My Notes on Stanford University Online Course Machine Learning (CS229) by Andrew Ng

Chenglong Chen[*]

May 10, 2013

1 Notes on: Lecture notes 1

1.1 Page 8: $\mathbf{tr}AB = \mathbf{tr}BA$

Proof. Without loss of generality, we assume $A = (A_{ij})_{m\times n}$ and $B = (B_{ij})_{n\times m}$. Thus, the diagonal elements of AB is $\sum_{k=1}^{n} A_{ik}B_{ki}$ $(i = 1, 2, \dots, m)$, and those of BA is $\sum_{i=1}^{m} B_{ki}A_{ik}$ $(k = 1, 2, \dots, n)$. Finally, we have

$$\mathrm{tr}AB = \sum_{i=1}^{m}\left(\sum_{k=1}^{n} A_{ik}B_{ki}\right) = \sum_{k=1}^{n}\left(\sum_{i=1}^{m} B_{ki}A_{ik}\right) = \mathrm{tr}BA.$$

□

1.2 Page 9: $\nabla_A \mathbf{tr}AB = B^T$

Proof. Let $A = (A_{ij})_{m\times n}$ and $B = (B_{ij})_{n\times m}$. Thus $AB = \left(\sum_{k=1}^{n} A_{ik}B_{kj}\right)_{m\times m}$ and $\mathrm{tr}AB = \sum_{s=1}^{m}\sum_{k=1}^{n} A_{sk}B_{ks}$. Furthermore, we have

$$\frac{\partial \mathrm{tr}AB}{\partial A_{ij}} = B_{ji}$$

Therefore,

$$\frac{\partial \mathrm{tr}AB}{\partial A} = \left(\frac{\partial \mathrm{tr}AB}{\partial A_{ij}}\right)_{m\times n} = (B_{ji})_{m\times n} = B^T.$$

□

1.3 Page 9: $\nabla_{A^T} f(A) = (\nabla_A f(A))^T$

Proof. Let $A = (A_{ij})_{m\times n}$. According to the definition of $\nabla_A f(A)$, we have

$$\nabla_{A^T} f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{21}} & \cdots & \frac{\partial f(A)}{\partial A_{m1}} \\ \frac{\partial f(A)}{\partial A_{12}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{1n}} & \frac{\partial f(A)}{\partial A_{2n}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix} = (\nabla_A f(A))^T.$$

□

[*]e-mail: c.chenglong@gmail.com.

1.4 Page 9: $\nabla_A \mathbf{tr} ABA^TC = CAB + C^TAB^T$

Proof. To derive this property, we need some facts of matrix differential calculus. As shown in Table 2 on Page 199 of the book [3], if

$$\mathrm{d}\phi(X) = \mathrm{tr}A^T\mathrm{d}X = (\mathrm{vec}\,A)^T\mathrm{d}\,\mathrm{vec}\,X,$$

then the Jacobian matrix is

$$\mathrm{D}\phi(X) = (\mathrm{vec}\,A)^T.$$

where $\phi(X)$ is a scalar function of m-by-n matrix X. Furthermore, as indicated in the Exercise 2 on Page 197 that

$$\mathrm{D}\phi(X) = \left(\mathrm{vec}\,\frac{\partial\phi(X)}{\partial X}\right)^T,$$

we have the derivative of $\phi(X)$ as

$$\frac{\partial\phi(X)}{\partial X} = A.$$

Furthermore, we also need some fundamental rules of differential calculus (Page 167-168 of [3]):

$$\mathrm{d}A = 0,$$

$$\begin{aligned}\mathrm{d}(U \pm V) &= \mathrm{d}U \pm \mathrm{d}V,\\ \mathrm{d}(UV) &= (\mathrm{d}U)V + U\mathrm{d}V,\\ \mathrm{d}U^T &= (\mathrm{d}U)^T,\\ \mathrm{d}\,\mathrm{tr}U &= \mathrm{tr}\,\mathrm{d}U.\end{aligned}$$

where U and V are matrix functions, and A is a matrix of real constants.

Now to compute $\nabla_A\mathrm{tr}ABA^TC$, we first obtain[1]

$$\begin{aligned}\mathrm{d}(\mathrm{tr}ABA^TC) &= \mathrm{tr}\,\mathrm{d}(ABA^TC)\\ &= \mathrm{tr}\big((\mathrm{d}AB)A^TC + AB\mathrm{d}(A^TC)\big)\\ &= \mathrm{tr}\big((\mathrm{d}A)BA^TC + AB(\mathrm{d}A^T)C\big)\\ &= \mathrm{tr}\big((\mathrm{d}A)BA^TC + AB(\mathrm{d}A)^TC\big)\\ &= \mathrm{tr}(\mathrm{d}A)BA^TC + \mathrm{tr}AB(\mathrm{d}A)^TC\\ &= \mathrm{tr}(\mathrm{d}A)BA^TC + \mathrm{tr}\big(AB(\mathrm{d}A)^TC\big)^T\\ &= \mathrm{tr}(\mathrm{d}A)BA^TC + \mathrm{tr}C^T(\mathrm{d}A)B^TA^T\\ &= \mathrm{tr}BA^TC(\mathrm{d}A) + \mathrm{tr}B^TA^TC^T(\mathrm{d}A)\\ &= \mathrm{tr}(BA^TC + B^TA^TC^T)\mathrm{d}A.\end{aligned}$$

Finally, we have

$$\nabla_A\mathrm{tr}ABA^TC = \frac{\partial\mathrm{tr}ABA^TC}{\partial A} = (BA^TC + B^TA^TC^T)^T = CAB + C^TAB^T.$$

With the property illustrated here, the proof of $\nabla_A\mathrm{tr}AB = B^T$ can be much simpler:

$$\mathrm{d}\,\mathrm{tr}AB = \mathrm{tr}(\mathrm{d}AB) = \mathrm{tr}(\mathrm{d}A)B = \mathrm{tr}B\mathrm{d}A$$

and

$$\nabla_A\mathrm{tr}AB = \frac{\partial\mathrm{tr}AB}{\partial A} = B^T$$

□

[1] Also see `http://en.wikipedia.org/wiki/Matrix_calculus`.

1.5 Page 11: Another method to derive $\nabla_\theta J(\theta)$

Proof. According to Page 10 of the lecture note, we have

$$J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}).$$

We first compute the differential of $J(\theta)$, i.e.,

$$\begin{aligned}
\mathrm{d}J(\theta) &= \frac{1}{2}\mathrm{d}\left((X\theta - \vec{y})^T(X\theta - \vec{y})\right) \\
&= \frac{1}{2}\left[\left(\mathrm{d}(X\theta - \vec{y})^T\right)(X\theta - \vec{y}) + (X\theta - \vec{y})^T\left(\mathrm{d}(X\theta - \vec{y})\right)\right] \\
&= \frac{1}{2}\left[(X\mathrm{d}\theta)^T(X\theta - \vec{y}) + (X\theta - \vec{y})^T X\mathrm{d}\theta\right] \\
&= \frac{1}{2}\left[\mathrm{tr}(X\mathrm{d}\theta)^T(X\theta - \vec{y}) + \mathrm{tr}(X\theta - \vec{y})^T X\mathrm{d}\theta\right] \\
&= \frac{1}{2}\left[\mathrm{tr}\left((X\mathrm{d}\theta)^T(X\theta - \vec{y})\right)^T + \mathrm{tr}(X\theta - \vec{y})^T X\mathrm{d}\theta\right] \\
&= \mathrm{tr}(X\theta - \vec{y})^T X\mathrm{d}\theta.
\end{aligned}$$

Finally, we have

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = \left((X\theta - \vec{y})^T X\right)^T = X^T(X\theta - \vec{y}).$$

Note that this is the same as that given in the lecture note. □

1.6 Page 15: Locally weighted linear regression and Non-parametric methods

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

For an exercise of locally weighted linear regression, pls refer to prob2 in Problem Set #1 (Autumn 2012 version). For an exercise of locally weighted logistic regression, pls refer to prob2 in Problem Set #1 (Old version).

For more information about non-parametric method, pls check wikipedia page `http://en.wikipedia.org/wiki/Non-parametric_statistics`

1.7 Regularized Linear Regression (From Ng's ML course of Coursera)

Proof. In Ng's ML course of Coursera, the cost function of regularized linear regression is given by

$$\begin{aligned}
J(\theta) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2,
\end{aligned}$$

where $\frac{\lambda}{2}\sum_{j=1}^{n}\theta_j^2$ is the regularization term and λ is the regularization parameter. Note that in the regularization term, the θ_0 ($\theta_0 = 1$) is not penalized by convention. This equation can be formulated as a more compact one, i.e.,

$$J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) + \frac{\lambda}{2}\theta^T D\theta,$$

where D is expressed as

$$D = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

To derive $\nabla_\theta J(\theta)$, we only need to take care of the regularization term. Since

$$\begin{aligned} \mathrm{d}\,\frac{\lambda}{2}\theta^T D\theta &= \frac{\lambda}{2}\big((\mathrm{d}\theta^T)D\theta + \theta^T \mathrm{d}(D\theta)\big) \\ &= \frac{\lambda}{2}\big((\mathrm{d}\theta)^T D\theta + \theta^T D\mathrm{d}\theta\big) \\ &= \frac{\lambda}{2}\big(\mathrm{tr}((\mathrm{d}\theta)^T D\theta)^T + \mathrm{tr}\theta^T D\mathrm{d}\theta\big) \\ &= \lambda\mathrm{tr}\theta^T D\mathrm{d}\theta, \end{aligned}$$

we have

$$\nabla_\theta \frac{\lambda}{2}\theta^T D\theta = (\lambda\theta^T D)^T = \lambda D\theta,$$

and

$$\nabla_\theta J(\theta) = X^T(X\theta - \vec{y}) + \lambda D\theta.$$

Now that the normal equation can be obtained by setting $\nabla_\theta J(\theta)$ to zero and solving for θ, which yields

$$\theta = (X^T X + \lambda D)^{-1}X^T\vec{y}.$$

Note that this is the same as that given in Ng's ML course of Coursera. □

Note: Indeed, θ_0 can also be added to the regularization term, which will becomes

$$\frac{\lambda}{2}||\theta||_2^2 = \frac{\lambda}{2}\theta^T\theta$$

In that case we have $D = I$. This model is called *ridge regression.* Adding the regularization term $\frac{\lambda}{2}\theta^T\theta$ can help to avoid $X^T X$ being non invertible. That is to say, $X^T X + \lambda I$ is (always) invertible provided $\lambda > 0$ (See Note 1.8 below).

1.8 The invertibility of $A^T A$

Proof. We assume that $A \in \mathbb{R}^{m\times n}$. If $Ax = 0$, then $A^T Ax = 0$. Conversely, if $A^T Ax = 0$, then we have

$$0 = x^T A^T Ax = (Ax)^T Ax = ||Ax||_2^2.$$

Therefore, $Ax = 0$. This implies that equations $Ax = 0$ and $A^T Ax = 0$ share the same solutions. Hence, the numbers of the solution in the basic are the same, i.e.,

$$n - \mathrm{rank}A = n - \mathrm{rank}(A^T A),$$

or

$$\mathrm{rank}A = \mathrm{rank}(A^T A).$$

Similarly, we also have

$$\mathrm{rank}(AA^T) = \mathrm{rank}A^T = \mathrm{rank}A.$$

If A^TA is invertible, $\text{rank}(A^TA) = n$ since A^TA is a $n \times n$ matrix. Thus $\text{rank}A = n$, which implies that the column of A is linearly independent (so that $m \geq n$, because "any $n+1$ vectors of n-D are and must be linearly dependent").

Otherwise, if $m < n$, then $\text{rank}A \leq m < n$, which implies that A^TA is not invertible. However, as we will show below, adding a small diagonal matrix, say λI, will make the matrix $A^TA + \lambda I$ invertible. □

1.9 The invertibility of $A^TA + \lambda I$

Proof. We assume that $A \in \mathbb{R}^{m \times n}$. Decompose A by SVD:

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T = U \Sigma V^T.$$

Then we have

$$\begin{aligned} A^TA &= V\Sigma U^T U \Sigma V^T \\ &= V\Sigma\Sigma V^T \\ &= VDV^T \\ &= \sum_{i=1}^{n} \sigma_i^2 v_i v_i^T \end{aligned}$$

Note that $\sigma_i^2 \geq 0$ is eigenvalue of A^TA, indicating that A^TA is positive semi-definite. (It can also be shown by $x^TA^TAx = (Ax)^TAx = ||Ax||_2^2 \geq 0$ for any given vector $x \in \mathbb{R}^n$.)

After adding a small diagonal matrix, say λI, we have

$$\begin{aligned} A^TA + \lambda I &= VDV^T + \lambda VV^T \\ &= V(D + \lambda I)V^T \\ &= \sum_{i=1}^{n} (\sigma_i^2 + \lambda) v_i v_i^T. \end{aligned}$$

Now, since $\forall i$, $\sigma_i^2 + \lambda > 0$, so $A^TA + \lambda I$ is positive definite. This is so because "A matrix being positive definite" is equivalent to "All its eigenvalues are positive".[2]

Hence, $A^TA + \lambda I$ is invertible. This is so because "Every positive definite matrix is invertible".[3] This could be understood in another way around. Recall "The determinant $\det(A)$ of a matrix A is non-zero *if and only if* A is invertible" and "The determinant of a matrix is equal to the product of its eigenvalues".[4] Now since every eigenvalue $\sigma_i^2 + \lambda > 0$, we have $\det(A^TA + \lambda I) \neq 0$ and thus $A^TA + \lambda I$ is invertible. □

2 Notes on: Lecture notes 2

2.1 Page 1: Why does logistic regression try to find a straight line?

Proof. Recalling the hypothesis $h_\theta(x)$, i.e.,

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} = p(y = 1 \mid x; \theta),$$

a natural choice for the decision would be $p(y = 1 \mid x; \theta) = 0.5$, i.e.,

$$0.5 = \frac{1}{1 + e^{-\theta^T x}}.$$

[2] Also see `http://en.wikipedia.org/wiki/Positive-definite_matrix`

[3] Also see `http://en.wikipedia.org/wiki/Positive-definite_matrix`

[4] Also see `http://en.wikipedia.org/wiki/Determinant`

The above equation turns out to be

$$\theta^T x = 0,$$

which itself defines a straight line. □

2.2 Page 3: The two definitions of covariance

Proof. Since $\mathrm{Cov}(Z) = E[(Z - E[Z])(Z - E[Z])^T]$, we have

$$\begin{aligned}
\mathrm{Cov}(Z) &= E[(Z - E[Z])(Z - E[Z])^T \\
&= E[ZZ^T - Z(E[Z])^T - E[Z]Z^T + (E[Z])(E[Z])^T] \\
&= E[ZZ^T] - (E[Z])(E[Z])^T - (E[Z])(E[Z])^T + (E[Z])(E[Z])^T \\
&= E[ZZ^T] - (E[Z])(E[Z])^T.
\end{aligned}$$

□

2.3 Page 6: Derivation of the MLE of the parameters of GDA

See my solution to problem 4(b) and (c) in the problem set #1, Autumn 2012.

2.4 Page 6: Derivation of the decision boundary of GDA

The decision boundary of GDA satisfies $p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma) = 0.5$. As shown in my solution to problem 4(b) and (c) in the problem set #1, Autumn 2012, $p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma)$ can be expressed as a logistic function, i.e.,

$$p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-\theta^T x)},$$

where a constant intercept term $x_0 = 1$ is added to x and θ is

$$\begin{bmatrix} \frac{1}{2}(\mu_0^T \Sigma^{-1}\mu_0 - \mu_1^T \Sigma^{-1}\mu_1) - \log\frac{1-\phi}{\phi} \\ \Sigma^{-1}(\mu_1 - \mu_0) \end{bmatrix}.$$

Note that $\log(\cdot)$ stands for the natural logarithm.

2.5 Page 9: Independent and conditionally independent

If two event A and B are independent, we have

$$p(A|B) = p(A).$$

For example, let the event A be 'I have a new car'; event B be 'The weather is fine.' In this specific, the two events are (generally and reasonably) not related. That is to say, the probability of occurrence of the event A has nothing to do with the occurrence of the event B.

In conditional independence two events A and B which are initially dependent become independent given the occurrence of a third event C. Formally, we have

$$p(A|B, C) = p(A|C).$$

For example, let the event A be 'The email contains 'buy''; event B be 'The email contains 'price''; event C be 'The email is a spam email.' Obviously, regardless of event C, event A and B most likely happen together: if event A happens, it is most likely event B also happens; thus they are dependent. However, if one was told that event C occurs, events A and B become independent (at least to some extent): since it is a spam email, even without the knowledge of the word 'price,' one may still believe that it is most likely that the word 'buy' will occur.

More related details can be found in the paper [2].

2.6 Page 10: Derivation of the MLE of the parameters of Naive Bayes

See my solution to problem 4 in the problem set #1, Public Course (i.e., old version).

2.7 Page 14: Derivation of the MLE of the parameters of multinomial event model

We first write the joint distribution of (x, y) as follow

$$\begin{aligned}
p(x,y) &= p(x|y)p(y) \\
&= \left(\prod_{j=1}^{n} p(x_j|y)\right) p(y) \\
&= \left[\prod_{j=1}^{n}\left(\prod_{s=1}^{N-1}(\phi_{s|y})^{I\{x_j=s\}}\right)\cdot(\phi_{N|y})^{I\{x_j=N\}}\right](\phi_y)^y(1-\phi_y)^{1-y}
\end{aligned}$$

where $N = |V|$. We note that similar with the case of multinomial distribution, there is $N-1$ parameter in $\phi_{s|y=l}$'s ($l \in \{0,1\}$), since $\phi_{N|y=l}$ can be expressed as $\phi_{N|y=l} = 1 - \sum_{s=1}^{N-1}\phi_{s|y=l}$.

Based on $p(x, y)$, we now write the log-likelihood of the data:

$$\begin{aligned}
&\ell(\phi_y, \phi_{k|y=0}, \phi_{k|y=1}) \\
&= \log\left(\prod_{i=1}^{m} p(x^{(i)}, y^{(i)})\right) \\
&= \sum_{i=1}^{m}\log\left(p(x^{(i)}, y^{(i)})\right) \\
&= \sum_{i=1}^{m}\left[\sum_{j=1}^{n_i}\left(\sum_{s=1}^{N-1} I\{x_j^{(i)}=s\}\log(\phi_{s|y^{(i)}}) + I\{x_j^{(i)}=N\}\log(\phi_{N|y^{(i)}})\right) + y^{(i)}\log(\phi_y) + (1-y^{(i)})\log(1-\phi_y)\right]
\end{aligned}$$

For ϕ_y, we have

$$\begin{aligned}
\frac{\partial \ell}{\partial \phi_y} &= \sum_{i=1}^{m}\left(y^{(i)}\frac{1}{\phi_y} - (1-y^{(i)})\frac{1}{1-\phi_y}\right) \\
&= \sum_{i=1}^{m}\frac{y^{(i)}-\phi_y}{\phi_y(1-\phi_y)}.
\end{aligned}$$

Setting $\frac{\partial \ell}{\partial \phi_y} = 0$ and solving for ϕ_y, we get the MLE of ϕ_y, i.e.,

$$\hat{\phi}_y = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} = \frac{1}{m}\sum_{i=1}^{m} I\{y^{(i)} = 1\}.$$

For $\phi_{k|y=l}$, ($k \in \{1, \ldots, N-1\}$ and $l \in \{0, 1\}$) we have

$$\begin{aligned}
\frac{\partial \ell}{\partial \phi_{k|y=l}} &= \sum_{i=1}^{m}\sum_{j=1}^{n_i}\left(I\{x_j^{(i)}=k\}\frac{\partial \log(\phi_{k|y^{(i)}})}{\partial \phi_{k|y=l}} + I\{x_j^{(i)}=N\}\frac{\partial \log(\phi_{N|y^{(i)}})}{\partial \phi_{k|y=l}}\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n_i}\left(\frac{I\{x_j^{(i)}=k\}}{\phi_{k|y^{(i)}}}\frac{\partial \phi_{k|y^{(i)}}}{\partial \phi_{k|y=l}} + \frac{I\{x_j^{(i)}=N\}}{\phi_{N|y^{(i)}}}\frac{\partial \phi_{N|y^{(i)}}}{\partial \phi_{k|y=l}}\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n_i}\left(\frac{I\{x_j^{(i)}=k\}}{\phi_{k|y=l}} I\{y^{(i)}=l\} + \frac{I\{x_j^{(i)}=N\}}{\phi_{N|y=l}}\left(-I\{y^{(i)}=l\}\right)\right),
\end{aligned}$$

where the last step is obtained using the fact that

$$\frac{\partial \phi_{N|y^{(i)}}}{\partial \phi_{k|y=l}} = \frac{\partial(1-\sum_{s=1}^{N-1}\phi_{s|y^{(i)}})}{\partial \phi_{k|y=l}} = -I\{y^{(i)}=l\}.$$

Setting $\frac{\partial \ell}{\partial \phi_{k|y=l}} = 0$ and solving for $\phi_{k|y=l}$, we get the MLE of $\phi_{k|y=l}$, i.e.,

$$\hat{\phi}_{k|y=l} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=k\}I\{y^{(i)}=l\}}{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=N\}I\{y^{(i)}=l\}} \cdot \phi_{N|y=l}.$$

Using the fact that for a given l, $\sum_{k=1}^{N}\hat{\phi}_{k|y=l}=1$, we finally obtain

$$\begin{aligned}
\phi_{N|y=l} &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=N\}I\{y^{(i)}=l\}}{\sum_{k=1}^{N}\left(\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=k\}I\{y^{(i)}=l\}\right)} \\
&= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=N\}I\{y^{(i)}=l\}}{\sum_{i=1}^{m} I\{y^{(i)}=l\}\left(\sum_{j=1}^{n_i}\sum_{k=1}^{N} I\{x_j^{(i)}=k\}\right)} \\
&= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=N\}I\{y^{(i)}=l\}}{\sum_{i=1}^{m} I\{y^{(i)}=l\}\left(\sum_{j=1}^{n_i} 1\right)} \\
&= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=N\}I\{y^{(i)}=l\}}{\sum_{i=1}^{m} I\{y^{(i)}=l\}n_i}.
\end{aligned}$$

Substituting $\phi_{N|y=l}$ back into $\hat{\phi}_{k|y=l}$, we finally get

$$\begin{aligned}
\hat{\phi}_{k|y=l} &= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=k\}I\{y^{(i)}=l\}}{\sum_{i=1}^{m} I\{y^{(i)}=l\}n_i} \\
&= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=k \wedge y^{(i)}=l\}}{\sum_{i=1}^{m} I\{y^{(i)}=l\}n_i}.
\end{aligned}$$

As shown, the above equation also holds for $k=N$. After applying Laplace smoothing, we obtain

$$\hat{\phi}_{k|y=l} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n_i} I\{x_j^{(i)}=k \wedge y^{(i)}=l\}+1}{\sum_{i=1}^{m} I\{y^{(i)}=l\}n_i+|V|},$$

which is the same as that given in the lecture note.

3 Notes on: Lecture notes 3

3.1 Page 4: w is orthogonal (at $90°$) to the separating hyperplane

Proof. For any two different points x_1 and x_2 on the decision boundary $w^Tx+b=0$, we have

$$w^Tx_1+b=0,$$
$$w^Tx_2+b=0.$$

Thus, we have

$$w^Tx_1=w^Tx_2,$$

i.e.,

$$w^T(x_1-x_2)=0,$$

implying that w is orthogonal to vector x_1-x_2, which is aligned with the decision boundary. □

3.2 Page 13: The optimal value for the intercept term b

Method 1 (see also Page 330 of [1])
Let $S = \{i : \alpha_i^* > 0, i \in \{1, \ldots, m\}\}$ denote the index set of *support vectors*, where α^* is the solution to the SVM dual optimization problem. Recalling the KKT dual complementary condition, i.e.,

$$\alpha_i^* g_i(w^*) = 0,\ i = 1, \ldots, m,$$

we have

$$g_i(w^*) = -y^{(i)}(w^{*T}x^{(i)} + b) + 1 = 0,$$

or

$$y^{(i)}(w^{*T}x^{(i)} + b) = 1$$

for the support vector ($i \in S$) since $\alpha_i^* > 0$. The above expression implies that the support vectors have functional margin exactly equal to 1.

Multiplying both sides by $y^{(i)}$ (which is $y^{(i)} \in \{-1, 1\}$, so that $(y^{(i)})^2 = 1$) then we have for $i \in S$

$$b^* = y^{(i)} - w^{*T}x^{(i)}.$$

As shown, b^* can be obtained by any of the training examples satisfying the dual complementary condition, or any of the support vectors.
Note: In practice, given the optimization problem is solved to *some degree of accuracy* (the obtained solution is approximate to the theoretical solution), it is standard to set b^* to the average such value computed over all the support vectors:

$$b^* = \frac{1}{N_S} \sum_{i \in S} \left(y^{(i)} - w^{*T}x^{(i)} \right).$$

where N_S is the total number of support vectors in S.
Method 2
Recall the primal optimization problem

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1,\ i = 1, \ldots, m. \end{aligned}$$

For optimal solution α^* (and also w^*), the constraint can be expressed as

$$\min \left(y^{(i)}(w^{*T}x^{(i)} + b) \right) = 1,$$

i.e., the functional margin of the training examples is equal to 1. This also indicates that the functional margins of both positive and negative training examples are 1. Thus,

$$\min_{i:y^{(i)}=1} \left(w^{*T}x^{(i)} + b \right) = 1,$$

and

$$\max_{i:y^{(i)}=-1} \left(w^{*T}x^{(i)} + b \right) = -1.$$

Adding both sides of the above two equations and solving for b results in the optimal value:

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T}x^{(i)} + \min_{i:y^{(i)}=1} w^{*T}x^{(i)}}{2}.$$

Note: From the lecture video (video 8, 1:09:30), the intuition behind this formula is that: find the worst positive and negative examples and place the separating hyperplane between them, such that the two examples have the same distance to the hyperplane.

3.3 Page 20: The dual form of the ℓ_1 norm soft margin SVM

The primal problem of the ℓ_1 norm soft margin SVM is given by

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ i = 1, \dots, m \\ & \xi_i \geq 0,\ i = 1, \dots, m. \end{aligned}$$

The Lagrangian is given as

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i \xi_i.$$

To find the dual form of the problem, we need to first minimize $\mathcal{L}(w, b, \xi, \alpha, r)$ with respect to w, b, and ξ (for fixed α and r), to get $\theta_{\mathcal{D}}$, which we'll do by setting the derivatives of $\mathcal{L}$ with respect to w, b, and ξ to zero. We have:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0, \\ \frac{\partial \mathcal{L}}{\partial b} &= -\sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \\ \frac{\partial \mathcal{L}}{\partial \xi_i} &= C - \alpha_i - r_i = 0. \end{aligned}$$

These imply that

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}, \tag{1}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \tag{2}$$

$$C - \alpha_i - r_i = 0. \tag{3}$$

Plugging all these back into the Lagrangian, and simplify, we get

$$\mathcal{L}(w, b, \xi, \alpha, r) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle.$$

Therefore, the dual form of the problem is given by

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C,\ i = 1, \dots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \end{aligned}$$

3.4 Page 20: The KKT dual complementary conditions of the ℓ_1 norm soft margin SVM

The KKT dual complementary conditions are given by

$$\alpha_i[y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i] = 0,\ i = 1, \dots, m \tag{4}$$
$$r_i \xi_i = 0,\ i = 1, \dots, m \tag{5}$$
$$\alpha_i \geq 0, r_i \geq 0,\ i = 1, \dots, m. \tag{6}$$

Since $C - \alpha_i - r_i = 0$, we have

$$\alpha_i = 0 \Rightarrow r_i = C.$$

Furthermore, $r_i \xi_i = 0$ implies $\xi_i = 0$. Finally, since $y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i$, we have $y^{(i)}(w^T x^{(i)} + b) \geq 1$, i.e.,

$$\alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1.$$

For $\alpha_i \neq 0$, we have

$$\alpha_i[y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i] = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1 - \xi_i.$$

If $\alpha_i = C$, then

$$C - \alpha_i - r_i = 0 \Rightarrow r_i = 0 \Rightarrow \xi_i \geq 0.$$

Therefore

$$\alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1.$$

If $0 < \alpha_i < C$, then

$$C - \alpha_i - r_i = 0 \Rightarrow r_i \neq 0 \Rightarrow \xi_i = 0.$$

Hence

$$0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1.$$

To sum up, we have

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1. \end{aligned}$$

4 Notes on: Lecture notes 4

4.1 Page 6: The size of training set m (sample complexity)

Proof. The question: Given γ and some $\delta > 0$, how large must m be before we can guarantee that with probability at least $1 - \delta$, training error will be within γ of generalization error?

Since

$$P(\forall h \in \mathcal{H}, |\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma) \geq 1 - 2k \exp(-2\gamma^2 m),$$

we set

$$2k \exp(-2\gamma^2 m) = \delta.$$

Solving for m, we find that if

$$m \geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta},$$

then with probability at least $1 - \delta$, we have that $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$ for all $h \in \mathcal{H}$.

□

4.2 Page 7: The error bound γ

Proof. We can also hold m and δ fixed and solve for γ in

$$2k \exp(-2\gamma^2 m) = \delta,$$

which results in

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

Therefore, with probability at least $1 - \delta$, we have that for all $h \in \mathcal{H}$,

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

□

4.3 Page 8: Corollary

Proof. Let $|\mathcal{H}| = k$, and let any δ, γ be fixed. Then for $\varepsilon(\hat{h}) \leq \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$ to hold with probability at least $1 - \delta$, it suffices that

$$\gamma \geq \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

Solving for m gives us

$$\begin{aligned} m &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right) \psi. \end{aligned}$$

□

5 Notes on: Lecture notes 5

5.1 Page 7: The posterior distribution $p(y|x, S)$

To compute the posterior distribution $p(y|x, S)$, we can integrate the joint density $p(y, \theta|x, S)$ over θ. That is

$$p(y|x, S) = \int_\theta p(y, \theta|x, S)\, d\theta.$$

Since $p(u, v) = p(u|v)p(v)$, we can write

$$p(y, \theta|x, S) = p(y|\theta, x, S)p(\theta|x, S).$$

Now, we simplify the above equation.

The distribution $p(\theta|x, S)$ Note that the parameter θ is computed (or estimated) using only the information of training set S. Therefore, the distribution of θ is completely known once given S. This implies

$$p(\theta|x, S) = p(\theta|S).$$

The distribution $p(y|\theta, x, S)$ Note that once given x and the parameter θ, the distribution (or probability) of y is completely known (since the probability y is a function of x and θ; see MLE or Eq (1) in Lecture Note 5). This implies

$$p(y|\theta, x, S) = p(y|\theta, x).$$

Plugging these into the above integral, we finally have

$$p(y|x, S) = \int_\theta p(y|\theta, x)p(\theta|S)\, d\theta,$$

as shown in Eq (5) in Lecture Notes 5 (Page 7).

6 Notes on: Lecture notes 7a

6.1 Implementation of k-means

The k-means clustering algorithm is as follows:

1. Initialize **cluster centroids** $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$ randomly, where K is the number of clusters.
2. Repeat until convergence: {

   Class Assignment For every i, set

   $$c^{(i)} := \arg\min_k ||x^{(i)} - \mu_k||^2.$$

   Centroid Update For each k, set

   $$\mu_k := \frac{\sum_{i=1}^m I\{c^{(i)} = k\}x^{(i)}}{\sum_{i=1}^m I\{c^{(i)} = k\}}.$$

   }

6.1.1 Class Assignment

The key of class assignment is to compute the distance between each training example $x^{(i)}$ with respect to each centroid μ_k. This can be done efficiently if we use vectorization. Denote X as the conventional $m \times n$ design matrix and μ as the $K \times n$ matrix that contains all the K centroids. We further define D as the distance matrix, whose (i, k)-entry, i.e., $D(i, k)$ satisfies

$$D_{ik} = ||x^{(i)} - \mu_k||^2.$$

Notice that D is of size $m \times K$. There are mainly two ways for computing D. First, we can use for-loop, as below,

```
D = zeros(size(X, 1), K);
for k = 1 : K
    d = bsxfun(@minus, X, mu(k, :));
    D(:, k) = sum(d.^2, 2);
end
```

Indeed, the above computation can also be carried out using vectorization. To that end, we note that

$$||x^{(i)} - \mu_k||^2 = ||x^{(i)}||^2 - 2(x^{(i)})^T\mu_k + ||\mu_k||^2.$$

Further note that there is no need to include the term $||x^{(i)}||^2$ in the computation of $||x^{(i)} - \mu_k||^2$, since it is the same for each cluster centroid μ_k and thus will not affect finding the closet cluster.

To develop a vectorization implementation for computing D, we denote $\hat{x}$ as a $m \times 1$ vector that $\hat{x}_i = ||x^{(i)}||^2$ and $\hat{\mu}$ as a $K \times 1$ vector that $\hat{\mu}_k = ||\mu_k||^2$. Then D can be computed through

$$D = \hat{x} \cdot \mathbf{1}_{1\times K} - 2 \cdot X \cdot \mu^T + \mathbf{1}_{m\times 1} \cdot \hat{\mu}^T.$$

There are three implementations of this computation

```
% The following three implementations are in efficient descent order
% 1:
D = bsxfun(@plus, bsxfun(@plus, -2*X*mu', sum(mu.^2,2)'), sum(X.^2,2));
% D = bsxfun(@plus, -2*X*mu', sum(mu.^2,2)');  % simplified version
% % 2:
% D = sum(X.^2,2)*ones(1,K) - 2*X*mu' + ones(m,1)*sum(mu.^2,2)';
% % D = - 2*X*mu' + ones(m,1)*sum(mu.^2,2)';  % simplified version
% % 3:
% D = repmat(sum(X.^2,2), [1,K]) - 2*X*mu' + repmat(sum(mu.^2,2)', [m,1]);
% % D = - 2*X*mu' + repmat(sum(mu.^2,2)', [m,1]);  % simplified version
```

After computing the distance matrix D, the class vector c is obtained by

```
[~, c] = min(D, [], 2);
```

The overall implementation of this step is given by `findClosestCentroids` function as below

```
function c = findClosestCentroids(X, mu)
% ***************************************************************
% This function performs centroid assignment
% Copyright Chenglong Chen, May 30, 2013
% ***************************************************************
% -- Input:
% X: training set (m by n matrix)
% mu: matrix of all mean vectors (K by n matrix)
%       where mu(k, :) = mu_k
% -- Output:
% c: class label vector (m by 1 vector)
% ***************************************************************
% Pls refer to Ng's Lecture Note 7a Page 1-3 for detail.
% ***************************************************************

m = size(X, 1);
K = size(mu, 1);

% % ----------------------------------------
% % First method: use a for loop
% % ----------------------------------------
% D = zeros(size(X, 1), K);
% for k = 1 : K
%     d = bsxfun(@minus, X, mu(k, :));
%     D(:, k) = sum(d.^2, 2);
% end

% ---------------------------------------------------------------------
% Second Method: use a vectorized implementation
% ------------------------------------------------------------------------------
% Note that for a signle example x^(i), the distance between
% x^(i) and centroid u_k can be computed through:
% || x^(i) - u_k ||^2 = || x^(i) ||^2 - 2 * (x^(i))^T * u_k + || u_k ||^2
% ------------------------------------------------------------------------------
% The following three implementations are in efficient descent order
% 1:
D = bsxfun(@plus, bsxfun(@plus, -2*X*mu', sum(mu.^2,2)'), sum(X.^2,2));
% D = bsxfun(@plus, -2*X*mu', sum(mu.^2,2)');  % simplified version
% % 2:
% D = sum(X.^2,2)*ones(1,K) - 2*X*mu' + ones(m,1)*sum(mu.^2,2)';
% % D = - 2*X*mu' + ones(m,1)*sum(mu.^2,2)';  % simplified version
% % 3:
% D = repmat(sum(X.^2,2), [1,K]) - 2*X*mu' + repmat(sum(mu.^2,2)', [m,1]);
% % D = - 2*X*mu' + repmat(sum(mu.^2,2)', [m,1]);  % simplified version

% find the closet centroid
[~, c] = min(D, [], 2);

% % just one line code using "pdist2" function; slower
% [~, c] = pdist2(mu, X, 'euclidean', 'Smallest', 1);

end  % of function findClosestCentroids
```

6.1.2 Centroid Update

Using a for-loop, it is not hard to update the centroid

```
mu = zeros(K, n);
for k = 1 : K
    mu(k, :) = mean(X(c==k, :));
end  % of for
```

To develop a vectorization of this computation, we first play with a small examples. To this end, we simply assume $m = 4$ and $K = 2$. Thus

$$X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ (x^{(3)})^T \\ (x^{(4)})^T \end{bmatrix}$$

We further assume c is given as follow

$$c = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix}.$$

Therefore,

$$\mu_1 = \frac{x^{(1)} + x^{(2)}}{2}$$

and

$$\mu_2 = \frac{x^{(3)} + x^{(4)}}{2}.$$

Finally, we have

$$\mu = \begin{bmatrix} (\mu_1)^T \\ (\mu_2)^T \end{bmatrix}.$$

Indeed, the above computation can be expressed in a more compact manner. We first convert c to an $m \times K$ index matrix C, i.e.,

$$C = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix},$$

It is easy to verify that the following equation holds

$$\begin{aligned} \frac{1}{2}C^T \cdot X &= \frac{1}{2}\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ (x^{(3)})^T \\ (x^{(4)})^T \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} (x^{(1)})^T + (x^{(2)})^T \\ (x^{(3)})^T + (x^{(4)})^T \end{bmatrix} \\ &= \mu. \end{aligned}$$

For a more general case, it is computed used the following code

```
C = sparse(1:m, c, 1, m, K);  % the index matrix; it is much faster
% without converting it to "full" matrix, i.e., C = full(sparse(...))
% update the centroids
mu = bsxfun(@rdivide, C' * X, sum(C)');  % use "bsxfun"; faster
% mu = (C * spdiags(1./sum(C)', 0, K, K))' * X;  % or use "spdiags"; slower
% mu = (C * spdiags(spfun(@(x) 1./x,sum(C)'), 0, K, K))' * X;
```

This vectorization maybe a little memory consuming (due to the index matrix C) while it is generally computationally efficient. For a very large training set, it maybe better to use the for-loop implementation.

The overall implementation of this step is given by `computeCentroids` function as below

```
function mu = computeCentroids(X, c, K)
% *****************************************************************
% This function updates the centroids
% Copyright Chenglong Chen, May 30, 2013
% *****************************************************************
% -- Input:
% X: training set (m by n matrix)
% c: class label vector (m by 1 vector)
% K: number of clusters
% -- Output:
% mu: matrix of all mean vectors (K by n matrix)
%       where mu(k, :) = mu_k
% *****************************************************************
% Pls refer to Ng's Lecture Note 7a Page 1-3 for detail.
% *****************************************************************

% Useful variables
m = size(X, 1);

% % First Method: use a for loop
% mu = zeros(K, n);
% for k = 1 : K
%     mu(k, :) = mean(X(c==k, :));
% end  % of for

% Second Method: use a vectorized implementation
C = sparse(1:m, c, 1, m, K);  % the index matrix; it is much faster
% without converting it to "full" matrix, i.e., C = full(sparse(...))
% update the centroids
mu = bsxfun(@rdivide, C' * X, sum(C)');  % use "bsxfun"; faster
% mu = (C * spdiags(1./sum(C)', 0, K, K))' * X;  % or use "spdiags"; slower
% mu = (C * spdiags(spfun(@(x) 1./x,sum(C)'), 0, K, K))' * X;

% % just one line code using "grpstats" function; slower
% mu = grpstats(X, c);

end  % of function computeCentroids
```

6.1.3 Distortion function

According the Lecture notes 7a, Page 2, k-means is the coordinate descent on the following cost function (or distortion function) J

$$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2.$$

(In Ng's ML course in Coursera, $J(c, \mu)$ is the average distortion over the entire training examples. But, they are essentially the same thing.) Specifically, the inner-loop of k-means repeatedly minimizes J with respect to c while holding μ fixed (i.e., the **cluster assignment** step), and then minimizes J with respect to μ while holding c fixed (i.e., the **centroid update** step). Thus, J must monotonically decrease, and the value of J must converge. Therefore, J can be used to check if k-means converges.

We define a $m \times K$ matrix `Mu` whose i-th row is $\mu_{c^{(i)}}$. Then J can be computed using

```
J = sum(sum((X-Mu).^2));
```

All that remained to be computed is the matrix `Mu`. The easiest way is to use a for-loop, i.e.,

```
Mu = zeros(size(X));
for k = 1 : K
    Mu(c==k, :) = repmat(mu(k, :), [sum(c==k), 1]);
end  % of for
```

There also exists an efficient vectorization, i.e.,

```
C = sparse(1:m, c, 1, m, K);  % the index matrix; it is much faster
% without converting it to "full" matrix, i.e., C = full(sparse(...))
Mu = C*mu;  % the centroids matrix
```

To verify the correctness of the above implementation, we again use the previous small example. It is obvious that

$$C \cdot \mu = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (\mu_1)^T \\ (\mu_2)^T \end{bmatrix}$$
$$= \begin{bmatrix} (\mu_1)^T \\ (\mu_1)^T \\ (\mu_2)^T \\ (\mu_2)^T \end{bmatrix}.$$

The overall implementation of distortion function is given by `computeDistortion` function as below

```
function J = computeDistortion(X, mu, c)
% ***************************************************************
% This function computes the distortion
% Copyright Chenglong Chen, May 30, 2013
% ***************************************************************
% -- Input:
% X: training set (m by n matrix)
% mu: matrix of all mean vectors (k by n matrix)
%       where mu(j, :) = mu_j
% c: class label vector (m by 1 vector)
% -- Output:
% J: the distortion
% ***************************************************************
% Pls refer to Ng's Lecture Note 7a Page 1-3 for detail.
% ***************************************************************

K = size(mu, 1);

% % First Method: use a for loop
% Mu = zeros(size(X));
% for k = 1 : K
%     Mu(c==k, :) = repmat(mu(k, :), [sum(c==k), 1]);
% end  % of for

% Second Method: use a vectorized implementation
m = size(X, 1);
C = sparse(1:m, c, 1, m, K);  % the index matrix; it is much faster
% without converting it to "full" matrix, i.e., C = full(sparse(...))
Mu = C*mu;  % the centroids matrix

% compute the distortion
J = sum(sum((X-Mu).^2));

end  % of function computeDistortion
```

6.1.4 The overall implementation of k-means

Putting previous codes together, we have the following implementation of k-means[5]

[5]All the codes can be found in the corresponding folder. There are also some examples of running k-means on different learning tasks.

```
function [mu, c, J] = kmeansCluster(X, K)
% *****************************************************************
% This function performs k-means clustering
% Copyright Chenglong Chen, May 30, 2013
% *****************************************************************
% -- Input:
% X: training set (m by n matrix)
% K: number of centroids
% -- Output:
% mu: matrix of all mean vectors (k by n matrix)
%       where mu(j, :) = mu_j
% c: class label vector (m by 1 vector)
% J: value of distortion function at each iteration
% *****************************************************************
% Pls refer to Ng's Lecture Note 7a Page 1-3 for detail.
% *****************************************************************

m = size(X, 1);

% initialize the centroids as k randomly chosen training examples
rand('state', 0);
randOrder = randperm(m);
mu = X(randOrder(1:K), :);
% mu = rand(K, n);  % or just randomly initialized it

% vector for the value of distortion function
J = [0, Inf];  % we initialize it with two values before the iteration
Tol = 1e-3;  % the tolerance to terminate the iteration


while abs(J(end) - J(end-1)) > Tol

    % ----------------
    % assign class
    % ----------------
    c = findClosestCentroids(X, mu);

    % ---------------------------------------------------------------------
    % delete the centroid to which no point has been assigned.
    % ---------------------------------------------------------------------
    % the number of points assigned to the k-th centroid

%     % First Method: use a for loop
%     num = zeros(1, K);
%     for k = 1 : K
%         num(k) = sum(c==k);
%     end

    % Second Method: use sparse matrix
    num = sum(sparse(1:m, c, 1, m, K));  % faster than for loop
    mu(num==0, :) = [];  % delete the centroid
    K = size(mu, 1);  % new K
    % adjust the class vector accordingly based on new K
    [~,~,idx] = unique(c);
    b = 1:K;
    c = b(idx);

    % -------------------------------
    % update the centroids
    % -------------------------------
    mu = computeCentroids(X, c, K);

    % -----------------------------------------------
    % compute the value of distortion function
    % -----------------------------------------------
    J(end+1) = computeDistortion(X, mu, c);

end  % of while

J(1:2) = [];  % delete those two values

end  % of function kmeansCluster
```

7 Notes on: Lecture notes 7b

7.1 Page 2: The ML estimates of Gaussian Mixture Model

Different from Gaussian Discriminant Analysis model (GDA), we here use z to play the role of the class labels in the Gaussian Mixture Model (GMM). There are other minor differences in GMM from GDA, 1) it is generalized the z to be multinomial rather than Bernoulli, and 2) different Σ_j is used for each Gaussian.

Proof. Since

$$p(x|z) = \frac{1}{(2\pi)^{n/2}|\Sigma_z|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_z)^T\Sigma_z^{-1}(x-\mu_z)\right),$$

the log-likelihood of the data can then be expressed as

$$\begin{aligned}
\ell(\phi,\mu,\Sigma) &= \sum_{i=1}^{m} \log p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi) \\
&= \sum_{i=1}^{m} \log \frac{1}{(2\pi)^{n/2}|\Sigma_{z^{(i)}}|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)}-\mu_{z^{(i)}})^T\Sigma_{z^{(i)}}^{-1}(x^{(i)}-\mu_{z^{(i)}})\right) \cdot \phi_{z^{(i)}}.
\end{aligned}$$

After simplification, we obtain

$$\ell(\phi,\mu,\Sigma) = \sum_{i=1}^{m} \left[-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_{z^{(i)}}| - \frac{1}{2}(x^{(i)}-\mu_{z^{(i)}})^T\Sigma_{z^{(i)}}^{-1}(x^{(i)}-\mu_{z^{(i)}}) + \log(\phi_{z^{(i)}})\right].$$

By computing the partial derivatives of the log-likelihood $\ell(\phi,\mu,\Sigma)$ with respect to the parameters and then solving for zero, we can find the maximum likelihood estimate of the parameters.

1. For ϕ_j, we have the following terms that depend on ϕ_j

$$\sum_{i=1}^{m} \log(\phi_{z^{(i)}}).$$

With the constraint that the ϕ_j's sum to 1 since they represent the probabilities $\phi = p(z^{(i)} = j; \phi)$, we construct the following Lagrangian

$$\mathcal{L}(\phi) = \sum_{i=1}^{m} \log(\phi_{z^{(i)}}) + \beta(\sum_{j=1}^{k} \phi_j - 1),$$

where β is the Lagrange multiplier. Taking derivatives, we find

$$\frac{\partial \mathcal{L}}{\partial \phi_j} = \sum_{i=1}^{m} \left(\frac{1}{\phi_{z^{(i)}}} \frac{\partial \phi_{z^{(i)}}}{\partial \phi_j} \right) + \beta = \sum_{i=1}^{m} \frac{I\{z^{(i)} = j\}}{\phi_j} + \beta.$$

Setting this to zero and solving, we get

$$\phi_j = \frac{\sum_{i=1}^{m} I\{z^{(i)} = j\}}{-\beta}.$$

Using the constraint that $\sum_j \phi_j = 1$, we easily find that

$$-\beta = \sum_{i=1}^{m} \sum_{j=1}^{k} I\{z^{(i)} = j\} = \sum_{i=1}^{m} 1 = m$$

This used the fact that $z^{(i)}$ is drawn from $\{1, 2 \ldots, k\}$, thus $\sum_{j=1}^{k} I\{z^{(i)} = j\} = 1$. Finally, we obtain

$$\phi_j = \frac{1}{m} \sum_{i=1}^{m} I\{z^{(i)} = j\}$$

as presented in the Lecture Notes.

2. To derive partial derivative $\frac{\partial \ell}{\partial \mu_j}$, we need the following fact for vector derivative

$$\frac{\partial x^T A x}{\partial x} = (A + A^T)x,$$

where $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$. Specially, if A is symmetric, i.e., $A^T = A$, we have

$$\frac{\partial x^T A x}{\partial x} = 2Ax.$$

Proof. Using the product rule, the differential of $x^T A x$ follows[6]

$$\begin{aligned} \mathrm{d}(x^T A x) &= (\mathrm{d}x^T) A x + x^T A \mathrm{d}x \\ &= (\mathrm{d}x)^T A x + x^T A \mathrm{d}x \\ &= \mathrm{tr}(\mathrm{d}x)^T A x + \mathrm{tr} x^T A \mathrm{d}x \\ &= \mathrm{tr}\big((\mathrm{d}x)^T A x\big)^T + \mathrm{tr} x^T A \mathrm{d}x \\ &= \mathrm{tr}(x^T A^T + x^T A)\mathrm{d}x. \end{aligned}$$

[6]See my notes on Lecture Note 1 for more details about the differential of a vector or matrix, in particular Note 1.4 and 1.5.

So

$$\frac{\partial x^T A x}{\partial x} = (x^T A^T + x^T A)^T = (A + A^T)x.$$

□

Now to derive $\frac{\partial \ell}{\partial \mu_j}$ $(j = \{0, 1\})$ for

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{m} \left[-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_{z^{(i)}}| - \frac{1}{2}(x^{(i)} - \mu_{z^{(i)}})^T \Sigma_{z^{(i)}}^{-1}(x^{(i)} - \mu_{z^{(i)}}) + \log(\phi_{z^{(i)}}) \right].$$

we have

$$\begin{aligned} \frac{\partial \ell}{\partial \mu_j} &= -\frac{1}{2}\sum_{i=1}^{m} \frac{\partial}{\partial \mu_j}\Big((x^{(i)} - \mu_{z^{(i)}})^T \Sigma_{z^{(i)}}^{-1}(x^{(i)} - \mu_{z^{(i)}})\Big) \\ &= \sum_{i=1}^{m} \Sigma_j^{-1}(x^{(i)} - \mu_j) I\{z^{(i)} = j\}, \end{aligned}$$

where the second equation is obtained using the fact that the covariance matrix Σ_j (and also Σ_j^{-1}) is symmetric.

Setting $\frac{\partial \ell}{\partial \mu_j} = 0$ and solving for μ_j, we get the MLE of μ_j, i.e.,

$$\mu_j = \frac{\sum_{i=1}^{m} I\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} I\{z^{(i)} = j\}},$$

as presented in the Lecture Notes.

3. To derive partial derivative $\frac{\partial \ell}{\partial \Sigma_j}$, we need the following fact for matrix derivative

$$\frac{\partial x^T A x}{\partial A} = x x^T,$$

where $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$.

Proof. The differential of $x^T A x$ follows

$$\begin{aligned} \mathrm{d}(x^T A x) &= \mathrm{d}\,\mathrm{tr} x^T A x \\ &= \mathrm{tr}\,\mathrm{d}(x^T A x) \\ &= \mathrm{tr} x^T (\mathrm{d}A) x \\ &= \mathrm{tr} x x^T \mathrm{d}A. \end{aligned}$$

The above equation implies

$$\frac{\partial x^T A x}{\partial A} = (x x^T)^T = x x^T.$$

□

However, it is a rather hard job to derive $\frac{\partial \ell}{\partial \Sigma_j}$ for

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{m} \left[-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_{z^{(i)}}| - \frac{1}{2}(x^{(i)} - \mu_{z^{(i)}})^T \Sigma_{z^{(i)}}^{-1}(x^{(i)} - \mu_{z^{(i)}}) + \log(\phi_{z^{(i)}}) \right],$$

since it involves the term $\frac{1}{2}(x^{(i)}-\mu_{z^{(i)}})^T\Sigma^{-1}_{z^{(i)}}(x^{(i)}-\mu_{z^{(i)}})$ and in particular $\Sigma^{-1}_{z^{(i)}}$. Therefore, rather than directly derive the MLE of Σ_j itself, we choose to derive the MLE of Σ_j^{-1}. To this end, we rewrite the log-likelihood slightly as

$$\ell(\phi,\mu,\Sigma^{-1}) = \sum_{i=1}^m \left[-\frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\Sigma^{-1}_{z^{(i)}}| - \frac{1}{2}(x^{(i)}-\mu_{z^{(i)}})^T\Sigma^{-1}_{z^{(i)}}(x^{(i)}-\mu_{z^{(i)}}) + \log(\phi_{z^{(i)}})\right].$$

where the fact $|A^{-1}| = |A|^{-1}$ is used. Now we compute the partial derivative w.r.t. Σ_j^{-1}, obtaining

$$\begin{aligned}\frac{\partial \ell}{\partial \Sigma_j^{-1}} &= \frac{1}{2}\sum_{i=1}^m \left[\frac{1}{|\Sigma^{-1}_{z^{(i)}}|}\frac{\partial |\Sigma^{-1}_{z^{(i)}}|}{\partial \Sigma_j^{-1}} - (x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T I\{z^{(i)}=j\}\right] \\ &= \frac{1}{2}\sum_{i=1}^m \left[\Sigma_j - (x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T\right] I\{z^{(i)}=j\}.\end{aligned}$$

where we use the fact that $\frac{\partial |A|}{\partial A} = |A|(A^{-1})^T$ from Page 9 of Lecture Note 1 and $\Sigma_j^T = \Sigma_j$ because it is symmetric. Setting $\frac{\partial \ell}{\partial \Sigma_j^{-1}} = 0$ and solving for Σ_j, we get the MLE of Σ_j, i.e.,

$$\Sigma_j = \frac{\sum_{i=1}^m I\{z^{(i)}=j\}(x^{(i)}-\mu_j)(x^{(i)}-\mu_j)^T}{\sum_{i=1}^m I\{z^{(i)}=j\}}.$$

as presented in the Lecture Notes.

□

8 Notes on: Lecture notes 8

8.1 Page 1: 1-D proof of Jensen's inequality

Proof. The tangent line of $f(x)$ through point $(E[X], f(E[X]))$ is given by

$$y = k(E[X])(x - E[X]) + f(E[X]),$$

where $k(E[X])$ is the slope at point $(E[X], f(E[X]))$. Since $f(x)$ is convex, we have

$$f(x) \geq k(E[X])(x - E[X]) + f(E[X]).$$

Let $x = X$, we then have

$$f(X) \geq k(E[X])(X - E[X]) + f(E[X]).$$

Taking the expectation of both sides yields

$$E[f(X)] \geq f(E[X]).$$

□

8.2 Page 6-8: The EM algorithm of Gaussian Mixture Model

In the Lecture Notes, the update rule for ϕ and μ are already derived. Herein, we derive the update rule for Σ. Recall that in the E-step, we set

$$w_j^{(i)} = Q_i(z^{(i)}=j) = p(z^{(i)}=j|x^{(i)};\phi,\mu,\Sigma).$$

In the M-step, we need to maximize, with respect to our parameters ϕ, μ, Σ, the quantity

$$\begin{aligned}
&\sum_{i=1}^{m}\sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)}{Q_i(z^{(i)})} \\
&\quad = \sum_{i=1}^{m}\sum_{z^{(i)}} Q_i(z^{(i)} = j) \log \frac{p(x^{(i)}|z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{Q_i(z^{(i)} = j)} \\
&\quad = \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \log \frac{\frac{1}{(2\pi)^{n/2}|\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)\right) \cdot \phi_j}{w_j^{(i)}}.
\end{aligned}$$

Grouping together only the terms that depend on Σ_j, we find that we need to maximize

$$\mathcal{L}(\Sigma) = \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \left[-\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)\right].$$

However, it is a rather hard job to derive $\frac{\partial \mathcal{L}}{\partial \Sigma_j}$ since it involves the term $\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)$ and in particular Σ_j^{-1}. Therefore, rather than directly derive the MLE of Σ_j itself, we choose to derive the MLE of Σ_j^{-1}. To this end, we rewrite $\mathcal{L}(\Sigma)$ slightly as

$$\mathcal{L}(\Sigma^{-1}) = \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \left[\frac{1}{2}\log|\Sigma_j^{-1}| - \frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)\right].$$

where the fact $|A^{-1}| = |A|^{-1}$ is used. We then compute the derivative $\frac{\partial \mathcal{L}}{\partial \Sigma_j^{-1}}$, i.e.,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \Sigma_j^{-1}} &= \frac{1}{2}\sum_{i=1}^{m} w_j^{(i)} \left[\frac{1}{|\Sigma_j^{-1}|}\frac{\partial |\Sigma_j^{-1}|}{\partial \Sigma_j^{-1}} - (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T\right] \\
&= \frac{1}{2}\sum_{i=1}^{m} w_j^{(i)} \left[\Sigma_j - (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T\right],
\end{aligned}$$

where we use the following facts

$$\begin{aligned}
\frac{\partial x^T A x}{\partial A} &= x x^T, \\
\frac{\partial |A|}{\partial A} &= |A|(A^{-1})^T,
\end{aligned}$$

and $\Sigma_j^T = \Sigma_j$ because it is symmetric.

Setting $\frac{\partial \mathcal{L}}{\partial \Sigma_j^{-1}}$ to zero and solving for Σ_j, we get

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_j^{(i)}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}},$$

as presented in Page 3 in Lecture Note 7b.

8.3 Derivation of the EM algorithm of Naive Bayes

For the derivation of the MLE of Naive Bayes, please see my solution to problem 4 in the problem set #1, Public Course (i.e., old version). We herein derive the EM algorithm for Naive Bayes, which is presented in the Lecture Video 13 (around 0:45:00). To recap a little bit, we first describe the problem.

Here, the inpute features x_j, $j = 1, \dots, n$ to our model are discrete, binary-valued variables, so $x_j \in \{0, 1\}$. We call $x = [x_1, x_2, \dots, x_n]^T$ to be input vector. For each training example $x^{(i)}$, we assume it comes from some clusters specifying using the laten variable z. (Different from MLE of Naive Bayes, we use z to play the role of class label. Note that z is unobserved.) To simplify the analysis, we assume z is a single binary-valued $z \in \{0, 1\}$. Our model is then parameterized by

$$\begin{aligned}
\phi_{j|z=0} &= p(x_j = 1|z = 0),\\
\phi_{j|z=1} &= p(x_j = 1|z = 1),\\
\phi_z &= p(z = 1).
\end{aligned}$$

We model the joint distribution of (x, z) according to

$$\begin{aligned}
p(z) &= (\phi_z)^z (1 - \phi_z)^{1-z},\\
p(x|z = 1) &= \prod_{j=1}^n p(x_j|z = 1)\\
&= \prod_{j=1}^n (\phi_{j|z=1})^{x_j} (1 - \phi_{j|z=1})^{1-x_j},\\
p(x|z = 0) &= \prod_{j=1}^n p(x_j|z = 0)\\
&= \prod_{j=1}^n (\phi_{j|z=0})^{x_j} (1 - \phi_{j|z=0})^{1-x_j}.
\end{aligned}$$

All these result in

$$p(x, z) = p(x|z)p(z) = \left(\prod_{j=1}^n (\phi_{j|z})^{x_j} (1 - \phi_{j|z})^{1-x_j}\right)(\phi_z)^z (1 - \phi_z)^{1-z}.$$

In the E-step, we set

$$w^{(i)} = Q_i(z^{(i)} = 1) = p(z^{(i)} = 1|x^{(i)}; \phi_{j|z}, \phi_z).$$

Next, in the M-step, we need to maximize, with respect to our parameters $\phi_{j|z}, \phi_z$, the quantity

$$\begin{aligned}
&\sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi_{j|z}, \phi_z)}{Q_i(z^{(i)})}\\
&= \sum_{i=1}^m \sum_{l=0}^1 Q_i(z^{(i)} = l) \log \frac{p(x^{(i)}, z^{(i)} = l; \phi_{j|z}, \phi_z)}{Q_i(z^{(i)} = l)}\\
&= \sum_{i=1}^m \Bigg[Q_i(z^{(i)} = 0) \log \frac{p(x^{(i)}, z^{(i)} = 0; \phi_{j|z}, \phi_z)}{Q_i(z^{(i)} = 0)}\\
&\qquad + Q_i(z^{(i)} = 1) \log \frac{p(x^{(i)}, z^{(i)} = 1; \phi_{j|z}, \phi_z)}{Q_i(z^{(i)} = 1)}\Bigg]\\
&= \sum_{i=1}^m \Bigg[(1 - w^{(i)}) \log \frac{\left(\prod_{j=1}^n (\phi_{j|z=0})^{x_j} (1 - \phi_{j|z=0})^{1-x_j}\right) \cdot (1 - \phi_z)}{1 - w^{(i)}}\\
&\qquad + w^{(i)} \log \frac{\left(\prod_{j=1}^n (\phi_{j|z=1})^{x_j} (1 - \phi_{j|z=1})^{1-x_j}\right) \cdot \phi_z}{w^{(i)}}\Bigg].
\end{aligned}$$

1. For ϕ_z. Grouping together only the terms that depend on ϕ_z, we find that we need to maximize

$$\sum_{i=1}^{m} \left[(1 - w^{(i)}) \log(1 - \phi_z) + w^{(i)} \log \phi_z \right].$$

   Setting the derivative, i.e.,

$$\frac{\partial}{\partial \phi_z} \sum_{i=1}^{m} \left[(1 - w^{(i)}) \log(1 - \phi_z) + w^{(i)} \log \phi_z \right] = \sum_{i=1}^{m} \left[-\frac{1 - w^{(i)}}{1 - \phi_z} + \frac{w^{(i)}}{\phi_z} \right],$$

   to zero and solving, we get

$$\phi_z = \frac{1}{m} \sum_{i=1}^{m} w^{(i)},$$

   as presented in the Lecture Video 13 (around 0:45:00).

2. For $\phi_{j|z=0}$ and $\phi_{j|z=1}$. Grouping together only the terms that depend on $\phi_{j|z=0}$, we find that we need to maximize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (1 - w^{(i)}) \big(x_j \log \phi_{j|z=0} + (1 - x_j) \log(1 - \phi_{j|z=0}) \big).$$

   Setting the derivative, i.e.,

$$\frac{\partial}{\partial \phi_{j|z=0}} \sum_{i=1}^{m} \sum_{j=1}^{n} (1 - w^{(i)}) \big(x_j \log \phi_{j|z=0} + (1 - x_j) \log(1 - \phi_{j|z=0}) \big) = \sum_{i=1}^{m} (1 - w^{(i)}) \left[\frac{x_j}{\phi_{j|z=0}} - \frac{1 - x_j}{1 - \phi_{j|z=0}} \right],$$

   to zero and solving, we get

$$\begin{aligned} \phi_{j|z=0} &= \frac{\sum_{i=1}^{m} (1 - w^{(i)}) x_j}{\sum_{i=1}^{m} (1 - w^{(i)})} \\ &= \frac{\sum_{i=1}^{m} (1 - w^{(i)}) I\{x_j = 1\}}{\sum_{i=1}^{m} (1 - w^{(i)})}, \end{aligned}$$

   as presented in the Lecture Video 13 (around 0:45:00).

   Similarly, we can obtain

$$\begin{aligned} \phi_{j|z=1} &= \frac{\sum_{i=1}^{m} w^{(i)} x_j}{\sum_{i=1}^{m} w^{(i)}} \\ &= \frac{\sum_{i=1}^{m} w^{(i)} I\{x_j = 1\}}{\sum_{i=1}^{m} w^{(i)}}. \end{aligned}$$

9 Notes on: Lecture notes 9

9.1 Page 1: The covariance matrix Σ is singular

Denote X as the $m \times n$ design matrix (suppose we have already zero out the mean for each column; so X have zero mean for each column). The covariance matrix Σ can then be expressed as

$$\Sigma = \frac{1}{m} X^T X.$$

If $m < n$, i.e., the number of examples is less than the dimension of the data, it can be shown that $X^T X$ and therefore Σ are not invertible, or namely singular. The proof can be found in Note 1.7 of this Note.

9.2 Page 2: The MLE for restriction of Σ being diagonal

Proof. The log-likelihood of the parameters given the data is

$$\begin{aligned}\ell(\mu, \Sigma) &= \sum_{i=1}^{m} \log \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu)\right) \\ &= \sum_{i=1}^{m}\left[-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu)\right].\end{aligned}$$

If Σ is diagonal, i.e.,

$$\Sigma = \begin{bmatrix}\Sigma_{11} & 0 & \cdots & 0 \\ 0 & \Sigma_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Sigma_{nn}\end{bmatrix},$$

we have

$$|\Sigma| = \Sigma_{11}\Sigma_{22}\cdots\Sigma_{nn} = \prod_{j=1}^{n}\Sigma_{jj},$$

and

$$\Sigma^{-1} = \begin{bmatrix}\Sigma_{11}^{-1} & 0 & \cdots & 0 \\ 0 & \Sigma_{22}^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Sigma_{nn}^{-1}\end{bmatrix}.$$

Substituting these into ℓ results in

$$\ell(\mu, \Sigma) = \sum_{i=1}^{m}\left[-\frac{n}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{n}\log\Sigma_{jj} - \frac{1}{2}\sum_{j=1}^{n}(x_j^{(i)}-\mu_j)^2\Sigma_{jj}^{-1}\right].$$

Taking the derivative with respect to Σ_{jj}, we find

$$\frac{\partial\ell}{\partial\Sigma_{jj}} = \sum_{i=1}^{m}\left[-\frac{1}{2}\Sigma_{jj}^{-1} + \frac{1}{2}(x_j^{(i)}-\mu_j)^2\Sigma_{jj}^{-2}\right].$$

Setting to zero and solving for Σ_{jj}, we get

$$\Sigma_{jj} = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)}-\mu_j)^2,$$

as presented in the Lecture Notes. □

9.3 Page 2: The MLE for restriction of Σ being diagonal with equal entries

Proof. If Σ is diagonal with equal entries, i.e.,

$$\Sigma = \begin{bmatrix}\sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^2\end{bmatrix} = \sigma^2 I,$$

we have

$$|\Sigma| = \prod_{j=1}^{n} \sigma^2 = (\sigma^2)^n,$$

and

$$\Sigma^{-1} = \sigma^{-2} I = (\sigma^2)^{-1} I.$$

Substituting these into ℓ results in

$$\ell(\mu, \Sigma) = \sum_{i=1}^{m} \left[-\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\sigma^2 - \frac{1}{2}(\sigma^2)^{-1} \cdot \sum_{j=1}^{n}(x_j^{(i)} - \mu_j)^2 \right].$$

Taking the derivative with respect to σ^2, we find

$$\frac{\partial \ell}{\partial \sigma^2} = \sum_{i=1}^{m} \left[-\frac{n}{2}(\sigma^2)^{-1} + \frac{1}{2}(\sigma^2)^{-2} \sum_{j=1}^{n}(x_j^{(i)} - \mu_j)^2 \right].$$

Setting to zero and solving for σ^2, we get

$$\sigma^2 = \frac{1}{mn} \sum_{j=1}^{n} \sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2,$$

as presented in the Lecture Notes. □

9.4 Page 7: The derivation of EM for factor analysis

In the E-step, we set

$$Q_i(z^{(i)}) = \frac{1}{(2\pi)^{k/2}|\Sigma_{z^{(i)}|x^{(i)}}|^{1/2}} \exp\left(-\frac{1}{2}(z^{(i)} - \mu_{z^{(i)}|x^{(i)}})^T \Sigma_{z^{(i)}|x^{(i)}}^{-1} (z^{(i)} - \mu_{z^{(i)}|x^{(i)}}) \right),$$

where

$$\begin{aligned} \mu_{z^{(i)}|x^{(i)}} &= \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(x^{(i)} - \mu), \\ \Sigma_{z^{(i)}|x^{(i)}} &= I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda. \end{aligned}$$

In the M-step, we need to maximize the following quantity

$$\begin{aligned} \mathcal{L}(\mu, \Lambda, \Psi) &= \sum_{i=1}^{m} \mathrm{E}\left[\log \frac{1}{(2\pi)^{n/2}|\Psi|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right) \right] \\ &= \sum_{i=1}^{m} \mathrm{E}\left[-\frac{1}{2}\log|\Psi| - \frac{n}{2}\log(2\pi) - \frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right]. \end{aligned}$$

Note that the expectation is with respect to $z^{(i)}$ drawn from Q_i. Therefore, $\mathrm{E}[\cdot]$ is short for $\mathrm{E}_{z^{(i)} \sim Q_i}[\cdot]$.

9.4.1 The M-step update for Λ

Taking the derivative with respect to Λ, we get

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \Lambda} &= \frac{\partial}{\partial \Lambda} \sum_{i=1}^{m} \mathrm{E}\left[-\frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)}) \right] \\ &= \frac{\partial}{\partial \Lambda} \sum_{i=1}^{m} \mathrm{E}\left[-\frac{1}{2}(z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} + (z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu) \right] \\ &= \sum_{i=1}^{m} \mathrm{E}\left[-\frac{1}{2}\frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} + \frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu) \right]. \end{aligned}$$

First part: $\frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)}$ We compute the differential, i.e.,

$$\begin{aligned}
\mathrm{d}(z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} &= (\mathrm{d}(z^{(i)})^T \Lambda^T) \cdot \Psi^{-1} \Lambda z^{(i)} + (z^{(i)})^T \Lambda^T \cdot (\mathrm{d}\Psi^{-1} \Lambda z^{(i)}) \\
&= (z^{(i)})^T (\mathrm{d}\Lambda)^T \Psi^{-1} \Lambda z^{(i)} + (z^{(i)})^T \Lambda^T \Psi^{-1} (\mathrm{d}\Lambda) z^{(i)} \\
&= \operatorname{tr} (z^{(i)})^T (\mathrm{d}\Lambda)^T \Psi^{-1} \Lambda z^{(i)} + \operatorname{tr} (z^{(i)})^T \Lambda^T \Psi^{-1} (\mathrm{d}\Lambda) z^{(i)} \\
&= \operatorname{tr} (\mathrm{d}\Lambda)^T \Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T + \operatorname{tr} z^{(i)} (z^{(i)})^T \Lambda^T \Psi^{-1} (\mathrm{d}\Lambda) \\
&= \operatorname{tr} \left((\mathrm{d}\Lambda)^T \Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T\right)^T + \operatorname{tr} z^{(i)} (z^{(i)})^T \Lambda^T \Psi^{-1} (\mathrm{d}\Lambda) \\
&= 2 \operatorname{tr} z^{(i)} (z^{(i)})^T \Lambda^T \Psi^{-1} (\mathrm{d}\Lambda).
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} &= 2\left(z^{(i)} (z^{(i)})^T \Lambda^T \Psi^{-1}\right)^T \\
&= 2 \Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T.
\end{aligned}$$

Second part: $\frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu)$ We compute the differential, i.e.,

$$\begin{aligned}
\mathrm{d}(z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu) &= (z^{(i)})^T (\mathrm{d}\Lambda)^T \Psi^{-1} (x^{(i)} - \mu) \\
&= \operatorname{tr} (z^{(i)})^T (\mathrm{d}\Lambda)^T \Psi^{-1} (x^{(i)} - \mu) \\
&= \operatorname{tr} (\mathrm{d}\Lambda)^T \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T \\
&= \operatorname{tr} \left((\mathrm{d}\Lambda)^T \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T\right)^T \\
&= \operatorname{tr} z^{(i)} (x^{(i)} - \mu)^T \Psi^{-1} (\mathrm{d}\Lambda)
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\frac{\partial}{\partial \Lambda}(z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu) &= \left(z^{(i)} (x^{(i)} - \mu)^T \Psi^{-1}\right)^T \\
&= \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T.
\end{aligned}$$

Putting these into $\frac{\partial \mathcal{L}}{\partial \Lambda}$, we finally have

$$\frac{\partial \mathcal{L}}{\partial \Lambda} = \sum_{i=1}^{m} \mathrm{E}\left[-\Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T + \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T\right],$$

as presented in the Lecture Notes.

Setting this to zero and simplifying, we get

$$\Lambda \sum_{i=1}^{m} \mathrm{E}\left[z^{(i)} (z^{(i)})^T\right] = \sum_{i=1}^{m} (x^{(i)} - \mu) \mathrm{E}\left[(z^{(i)})^T\right].$$

Hence, solving for Λ, we obtain

$$\Lambda = \left(\sum_{i=1}^{m} (x^{(i)} - \mu) \mathrm{E}\left[(z^{(i)})^T\right]\right) \left(\sum_{i=1}^{m} \mathrm{E}\left[z^{(i)} (z^{(i)})^T\right]\right)^{-1}.$$

From the definition of Q_i being Gaussian with mean $\mu_{z^{(i)}|x^{(i)}}$ and covariance $\Sigma_{z^{(i)}|x^{(i)}}$ (see E-step), we easily find

$$\begin{aligned}
\mathrm{E}\left[(z^{(i)})^T\right] &= \mu_{z^{(i)}|x^{(i)}}^T, \\
\mathrm{E}\left[z^{(i)} (z^{(i)})^T\right] &= \mu_{z^{(i)}|x^{(i)}} \mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}.
\end{aligned}$$

The later comes from the fact that, for a random variable Y, $\mathrm{Cov}(Y) = \mathrm{E}[YY^T] - \mathrm{E}[Y]\mathrm{E}[Y]^T$, and hence $\mathrm{E}[YY^T] = \mathrm{E}[Y]\mathrm{E}[Y]^T + \mathrm{Cov}(Y)$. Substituting this back into the equation for Λ, we get the M-step update for Λ

$$\Lambda = \left(\sum_{i=1}^{m}(x^{(i)} - \mu)\mu_{z^{(i)}|x^{(i)}}^T\right)\left(\sum_{i=1}^{m}\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}\right)^{-1},$$

as presented in the Lecture Notes.

9.4.2 The M-step update for μ

Taking the derivative with respect to μ, we get

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mu} &= \frac{\partial}{\partial \mu}\sum_{i=1}^{m}\mathrm{E}\left[-\frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T\Psi^{-1}(x^{(i)} - \mu - \Lambda z^{(i)})\right] \\ &= \sum_{i=1}^{m}\mathrm{E}\left[\Psi^{-1}(x^{(i)} - \mu - \Lambda z^{(i)})\right],\end{aligned}$$

where the facts

$$\frac{\partial x^T A x}{\partial x} = (A + A^T)x,$$

and $\Sigma = \Sigma^T$ are used.

Setting this to zero and solving for μ, we obtain

$$\mu = \frac{1}{m}\sum_{i=1}^{m}x^{(i)},$$

as presented in the Lecture Notes.

9.4.3 The M-step update for Ψ

We first rewrite the quantity we need to maximize as

$$\mathcal{L}(\mu, \Lambda, \Psi^{-1}) = \sum_{i=1}^{m}\mathrm{E}\left[\frac{1}{2}\log|\Psi^{-1}| - \frac{n}{2}\log(2\pi) - \frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T\Psi^{-1}(x^{(i)} - \mu - \Lambda z^{(i)})\right].$$

Taking the derivative with respect to Ψ^{-1}, we get

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \Psi^{-1}} &= \frac{\partial}{\partial \Psi^{-1}} \sum_{i=1}^{m} \mathrm{E}\left[\frac{1}{2}\log|\Psi^{-1}| - \frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)})\right] \\
&= \sum_{i=1}^{m} \mathrm{E}\left[\frac{1}{2}\frac{1}{|\Psi^{-1}|}|\Psi^{-1}|\Psi - \frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})(x^{(i)} - \mu - \Lambda z^{(i)})^T\right] \\
&= \sum_{i=1}^{m} \mathrm{E}\left[\frac{1}{2}\Psi - \frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})(x^{(i)} - \mu - \Lambda z^{(i)})^T\right] \\
&= \frac{1}{2}\sum_{i=1}^{m} \mathrm{E}\Big[\Psi - (x^{(i)} - \mu)(x^{(i)} - \mu)^T + (x^{(i)} - \mu)(z^{(i)})^T \Lambda^T \\
&\qquad\qquad + \Lambda z^{(i)}(x^{(i)} - \mu)^T - \Lambda z^{(i)}(z^{(i)})^T \Lambda^T\Big] \\
&= \frac{1}{2}\sum_{i=1}^{m} \Big(\Psi - (x^{(i)} - \mu)(x^{(i)} - \mu)^T + (x^{(i)} - \mu)\mathrm{E}[(z^{(i)})^T]\Lambda^T \\
&\qquad\qquad + \Lambda \mathrm{E}[z^{(i)}](x^{(i)} - \mu)^T - \Lambda \mathrm{E}[z^{(i)}(z^{(i)})^T]\Lambda^T\Big) \\
&= \frac{1}{2}\sum_{i=1}^{m} \Big(\Psi - (x^{(i)} - \mu)(x^{(i)} - \mu)^T + (x^{(i)} - \mu)\mu_{z^{(i)}|x^{(i)}}^T \Lambda^T \\
&\qquad\qquad + \Lambda \mu_{z^{(i)}|x^{(i)}}(x^{(i)} - \mu)^T - \Lambda(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}})\Lambda^T\Big),
\end{aligned}$$

where we use the following facts

$$\frac{\partial x^T A x}{\partial A} = x x^T,$$

$$\frac{\partial |A|}{\partial A} = |A|(A^{-1})^T,$$

and $\Psi^T = \Psi$ because it is symmetric.

Setting this to zero and solving for Ψ, we obtain

$$\begin{aligned}
\Psi = \mathrm{diag}\Bigg\{\frac{1}{m}\sum_{i=1}^{m} \Big(&(x^{(i)} - \mu)(x^{(i)} - \mu)^T - (x^{(i)} - \mu)\mu_{z^{(i)}|x^{(i)}}^T \Lambda^T \\
&- \Lambda \mu_{z^{(i)}|x^{(i)}}(x^{(i)} - \mu)^T + \Lambda(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}})\Lambda^T\Big)\Bigg\},
\end{aligned}$$

where the diag$\{\bullet\}$ operator sets all of the nondiagonal elements of a matrix to zero.

Notice that this is slightly different from the equation presented in the Lecture Notes. There the mean μ is not subtracted from $x^{(i)}$. However, we can obtain the same expression if we pre-process the data before running EM. Specifically, recall that the M-step update for μ is

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}.$$

Notice that μ does not change during the iteration and can be calculated just once. Therefore, we can zero out the mean from the data before running EM. In that case, μ will always be 0 and the M-step update for Ψ is simplified as

$$\Psi = \mathrm{diag}\Bigg\{\frac{1}{m}\sum_{i=1}^{m} \Big(x^{(i)}(x^{(i)})^T - x^{(i)}\mu_{z^{(i)}|x^{(i)}}^T \Lambda^T - \Lambda \mu_{z^{(i)}|x^{(i)}}(x^{(i)})^T + \Lambda(\mu_{z^{(i)}|x^{(i)}}\mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}})\Lambda^T\Big)\Bigg\},$$

which is the same as presented in the Lecture Notes. Accordingly, in the computation for $\mu_{z^{(i)}|x^{(i)}}$ and $\Sigma_{z^{(i)}|x^{(i)}}$, and the M-step update for Λ, μ is therefore also 0.[7]

10 Notes on: Lecture notes 10

10.1 Page 5: The derivation of PCA

The optimization problem for PCA is formalized as follow

$$\begin{aligned} \max_u \quad & u^T \Sigma u, \\ \text{s.t.} \quad & u^T u = 1, \end{aligned}$$

where

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} (x^{(i)})^T,$$

which is just the empirical covariance matrix of the data (assuming it has zero mean; If not, **mean normalization** should be performed beforehand). We then optimize using the Lagrangian

$$\mathcal{L}(u) = u^T \Sigma u - \lambda (u^T u - 1),$$

with $-\lambda$ being the Lagrange multiplier. Taking derivative with respect to u, we get

$$\frac{\partial \mathcal{L}}{\partial u} = 2(\Sigma u - \lambda u).$$

Setting this to zero and simplifying, we obtain

$$\Sigma u = \lambda u.$$

This indicates that λ is the principal (i.e., the largest) eigenvalue of Σ with u being the corresponding eigenvector, i.e., principal eigenvector. It can be further shown that the maximum value of $u^T \Sigma u$ is

$$\max u^T \Sigma u = u^T \lambda u = \lambda,$$

i.e., the principal eigenvalue. More generally, the top k eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$ ($k \leq n$) correspond to the top k maximum values of $u^T \Sigma u$.

10.2 Page 5: The orthogonal property of eigenvectors of Σ

Footnote 2 says "Because Σ is symmetric, the u_i's will (or always can be chosen to be) orthogonal to each other." Herein, we prove this statement.

Proof. Let A be an $n \times n$ symmetric matrix, $A^T = A$. Let $\lambda_i \neq \lambda_j$ be two distinct eigenvalues. From the definitions we have

$$\begin{aligned} Av_i &= \lambda_i v_i, \\ Av_j &= \lambda_j v_j. \end{aligned}$$

Multiplying the first equation on the left by v_j^T and the transpose of the second equation on the right by v_i, we obtain

$$v_j^T A v_i - v_j^T A v_i = 0 = (\lambda_i - \lambda_j) v_j^T v_i.$$

Thus, $v_j^T v_i = 0$. Furthermore, it can be shown that even if the eigenvalues are not distinct, we can still find a set of orthogonal eigenvectors. (This statement is from the book "Pattern Recognition"; though I can not prove it yet.) $\square$

[7] This is equivalently to say that the parameters for our model are reduced to just Λ and Ψ, since μ is always 0.

10.3 Computation of eigenvectors of Σ

Denote X as the conventional $m \times n$ design matrix. Then, the empirical covariance matrix Σ can be computed as

$$\Sigma = \frac{1}{m} X^T X,$$

which is a $n \times n$ matrix. This is computed in Matlab/Octave as

```
Sigma = (1/m) * (X') * X;  % compute the empirical covariance matrix
```

The key to perform PCA is to compute the eigenvectors (and the eigenvalues as well) of the covariance matrix Σ. We herein discuss three ways to do that.

10.3.1 Use `eig` routine

The natural way to compute the eigenvectors is through the `eig` routine:

```
% compute the eigenvectors U and eigenvalues S of Sigma
[U, S] = eig(Sigma);
```

where `U` is a matrix of eigenvectors (one eigenvector per column) and `S` is a diagonal matrix of eigenvalues. However, to perform PCA with the top K principal components, we further need to sort matrix `U` in the order from top to bottom eigenvector and matrix `S` accordingly. Taking this issue into account, we can perform PCA with the `eig` routine using the following piece of codes:

```
% compute the eigenvectors U and eigenvalues S of Sigma
[U, S] = eig(Sigma);
[s, ind] = sort(diag(S), 'descend');  % sort the eigenvalues
S = diag(s);  % re-arrange the eigenvalue matrix S
U = U(:,ind);  % re-arrange the eigenvector matrix U
```

10.3.2 Use `svd` routine

As an alternative way, we can use singular value decomposition (SVD) through the `svd` routine. To that end, we need some preliminary results of linear algebra about SVD. (It may also serve as a refresher.)

Unitary matrix If a $m \times m$ matrix A satisfies $A^T A = AA^T = I$, then A is a unitary matrix. For the definition, we can easily tell that $A^{-1} = A^T$ if A is a unitary matrix.

Singular value Suppose $A \in \mathbb{R}^{m \times n}$ and the eigenvalues of $A^T A$ are

$$\lambda_1 \geq \lambda_2 \cdots \lambda_r > \lambda_{r+1} = \cdots = \lambda_n = 0,$$

where r is the rank of $A^T A$. (Recall that in Note 1.7, we have $\text{rank}(AA^T) = \text{rank}A^T = \text{rank}A = \text{rank}(A^T A)$.) Then $\sigma_i = \sqrt{\lambda_i}$ $(i = 1, 2, \ldots, n)$ is the **singular value** of A.

$A^T A$ and AA^T have the same non-zero eigenvalues

Proof. Suppose $A^T Ax = \lambda x$, where $\lambda \neq 0$ and $x \neq 0$. For $y = Ax$, we then have

$$AA^T y = AA^T(Ax) = A(\lambda x) = \lambda y,$$

indicating that λ is also eigenvalue of AA^T. It can be shown that any non-zero eigenvalue of AA^T is also eigenvalue of $A^T A$. $\square$

Singular Value Decomposition We assume that $A \in \mathbb{R}^{m\times n}$. Let

$$S = \left[\begin{array}{cc} \Lambda & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{array} \right] \in \mathbb{R}^{m\times n},$$

$$\Lambda = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r},$$

$$U = \left[\begin{array}{cccc} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{array} \right] \in \mathbb{R}^{m\times m},$$

$$V = \left[\begin{array}{cccc} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{array} \right] \in \mathbb{R}^{n\times n},$$

where u_i is the (unit-length and orthogonal) eigenvector of AA^T and v_i is the (unit-length and orthogonal) eigenvector of A^TA. The matrix A can then be decomposed as

$$A = USV^T,$$

This is the **Singular Value Decomposition** or abbreviated as **SVD**. Note that U and V are also unitary matrices.

Method 1 using `svd` From SVD, we know that U is the eigenvector matrix of AA^T. Using $n \times m$ matrix Y, i.e.,

$$Y = \frac{1}{\sqrt{m}} X^T,$$

we can express the covariance matrix Σ as

$$\Sigma = YY^T.$$

Therefore, decompose Y by SVD:

$$Y = USV^T,$$

then matrix U contains the eigenvectors of the matrix YY^T, i.e., the covariance matrix Σ.

Furthermore, the diagonal entries of the diagonal matrix S correspond to the singular values of Y, i.e., the squared root of the eigenvalues of YY^T. (Actually from the previous definition of singular value, this should be the squared root of the eigenvalues of Y^TY. However, from previous notes, we also note that Y^TY and YY^T have the same non-zero eigenvalues.) Thus, the eigenvalues of YY^T (i.e., covariance matrix Σ) can be computed through

$$S = S \circ S,$$

where we use "$\circ$" to denote the element-wise product operator (denoted ".*" in Matlab or Octave, and also called the Hadamard product), so that if $A = B \circ C$, then $A_{ij} = B_{ij} * C_{ij}$.

The code for this method is given as

```
Y = (1/sqrt(m)) * X';  % construct matrix Y (n x m)
% compute the eigenvectors U of Sigma = Y*Y^T and
% singular values S of Y
[U, S] = svd(Y);
% convert singular values of Y to eigenvalues of
% Sigma = Y*Y^T
S = S .* S;
```

Method 2 using `svd` The above method directly performs SVD on the training data, X (or effectively $Y = \frac{1}{\sqrt{m}} X^T$). It turns out that there exists another implementation by using SVD on the covariance matrix Σ. From the implement notes on PCA in UFLDL:

Next, PCA computes the eigenvectors of Σ. One could do this using the Matlab `eig` function. However, because Σ is a symmetric positive semi-definite matrix, it is more numerically reliable to do this using the `svd` function. Concretely, if you implement

```
[U,S,V] = svd(Sigma);
```

then the matrix `U` will contain the eigenvectors of `Sigma` (one eigenvector per column, sorted in order from top to bottom eigenvector), and the diagonal entries of the matrix `S` will contain the corresponding eigenvalues (also sorted in decreasing order). The matrix `V` will be equal to transpose of `U`, and can be safely ignored. (**Note: The `svd` function actually computes the singular vectors and singular values of a matrix, which for the special case of a symmetric positive semi-definite matrix—which is all that we're concerned with here—is equal to its eigenvectors and eigenvalues. A full discussion of singular vectors vs. eigenvectors is beyond the scope of these notes.**)

We herein explain why matrix `U` contains all the eigenvectors of `Sigma`. To show this, we only need to prove that a symmetric matrix A and its square AA (equivalent to AA^T by symmetry) have the same eigenvectors.

Proof. By the definition of the eigenvector, we have

$$Av = \lambda v.$$

Multiplying A on both side,

$$AAv = A\lambda v = \lambda^2 v.$$

This indicates that the eigenvector v of A is also an eigenvector of AA^T.

Note that the singular value of A is the squared root of the eigenvalue of AA^T. From the above equation, the singular value of A is thus given by $\sigma = \sqrt{\lambda^2} = |\lambda|$. If A is further positive semi-definite (like the covariance matrix Σ we're concerned with here), it can be shown that the eigenvalue $\lambda \geq 0$ (see Note 1.8). Therefore, the singular value of a symmetric positive semi-definite matrix A, given by $\sigma = |\lambda| = \lambda$, is exactly the eigenvalue of A. $\square$

Recall that the covariance matrix Σ is symmetric positive semi-definite matrix (see Note 1.8). Therefore, in code

```
[U,S,V] = svd(Sigma);
```

matrix `U` contains the eigenvectors of `Sigma` and also `S` contain the corresponding eigenvalues of `Simga`.

One more note. From the implement notes on PCA in UFLDL: "The matrix `V` will be equal to transpose of `U`, and can be safely ignored." This seems wrong. Note that from the definition of SVD, U contains the eigenvectors of AA^T and V the eigenvectors of A^TA. Since A or `Sigma` exactly is symmetric, $AA^T = A^TA = AA$. Thus, U and V should be equal instead. This could be verified through the following codes:

```
X = magic(5);
Sigma = (1/5) * (X' * X);
[U, S, V] = svd(Sigma);
norm(U(:)-V(:))
```

10.4 Page 5: Projecting and recovering the data

See `http://ufldl.stanford.edu/wiki/index.php/PCA` for details.

10.5 Page 6: The first k principal components maximize $\sum_i ||y^{(i)}||_2^2$

To be finished.

10.6 Collections

Following are some materials related to PCA (and SVD)

1. A Tutorial on Principal Component Analysis. `http://www.snl.salk.edu/~shlens/pca.pdf`
2. Principal Component Analysis. `http://www.stat.columbia.edu/~fwood/Teaching/w4315/Spring2010/PCA/slides.pdf`
3. What is the intuitive relationship between SVD and PCA. `http://math.stackexchange.com/questions/3869/what-is-the-intuitive-relationship-between-svd-and-pca`
4. We Recommend a Singular Value Decomposition. `http://www.ams.org/samplings/feature-column/fcarc-svd`

11 Miscellaneous

11.1 Confusion Matrix

- **true positive (TP)** eqv. with hit
- **true negative (TN)** eqv. with correct rejection
- **false positive (FP)** eqv. with false alarm, Type I error
- **false negative (FN)** eqv. with miss, Type II error
- **sensitivity or true positive rate (TPR)** eqv. with hit rate, recall

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

- **false positive rate (FPR)** eqv. with fall-out

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

- **accuracy (ACC)**

$$ACC = \frac{TP + TN}{P + N}$$

- **specificity (SPC) or True Negative Rate**

$$SPC = \frac{TN}{N} = \frac{TN}{FP + TN} = 1 - FPR$$

- **positive predictive value (PPV)** eqv. with precision

$$PPV = \frac{TP}{P'} = \frac{TP}{TP + FP}$$

- **negative predictive value (NPV)**

$$NPV = \frac{TN}{N'} = \frac{TN}{TN + FN}$$

- **false discovery rate (FDR)**

$$FDR = \frac{FP}{P'} = \frac{FP}{FP + TP}$$

- **Matthews correlation coefficient (MCC)**

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{PNP'N'}}$$

- **F1 score** is the harmonic mean of precision and recall

$$F1 = \frac{2TP}{P + P'} = \frac{2TP}{2TP + FP + FN}$$

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] A. P. Dawid. Conditional independence in statistical theory. *J. Roy. Statist. Soc. Ser. B*, 41(1):1–31, 1979.

[3] Jan Magnus and Heinz Neudecker. *Matrix Differential Calculus With Applications In Statistics And Econometrics*. John Wiley & Sons, third edition, Jan. 16 2007.