CIS 520, Machine Learning, Fall 2018: Assignment 5

Wentao He

Collaborator: N/A

1 Multiclass Boosting

1. Proof:

$$\begin{aligned} D_{T+1}(i) &= \frac{D_t(i)e^{-\alpha_t \times \hat{h}_{t,y_i}(x_i)}}{Z_T} \\ &= \frac{D_{t-1}(i)e^{-\alpha_{t-1} \times \hat{h}_{t-1,y_i}(x_i) - \alpha_t \times \hat{h}_{T,y_i}(x_i)}}{Z_{T-1}Z_T} \\ &= \frac{D_1(i)e^{\sum_{t=1}^{T} -\alpha_t \times \hat{h}_{t,y_i}(x_i)}}{\prod_{t=1}^{T} Z_t} \\ &= \frac{\frac{1}{m}e^{-F_{T,y_i}(x_i)}}{\prod_{t=1}^{T} Z_t} \end{aligned}$$

2. Proof:

Let us first consider the case $H(x_i) = y_i$. Then both $\mathbf{1}(H(x_i) \neq y_i) = 0$ and the inequality trivially hold. Then let us consider the case $H(x_i) \neq y_i$. Then we should be able to clain that $F_{T,y_i}(x_i) < 0$ holds true, because if $F_{T,y_i}(x_i) > 0$, then we will get $F_{T,y_i}(x_i) > F_{T,k}(x_i)$ for all $k \neq y_i$. Otherwise we will get$\sum_{k=1}^{K} F_{T,k}(x_i) > -(K-2)\sum_{t=1}^{T} \alpha_t$, which will then lead to $H(x_i) = y_i$ and contradict our assumption in this particular problem. The inequality holds because both sides of the inequality will be 1, $\Rightarrow 1 \leq 1$ holds. Therefore in conclusion, $\mathbf{1}(H(x_i) \neq y_i) \leq \mathbf{1}(F_{T,y_i}(x_i) \leq 0)$.

3. Proof:

$$
\begin{aligned}
er_S[H] &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}(H(x_1)\neq y_1) \\
&\leq \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}(F_{T,y_i}(x_i)<0) \\
&\leq \frac{1}{m}\sum_{i=1}^{m}e^{F_{T,y_i}(x_i)} \\
\text{since } Z_t &= \sum_{j=1}^{m}D_t(j)e^{-\alpha_t\times h_{t,y_i}(x_i)} \text{ and } \sum_i D_{t+1}(i)=1 \\
er_S[H] &\leq \frac{1}{m}\sum_{i=1}^{m}e^{F_{T,y_i}(x_i)} = (\prod_{t=1}^{T}Z_t)\times(\sum_i D_{t+1}(i)) = \prod_{t=1}^{T}Z_t
\end{aligned}
$$

4. Proof:

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{m} D_t(j) e^{-\alpha_t \times \hat{h}_{t,y_i}(x_i)} \\
&= \sum_{i=1}^{m} D_t(i) e^{-\alpha_t} \mathbf{1}(H(x_i) = y_i) + \sum_{i=1}^{m} D_t(i) e^{\alpha_t} \mathbf{1}(H(x_i) \neq y_i) \\
&= \sum_{i=1}^{m} D_t(i) e^{-\alpha_t} (1 - \mathbf{1}(H(x_i) \neq y_i)) + \sum_{i=1}^{m} D_t(i) e^{\alpha_t} \mathbf{1}(H(x_i) \neq y_i) \\
&= (\sum_{i=1}^{m} D_t(i) - \sum_{i=1}^{m} D_t(i) \mathbf{1}(H(x_i) \neq y_i) e^{-\alpha_t} + \sum_{i=1}^{m} D_t(i) e^{\alpha_t} \mathbf{1}(H(x_i) \neq y_i) \\
&= (1 - er_t) e^{-\alpha_t} + er_t e^{\alpha_t} \\
\text{since } e^{\alpha_t} &= e^{\frac{1}{2} \ln \frac{1-er_t}{er_t}} = \sqrt{\frac{1-er_t}{er_t}} \\
\text{so } Z_t &= 2 \times \sqrt{er_t(1-er_t)}
\end{aligned}
$$

5. Proof: Since $er_t \leq \frac{1}{2} - \gamma$ for all t, we can choose $max(er_t) = \frac{1}{2}$ to prove the statement.

$$\begin{aligned} er_S[H] &\leq \prod_{t=1}^{T} Z_t \\ &= \prod_{t=1}^{T} 2 \times \sqrt{er_t(1-er_t)} \\ &= 2^T \sqrt{(\frac{1}{2}-\gamma)(\frac{1}{2}+\gamma)}^T \\ &= (1-4\gamma)^{\frac{1}{2}T} \\ &\leq e^{-2\gamma^2 T} \end{aligned}$$

2 Perceptron vs. Winnow

1. For perceptron:
   $||u||_2 = \sqrt{k}$, $R_2 = \sqrt{d}$
   Number of mistakes $\leq \dfrac{R_2^2 \cdot ||u||_2^2}{\gamma^2} = \dfrac{kd}{\gamma^2}$
   For Winnow:
   $||u||_1 = k$, $R_\infty = 1$
   Number of mistakes $\leq \dfrac{R_\infty^2 \cdot ||u||_1^2}{\gamma^2} \ln(d) = \dfrac{2d^2}{\gamma^2 \ln(d)}$
   Therefore Winnow is a better choice.

2. Similar to the previous problem, for perceptron:
   $||u||_2 \leq 2\sqrt{d}$, $R_2 = \sqrt{k}$
   Number of mistakes $\leq \dfrac{4kd}{\gamma^2}$
   For Winnow:
   $||u||_1 = d$, $R_\infty = 1$
   Number of mistakes $\leq \dfrac{2d^2}{\gamma^2 \ln(d)}$
   Therefore Perceptron is a better choice.

3. No. Winnow is maintaining a non-negative weight vector so it will have a non-negative dot product with all feature vectors in this particular case. Therefore Winnow will predict every label as positive, namely > 0.

3 Principal Component Analysis

1. Resulting Image, please see Figure 1.

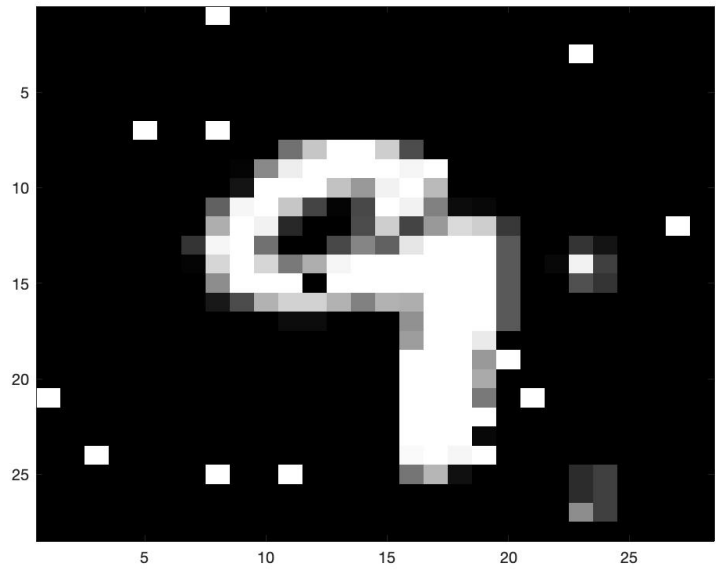

Figure 1: Resulting Image 1.

Code:

```
load('MNIST_train.mat')
last=X_train(12000,:);
last_ex=reshape(last,[28 28]);
imagesc(last_ex')
colormap('gray')
```

2. Resulting Images, please see Figure 2, Figure 3 and Figure 4.

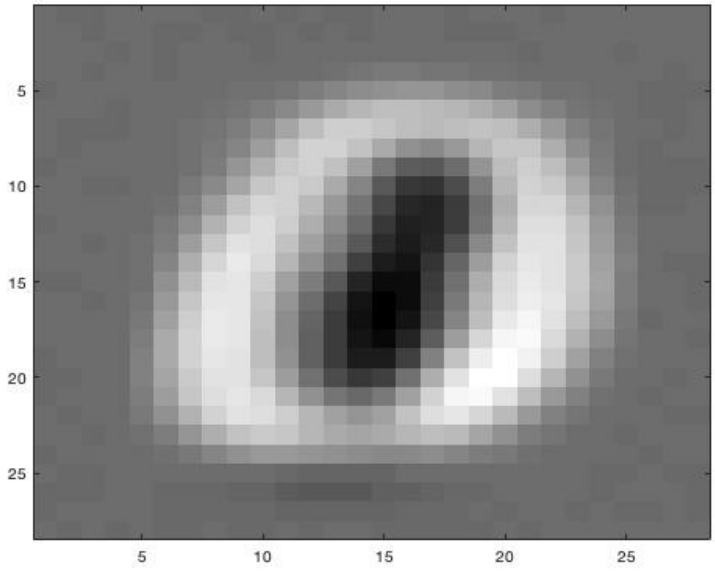

Figure 2: Resulting Image 1.

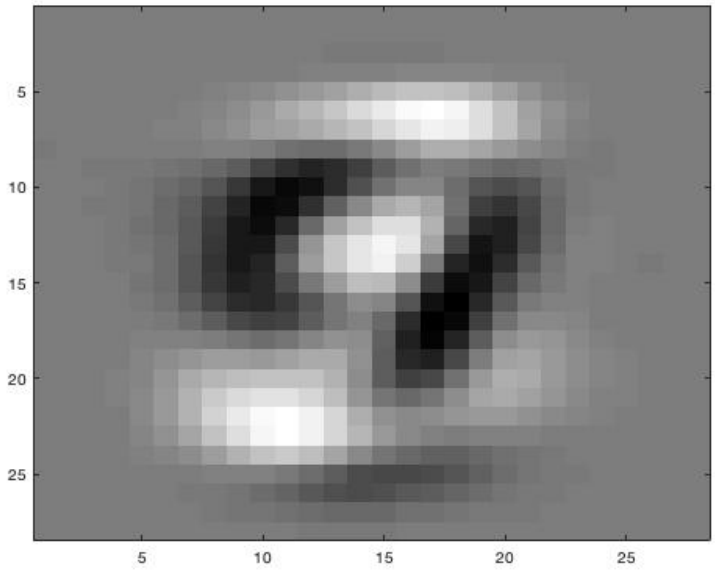

Figure 3: Resulting Image 2.

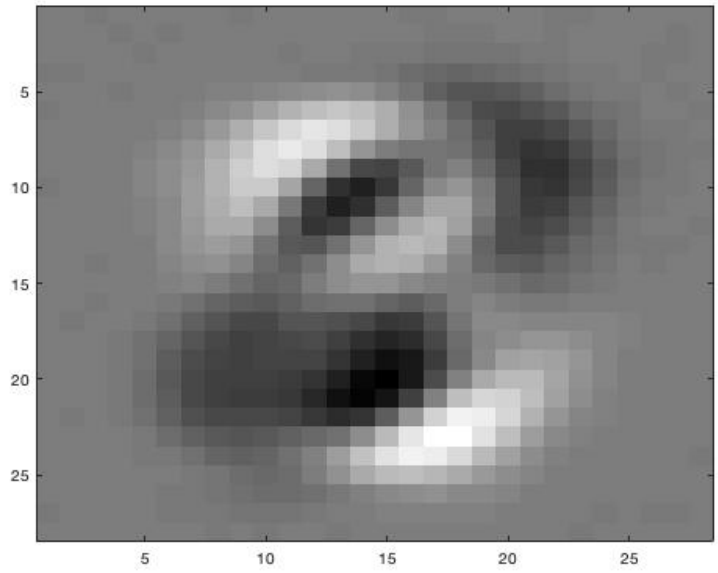

Figure 4: Resulting Image 3.

These images make sense because the first principal component is the best straight line you can fit to the data and it has more information than others. Namely, the first principal component vector will tell you the most about the point cloud after projecting all points onto the vector.

3. For 1^{st} and 2^{nd} PC, please see Figure 5. For 100^th and 101^st PC, please see Figure 6.

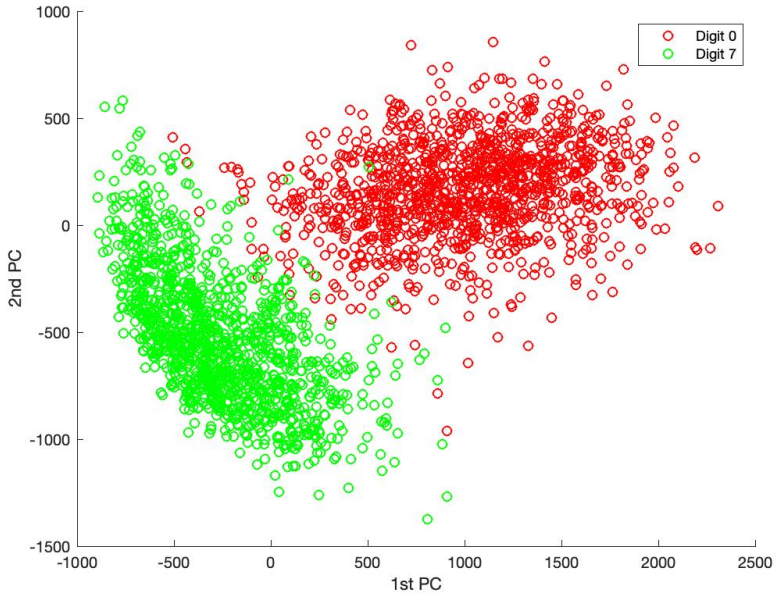

Figure 5: 1^{st} and 2^{nd} PC.

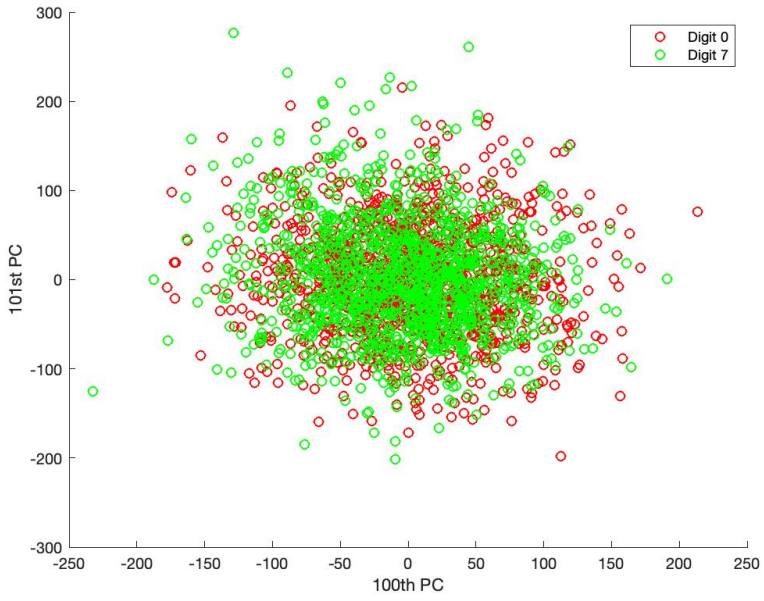

Figure 6: 100^{th} and 101^{st} PC.

What we can observe from those two graphs is that for the 1st and 2nd PC dimensions, images are separated pretty well simply because the first two PC dimensions contain more information regarding the characters of the images, namely the difference between the 0 and the 7 characters that are contained in the first two PC dimensions. Since the first principal component vector will tell you the most about

the point cloud after projecting all points onto the vector, in the second graph, 100th and 101st PC dimensions contain much more noice and less useful information regarding the differences between those two characters, so the graph is not very separated.

4. Please see Figure 7.

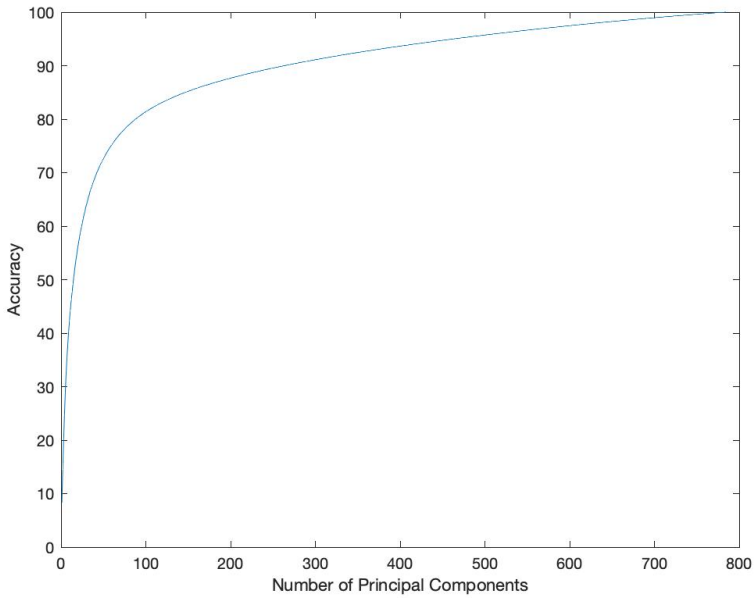

Figure 7: Fractional reconstruction accuracy as a function of the number of principal components.

The number of principal components needed to achieve each of 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10% reconstruction accuracy:

	10%	20%	30%	40%	50%	60%	70%	80%	90%
# of principal components	2	4	6	9	15	25	43	88	264

5. For 500^{th} examples, please see Figure 8. For 6000^{th} examples, please see Figure 9. For 10000^{th} examples, please see Figure 10.

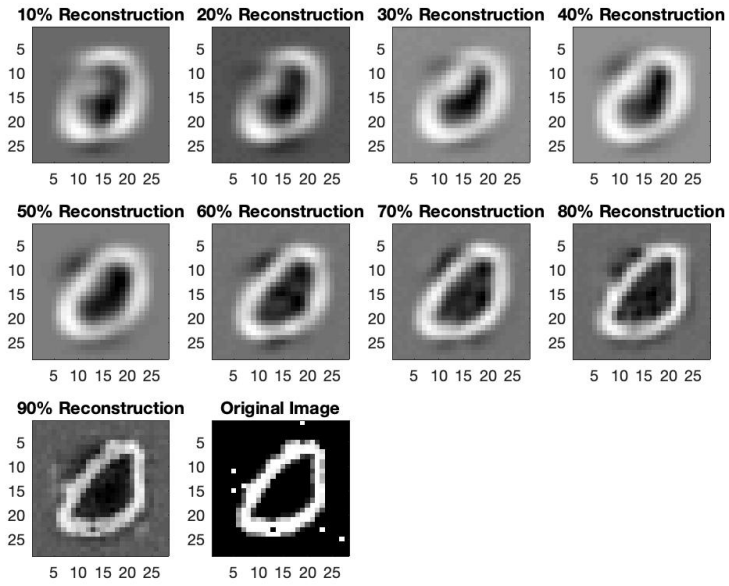

Figure 8: 500^{th} examples

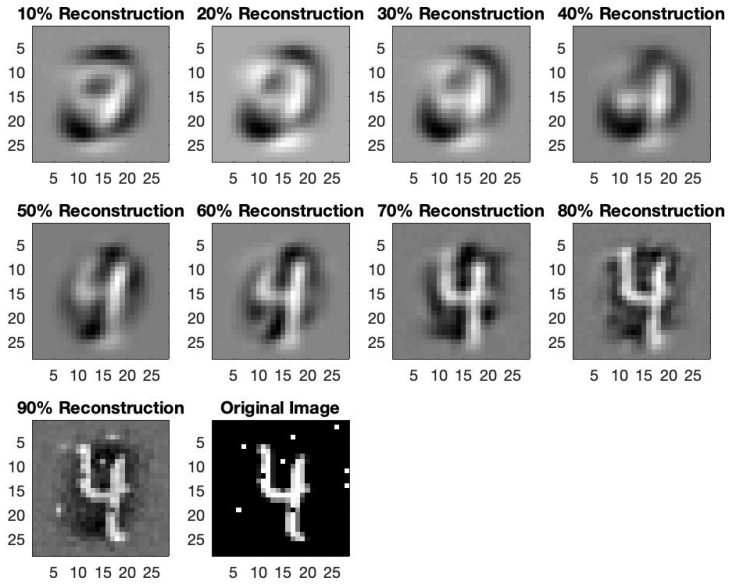

Figure 9: 6000^{th} examples

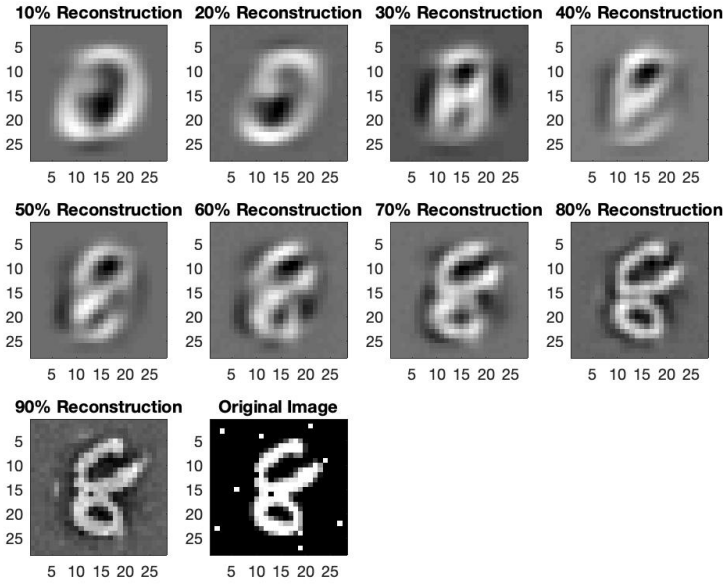

Figure 10: 10000^{th} examples

6. For reconstruction accuracy, please see Figure 11.

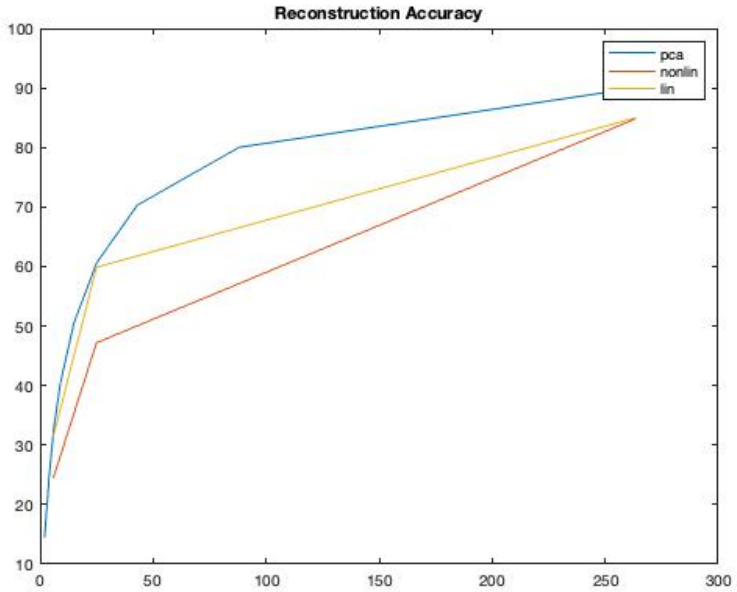

Figure 11: 500^{th} examples

From the graph we can tell that PCA and autoEncoder with linear transfer functions. Both PCA and the autoEncoder works similarly. PCA is restricted to a linear map, while autoEncoders can have nonlinear enoder/decoders. A single layer autoEncoder with linear transfer function is nearly equivalent to PCA, which means that when autoEncoder has 1 hidder layer, they will have similar reconstruction accuracy, which will be higher than nonlinear function.

7. It can dependent on the number of features(neurons in input layer). Higher number of hidden layers increase order of weights and it helps to make a higher order decision boundary. Therefore increasing the number of hidden layers can decrease the reconstruction error if the autoEncoder is trained properly. As we increase the number of hidden layers then accuracy can be obtained up to great extent but neural network became complex then previous. Increasing the number of hidden layers will cause your network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data. There is no question to go for more and more hidden layers as the Back-Propagation algorithm will be less effective. The test set error will shoot up when you use more hidden layers though you might have got near perfect accuracy for train sets.

4 Eigenvectors

1. Since X is full rank, and $rank(X) = p$, $X^T X$ is not singular. Thus we are able to write $\hat{w} = (X^T X)^{-1} X^T y$. Since $X = U\Lambda V^T$, together with SVD, we are able to get $(X^T X)^{-1} X^T = (V\Lambda^T \Lambda V^T)^{-1} V \Lambda U^T = (V^T)^{-1} \Lambda^{-1} V^{-1} V \Lambda U^T$. Since $V^T V = I$, $\Lambda^T = \Lambda$, and $(V_P^T)^{-1} = V_P$ for thin SVD, we have $(X^T X)^{-1} X^T = V_P \Lambda_P^{-1} U_P^T$, which is the same form as X^+, and X^+ is the left inverse. In conclusion, we can say that X^+ with the form of $V_P \Lambda_P^{-1} U_P^T$ is the solution of the least square problem.

2. SVD of X shows that X^TX and XX^T share the same eigenvalues: $X^TX = VD^TDV^T = VD^2V^T$ and $XX^T = UDD^TU^T = UD^2U^T$. Thus, for the right singular vector v_i of X^TX, $X^TXv_i = \lambda_i v_i$. For the left singular vector u_i, $XX^Tu_i = \lambda_i u_i$. Also, both left and right singular vectors have a relation which is $X^Tu_i = \sqrt{\lambda_i}u_i$. In conclusion, v_i can be found by $v_i = \dfrac{X^Tu_i}{\sqrt{\lambda_i}}$.