2 Classification

An application that is closely related to dimensionality reduction is classification, a process to identify to which of a set of categories a new observation belongs, often on the basis of a training set of data. Generally speaking, classification can occur at the data space, or feature space. The feature vector can be extracted in many different ways. One possible source of features vectors is from dimensionality reduction as discussed in section 1: in this setting, the feature vector consists of the coefficients for the linear expansion in another basis. In the following, we use the term *feature vector* to denote the variables on which the classification performs. Here we discuss some simple yet popular methods for classification and pattern recognition.

2.1 Linear Methods for Classification

Given a feature vector $\mathbf{x}$, we can have a predictor $G(\mathbf{x})$ that takes value in a discrete set $\mathcal{G}$ consisting of all class labels which, for convenience, labeled as $1, 2, \cdots, K$. We can always divide the input space into a collection of regions labeled according to the classification; these boundaries can be rough or smooth, depending on the prediction function. An important class of procedures use linear decision boundaries; this is what we will talk about here in this chapter.

2.1.1 Linear Regression for an Indicator Matrix

For the first method, we will re-visit the method of linear regression we learned in Project 2. In project 2, we looked at how to learn a regression function or predictor

$$y = \mathbf{w}^T \mathbf{z} = \sum_j w_j z_j.$$

where $\mathbf{z}$ is the feature vector for each data point, and y the scalar output. Through simple least-square method, we also derived the optimal value for $\mathbf{w}$ as:

$$\mathbf{M}\mathbf{w} = \mathbf{s}$$

where $\mathbf{M} \leftarrow \sum_i \mathbf{z}_i \mathbf{z}_i^T$ and $\mathbf{s} \leftarrow \sum_i \mathbf{z}_i y_i$ from all training data. In matrix form, the above equation can be simply written as:

$$\left(\mathbf{Z}\mathbf{Z}^T\right)\mathbf{w} = \mathbf{Z}\mathbf{y}.$$

where

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_N \end{bmatrix}. \tag{2.1}$$

and

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{2.2}$$

for all N sets of $\{\mathbf{z}_i, y_i\}_{i=1}^N$.

To apply linear regression to classification involves two straightforward extensions.

Linear regression with multiple outputs

Suppose now that, for each feature vector $\mathbf{z}$, we have multiple instead of a single output that we wish to predict. We assume a linear model for each output:

$$y_{k,i} = \mathbf{w}_k^T \mathbf{z}_i, \qquad k = 1, \cdots, K$$

where the subscript i corresponds to the i-th data point, and k the k-th output for that data point. Note that each $y_{i,k}$ corresponds to the same vector $\mathbf{z}_i$; it's the prediction model, $\mathbf{w}_k$, that changes. This can be written into a vector form:

$$\mathbf{y}_i = \mathbf{W}\mathbf{z}_i$$

where $\mathbf{y}_i$ is a vector that includes all K outputs for each $\mathbf{z}_i$. $\mathbf{W}$ is a $K \times p$ matrix where each row corresponds to $\mathbf{w}_k$, and p is the dimension of $\mathbf{z}_i$.

A straightforward generalization of the least-square optimization criteria used in Project 2 will give us:

$$\arg\min_{\mathbf{W}} R = \sum_i ||\mathbf{W}\mathbf{z}_i - \mathbf{y}_i||_2^2$$

We will not go into the derivation of the above problem, but the optimal form of $\mathbf{W}$ is very similar to what we did in Project 2:

$$\mathbf{W} = \mathbf{Y}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1}$$

where $\mathbf{Z}$ is a $p \times N$ matrix as defined before (p being the dimension of the feature vector $\mathbf{z}$)

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_N \end{bmatrix}. \tag{2.3}$$

$\mathbf{Y}$ is a $k \times N$ matrix that consists of all the $\mathbf{y}_i$ vectors for all data points:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_N \end{bmatrix}. \tag{2.4}$$

and the output $\mathbf{W}$ is a $k \times p$ matrix where each row gives the coefficient for predicting the corresponding element in the output vector $\mathbf{y}$.

Classification with indicator response matrix To apply the above method for classification, we code the response of each feature vector to each class type via an indicator variable. Therefore, if $\mathcal{G}$ has K classes, there will be K such indicators $Y_k, k = 1, \cdots, K$ with $Y_k = 1$ if $G = k$ else 0. For each feature vector, these are collected together in a vector $\mathbf{y} = [Y_1, \cdots, Y_k]^T$. For example, if we are looking at a 3-class classification, and a known data point in the training data belongs to class 2, then its corresponding indicator vector $\mathbf{y}$ is:

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}. \tag{2.5}$$

Given a training set of N data points, according to equation (2.4), we know that $\mathbf{Y}$ is linear regression is a matrix that is of the size $K \times N$, where K is the number of classes; the values in $\mathbf{Y}$ is either 0 or 1, and each column has only a single 1. Matrix $\mathbf{Y}$ is known as *indicator response matrix*.

Given a new observation, it is classified as follows:

- Assemble the feature vector $\mathbf{x}$ as the way the training data $\mathbf{z}$is assembled (see more later).
- Compute the fitted output $\mathbf{y}_p = \mathbf{W}\mathbf{x}$. Note that here your output is a vector.
- Identify the largest component and classify the corresponding class label to the new data.

Example (2-Class): Using first-order polynomial (affine function) for linear regression, test the above method in two different training and test datasets provided through myCourses: Write a sub-function to calculate the percentage of error (mis-classification).

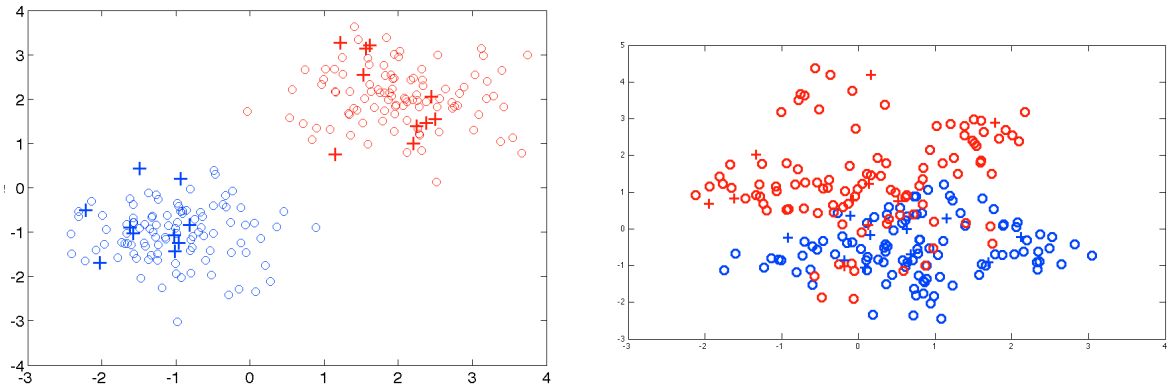

Figure 2.1: Training (circle) and test (plus sign) data of the 2D examples

1. The first example is shown on the left of Fig. 2.1. Corresponding data and labels for the training and test sets are provided in text files labeled with "2C.txt". Each class is generated from one bivariate Gaussian. You should expect a 100% accuracy (0 mis-classification) in this example. Your $\mathbf{W}$ trained from linear regression should be:
$$\mathbf{W} = \begin{bmatrix} 0.6511 & -0.1282 & -0.1710 \\ 0.3489 & 0.1282 & 0.1710 \end{bmatrix}.$$
In this type of classes, a linear decision boundary is the best one can do and our estimate is almost optimal. In some cases, the region of overlap is inevitable and future data to be predicted will be plagued by this overlap as well.

2. The second example is shown on the right of Fig. 2.1. Corresponding data and labels for the training and test sets are provided in text files labeled with "ESL2C.txt". In this case, each class is represented by a mixture of tightly clustered Gaussian distributions, the mean of each is in turn generated from a bivariate Gaussian. In this case, you should expect a 80% accuracy as shown in Fig. 2.2 where the mis-classified test points are marked in black. Your $\mathbf{W}$ trained from linear regression should be:
$$\mathbf{W} = \begin{bmatrix} 0.5387 & 0.1063 & -0.2261 \\ 0.4613 & -0.1063 & 0.2261 \end{bmatrix}.$$
In this type of classes, a linear decision boundary is unlikely to be optimal.

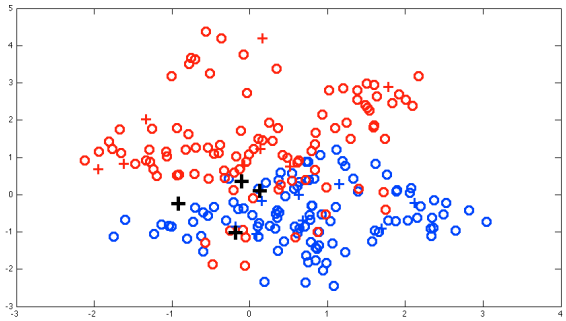

Figure 2.2: Results of classification by linear regression in the second 2-class example (right panel of Fig. 2.1). Mis-classification points are marked by black.

Example (3-Class): Similarly, test the same first-order polynomial (affine function) linear regression, in the 3-class training and test datasets provided through myCourses ((trainingdata3C.txt, etc).

1. The first example is a 3-class version of the second example (right panel of Fig. 2.1) discussed above. Each class is a mixture of Gaussian whose means are generated

from a Gaussian distribution ("ESL3C.txt"). As shown below, you should expect a 86.7% classification accuracy, with your trained $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} 0.3900 & 0.0843 & -0.2157 \\ 0.5143 & -0.2222 & 0.0478 \\ 0.0957 & 0.1379 & 0.1679 \end{bmatrix}.$$

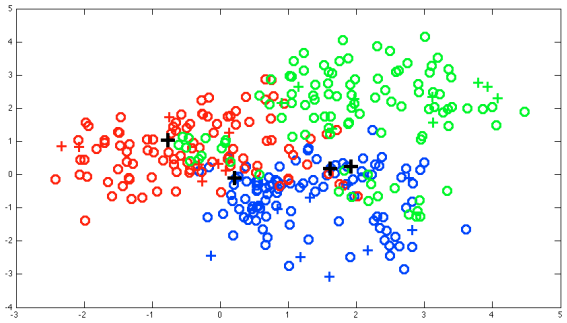

Figure 2.3: Results of classification by linear regression in the first 3-class example. Misclassification points are marked by black.

2. The second example is a simple case as a 3-class extension of the first example discussed in the 2-class examples. Each class is modeled by one Gaussian distribution ("3C.txt"):

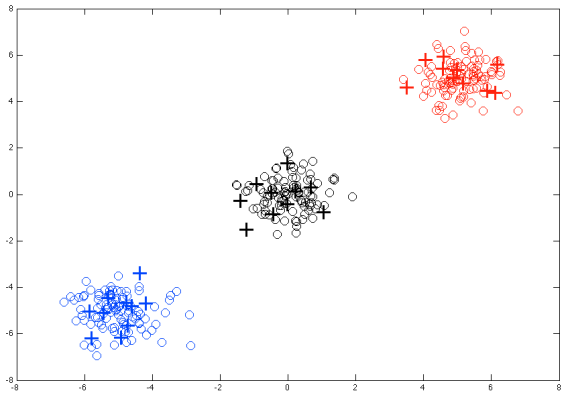

Figure 2.4: Training (circle) and test (plus sign) data of the 3D example

From what we observed during the 2-class examples, we should expect a perfect classification on this simple set of data, right? Unfortunately, there is a serious problem with the regression approach when the number of classes $K \geq 3$. If we use affine function for regression, because of the rigid nature of the regression model,

classes can be *masked* by others. For this particular example, you will encounter a 30% mis-classification rate with your $\mathbf{W}$ trained from linear regression as:

$$\mathbf{W} = \begin{bmatrix} 0.3347 & -0.0482 & -0.0496 \\ 0.3315 & 0.0562 & 0.0414 \\ 0.3338 & -0.0079 & 0.0083 \end{bmatrix}.$$

As shown below, the class in the middle is completely masked.

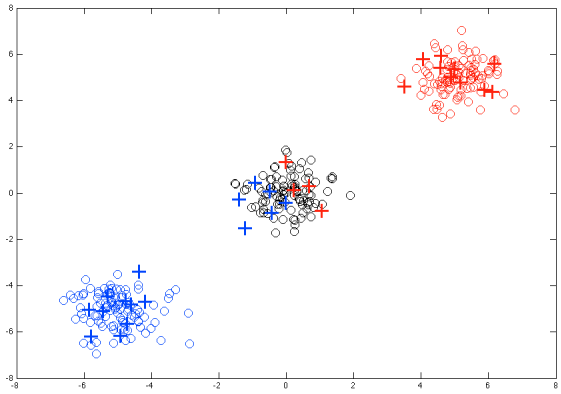

Figure 2.5: Results from linear regression using affine models on the example in Fig. 2.4.

A solution to this is to raise the degree of the polynomial for linear regression. For example, we can expand our feature vectors to include their squares and cross-products, thereby adding $p(p+1)/2$ additional variables. Linear function in this augmented space maps down to quadratic function in the original space; hence linear decision boundaries in the augmented feature space maps to quadratic decision boundaries in the original data space. The idea is similar to what we did previously in Project 2, using simple linear method to learn complex functions.

Example: On the same 3-class training and test datasets in Fig. 2.4, test your linear regression for classification method using $p = 2$ polynomial. Your $\mathbf{W}$ is now:

$$\mathbf{W} = \begin{bmatrix} 0.0274 & -0.0494 & -0.0487 & 0.0093 & 0.0083 \\ 0.0236 & 0.0559 & 0.0414 & 0.0099 & 0.0077 \\ 0.9490 & -0.0065 & 0.0072 & -0.0192 & -0.0160 \end{bmatrix}.$$

And you will again obtain a 100% classification accuracy on this example.

Similarly, test your linear regression for classification method using $p = 2$ polynomial on the first example in Fig. 2.3. You will see an improvement of performance to 90% classification accuracy with the following trained $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} 0.4365 & 0.1590 & -0.2267 & -0.0483 & 0.0131 \\ 0.4973 & -0.2627 & 0.0776 & 0.0332 & -0.0248 \\ 0.0662 & 0.1037 & 0.1491 & 0.0151 & 0.0118 \end{bmatrix}.$$

What if increasing the degree of the polynomial p? What is the optimal value for p? A loose but general rule here is that, if $K \geq 3$ classes are lined up, polynomial terms up to degree $K - 1$ might be needed to resolve them. The complexity of this method therefore rises quickly with the number of classes.