = \underset{Q}{\text{argmin}}\ \text{tr}(QDQ^\top)$$

; where we have $\dim(\text{Im}(Q)) = d - m$. expand the trace and we have:

$$\begin{aligned}\text{tr}(QDQ^\top) &= \sum_{i=1}^{d}(QDQ^\top)_{i,i} = \sum_{i=1}^{d}\sum_{j=1}^{d}(QD)_{ij}Q^\top_{ji} \\ &= \sum_{i=1}^{d}\sum_{j=1}^{d}Q_{i,j}D_{j,j}Q^\top_{j,1} = \sum_{i=1}^{d}D_{j,j}\sum_{j=1}^{d}D_{j,j}\sum_{i=1}^{d}Q_{i,j}Q^\top_{j,i} \\ &= \sum_{j=1}^{d}D_{j,j}\sum_{i=1}^{d}Q^2_{i,j}\end{aligned}$$

since we need to minimize this term against Q, and $rank(Q) = d - m$. essentially we need to find the first m largest diagonal elements of D and set them to zero, and set the remaining column vectors of D to basis vector e_i.

$$Q = (I - P)U \implies P = I - QU^\top$$

P keeps the m eigenvectors of U corresponding to the largest eigenvalues of D.Or equivalently, we pick the first M largest eigenvectors of $S = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^\top$.

Probabilistic interpretation and Latent variable model

PCA has certain limitations, including:
(1) Non-parametric: no probabilistic model for observed data.
(2) Covariance matrix computation is complex and expensive.
(3) Does not deal properly with missing data.
(4) Outlying data observation can unduly affect the analysis.
(5) Has to be zero centered linear model.

Probabilistic PCA addresses limitations of PCA, and uses PCA as a general.
Gaussian density model.

Latent Variable Model: x: latent variables (unobserved ones) offer a lower dimensional representation of the data (y) and their dependencies:

$$\begin{aligned}&y = Wx + \mu + \epsilon,\ \text{where } p(x) = N(0, I_m), \epsilon = N(-, \sigma^2 I_d)c \\ &p(y|z) = N(y|Wx + \mu, \sigma^2 I_d)\end{aligned}$$

Limitations of PCA
some datasets are essentially not suitable for PCA, an example is presented in homework2. For example, if we have a one-dimensional dataset described by function:

$$c(t) = (\cos(t), \sin(t), t), \text{ so that } c : \mathbb{R} \to \mathbb{R}^3$$

this describes curve in 3-D space, but a projection onto any subspace will lose alot of data. In fact, for a general case of $c(t) = (t, t^2, ..., t^d)$, PCA is not plausible.
If we have nonlinear model. using a single Gaussian is bad too. An example is a 2-D parabola that has points scattered on the curve. Its mean is on the center of the curve, where there is no point. A Gaussian model, however, says that the most likely occurence would be on the mean, which is not the case.
To deal with this problem, a solution is to use **a mixture of Gaussians**: partition the data points and group some nearest points as in one distribution, and some others in another distribution. The resulting formula for the probability distribution of a single point x would be:

$$p(x) = \sum_{k=1}^{K} \pi_k N(x, u_k, \Sigma_k)$$

where π_k is the probability of choosing a particular Gaussian, and the Gaussian term after is the probability of x coming from this particular Gaussian. The property of pdf says the integral of above equation gives 1, which means:

$$\sum_{k=1}^{K} \pi_k \int N(x, \mu_k, \Sigma_k)dx = \sum_{k=1}^{K} \pi_k = 1$$

which makes sense since π_k represents partition of the whole dataset as a single distribution. We can again use maximum likelihood to compute the probability distribution and maximize it using an iterative algorithm. It will be alot of computations, and choice of partition number K matters: a K too large results in overfitting, whereas too small result in underfitting the dataset.
Another solution is to use the **kernel method**. We need a smooth function that passes through the datapoints, and to do so, introduce the kernel function with Gaussian:

$$k(x) = \frac{1}{(2\pi h^2)^{D/2}} exp(-||x||^2/2h^2)$$

where h is the bandwidth parameter that controlls how wide each kernel function is for certain groups of points. For an outliar, the bandwidth should be very large, whereas for neighbor (K closest neighbors), bandwidth should be small.

3 Classification Methods

1. K-nearest Neighbor

Give datasets $D_1 = \{x_1, x_2, ..., x_k\}$ and $D_2 = \{x_{k+1}, ..., x_N\}$. Build probability distributionP_1 from D_1 and P_2 from D_2. The probability distribution could be Gaussian, mixture Gaussian, or built from kernel function. K-nearest neighbor computes the k nearest neighbors of x in both D_1 and D_2. If more neighbors of x is in D_1 assign x to D_1, else assign x to D_2.

Brute force implementation: in short, calculate the pairwise norm $||x - x_n||$, and then sort the resport. It is computationally complex: for M points, we have a complexity of $O(dNM + MN \log N)$
Some better ways to implement is to use **k-d trees** and **ball trees**.

2. Basic Idea of Linear Classification

For points in $\mathbb{R}^d$, define a hyperplane $\{v : w^\top v + w_0 = 0\}$. If for any $v \in \mathbb{R}^d$, we have $w^\top v + w_0 > 0$, it is in class 1. If $w^\top v + w_0 < 0$, it is in class 0. Sometimes, the parameters w, w_0 are chosen such that points are mapped to the set $\{1, -1\}$, but that could be troublesome, for points scattered around the hyperpland and far away from the hyperplane: If we have a plane orthogonal to the hyperplane and passes points from these two categories, the far points are more likely to be 1 and close points are more likely to not be 1.
2-class Linear Classifier
In the case where we try to classifiy the dataset D into $D_1 = \{x_1, x_2, ..., x_k\}$ and $D_2 = \{x_{k+1}, ..., x_N\}$, we try to use a classifier $w^\top x + w_0$ that maps all elements in D_1 into 1 and all elements of D_2 into -1. The choice of function would be $\tanh(x)$, which maps data into the range of [-1,1], with the split point at $x = 0$. Let $\tilde{w} = (w, w_0) \in \mathbb{R}^{d+1}$ and $\tilde{x_n} = (x_n, 1) \in \mathbb{R}^{d+1}$ The cost function in this case is

$$J(\tilde{w}) = \sum_{n=1}^{N} ||\tanh(\tilde{w}^\top \tilde{x}) - t_n||^2$$

Multi-class classifier
To classify D into subsets $D_1, D_2, ..., D_r$, we need a transformation $W : \mathbb{R}^d \longrightarrow \mathbb{R}^r$, $w_0 \in \mathbb{R}^r$. Essentially, we choose x based on largest coordinate after the transformatio. Item in the partition D_i should be transformed to e_i, the ith unit basis vector $[0, ..., 1, ..., 0]^\top$ for the subspace $\mathbb{R}^r$. The definition of the mapping is in the form

$$\sigma_i(y) = \frac{e^{y_i}}{\sum\limits_{j=1}^{r} e^{y_i}}$$

For all the points in a partition set D_i. If e^{x_i} is large, then $\sigma(x_i) \longrightarrow 1$. Otherwise, some other terms dominates the denominator term and it goes to zero. This results in e_i for every partition set. The cost function in this case is:

$$J(\tilde{w}) = \sum_{n=1}^{N} ||\sigma(\tilde{w}^\top \tilde{x}) - t_n||^2$$

3. Smooth Optimization

Most cases, setting gradient to zero and solve is very hard analytically. Finding the global minimum of a non-convex function is NP-hard, and mostly iterative methods, such as gradient descent is used.

Gradient Descent has the formulation:

$$x^{(n+1)} = x^{(n)} - \tau \nabla f(x^{(n)})$$

where τ is the step size or learning rate. In most cases, we choose 1 over the **Lipschitz Constant** of ∇f as the step size. The constant is defined as:

$$L = max_x ||\nabla^2 f(x)|| = max_{(x,y)} \frac{||\nabla f(x) - \nabla f(y)||}{||x-y||}$$

It can be interpreted as the maximum eigenvalue of the Hessian matrix, or the largest descent direction of the gradient. The reason to choose it is based on the inequality:

$$f(x) \leq f(y) + \langle \nabla f(y), x-y \rangle + \frac{L}{2}||x-y||^2$$

The resulting gradient descent formulation becomes

$$x^{(n+1)} = x^{(n)} - \frac{1}{L} \nabla f(x^{(n)})$$

The Lipschitz constant is usually found analytically. Back in the case of multi-class classification, we can minimize the cost function by gradient descent, in the form

$$W^{(n+1)} = W^{(n)} - \frac{1}{L} \nabla_w J(w^{(n)})$$

In the 2-class example, we have

$$\begin{aligned} J(w) &= \sum_{i=1}^{N} (\tanh(wx_n) - t_n)^2 \\ \nabla_w \tanh(wx_n) &= \tanh'(wx_n) x_n \\ \nabla_w^2 \tanh(wx_n) &= \tanh''(wx_n) x_n x_n^\top \end{aligned}$$

For convex function, gradient descent converges to global min at rate of $O(\frac{1}{N})$ and N is the number of iterations.
Newton's method makes use of the second order Taylor expansion of a function to approximate it:

$$f(x) \approx f(y) + \langle \nabla f(y), x-y \rangle + \frac{1}{2}(x-y)^\top \nabla^2 f(y)(x-y)$$

in our case, we have

$$f(x) \approx f(x^{(k)}) + (x - x^{(k)})^T g^{(k)} + \frac{1}{2}(x - x^{(k)})^T F(x^{(k)})(x - x^{(k)})$$

where $g^{(k)} = \nabla f(x^{(k)})$ and $F(x^{(k)}) = \nabla^2 f(x^{(n)})$. If x is a local minimizer, by interior case of first order necessary condition, we must have the gradient of above equation equal to zero.

$$g^{(k)} + F(x^{(k)})(x - x^{(k)}) = 0$$

by second order sufficient condition, if $F(x^{(k)}) > 0$, the taylor expansion function reaches its minimum. Therefore, the iterative algorithm for Newton's method is given as follows:

$$x^{(k+1)} = x^{(k)} - F(x^{(k)})^{-1}g^{(k)}$$

When Newton's method converges, it has a much better convergence rate than gradient descent, but it is much harder to compute the Hessian, when the input has large number of dimensions.

4. Nonlinear classifier

IN the case where it is hard to simply using a linear classifier, a method to consider is to use change of coordinate. i.e. change from Cartesian to polar coordinate. Call the transformation as ϕ, then the cost function can be rewritten as

$$J(w) = \sum_{n=1}^{N} ||\sigma(\tilde{w}\tilde{\phi}(x_n)) - t_n||^2$$

where we choose $\phi(x)$ as a linear combination of simpler nonlinear function $\phi_j(x)$. The key is then to pick this nonlinear function $\phi_j(x)$. In **Neural Nets**, the transformation is composite of linear functions and the same nonlinear function

$$\phi = \tilde{\phi} \circ A_1 \circ \tilde{\phi} \circ A_2 \circ ... \circ \tilde{\phi} \circ A_m$$

where A_i are linear transformations, and $\tilde{\phi}_i$ is a nonlinear function. In this case, we have many different parameters to consider, and the cost function is:

$$J(w, A_1, ..., A_m) = \sum_{n=1}^{N} ||\sigma(w\phi(x_n)) - t_n||^2$$

where ϕ depends on $A_1, A_2, ..., A_m$. The nonlinear function is often chosen as: $max_x(0, x)$.

5. Probabilistic Interpretation

For a two class scenario $D = \{D_1, D_2\}$, construct probability distribution p_1 from D_1 and p_2 from D_2. Decision rule set such that for $x \in D$, if $p_1(x) > p_2(x)$, pick x in class 1.
Objective: maximize posterior probability $p(D_1|x)$. Known assumption and tools:

$$p(D_1|x) = \frac{1}{1 + \frac{p(x|D_2)p(D_2)}{p(x|D_1)p(D_1)}}$$
$$a = \log(\frac{p(x|D_2)p(D_2)}{p(x|D_1)p(D_1)})$$
$$p(x|D_1) = N(\mu_1, \Sigma), p(x|D_2) = N(\mu_2, \Sigma)$$

Then we have that

$$a = (\mu_1 - \mu_2)^\top \Sigma^{-1} x - (\mu_1 - \mu_2)^\top \Sigma^{-1}(\frac{\mu_1 + \mu_2}{2}) + \log(\frac{p(D_2)}{p(D_1)}) = W^\top x + W_0$$

Maximize the posterior probability of points assigned to correct class label, let $t_n \in \{0,1\}$ and take log.

$$\prod_{n=1}^{N} p(D_1|x_n)^{t_n} p(D_2|x_n)^{1-t_n}$$
$$\implies \sum_{n=1}^{N} t_n \log(p(D_1|x_n)) + (1-t_n)\log(p(D_2|x_n))$$
$$\implies \nabla E(w) = -\sum_{n=1}^{N} \nabla_w a(w)(t_n - \sigma(a))$$

Where $\sigma(x)$ is the sigmoid function. The global minimum can be found since it is convex function and gradient descent can be used.

4 Book reading note

1.maximization of variance on the component
PCA as defined as orthogonal projection of the data onto a lower dimensional linear space, konwn as *principal subspace* that maximizes the variance of the projected data.
Or it is defined as the linear projection that minimizes the average projection cost, defined as the mean squared distance between the data points and their projections.
1-D scenario: project to one dimensional space, with normal vector u_1, each data is then projected onto a scalar $u_1^\top x_n$, and the mean of projection is $u_1^\top \bar{x}$, where $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ is the sample mean of the data set. Variance of the dataset is given by: $u_1^\top S u_1$, where $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^\top$ is the covariance matrix of the dataset. Maximization of it gives us

$$Su_1 = \lambda u_1$$

which says that u_1 must be an eigenvector of S. Given $u_1^\top u_1 = 1$, we have that

$$u_1^\top S u_1 = \lambda_1$$

so that u_1 must equal to the largest eigenvalue.
in general, PCA involves evaluating mean $\bar{x}$ and covariance matrix S of the dataset and then finding the M eigenvectors of S correponding to the M largest eigenvalues.

2.minimization of projection error
let $\{u_1, u_2, ..., u_D\}$ by the D-dimensional basis vector for the vector space of the data, such that

$$x_n = \sum_{i=1}^{D}(x_n^\top u_i)u_i$$

is unique. To project to dataset into a lower dimension M, we can approximate each point by

$$\tilde{x_n} = \sum_{i=1}^{M} z_{ni}u_i + \sum_{i=M+1}^{D} b_i u_i$$

and the task is to minimize

$$J = \frac{1}{N}\sum_{n=1}^{N} ||x_n - \tilde{x_n}||^2$$

and this gives us that $z_{nj} = x_n^\top u_j$ and $b_j = \bar{x}^\top u_j$ This in the end gives us

$$J = \sum_{i=M+1}^{D} u_i^\top S u_i$$

miniimzation of this function also gives us

$$Su_i = \lambda_i u_i$$

which is the same result as in part 1.