Linear Discriminant Analysis(LDA)

November 15, 2018

Group members names:- Kanjaria Mihir(AU1741065) Kukadiya Kevin(AU1741066) Krushna Shah(AU1741086) Yesha Shastri(AU1741035) Muskan Matwani(AU1741027)

Abstraction

Linear Discriminant Analysis (LDA) had been a popular method for extracting features which preserve class separability. LDA works better with large dataset having multiple classes. Class separability is an important factor while DR. The projection matrix of LDA is usually obtained by simultaneously maximizing the between-class covarince and minimizing the with-in class covariance. However, the computation of LDA involves dense matrices eigen-decomposition which is computationally expensive both in time and memory requirement when the number of samples and the number of features are large which is not the case here because here the number of features are very small. We can easily project data-points corresponds to features into lesser dimeson of space using LDA. It is widely used for image compression, SR and PR. ILDA, which is extended version of LDA also helps us to add more feature in classes without recalculating.

Keywords

Dimensional Reduction(DR) ,Data Classification (DC) ,Pattern Recognition(PR) ,Incremental Linear Discriminant Analysis(ILDA) ,Speech Recognition(SR)

Section I

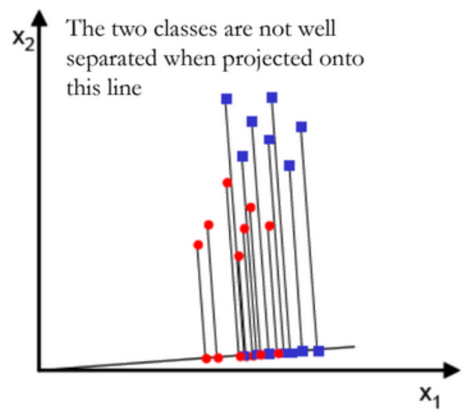
Introduction about the problem

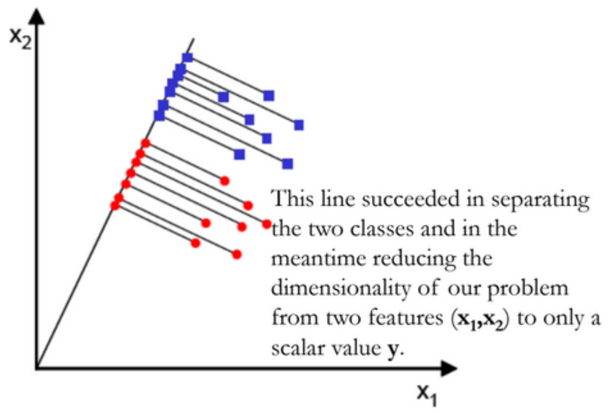
If there are more than two classes, then Linear Discriminant Analysis is the preferred linear classification technique. There are many possible techniques for classification of data. Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are two commonly used techniques for data classification and dimensionality reduction. Linear Discriminant Analysis easily handles the case where the within-class frequencies are unequal and their performances has been examined on randomly generated test data. This method maximizes the ratio of between-class variance to the within-class variance in any particular data set thereby guaranteeing maximal separability. The use of Linear Discriminant Analysis for data classification is applied to classification problem in speech recognition.We decided to implement an algorithm for LDA in hopes of providing better classification compared to Principal Components Analysis. The prime difference between LDA and PCA is that PCA does more of feature classification and LDA does data classification. In PCA, the shape and location of the original data sets changes when transformed to a different space whereas LDA doesn't change the location but only tries to provide more class separability and draw a decision region between the given classes.We first implement LDA algorithm which gives us profound eigen vector for projecting data point in matlab and then we build verilog programe for it such a way that it can be dumped into hardware i.e. on Zed Board.

Section II

Literature Review

Each class has N_i m-dimensional samples/vectors, where i = 1,2, ..., C. Hence we have a set of m-dimensional samples/vectors $\{x_1,x_2,...,x_Ni\}$ correspond to class C_i. Arranging these vectors from different classes into a big fat matrix X in which each column represents one vector. while projecting the vectors of X onto a hyperplane of dimension C-1 it result into Y from X. Assume we have m-dimensional samples $\{x_1,x_2,...,x_N\}$, In which N_1 belong to C_1 and N_2 belong to C_2. hence by projecting samples of X onto a line(C-1 space, C = 2) for each correspond X we would get scalar Y after projection. y = w^Tx (where w is the projection vectors used to project x to y.)


x_2
The two classes are not well separated when projected onto this line
x_1


x_2
This line succeeded in separating the two classes and in the meantime reducing the dimensionality of our problem from two features (x_1,x_2) to only a scalar value y.
x_1

Out of all this samples we would like to select only those vector which results into separability scalar value with maximum distance

$$\mu_i = \frac{1}{N_i}\sum_{x\in\omega_i} x \quad and \quad \tilde{\mu}_i = \frac{1}{N_i}\sum_{y\in\omega_i} y = \frac{1}{N_i}\sum_{x\in\omega_i} w^T x$$
$$= w^T \frac{1}{N_i}\sum_{x\in\omega_i} x = w^T \mu_i$$

i.e. for every X there is projection Y similarly for every mean of X there is mean of Y. We could then choose the distance between the projected means as our objective function
$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T\mu_1 - w^T\mu_2| = |w^T(\mu_1 - \mu_2)|$$
However, the distance between the classes is not the only good way to classify or deal with the separated data so to fulfill this requirement we are normalizing it with the with-in class distance which should be minimum. The solution proposed by Fisher is to maximize a function that represents the difference between the means, normalized by a measure of the within-class variability, or the so-called scatter. For each class we define the scatter, an equivalent of the variance, as; (sum of square differences between the projected samples and their class mean).

$$\tilde{s}_i^2 = \sum_{y\in\omega_i} (y - \tilde{\mu}_i)^2$$

S_i^2 (here capital S represent small s hat in the above equation) measures the variability within class C_i after projecting it on the y-space. Thus $S_1^2 + S_2^2$ measures the variability within the two classes at hand after projection, hence it is called within-class scatter of the projected samples. The Fisher linear discriminant is defined as the linear function w^Tx that maximizes the criterion function: (the distance between the projected means normalized by the within- class scatter of the projected samples.

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

Therefore, we will be looking for a projection where data points from the same class which are projected very close to each other and, at the same time, the projected means are as farther apart as possible. We will define a measure of the scatter in multivariate feature space x which are denoted as scatter matrices; $S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$

$$S_i = \sum_{x\in\omega_i} (x - \mu_i)(x - \mu_i)^T$$
$$S_w = S_1 + S_2$$

Where S_i is the covariance matrix of class C_i , and S_w is called the within-class scatter matrix.

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{w^T S_B w}{w^T S_W w}$$

Hence J(w) is a ratio of the difference between class means (which represented by the between-class scatter matrix) and within-class scatter matrix. Solving the generalized eigen value problem yields

$$S_W^{-1} S_B w = \lambda w \quad where \quad \lambda = J(w) = scalar$$

This is known as Fisher's Linear Discriminant, although it is not a discriminant but rather a specific choice of direction for the projection of the data down to one dimension. Using the same notation as PCA, the solution will be the eigen vector(s) of $S_x = S^-1_W S_B$

Verilog Approach

(1) HDL coder approach

We can directly convert the Matlab code to HDL code using HDL coder which comes as package inside the matlab. HDL coder uses fixed-point conversion method to handle floating number and it's operation. But the problem is HDL coder able to convert the code for a particular defined matrix (not generalized). So it's a good idea to write code in verilog manually rather than generating by HDL coder.

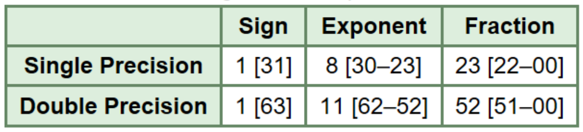
(2) Manual approach

While computing the LDA for a given matrix, in-between we have to deal with floating numbers like adding/subtraction/multiplication/division to floating number. So we need to represent those floating numbers in Binary IEEE Standards. Floating-point solves a number of representation problems. Fixed-point has a fixed window of representation, which limits it from representing both very large and very small numbers. Also, fixed-point is prone to a loss of precision when two large numbers are divided.

Floating-point, on the other hand, employs a sort of "sliding window" of precision appropriate to the scale of the number. This allows it to represent numbers from 1,000,000,000,000 to 0.0000000000000001 with ease, and while maximizing precision (the number of digits) at both ends of the scale.

IEEE floating point numbers have 3 basic component in it: (1)Mantissa (2)Exponent and (3)sign The following table shows the layout for single (32-bit) and double (64-bit) precision floating-point values. The number of bits for each field are shown, followed by the bit ranges in square brackets. 00 = least-significant bit. Here Sign bit represent the sign of Floating number, Exponent Bits represent the integer part of the Floating number, Fraction Bits represent the fraction part of the floating number.

Floating Point Components



	Sign	Exponent	Fraction
Single Precision	1 [31]	8 [30–23]	23 [22–00]
Double Precision	1 [63]	11 [62–52]	52 [51–00]

Laid out as bits, floating point numbers look like this:

Single(32-bit) :
SEEEEEEE EFFFFFFF FFFFFFFF FFFFFFFF
Double(64-bit) :
SEEEEEEE EEEEFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
S : Sign Bit
E : Exponent Bit
F : Fraction Bit

Below formula is used to calculate Decimal value from IEEE standards floating number :

Decimal Value = $(-1)^s\ (1+f)\ 2^{e-127}$

s : Binary Sign in Integer
e : Binary Exponent part in Integer
f : Binary Fraction part in Integer

• In order to compute Addition, Subtraction, and Multiplication of floating Number in verilog, we have used FPU(floating point unit) module of verilog which takes a single clock cycle to compute this operations.
• Apart from this operations, for Square-root and Division operation we have used in-built IP-Core(Intellectual property-core : it is an is a reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party.) libraries which takes 40-50 clock cycles to compute this operations.

Flow of Verilog code

Step-1 Creating Black and White classes from input matrix (Clock needed: 40)
Step-2 Calculating mean (Clock needed: 1)
Step-3 Subtraction of two means (Clock needed: 1)
Step-4 Multiplying subtracted matrix with it's transpose to calculate Sb (Clock needed: 1)
Step-5 Sb gets ready (Clock needed: 1)
Step-6 Calculating Sw (Clock needed: 1)
Step-7 Calculating Sw inverse (Clock needed: 50)
Step-8 Multiplying elements of Sw inverse and Sb (Clock needed: 1)
Step-9 Generating Objective matrix(i.e $Sw^{-1}*Sb$) (Clock needed: 1)
Step-10 Calculating eigen values of objective matrix (Clock needed: 50)
Step-11 Calculating eigen vectors of objective matrix (Clock needed: 50)

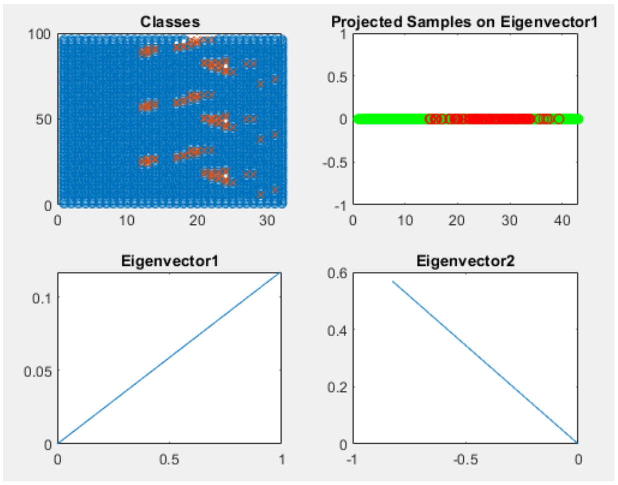
Things to remember:-
Addition or Multiplication or Subtraction operations takes a single clock cycle to compute and which has been implemented through FPU module

Square-root or Division operations takes 40 clock cycle to compute and which has been implemented through IP-Core directory
For safer side we are using 50 clock cycles in some steps

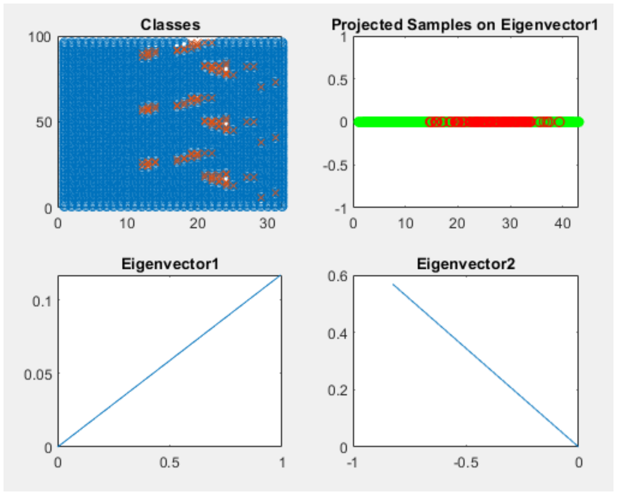
Section III

Approach Comparisons ::

Using Matlab Inbuilt Function:



Using Simulation through scripting:



Section IV

Conclusion

Since LDA (linear discriminant analysis) is used for data classification here we have classified matrix/image into two classes representing black and white pixels since number of classes is two so data classification is not too hard to compute but when the number of classes is a big number then lda is a effective method to separate out data and to classify it. In that case simulated and real data examples results suggest that, in the presence of irrelevant or redundant data, the LDA method can select important data for discriminant analysis and thereby yield improved classification.

References

(1) Shireen Elhabian and Aly A. Farag University of Louisville, CVIP Lab September 2009

(2) S. Balakrishnama, A. Ganapathiraju Institute for Signal and Information Processing Department of Electrical and Computer Engineering Mississippi State University Box 9571, 216 Simrall, Hardy Rd. Mississippi State, Mississippi 39762 Tel: 601-325-8335, Fax: 601-325-3149 Email: balakris, ganapath@isip.msstate.edu

(3) Introduction to Pattern Analysis Ricardo Gutierrez-Osuna Texas A and M University

(4) IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 35, NO. 5, OCTOBER 2005

(5) IEEE Standard 754 Floating Point Numbers by Steve Hollasch, 2019-08-24