CS 337, Fall 2023
Autoencoders and Boosting

Scribes: Johannes Roser, Ayush Agarwal, Shriyank Tatawat,
Adyasha Patra, Aryan Mathe, Hitesh Kumar
Edited by: Saurabh Kumar

October 16, 2023

Recap

We can explain the PCA algorithm in the following steps:

- Mean-center the input data
- Find the covariance matrix of the new data
- Use the first k principal eigenvectors of the covariance matrix as vectors in the projection matrix

1 Autoencoder

Autoencoder: An autoencoder is a type of artificial neural network that is used to learn efficient codings of input data. It consists of two parts: an encoder and a decoder

- **Encoder** - The encoder compresses the input data into a lower-dimensional latent representation
- **Decoder** - The decoder then reconstructs the original input data from the latent representation

Figure 1 shows the overall architecture of autoencoders.
Typically, autoencoders are feed-forward networks. The bottleneck helps by stopping the encoder and decoder from learning the identity function.

If dimension of latent space is k and $k << d$, then autoencoders can also be used for dimension reduction. We can use the reconstruction error minimization for training the autoencoder parameters as shown in equation 1.

$$L(x, \tilde{x}) = \|x - \tilde{x}\|^2 \tag{1}$$

1.1 Linear Autoencoder

A linear autoencoder is a type of autoencoder neural network where both the encoder and decoder are composed of only linear transformations. Suppose the weight matrix associated with **encoder** is denoted as W_1 of dimensions $k \times d$ and that associated with **decoder** is denoted by W_2 of dimensions $d \times k$, then for

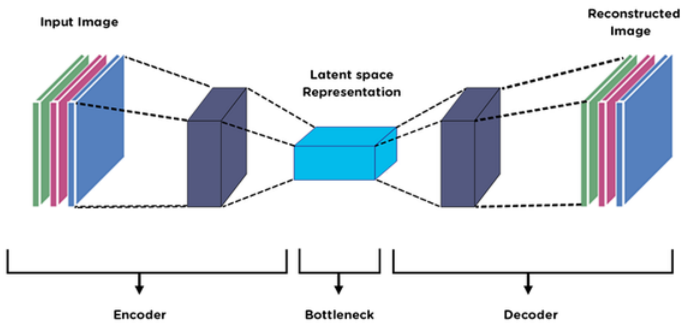

Figure 1: Autoenoder

a given input (x) the reconstructed output $(\tilde{x})$ can be deduced using the equation 2 and the reconstruction loss function will be given by the equation 3

$$\tilde{x} = W_2 \cdot W_1 \cdot x \tag{2}$$

$$L(x, \tilde{x}) = \|x - W_2 \cdot W_1 \cdot x\|^2 \tag{3}$$

On minimization of the above loss function, W_1 comes out to be U^T and W_2 comes out to be U where U is the projection matrix from PCA.

Commonly used Autoencoders

- **Voice Recognition**: Autoencoders can be used to extract useful acoustic features like speed, rythm and linguistic features like content into the bottleneck layer. Want to know more about it? Take CS753 next semester :)
- **Image segmentation**: Autoencoder can be used to remove noise, improve the quality of the image, and extract features that are useful for segmentation

1.2 Non Linear Autoencoder

When your autoencoder contains non-linear activation functions, then such type of auto-encoder lie in the family of non-linear auto-encoders. This is similar to how **Kernel PCA** extends traditional PCA by mapping data into a higher-dimensional space to capture non-linear relationships as shown in figure 2.

1.3 Kernel PCA

- Kernel PCA works in a similar way to that of a PCA, it first projects the data into a higher-dimensional space using a kernel function. This allows Kernel PCA to learn nonlinear relationships in the data.
- Once the data has been projected into the higher-dimensional space, Kernel PCA then finds the principal components of the data and projects it into a lower-dimensional space.
- Use of kernel function helps by allowing the possibility of using very-high-dimensional $\phi(x)$ if we never have to actually evaluate the data in that space, since the only thing we are interested in finding is $\phi(y)^T \cdot \phi(x)$ which is equal to $K(x, y)$

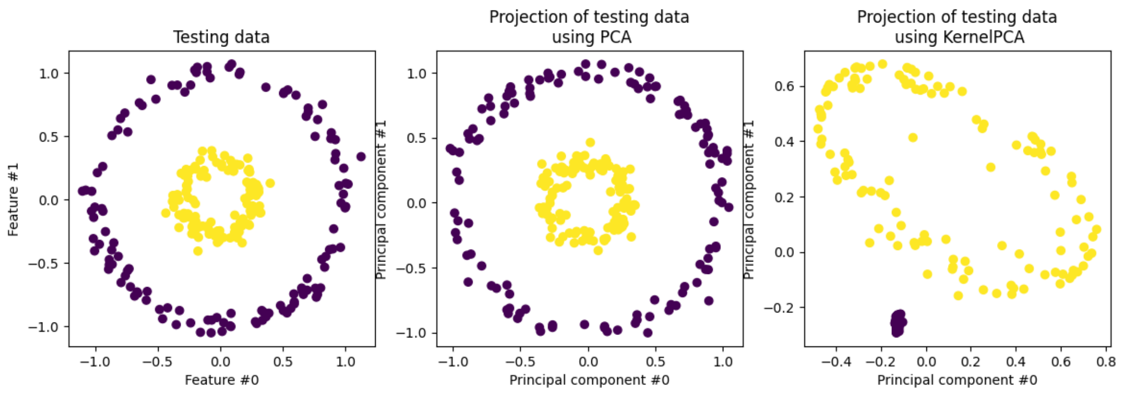

Figure 2: Kernel PCA visualization

2 Boosting

Boosting is a popular ensemble learning algorithm that aims to convert a weak ML model into a stronger version by optimizing the bias-variance trade-off for the training dataset.
The goal of the boosting algorithm is to combine several weak learners and build a strong learner in order to minimize training errors and improve performance. Although there are several kinds of Boosting algorithms such as **AdaBoost, Gradient Boost, XGBoost,** etc., but, we will focus only on AdaBoost in this lecture. Another algorithm of this family is the Random Forest algorithm that we briefly covered when we discussed decision trees.

Basic Intuition: Can we take a weak model[1] and make it stronger by focusing on the difficult examples, which are hard to classify.

2.1 Basic Algorithm

1. Initialize the **importance-weights** for each training example via a uniform distribution.
2. Train a weak learner on the training data.
3. Compute the error on the training set using the weak learner.
4. Increase the importance of those training examples that were misclassified.
5. Retrain the weak learner with importance-trained training examples.
6. repeat from **step 2-step 5** till convergence.

Examples for weak learners could include:

- Decision stumps
- Single perceptrons

One way to weight the learners is to scale the gradient descent function of the respective examples. The number of iterations the algorithm goes through is a hyperparameter.

[1] A weak model is defined as being only slightly more accurate than a random guess.

2.2 ADABOOST Algorithm

Adaboost is one of the most popular boosting algorithms. It was pioneered by Freund and Schapire in 1995. Given the training examples $\{x_i, y_i\}_{i=1}^{N}$, $y_i \epsilon \{-1; +1\}$, initalize all x_i, y_i instances with a weight defined as:

$$D(i) = \frac{1}{N} \qquad \forall i$$

For each iteration, $t = 1, \ldots, T$ (here T is the hyperparameter):

First, train the weak classifier $h \in H$ on the training data weighted as per $D_t(i)$:

$$h_t(x_i) = \{+1, -1\} \tag{4}$$

Compute the weighted error due to $h_t(x_i)$:

$$\epsilon_t = \Sigma_i D_t(i) \mathbb{1} \left[h_t(x_i) \neq y_i \right]$$

Compute an "importance" estimate α_t:

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Update the weights of the training instances:

$$D_{t+1}(i) \propto \begin{cases} D_t(i) \exp(-\alpha_t) & \text{if } h_t(x_i) = y_i \\ D_t(i) \exp(\alpha_t) & \text{if } h_t(x_i) \neq y_i \end{cases} \tag{5}$$

$$D_{t+1}(i) = \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)}{\sum_j D_t(j) \exp\left(-\alpha_t y_j h_t(x_j)\right)} \tag{6}$$

For loop ends here

Finally the boosted classifier H is equal to

$$H(x) = sign\left(\sum_t \alpha_t \cdot h_t(x) \right)$$

The error function that Adaboost minimizes is an exponential loss function.

$$L_{BOOST} = \frac{\sum_i^N exp(-y_i \cdot H(x_i))}{N}$$

Adaboost provides convergence guaranteed in the sense that training error will decrease exponentially as long as ϵ_i's correspond to learner's better than chance.

3 The Toy Example

3.1 Initial Classifier

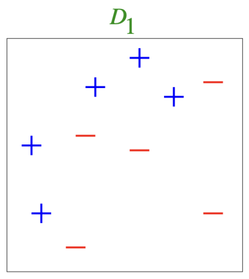

Figure 3: Initial Classifier

3.2 After completion of round 1

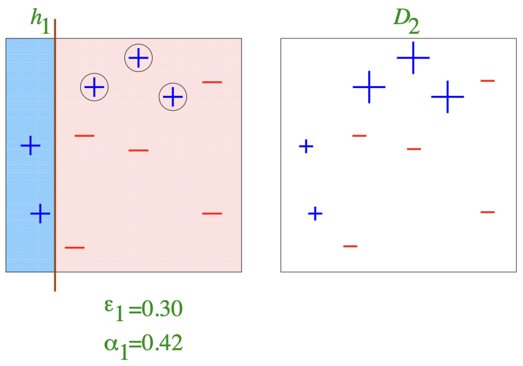

Figure 4: Classifier after completion of Round 1

ϵ_1 = the fraction of misclassified examples = 0.3
After the update of weights as seen in the algorithm above, the misclassified examples have larger weights which is demonstrated by increasing their size whereas the weights of the correctly classified examples have been reduced :

$$D_{t+1}(i) \propto \begin{cases} D_t(i)\exp(-\alpha_t) & \text{if } h_t(x_i) = y_i \\ D_t(i)\exp(\alpha_t) & \text{if } h_t(x_i) \neq y_i \end{cases} \tag{7}$$

3.3 After completion of round 2

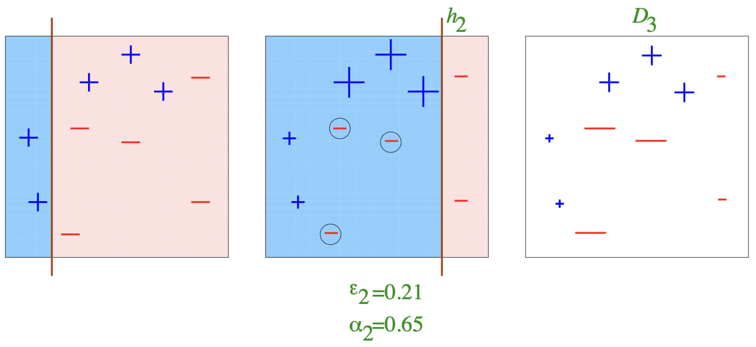

Figure 5: Classifier after completion of Round 2

Now, after training the model with newly updated weights, we have a new decision boundary for all the training examples with the corresponding values of ϵ and α.

3.4 After completion of round 3

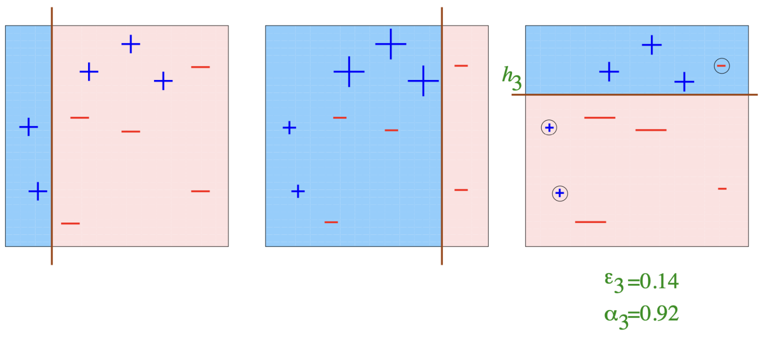

Figure 6: Classifier after completion of Round 3

Following a similar procedure for the third round we will get a new decision boundary.

3.5 Final Classifier

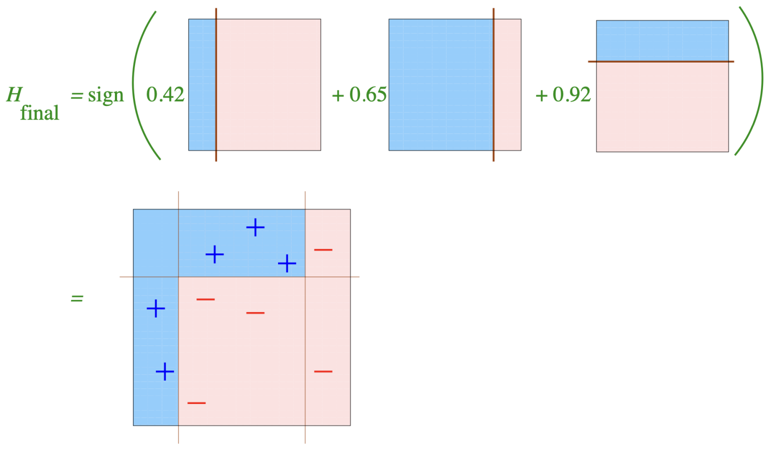

Figure 7: Final Classifier

Now using the models of all the training rounds and their corresponding values of α's our final boosted classifier (which represents an ensemble of weak classifiers) becomes

$$H(x) = sign(\Sigma_t \alpha_t h_t(x)) \tag{8}$$

Note

At every round of the boosting algorithm the value of $\alpha_t > 0$.

$$\alpha_t = \frac{1}{2} log(\frac{1 - \epsilon_t}{\epsilon_t})$$

$\epsilon_0 < 0.5$, because the initial model is better than chance which implies $\alpha_0 > 0$, and for each subsequent step of the algorithm the value of $\epsilon_t < 0.5$ because the training error decreases exponentially. Thus, α_t is always positive for every round.