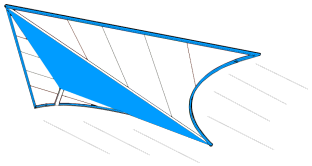

Report

Project: IRIS Classifiers

Implementer: Tran Bao Phuc

Ho Chi Minh City, Thursday 20th June, 2024

Table of contents

1 Goal of term project

The primary aim of this term project is to build and evaluate the efficacy of various classification models on the IRIS dataset. The project focuses on implementing multiple classification methods, including one-versus-one and one-versus-the-rest for binary class separation, as well as multi-class approaches such as Fisher's discriminant analysis and Bayesian classifiers. After employing these methods, I will evaluate their performance in terms of accuracy and confusion matrix by using a 5-fold cross-validation technique. This comprehensive evaluation helps enhance my understanding of the strengths and limitations of each model for classification.

2 Theoretical background

2.1 Introduction to Classification Algorithm

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

The algorithm which implements the classification on a dataset is known as a classifier. Classification problem have three distinct approaches, including:

- Discriminant Function Approach: Discriminant function classifiers use functions to discriminate between classes based on the input features. These functions assign scores to each class, and the class with the highest score is predicted for a given observation.
- Probabilistic Generative Models: Generative models determining the class-conditional densities for each class individually. Then use Bayes' theorem to find the posterior class probabilities and choose class has highest posterior.
- Probabilistic Discriminative Models: Discriminative models directly find the posterior class probabilities without explicitly modeling the distribution of each class. Then choose class has highest posterior.

2.2 Discriminant Function

2.2.1 Two classes

In machine learning, a discriminant function is used to classify input vectors into distinct classes. For simplicity, consider the case of two classes, C_1 and C_2. The discriminant function is a linear function of the input vector x, designed to separate these classes by a decision boundary, typically a hyperplane.

The simplest form of a linear discriminant function is represented as:

$$y(x) = w^T x + w_0$$

where w is a weight vector, and w_0 is a bias.

The decision rule for classifying a new input x is based on the sign of $y(x)$:

- If $y(x) \geq 0$, classify x as C_1.

- If $y(x) < 0$, classify x C_2

The corresponding decision boundary is therefore defined by the relation $y(\mathbf{x}) = 0$ which corresponds to a $(D-1)$ dimensional hyperplane within the D dimensional input space. This hyperplane separates the input space into two regions, each corresponding to one of the classes.

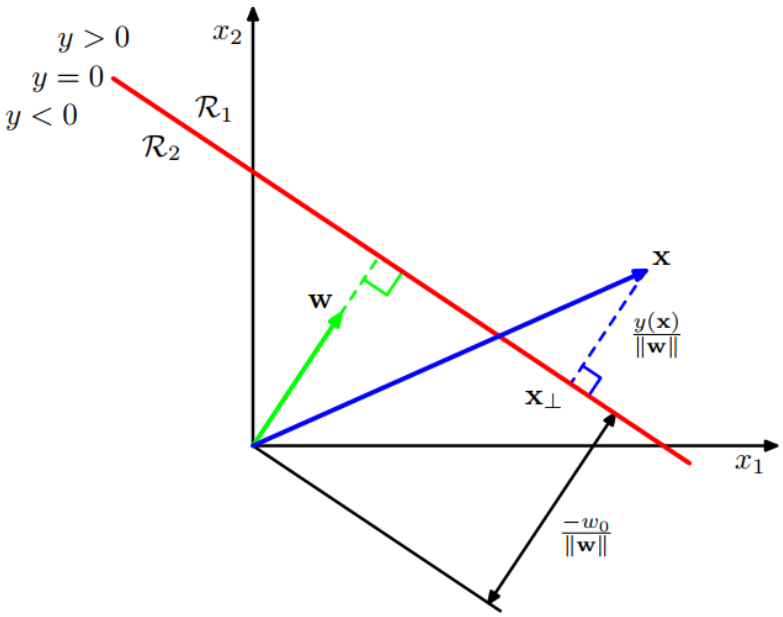

Figure 1: Illustration of the geometry of a linear discriminant function in two dimensions

To conveniently using, we will add dummy "input" value $x_0 = 1$ and theo define $\widetilde{w} = (w_0, w)$ and $\widetilde{x} = (x_0, x)$. So that:

$$y(x) = \widetilde{w}^T \widetilde{x}$$

In this case, the decision surfaces are D dimensional hyperplane passing through the origin of the $D+1$ dimensional expanded input space.

2.2.2 Multiple classes

The extension of linear discriminant to multiple classes, specifically $K > 2$ classes. The simple approach is combining multiple two-class discriminant functions and this strategy often leads to complications.

2.2.2.a One-Versus-The-Rest Classifier

A straightforward method is to use $K-1$ classifiers, each solving a two class problem to separate points in a particular class C_k from points not in that class. However, this approach

leads to regions of input space that are ambiguously classified. The Figure 2 shows an example involving three classes where this approach leads to regions of input space that are ambiguously classified.

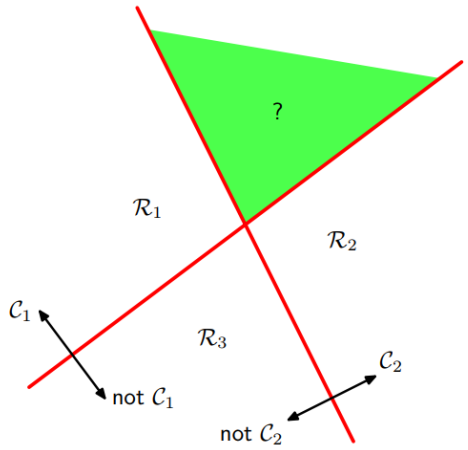

Figure 2: Illustration of the decision regions for a one-versus-the-rest classifier

2.2.2.b One-Versus-The-Rest classifier

Another method is to use $K(K-1)/2$ binary discriminant functions, one for every possible pair of classes. Classification is then based on a majority vote among the discriminant functions. Nevertheless, this method also leads the problem of ambiguous regions, as shown in the Figure-3.

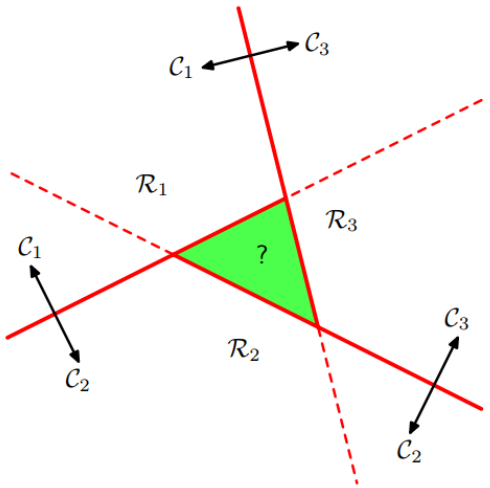

Figure 3: Illustration of the decision regions for a one-versus-the-one classifier

2.2.2.c Single K-Class Discriminant

To avoid these ambiguous classifications, a single K-class discriminant can be used. This approach involves defining K linear functions of the form:

$$y_k(x) = w_k^T x + w_{k0}$$

A point x is assigned to class C_k if $y_k(x) > y_j(x)$ for all $j \neq k$. The decision boundary between class C_k and class C_j is therefore given by $y_k(x) = y_j(x)$ and hence corresponds to a $(D-1)$ dimensional hyperplane defined by

$$(w_k - w_j)^T x + (w_{k0} - w_{j0}) = 0$$

This form is analogous to the decision boundary for the two-class case and retains similar geometrical properties.

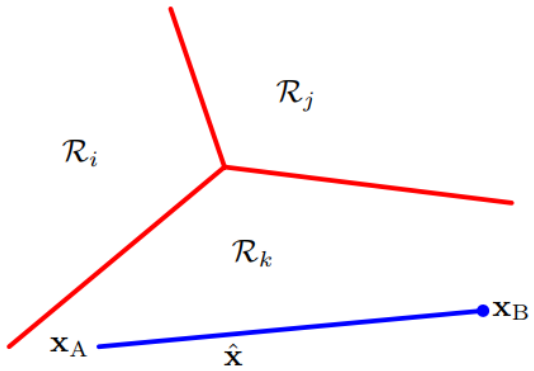

Figure 4: Illustration of the decision regions for a multiclass linear discriminant

2.2.3 Least squares for classification

For a classification problem with K classes, each class C_k is represented by a linear model:

$$y_k(x) = w_k^T x + w_{k0}$$

where $k = 1, ..., K$, We can conveniently group these together using vector notation so that:

$$y(x) = \widetilde{W}^T \widetilde{x}$$

where W is a matrix whose k^{th} column comprises the $D+1$ dimensional vector $\widetilde{w}_k = (w_{k0}, w_k^T)^T$ and $widetildex$ is the corresponding augmented input vector $(1, x^T)^T$ with a dummy input $x_0 = 1$. A new input x is then assigned to the class for which the output $y_k = \widetilde{w}_k^T \widetilde{x}$ is largest.

We now determine the parameter matrix $\widetilde{W}$, by minimizing a sum-of-squares error function. Consider a training data set $\{x_n, t_n\}$ where $n = 1, ..., N$, and define a matrix T whose n^{th} row is the vector t_n^T, together with a matrix $\widetilde{X}$ whose n^{th} row is $\widetilde{x}_n^T$. The sum-of-squares error function can then be written as

$$E_D(\widetilde{W}) = \frac{1}{2} Tr\{(\widetilde{X}\widetilde{W} - T)^T(\widetilde{X}\widetilde{W} - T)\}$$

Setting the derivative with respect to $\widetilde{W}$ to zero, and rearranging, we then obtain the solution for W, in the form

$$\widetilde{W} = (\widetilde{X}^T\widetilde{X})^{-1}\widetilde{X}^T T = \widetilde{X}^\dagger T$$

where $\widetilde{X}^\dagger$ is the pseudo-inverse of the matrix $\widetilde{X}$. We then obtain the discriminant function in the form

$$y(x) = \widetilde{W}^T\widetilde{x} = T^T(\widetilde{X}^\dagger)^T\widetilde{x}$$

2.2.4 Fisher's linear discriminant

Fisher's linear discriminant is a method used in the context of linear classification models to achieve dimensionality reduction. It is particularly useful in situations involving two classes. The primary idea is to project high-dimensional data onto a one-dimensional space while maximizing the separation between the two classes.

Consider a scenario with two classes, $C1$ and $C2$. We begin with a D-dimensional input vector x and project it onto a single dimension using the following transformation:

$$y = w^T x$$

where w is a weight vector. By placing a threshold on y, we can classify the data points: if $y \leq -w_0$, the point is classified as belonging to C_1; otherwise, it is classified as belonging to C_2.

When projecting onto one dimension, some information loss is inevitable. However, we can mitigate this by choosing w to maximize the separation between the projected means of the two classes. The mean vectors of the classes are defined as:

$$M_1 = \frac{1}{N_1}\sum_{n \in C_1} x_n$$

$$M_2 = \frac{1}{N_2}\sum_{n \in C_2} x_n$$

where $N1$ and $N2$ are the number of points in classes C_1 and C_2.

The goal is to choose w to maximize the difference between the projected means:

$$m_2 - m_1 = w^T(M_2 - M_1)$$

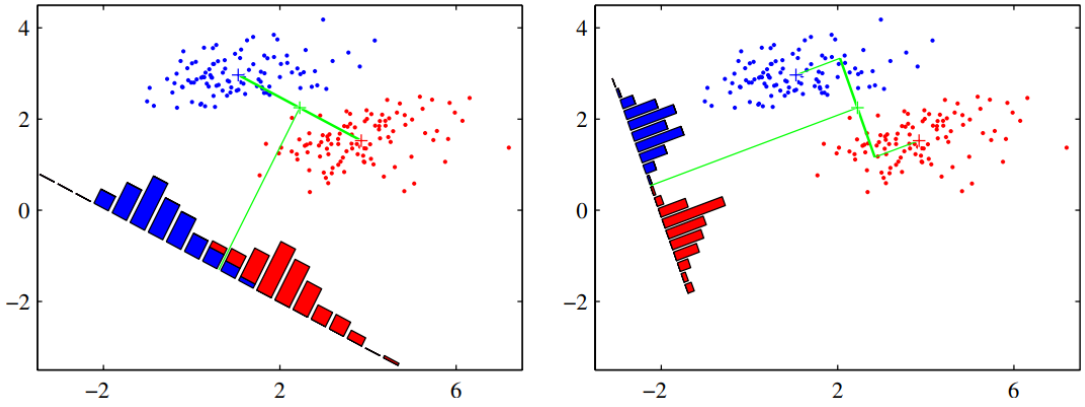

Figure 5: The left plot shows samples from two classes a long with the histograms resulting from projection onto the line joining the class means. Note that there is considerable class overlap in the projected space. The right plot shows the corresponding projection based on the Fisher linear discriminant, showing the greatly improved class separation.

The Fisher criterion provides a measure to maximize the separation of the projected class means while minimizing the variance within each class. The minimizing variance within each class helps minimizing the class overlap. This criterion is defined as:

$$J(w) = \frac{w^T S^B w}{w^T S^W w}$$

where S_B is the between-class covariance matrix and is given by

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

and S_W is the total within-class covariance matrix, given by

$$S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T$$

Maximizing $J(w)$ by S_w^{-1} we then obtain:

$$w \sim S_W^{-1}(m_2 - m_1)$$

2.2.5 Fisher's discriminant for multiple classes

We now consider the generalization of the Fisher discriminant to $K > 2$ classes, and we shall assume that the dimensionality D of the input space is greater than the number K of classes. Next, we introduce $D' > 1$ linear 'features' $y_k = w_k^T x$, where $k = 1, ..., D'$. These feature values can conveniently be grouped together to form a vector y. Similarly, the weight vectors $\{w_k\}$ can

be considered to be the columns of a matrix W, so that

$$y = W^T x$$

The within-class covariance matrix S_W for K classes is a generalization from the two-class case and is given by:

$$S_W = \sum_{k=1}^{K} S_k$$

where

$$S_k = \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{x}_n$$

and N_k is the number of patterns in class C_k

The total covariance matrix S_T is defined as:

$$S_T = \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{m})(\mathbf{x}_n - \mathbf{m})^T$$

where m is the mean of the total dataset

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n = \frac{1}{N} \sum_{k=1}^{K} N_k \mathbf{m}_k$$

and $N = \sum_k N_k$ is the total number of data points. The total covariance matrix can be decomposed into the sum of the within-class covariance matrix S_W and an additional matrix S_B, which measures the between-class covariance:

$$S_T = S_W + S_B$$

where

$$S_B = \sum_{k=1}^{K} N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T$$

These covariance matrices have been defined in the original x-space. We can now define similar matrices in the projected D'-dimensional y-space

$$s_W = \sum_{k=1}^{K} \sum_{n \in C_k} (\mathbf{y}_n - \mu_k)(\mathbf{y}_n - \mu_k)^T$$

and

$$s_B = \sum_{k=1}^{K} N_k(\mu_k - \mu)(\mu_k - \mu)^T$$

where

$$\mu_k = \frac{1}{N_k} \sum_{n \in C_k} \mathbf{y}_n$$

$$\mu = \frac{1}{N} \sum_{k=1}^{K} N_k \mu_k$$

Again we wish to construct a scalar that is large when the between-class covariance is large and when the within-class covariance is small. The criterion can then be written as an explicit function of the projection matrix W in the form:

$$J(w) = Tr\{(WS_W W^T)^{-1}(WS_B W^T)\}$$

Maximize criteria and solution of the weight values are determined by those eigenvectors of $S_W^{-1} S_B$ that correspond to the D' largest eigenvalues.

2.3 Probabilistic Generative Models

2.3.1 Normalized exponential

We turn next to a probabilistic view of classification and show how models with linear decision boundaries arise from simple assumptions about the distribution of the data. Here we shall adopt a generative approach in which we model the class-conditional densities $p(x|C_k)$, as well as the class priors $p(C_k)$, and then use these to compute posterior probabilities $p(C_k|x)$ through Bayes' theorem.

The posterior probability for class C_k can be written as

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

which is known as the *normalized exponential* and can be regarded as a multiclass generalization of the logistic sigmoid. Here the quantities a_k are defined by

$$a_k = ln(p(x|C_k)p(C_k))$$

The normalized exponential is also known as the softmax function, as it represents a smoothed version of the 'max' function because, if $ak \gg aj$ for all $j \neq k$, then $p(C_k|x) \simeq 1$, and $p(C_j|x) \simeq 0$.

Continue analysis, we obtain

$$a_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0} \tag{1}$$

where we have defined

$$\mathbf{w}_k = \Sigma^{-1}\boldsymbol{\mu}_k \tag{4.69}$$

$$w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T\Sigma^{-1}\boldsymbol{\mu}_k + \ln p(C_k) \tag{2}$$

We see that the $a_k(\mathbf{x})$ are again linear functions of $\mathbf{x}$ as a consequence of the cancellation of the quadratic terms due to the shared covariances. The resulting decision boundaries, corresponding to the minimum misclassification rate, will occur when two of the posterior probabilities (the two largest) are equal, and so will be defined by linear functions of $\mathbf{x}$.

2.3.2 Maximum likelihood solution

Consider a generative classification model for K classes defined by prior class probabilities $p(C_k) = \pi_k$ and general class-conditional densities $p(\phi \mid C_k)$, where ϕ is the input feature vector. Suppose we are given a training data set $\{\phi_n, t_n\}$ where $n = 1, \ldots, N$, and t_n is a binary target vector of length K that uses the 1-of-K coding scheme, so that it has components $t_{nj} = I_{jk}$ if pattern n is from class C_k. Assuming that the data points are drawn independently from this model and each class has a Gaussian class conditional density with a shared covariance matrix, and hence

$$p(\phi \mid C_k) = N(\phi \mid \mu_k, \Sigma)$$

The likelihood function is given by

$$p(\{\phi_n, t_n\}|\{\pi_k\}) = \prod_{n=1}^{N}\prod_{k=1}^{K}[p(\phi_n|C_k)\pi_k]^{t_{nk}}$$

and taking the logarithm, we obtain

$$\ln p(\{\phi_n, t_n\}|\{\pi_k\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\left\{\ln N(\phi \mid \mu_k, \Sigma) + \ln \pi_k\right\}$$

Setting the derivative with respect to π_k equal to zero and rearrangeing, we obtain

$$\pi_k = \frac{N_k}{N}$$

where N_k is the number of data points assigned to class C_k.

Setting the derivative with respect to μ_k equal to zero and rearrangeing, we obtain:

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N} t_{nk}\phi_n$$

Finally, consider the maximum likelihood solution for the shared covariance matrix Σ, we

obtain:

$$\Sigma = \sum_{k=1}^{K} \frac{N_k}{N} S_k$$

where

$$S_k = \frac{1}{N_k} \sum_{n=1}^{N} t_{nk}(\phi_n - \mu_k)(\phi_n - \mu_k)^T$$

We see, Σ is given by a weighted average of the covariances of the data associated with each class, in which the weighting coefficients are given by the prior probabilities of the classes.

3 Method and algorithm

3.1 Introduce Iris dataset

The Iris dataset is a well-known and widely used in the field of machine learning. The dataset consists of 150 samples, each samples represents an iris flower. There are three distinct species of iris: Iris setosa, Iris versicolor, and Iris virginica.

For each flower, four features are measured: Sepal Length (in centimeters), Sepal Width (in centimeters), Petal Length (in centimeters), Petal Width (in centimeters). These features are selected because they are believed to effectively classify the different species of iris. The objective in using this dataset is to develop a classification model that can accurately identify the species of an iris flower based on its measured features.

Given its balanced class distribution and the clear separability of its classes, the Iris dataset serves as an excellent starting point for demonstrating the effectiveness of various classification algorithms.

3.2 Data Preprocessing

The preprocessing pipeline can be detailed as follows:

- Data Loading: The FileReader class is designed to read the Iris dataset from a CSV file. The readIrisCSV method reads the data into a Pandas DataFrame for easy manipulation.
- Feature Selection and Scaling: The Id and Species columns are dropped from the DataFrame. The remaining features (including sepal length, sepal width, petal length, and petal width) are scaled down by a factor of 10.
- Adding a Bias Term: I add dummy 'input' by adding column of ones to the feature matrix.
- Target Encoding: The target variable, Species, is encoded into numerical values. Iris setosa is mapped to 0, Iris versicolor is mapped to 1, Iris virginica is mapped to 2.

3.3 Classifier

To effectively traning Iris dataset for classification, I define an abstract class *Model* for my machine learning models. This class includes abstract functions for training and predicting with a classification model.

The ClassificFunction class is used to provide a static method for creating discriminant functions used in classification tasks.

3.3.1 One-Versus-One Classifiers

The OneVersusOneModel class inherits Model class. It is designed for multi-class classification using a one-versus-one strategy. It include three method:

- classifyTwoClasses(self, class1, class2): Trains a discriminant function to distinguish between two specific classes.
- trainModel(self): Trains discriminant functions for all pairs of classes in a one-versus-one manner.
- predict(self, input): Makes a prediction for a given input sample by aggregating the results of the discriminant functions. During prediction, a voting mechanism is used to determine the final class. Default using class 0 for ambiguously area.

3.3.2 One-Versus-The-Rest Classifiers

The OneVersusRestModel class inherits from the Model class. It is designed for multi-class classification using a one-versus-rest strategy. It includes three methods:

- classifyOneClasses(self, class1): Trains a discriminant function to distinguish one specific class from all other classes.
- trainModel(self): Trains discriminant functions for each class versus the rest of the classes.
- predict(self, input): Makes a prediction for a given input sample by aggregating the results of the discriminant functions. During prediction, the class with the strongest response from its discriminant function is chosen as the final class. By default, class 0 is used for ambiguous areas.

3.3.3 Three-Class Discriminant Function Classifier

The MulticlassModel class inherits from the Model class. It is designed for multi-class classification using a single discriminant function to handle all classes simultaneously. It includes two methods:

- trainModel(self): Trains a single discriminant function using all classes and the entire dataset.
- predict(self, input): Makes a prediction for a given input sample by using the trained discriminant function and returning the class with the highest predicted score.

3.3.4 Fisher's Discriminant Function Classifier

The FisherModel class inherits from the Model class. It is designed for multi-class classification using Fisher's Linear Discriminant Analysis (LDA). It includes several methods for calculating necessary parameters, projecting data, training the model, and making predictions:

- _calculateMean(self): Computes the mean vectors for each class and the overall mean vector of the dataset.

- _calculateWithinClassCovarianceMatrix(self, meanEachClassList): Computes the within-class covariance matrix by summing the covariance matrices of each class.
- _calculateBetweenClassCovarianceMatrix(self, meanEachClassList, meanTotal): Computes the between-class covariance matrix by measuring the scatter of class means relative to the overall mean.
- findParameterToProject(self): Determines the projection matrix (W) by solving the eigenvalue problem for the matrix formed by the ratio of between-class scatter to within-class scatter, selecting the eigenvectors corresponding to the largest eigenvalues.
- projectDataToTwoDimensions(self): Projects the dataset into a two-dimensional space using the calculated projection matrix, adds a bias term (column of ones), and returns the transformed data and target labels.
- projectInputToTwoDimensions(self, input): Projects a new input sample into the same two-dimensional space using the previously calculated projection matrix and adds a bias term.
- trainModel(self): Projects the training data to two dimensions and trains a discriminant function using the projected data and one-hot encoded target matrix.
- predict(self, input): Projects a new input sample to the two-dimensional space and uses the trained discriminant function to make a prediction, returning the class with the highest predicted score.

3.3.5 Bayesian Classifier

The GenerativeModel class inherits from the Model class. It implements a generative model approach for classification tasks, specifically using Gaussian distributions. Here are the methods and their purposes:

- trainModel(self): Trains the generative model by computing the parameters needed for classification including the prior probabilities for each class, the mean vectors for each class, the shared covariance matrix, the weights w and the bias w_0.
- predict(self, input): Makes a prediction for a given input sample by computes the class probabilities using the softmax function. Finally, returns the class with the highest predicted probability.

3.4 Traning and Evaluation

To train and evaluate different classifiers using the provided CrossValidator class, I utilize several classifier methods (classifyOneVersusRest, classifyOneVersusOne, classifyThreeClass, classi-

fyFisher, classifyGeneration). Each method initializes a specific model, trains it on the training data, and then predicts on the test data for each fold of cross-validation.

Training

- Splitting data: The data is split into training and test sets (ratio 4:1) for each fold.
- Initialization: Each classifier method (classifyOneVersusRest, classifyOneVersusOne, etc.) initializes a specific model (OneVersusRestModel, OneVersusOneModel, etc.) with the training data, target labels, and classes.
- Training: The trainModel method of each model is called to train the model using the training data (dataTrain and targetTrain).

Evaluation

Cross-validation: The crossValidate method of CrossValidator performs k-fold cross-validation:

- Predictions: For each fold, the model predicts the classes for the test data using the trained model.
- Accuracy: Calculates the accuracy for each fold by comparing predicted labels with actual labels.
- Confusion Matrix: Updates the confusion matrix to evaluate the performance of the classifiers across different classes.

4 Simulation Results and Discussion

4.1 Simulation Results

4.1.1 One-Versus-One Classifiers

- Average Accuracy: 97.33%
- Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{bmatrix}$$

4.1.2 One-Versus-The-Rest Classifiers

- Average Accuracy: 72.00%
- Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 22 & 28 \\ 0 & 14 & 36 \end{bmatrix}$$

4.1.3 Three-Class Discriminant Function Classifier

- Average Accuracy: 84.00%
- Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 35 & 15 \\ 0 & 9 & 41 \end{bmatrix}$$

4.1.4 Fisher's Discriminant Function Classifier

- Average Accuracy: 84.00%
- Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 35 & 15 \\ 0 & 9 & 41 \end{bmatrix}$$

4.1.5 Bayesian Classifier

- Average Accuracy: 98.00%
- Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 48 & 2 \\ 0 & 1 & 49 \end{bmatrix}$$

4.2 Discussion

The Bayesian Classifier is the most accurate, achieving an impressive average accuracy of 98.00%. This was closely followed by the One-Versus-One Classifier, which attained an average accuracy of 97.33%.

On the other hand, the One-Versus-The-Rest Classifier demonstrated significant challenges with lower average accuracy of 72.00%. This suggests that this strategy is less effective in this particular context of multi-class classification.

Additionally, despite the theoretical advantages of Fisher's method in maximizing class separability, it did not show significant effectiveness in this problem, matching the performance of the Multiple Class Discriminant Function Classifiers, both of which achieved an average accuracy of 84%.

5 Conclusion

The project successfully implemented and evaluated various discriminant function classifiers and probabilistic generative models on the IRIS dataset. The Bayesian Classifier demonstrated the highest performance. In contrast, the One-Versus-The-Rest Classifier showed significant challenges, reflected in its lower average

Overall, the project provided comprehensive insights into the performance of different discriminant function classifiers on the IRIS dataset. The findings enhance the understanding of the strengths and limitations of each model, contributing valuable knowledge for the application of these methods in practical classification scenarios.