10.1 Binary Classification

- We have a **dataset** consisting of n examples, $(\underline{X}_1, Y_1), (\underline{X}_2, Y_2), \dots, (\underline{X}_n, Y_n)$. The i^{th} sample $(\underline{X}_i, Y_i)$ consists of a d-dimensional **feature vector** $\underline{X}_i \in \mathbb{R}^d$ (i.e., a column vector) as well as a **label** $Y_i \in \{+1, -1\}$.
- Goal: Learn how to classify every input vector $\underline{x}$ into the correct label.
- Intuition: Binary classification is similar to binary hypothesis testing, except that we do not know the conditional probability models, and do not have enough data to learn them well.
- It is often convenient to deal with the dataset as an $n \times d$ matrix of feature vectors and length-n column vector of labels,

$$\mathbf{X} = \begin{bmatrix} \underline{X}_1^{\mathsf{T}} \\ \underline{X}_2^{\mathsf{T}} \\ \vdots \\ \underline{X}_n^{\mathsf{T}} \end{bmatrix} \qquad \underline{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} .$$

  The i^{th} row of $\mathbf{X}$ is the transposed i^{th} feature vector $\underline{X}_i^{\mathsf{T}}$ and the i^{th} entry of $\underline{Y}$ is the i^{th} label Y_i.
- A **classifier** (or **decision rule**) $D(\underline{x})$ is a function that outputs a $+1$ or -1 label for any possible feature vector $\underline{x} \in \mathbb{R}^d$.
- The performance of a classifier is measure by the **error rate**, which is the fraction of misclassified examples. Define the function

$$g_{\text{error}}(y_{\text{guess}}, y_{\text{true}}) = \begin{cases} 1, & \text{if } y_{\text{guess}} \neq y_{\text{true}}, \\ 0, & \text{if } y_{\text{guess}} = y_{\text{true}}, \end{cases}$$

  which outputs 1 if the guessed label is wrong, and 0 if it is correct.

10.1.1 Training and Test Data

- It is always possible to attain an error rate of 0 by designing a complex decision rule that effectively memorizes the provided dataset. However, this may lead to poor performance for any examples outside of the provided dataset.
- One approach to avoid this issue is to split the dataset into (non-overlapping) training and test data.
- The **training data** is a (randomly-selected) subset of n_{train} examples from the dataset, denoted as

$$(\underline{X}_{\text{train},1}, Y_{\text{train},1}), \dots, (\underline{X}_{\text{train},n_{\text{train}}}, Y_{\text{train},n_{\text{train}}}) \ .$$

  - The training data is used to select the classifier and optimize its parameters.
  - The **training error** is the fraction of misclassified training examples

$$\text{Training Error Rate} = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} g_{\text{error}}(D(\underline{X}_{\text{train}}, i), Y_{\text{train}}, i) \ .$$

- ◦ It is often useful to organize the training data into a matrix of feature vectors of size $n_{\text{train}} \times d$ and a vector of labels of length n_{train}

$$\mathbf{X}_{\text{train}} = \begin{bmatrix} \underline{X}_{\text{train},1}^{\mathsf{T}} \\ \underline{X}_{\text{train},2}^{\mathsf{T}} \\ \vdots \\ \underline{X}_{\text{train},n_{\text{train}}}^{\mathsf{T}} \end{bmatrix}, \qquad \underline{Y}_{\text{train}} = \begin{bmatrix} Y_{\text{train},1} \\ Y_{\text{train},2} \\ \vdots \\ Y_{\text{train},n_{\text{train}}} \end{bmatrix}.$$

- The **test data** is the subset of $n_{\text{test}} = n - n_{\text{train}}$ examples from the dataset that were not selected for training, denoted as

$$(\underline{X}_{\text{test},1}, Y_{\text{test},1}), \dots, (\underline{X}_{\text{test},n_{\text{train}}}, Y_{\text{test},n_{\text{test}}}) \ .$$

  - ◦ The test data is only used to estimate and compare the performance of classifiers. It may not be used in any way to tune the parameters of a classifier.
  - ◦ The **test error** is the fraction of misclassified test examples

$$\text{Test Error Rate} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} g_{\text{error}}(D(\underline{X}_{\text{test}}, i), Y_{\text{test}}, i) \ .$$

  - ◦ It is often useful to organize the test data into a matrix of feature vectors of size $n_{\text{test}} \times d$ and a vector of labels of length n_{test}

$$\mathbf{X}_{\text{test}} = \begin{bmatrix} \underline{X}_{\text{test},1}^{\mathsf{T}} \\ \underline{X}_{\text{test},2}^{\mathsf{T}} \\ \vdots \\ \underline{X}_{\text{test},n_{\text{test}}}^{\mathsf{T}} \end{bmatrix}, \qquad \underline{Y}_{\text{test}} = \begin{bmatrix} Y_{\text{test},1} \\ Y_{\text{test},2} \\ \vdots \\ Y_{\text{test},n_{\text{test}}} \end{bmatrix}.$$

10.1.2 Nearest Neighbor Classifier

- The **nearest neighbor classifier** searches through the training data to find the index of the closest training vector and then returns the corresponding label as its guess,

$$D_{\text{NN}}(\underline{x}) = Y_{\text{train},i_{\text{closest}}} \quad \text{where} \quad i_{\text{closest}} = \underset{i=1,\dots,n_{\text{train}}}{\arg\min} \left\| \underline{x} - X_{\text{train},i} \right\| \ .$$

- The training error is always 0 for the nearest neighbor classifier.

10.1.3 Closest Average Classifier

- Let L_+ and L_- denote the indices in the training dataset with positive and negative labels, respectively,

$$L_+ = \big\{ i \in \{1, \dots, n_{\text{train}}\} : Y_{\text{train},i} = +1 \big\} \qquad L_- = \big\{ i \in \{1, \dots, n_{\text{train}}\} : Y_{\text{train},i} = -1 \big\} \ .$$

- Let $n_{\text{train},+}$ denote the number of positive training examples and let $n_{\text{train},-}$ denote the number of negative training examples.

- Let $\hat{\underline{\mu}}_+$ and $\hat{\underline{\mu}}_-$ denote the sample mean vectors for the positive and negative training examples, respectively,

$$\hat{\underline{\mu}}_+ = \frac{1}{n_{\text{train},+}} \sum_{i \in L_+} \underline{X}_{\text{train},i} \qquad \hat{\underline{\mu}}_- = \frac{1}{n_{\text{train},-}} \sum_{i \in L_-} \underline{X}_{\text{train},i} \ .$$

- The **closest average classifier** calculates the distance between the input vector $\underline{x}$ and the sample mean vectors $\underline{\hat{\mu}}_+$ and $\underline{\hat{\mu}}_-$. It returns the label of the closer sample mean vector,

$$D_{\text{avg}}(\underline{x}) = \begin{cases} +1, & \text{if } \|\underline{x} - \underline{\hat{\mu}}_+\| \leq \|\underline{x} - \underline{\hat{\mu}}_-\|, \\ -1, & \text{if } \|\underline{x} - \underline{\hat{\mu}}_+\| < \|\underline{x} - \underline{\hat{\mu}}_-\|. \end{cases}$$

- The closest average classifier can be interpreted as the ML rule applied to the following likelihoods: if $Y = +1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_+, \mathbf{I})$, and, if $Y_i = -1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_-, \mathbf{I})$ where $\mathbf{I}$ is the identity matrix. (In machine learning terminology, this is often called a Gaussian naïve Bayes classifier.)

10.1.4 Linear Discriminant Analysis Classifier

- Define L_+, L_-, $n_{\text{train},+}$, $n_{\text{train},-}$, $\underline{\hat{\mu}}_+$, and $\underline{\hat{\mu}}_-$ as above.

- Let $\hat{\mathbf{\Sigma}}_+$ and $\hat{\mathbf{\Sigma}}_-$ denote the sample covariance matrices for the positive and negative training examples, respectively,

$$\begin{aligned} \hat{\mathbf{\Sigma}}_+ &= \frac{1}{n_{\text{train},+} - 1} \sum_{i \in L_+} \left(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}_+\right)\left(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}_+\right)^{\mathsf{T}} \\ \hat{\mathbf{\Sigma}}_- &= \frac{1}{n_{\text{train},-} - 1} \sum_{i \in L_-} \left(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}_-\right)\left(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}_-\right)^{\mathsf{T}}, \end{aligned}$$

  and let $\hat{\mathbf{\Sigma}}$ denote the pooled sample covariance matrix,

$$\hat{\mathbf{\Sigma}} = \frac{1}{n_{\text{train}} - 2}\Big((n_{\text{train},+} - 1)\hat{\mathbf{\Sigma}}_+ + (n_{\text{train},-} - 1)\hat{\mathbf{\Sigma}}_-\Big).$$

- **Linear discriminant analysis** can be interpreted as the ML rule applied to the following likelihoods: if $Y = +1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_+, \hat{\mathbf{\Sigma}})$, and, if $Y_i = -1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_-, \hat{\mathbf{\Sigma}})$. After simplifying the log-likelihood ratio, we obtain the following classifier:

$$D_{\text{LDA}}(\underline{x}) = \begin{cases} +1 & \underline{x}^{\mathsf{T}}\hat{\mathbf{\Sigma}}^{-1}(\underline{\hat{\mu}}_+ - \underline{\hat{\mu}}_-) \ \geq \ \frac{1}{2}\big(\underline{\hat{\mu}}_+^{\mathsf{T}}\hat{\mathbf{\Sigma}}^{-1}\underline{\hat{\mu}}_+ - \underline{\hat{\mu}}_-^{\mathsf{T}}\hat{\mathbf{\Sigma}}^{-1}\underline{\hat{\mu}}_-\big), \\ -1 & \text{otherwise.} \end{cases}$$

- If the pooled sample covariance matrix is not full rank, then the inverse of $\hat{\mathbf{\Sigma}}$ can be replaced with its pseudoinverse. This often happens when the number of training examples n_{train} is less than the dimension d.

10.1.5 Quadratic Discriminant Analysis Classifier

- Define L_+, L_-, $n_{\text{train},+}$, $n_{\text{train},-}$, $\underline{\hat{\mu}}_+$, $\underline{\hat{\mu}}_-$, $\hat{\mathbf{\Sigma}}_+$, and $\hat{\mathbf{\Sigma}}_-$ as above.

- **Quadratic discriminant analysis** can be interpreted as the ML rule applied to the following likelihoods: if $Y = +1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_+, \hat{\mathbf{\Sigma}}_+)$, and, if $Y_i = -1$, then $\underline{X}$ is Gaussian$(\underline{\hat{\mu}}_-, \hat{\mathbf{\Sigma}}_-)$. After simplifying the log-likelihood ratio, we obtain the following classifier:

$$D_{\text{QDA}}(\underline{x}) = \begin{cases} +1 & \big(\underline{x} - \underline{\hat{\mu}}_+\big)^{\mathsf{T}}\hat{\mathbf{\Sigma}}_+^{-1}\big(\underline{x} - \underline{\hat{\mu}}_+\big) + \sum_{i=1}^{d}\log(\lambda_{+,i}) \leq \big(\underline{x} - \underline{\hat{\mu}}_-\big)^{\mathsf{T}}\hat{\mathbf{\Sigma}}_-^{-1}\big(\underline{x} - \underline{\hat{\mu}}_-\big) + \sum_{i=1}^{d}\log(\lambda_{-,i}), \\ -1 & \text{otherwise.} \end{cases}$$

where $\lambda_{+,1}, \dots, \lambda_{+,d}$ are the eigenvalues of $\hat{\mathbf{\Sigma}}_+$ and $\lambda_{-,1}, \dots, \lambda_{-,d}$ are the eigenvalues of $\hat{\mathbf{\Sigma}}_-$.

- If either sample covariance matrix is not full rank, then we can use reduce the dimension of the problem (e.g., using PCA discussed below) until both sample covariance matrices are full rank in the reduced space. This often happens when the number of training examples n_{train} is less than the dimension d.

10.1.6 Perceptron Classifier

- The **perceptron** was the first artificial neural network, consisting of just a weighted sum of the inputs, $b_0 + \sum_{j=1}^{d} b_j X_j$, followed by a threshold. These weights $b_0, \dots, b_d$ are trained via the following pseudocode where $0 < r < 1$ is the learning rate and $X_{\text{train},i,j}$ is the j^{th} coordinate of the i^{th} training example:

```
$b_0 = \cdots = b_d = 0$                                   % initialize weights to zero
for $i = 1$ to $n_{\text{train}}$                          % loop over training data
    if $\left(b_0 + \sum_{j=1}^{d} b_j X_{\text{train},i,j} \geq 0\right)$     % guess current label
        $Y_{\text{temp}} = +1$
    else
        $Y_{\text{temp}} = -1$
    end
    $b_0 = b_0 + r\,(Y_{\text{train},i} - Y_{\text{temp}})$     % update weights if guess wrong
    for $j = 1$ to $d$
        $b_j = b_j + r\,(Y_{\text{train},i} - Y_{\text{temp}})X_{\text{train},i,j}$
    end
end
```

Once the weights are trained, the perceptron classifier is

$$D_{\text{perceptron}}(\underline{x}) = \begin{cases} +1 & b_0 + \sum_{j=1}^{d} b_j x_j \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

- We can write the algorithm in vector form. Let $\underline{b} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_d \end{bmatrix}$ denote the vector of weights.

```
$\underline{b} = \underline{0}$                                        % initialize weights to zero
for $i = 1$ to $n_{\text{train}}$                                      % loop over training data
    if $\left(\underline{b}^{\mathsf{T}} \begin{bmatrix} 1 \\ \underline{X}_{\text{train},i} \end{bmatrix} \geq 0\right)$     % guess current label
        $Y_{\text{temp}} = +1$
    else
```

```
            $Y_{\text{temp}} = -1$
        **end**
        $\underline{b} = \underline{b} + r(Y_{\text{train},i} - Y_{\text{temp}})\begin{bmatrix} 1 \\ \underline{X}_{\text{train},i}\end{bmatrix}$    % update weights if guess wrong
    **end**
```

Once the weights are trained, the perceptron classifier is

$$D_{\text{perceptron}}(\underline{x}) = \begin{cases} +1 & \underline{b}^{\mathsf{T}}\begin{bmatrix} 1 \\ \underline{x}\end{bmatrix} \geq 0 \\ -1 & \text{otherwise.}\end{cases}$$

10.2 Dimensionality Reduction

- It is often useful to reduce the dimension of a dataset from d to some $k < d$ to avoid overfitting.
- Dimensionality reduction can also allow us to visualize a high-dimensional dataset in 2 or 3 dimensions.
- **Principal component analysis (PCA)** is a popular technique for dimensionality reduction. The procedure for reducing the training and test data is

  1. Select the reduced dimension $k < d$.
  2. Calculate the sample mean vector and sample covariance matrix from the training data,
  $$\underline{\hat{\mu}} = \frac{1}{n_{\text{train}}}\sum_{i=1}^{n_{\text{train}}} \underline{X}_{\text{train},i} \qquad \hat{\mathbf{\Sigma}} = \frac{1}{n_{\text{train}}-1}\sum_{i=1}^{n_{\text{train}}}\big(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}\big)\big(\underline{X}_{\text{train},i} - \underline{\hat{\mu}}\big)^{\mathsf{T}} .$$
  3. Compute the eigendecomposition $\hat{\mathbf{\Sigma}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{T}}$ where
  $$\mathbf{V} = \begin{bmatrix} | & | & & | \\ \underline{v}_1 & \underline{v}_2 & \cdots & \underline{v}_d \\ | & | & & | \end{bmatrix}$$
  is the orthogonal matrix of eigenvectors and
  $$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix}$$
  is the diagonal matrix of eigenvalues. We assume that these real eigenvalues are pre-sorted in descending order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$.
  4. Let $\mathbf{V}_k$ denote the matrix consisting only of the first k eigenvectors (corresponding to the k largest eigenvalues),
  $$\mathbf{V}_k = \begin{bmatrix} | & | & & | \\ \underline{v}_1 & \underline{v}_2 & \cdots & \underline{v}_k \\ | & | & & | \end{bmatrix} .$$

5. Center the data,

$$\mathbf{X}_{\text{train,centered}} = \mathbf{X}_{\text{train}} - \underline{1}_{n_{\text{train}}}\underline{\hat{\mu}}^{\mathsf{T}} \qquad \mathbf{X}_{\text{test,centered}} = \mathbf{X}_{\text{test}} - \underline{1}_{n_{\text{test}}}\underline{\hat{\mu}}^{\mathsf{T}}$$

where $\underline{1}_n$ is the all-ones column vector of length-n.

6. Project the centered data onto $\mathbf{V}_k$,

$$\mathbf{X}_{\text{train,reduced}} = \mathbf{X}_{\text{train,centered}}\mathbf{V}_k \qquad \mathbf{X}_{\text{test,reduced}} = \mathbf{X}_{\text{test,centered}}\mathbf{V}_k \ .$$

7. Use $\mathbf{X}_{\text{train,reduced}}$ and $\mathbf{X}_{\text{test,reduced}}$ in place of the original training and test data matrices, along with the original training and test labels Y_{train} and Y_{test} to train a classifier and evaluate its performance.

- Intuition: PCA first centers the data so it has mean zero. It then finds a new coordinate system where the first coordinate vector is in the direction of maximum variance, the second coordinate is the direction with the maximum variance in the $(n-1)$-dimensional subspace that is orthogonal to the first coordinate vector, and so on, so that the $(i+1)^{\text{th}}$ coordinate is the direction with the maximum variance in the $(n-i)$-dimensional subspace that is orthogonal to the subspace spanned by the first i coordinate vectors. Thus, keeping the first k coordinates results in the maximal possible variance that can be retained in k dimensions under linear transformations.