HW 3

Garrison Neel
CSCE 636

October 30, 2020

I Answers to the non-Programming Part

1 SVD of matrix A with the following eigendecomposition

$$A = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{bmatrix}$$

This is equivalent to $A = Q\Lambda Q^T$, meaning $Q^{-1} = Q^T$. This means Q is orthogonal, and consists of the eigenvectors of A which are also orthogonal. Comparing the relationship $Aw = \lambda w$ used in eigendecomposition, and $Av_i = \sigma u_i$ used in SVD shows that eigenvectors of A are also valid singular vectors. Because singular values must be positive, we can use $\sigma = |\lambda|$ to find the singular values. The corresponding singular vectors are the eigenvectors of A since they are orthogonal, with $u = v$ for positive eigenvalues and $u = -v$ for negative eigenvalues. The singular values are also in descending value order, so the singular value decomposition of A is:

$$\begin{aligned} A &= \begin{bmatrix} u_{11} & -u_{13} & u_{12} \\ u_{21} & -u_{23} & u_{22} \\ u_{31} & -u_{33} & u_{32} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{13} & u_{23} & u_{33} \\ u_{12} & u_{22} & u_{32} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & u_{13} & u_{12} \\ u_{21} & u_{23} & u_{22} \\ u_{31} & u_{33} & u_{32} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ -u_{13} & -u_{23} & -u_{33} \\ u_{12} & u_{22} & u_{32} \end{bmatrix} \end{aligned}$$

2 Please describe how we can train a logistic regression classifier and perform testing using only the solver and data given.

To solve this problem, I assume the training data spans $\mathbb{R}^d$ where d is the dimensionality of the training data.

Since $k(x_i, x_j) = x_i^T x_j$, the kernel $k(x_i, x_j)$ for $i = 1, 2, ..., n$ and $j = 1, 2, ..., n$ can be represented as a matrix $K_1 = X_t X_t^T$ where X_t is a matrix whose rows are the training samples $\{x_i\}_{i=1}^n$. By this definition K_1 is a real symmetric positive semi-definite matrix with rank d, and therefore d nonzero singular values. K_1 is then diagonalizable to the form

$$\underset{n\times n}{K_1} = \underset{n\times d}{U} \times \underset{d\times d}{\Sigma} \times \underset{n\times d}{U^T}$$

through compact singular value decomposition, keeping only the nonzero singular values. Splitting Σ by taking the square root allows this to be rearranged as

$$\begin{aligned} K_1 &= U\Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}}U^T \\ &= U\Sigma^{\frac{1}{2}} \times \left(U\Sigma^{\frac{1}{2}}\right)^T \end{aligned}$$

Which is analogous to the original form using X_t. Because inner products do not have unique solutions, $U\Sigma^{\frac{1}{2}}$ and $X_t X_t^T$ are not necessarily equivalent, but are instead related by an orthogonal transformation $R_{d\times d}$.

$$X_t \times R = U\Sigma^{\frac{1}{2}}$$

Defining $X_t' = X_t R$ reveals that $U\Sigma^{\frac{1}{2}}$ is simply a rotation or reflection of the original data. The transformed dataset X_t' can be used in the primal form of logistic regression, and will produce equivalent results to X_t since the transformation preserves the nature of the data.

To find the testing dataset, $k(x_i, x_j)$ for $i = 1, 2, ..., n$ and $j = n+1, ..., n+m$ can be represented as the matrix $K_2 = X_t X_T^T$ where X_T is a matrix whose rows are the testing data, $\{x_i\}_{i=n+1}^{n+m}$. Testing data is only useful if we find it transformed by the same R, so we define $X_T' = X_T R$ and rearrange the equation for K_2 to find:

$$\begin{aligned} K_2 &= X_t X^T \\ &= X_t R R^T X^T \\ &= X_t' \times (X_T R)^T \\ &= U\Sigma^{\frac{1}{2}} \times (X_T')^T \end{aligned}$$

Because the columns of U are orthogonal, $U^T U = I_{d\times d}$. $\Sigma^{\frac{1}{2}}$ is diagonal and invertible. Left multiplying $\Sigma^{-\frac{1}{2}} U^T$ and taking the transpose gives

$$\begin{aligned} \Sigma^{-\frac{1}{2}} U^T U \Sigma^{\frac{1}{2}} \times (X_T')^T &= \Sigma^{-\frac{1}{2}} U^T K_2 \\ X_T' &= K_2^T U \Sigma^{-\frac{1}{2}T} \end{aligned}$$

So, functionally equivalent training and test data can both be recovered from the kernel data as

$$\begin{aligned} X_t' &= U\Sigma^{\frac{1}{2}} \\ X_T' &= K_2^T U \Sigma^{-\frac{1}{2}} \end{aligned}$$

Because this data is an orthogonal transformation of the original data, the inner product is preserved, meaning the length and relative angle between data points is preserved. This makes it functionally equivalent data for logistic regression. Optimal weights for this dataset will also be an orthogonal transformation of the optimal weights for the original data, meaning the l_2 norm regularization term is also unaffected.

3 Ky Fan Proof

Theorem. Let $H \in \mathbb{R}^{n\times n}$ be a symmetric matrix with the eigenvalues

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n,$$

and the corresponding eigenvectors $U = [u_1, ..., u_n]$. Then

$$\lambda_1 + ... + \lambda_k = \max_{A\in\mathbb{R}^{n\times k}: A^T A = I_k} \text{trace}\left(A^T H A\right)$$

And the optimal A^* is given by $A^* = [u_1, ..., u_k]Q$ with Q an arbitrary orthogonal matrix.
Proof. H is diagonalizable to

$$H = U\Sigma U^T$$

Substituting this diagonalization for H yields

$$A^T H A = A^T U \Sigma U^T A = (U^T A)^T \Sigma (U^T A)$$

Renaming $B = U^T A$ gives matrix $B \in \mathbb{R}^{n \times k}$ which is also orthonormal. The trace can be rewritten

$$\text{trace}(A^T H A) = \text{trace}(B^T \Sigma B) = \text{trace}(\Sigma B B^T) = \sum_{i=1}^{n} \lambda_i ||b_i||^2$$

Where b_i are the row vectors from B. Each element in B is $b_{i,j} = u_i^T a_j$ where u_i is i^{th} eigenvector and a_j is the j^{th} column vector of A. So, $||b_i||_2^2 = \sum_{j=1}^{k} (u_i^T a_j)^2$, and the trace can be rewritten as

$$\sum_{i=1}^{n} \lambda_i \sum_{j=1}^{k} (u_i^T a_j)^2$$

Because the sets $a_{j=1,..,k}$ and $u_{i=1,..,n}$ are both orthonormal,

$$\sum_{j=1}^{k} (u_i^T a_j)^2 \leq 1$$

This sum is maximized when a given u_i is in the subspace $S \subset \mathbb{R}^n$ for which $a_{j=1,..,k}$ is a basis. Summed over all i, this sum becomes

$$\sum_{i=1}^{n} \sum_{j=1}^{k} (u_i^T a_j)^2 = \sum_{j=1}^{k} \sum_{i=1}^{n} (u_i^T a_j)^2 = \sum_{j=1}^{k} ||a_j||^2 = k$$

Showing that the total is k at maximum. Because the first k eigenvalues are the largest, the trace is maximized if the first k eigenvectors are in S. Subsequent eigenvectors u_i for $i = k+1, ..., n$ will make then the sum

$$\sum_{j=1}^{k} (u_i^T a_j)^2 = 0, \text{ for } i > k$$

Because this is the case, the original trace sum can be written as:

$$\sum_{i=1}^{k} \lambda_i \sum_{j=1}^{k} (u_i^T a_j)^2 = \sum_{i=1}^{k} \lambda_i = \lambda_1 + ... + \lambda_k,$$

which is equivalent to the form above. The trace is maximized for $A* = [u_1, ..., u_k]Q$ with Q an arbitrary orthogonal matrix, because this means the first k eigenvectors $[u_1, ..., u_k]$ also form a basis for the subspace $S \subset \mathbb{R}^n$ for which $a_{j=1,..,k}$ is a basis.

II Programming Part

4 Kernel vs Neural Network

A kernel logistic regression (KLR) model was implemented. The kernel used is the radial basis function (RBF)

$$k(x_i, x_j) = \exp(-\frac{||x_i - x_j||_2^2}{2\sigma^2}) \tag{1}$$

A RBF network model, and a 2-layer feed-forward neural (FFN) network model were also implemented for comparison. The models have the following hyperparameters:

learning_rate	The training learning rate
max_epochs	Number of training epochs
batch_size	Size of training batches
hidden_dim	# centroids (RBF), or # neurons in 2nd layer (FFN)
σ	Variance used in kernel

(a) KLR

Training for the KLR method achieved 100% validation accuracy in under 30 epochs with a learning rate of $1e-3$. No change was observed when varying batch size, so it was fixed at the size of the dataset because training is quick. Sigma had a large effect on the accuracy of the model.

σ	Valid. Accuracy (%)
1	57.41
5	80.69
10	100.00
15	100.00
18	100.00
20	100.00
40	98.94

A value of sigma in the range 10-20 seems ideal. A final value of 18 was chosen. The final hyperparameters for the KLR were determined to be:

Parameter	Value
learning_rate	1×10^{-3}
max_epochs	30
batch_size	# of samples
σ	18

With these hyperparameters, the KLR model achieved a test accuracy of 99.13%.

(b) RBF

The RBF model uses k-means clustering to reduce the size of the kernel matrix generated. K-means clustering algorithm was implemented using the following process:

1. Select k random data points to initialize cluster centers
2. Calculate pairwise distance between cluster centers and all datapoints
3. Assign each datapoint to the nearest centroid
4. Recalculate cluster centroids
5. Repeat steps 2-4 until there is no change in the cluster assignment (centroids no longer move)

For training, learning rate of 0.001 was used, along with a batchsize of 64, and σ of 19. The following table shows the result of varying the number of clusters, hidden_dim.

hidden_dim	Valid. Accuracy (%)
1	57.41
2	99.21
4	99.74
8	99.74
12	100.00
16	96.56
32	100.00

A value of 12 was selected because it presented the highest validation accuracy, and smaller values are desirable since they require less computation in prediction. Final hyperparameters chosen for the RBF model are:

Parameter	Value
learning_rate	1×10^{-3}
max_epochs	30
batch_size	64
hidden_dim	12
σ	18

The RBF model achieved a final test accuracy of 99.1%.

(c) FFN

The FFN uses a 2nd hidden layer which enables a nonlinear decision boundary. For comparison, training hyperparameters were copied from the RBF model, and hidden_dim, the number of neurons in the second layer, was varied.

hidden_dim	Valid. Accuracy (%)
1	100.00
2	100.00
4	100.00
8	99.74
12	99.74
16	100.00
32	100.00

The FFN reaches 100% validation accuracy in very few training epochs, but investigation showed that further training did improve test accuracy. The final hyperparameters used with the FFN model were

Parameter	Value
learning_rate	1×10^{-3}
max_epochs	30
batch_size	64
hidden_dim	12

The FFN model achieved a final test accuracy of 98.92%.

(d) Comparison

Model	Test Accuracy (%)
KLR	99.1
RBF	99.1
FFN	98.92

The RBF model was able to achieve similar accuracy to the full kernel KLR model, with only 12 centroids. This is significantly less computationally expensive. KLR and RBF seem better suited to this simple classification task since they effectively use the training data as a database of similar digits. At first glance FFN didn't achieve the same accuracy, but with a test dataset of only 465 example digits, FFN only incorrectly classified 1 more than RBF and KLR. This difference can be accounted for by differences in weight initialization, and training rather than model effectiveness. FFN has far more trainable parameters than a similar RBF network (since in RBF, the centroids are found using k-means), so training difficulty is higher.

5 Principal Component Analysis vs Auto-Encoder

Principal component analysis (PCA) aims to find a basis for a dataset which maximizes the variance. The data can then be projected into this reduced space which guarantees maximum variance and minimal reconstruction error. This is helpful for training because lower dimensionality means smaller networks, and maximum variance makes training easier. Autoencoders (AEs) can be used for this same process. AEs are neural networks which aim to reduce the dimensionality of data, while minimizing the reconstruction error. Since they

can be trained iteratively, the computation time can be reduced compared to PCA. Another strength is complexity. PCA gives minimal reconstruction error for a linear transform of the data, while AEs can be augmented to form nonlinear dimensionality reduction, further minimizing error.

(e) Reconstruction error comparison

PCA and a single layer AE were implemented. For the given matrix, the reconstruction error (frobenius norm of difference) is summarized in the table below.

	Reduced Dimension		
Method	32	64	128
PCA	129.37	85.81	45.63
AE	129.89	86.23	46.43

The methods yield similar reconstruction errors. PCA gives lesser error than AE because PCA is explicitly computed where AE is effectively searching for the correct solution. The lower bound of error for AE is the error from PCA reconstruction.

(f) Relation between G and W

PCA results in a semi-orthogonal matrix G which projects data to the reduced dimension space with minimum reconstruction error. AE solves for the optimal W that minimizes reconstruction error. We know that solutions to PCA are not unique, since any orthogonal transformation of G still guarantees minimal reconstruction error. If we introduce some orthogonal transform R', such that $G' = GR'$, we can show that the error is unchanged from the original form $|X - GG^TX|$

$$\begin{aligned} \text{err} &= |X - G'G'^TX| \\ &= |X - GR'R'^TG^TX| \\ &= |X - GG^TX| \end{aligned}$$

So W is not necessarily attempting to converge to G, but rather some G' which is an orthogonal transformation of G. Because W is not a true solution, we can only find R, the linear transformation from G to W

$$\begin{aligned} GR &= W \\ G^TGR &= G^TW \\ R &= G^TW \end{aligned}$$

The nearest orthogonal transform can be found using SVD. If $\text{SVD}(R) = U\Sigma V^T$ then the nearest orthogonal matrix is $R' = UV^T$. Assuming that $GR' = G'$ is the nearest PCA solution to W, we can then compare how close AE came to solving PCA. For $p = d = 64$,

$$\begin{aligned} &\text{frob\_norm}(W, G') = 0.4226 \\ &\text{frob\_norm}(R, R') = 0.081 \\ &\text{frob\_norm}(A, G'G'^TA) = 85.82 \end{aligned}$$

The reconstruction error shows that G' does produce the same minimal reproduction error as G. The norm of the transformation matrices R and R' show how minimally R had to be altered to be valid. The distance between W and G shows just how close W is to an optimal solution. This is congruent with the result from the last section showing reconstruction error by AE was only slightly higher than PCA.

(g) Weights not shared

Perviously, weights were shared by the encoder and decoder halves of the AE. Decoupling these weights affects reconstruction error as shown in the table below.

	Reduced Dimension		
AE Type	32	64	128
Shared Weights	129.89	86.23	46.43
Not Shared	130.21	86.60	47.0

Decoupling weights appears to make the issue of training more difficult. Since both are still linear transforms, the absolute best reconstruction error doesn't improve. Effectively, the training difficulty has doubled while the upper bound on performance hasn't changed. It's unsurprising that the result is worse performance.

(h) Nonlinear AutoEncoder

To see the effect of nonlinear AEs, I made a multi-layer AE. The encoder consists of a fully connected (FC) layer with 512 neurons, ReLU activation, and another FC layer with hidden_dim neurons representing the reduced dimension state. The decoder is composed of a FC layer with 512 neurons again, ReLU activation and then output layer with n_features neurons. The AE was trained using the fixed hyperparameters in the provided code. This AE was able to improve performance beyond PCA, resulting in the following reconstruction error:

	Reduced Dimension		
Method	32	64	128
PCA	129.37	85.81	45.63
AE	129.89	86.23	46.43
Nonlinear AE	71.22	55.21	36.44

This AE showed significant improvement over PCA, and the single layer AE from before.