CS 532

Pattern Recognition

Project

Linear Discriminant Analysis

By

Daniel Griffin, dgriffin5@wisc.edu, (9075594433)

Sparsh Agarwal, agarwal39@wisc.edu, (9075905142)

CONTENTS

OVERVIEW

Applying machine learning on data sets with lots of features is extremely complex. Many times, it can be nearly impossible. As any seasoned data analysis expert would tell you, feature representations can make or break your model. So, how can we learn accurate, predictive models when the data has a large number of features that can be noisy, redundant, or not particularly informative? One solution to this problem is to try to reduce the number of features a data set contains. Dimensionality reduction methods do this by combining features into more meaningful ones, and removing redundant ones.

Earlier in the course, the dimensionality reduction method called Principle Component Analysis (PCA) was introduced. This method uses distances from data points to candidate feature subspaces to find a good reduction in the number of features for the data set. One of the down sides of PCA is that it only considers data features, and completely ignores data class labels. You might call it an 'unsupervised' dimensionality reduction method since it ignores class labels. This raises the question, is there some way to find a subspace that uses both the data features, and the class labels?

Linear Discriminant Analysis (LDA) does exactly this. LDA is one of the oldest and most widely used 'supervised' dimensionality reduction methods. LDA finds a lower dimensional subspace that not only fits the data well, but also a subspace that makes data points easier to classify. Furthermore, with only a few modifications, LDA can be extended for use as a classifier.

Someone who completes this lab will learn about the conceptual underpinnings for why LDA works, the graphical intuition of how LDA works from data plots, as well as the mathematical derivation of how it works. A reader will also learn how LDA relates to PCA, and under what circumstances LDA will outperform PCA. The reader will then learn how to apply LDA in a real world classification task with a Least Squares classifier, and how to extend LDA for classification tasks. Finally, a reader who completes this lab will know the low level algorithms necessary for performing LDA.

This lab starts off with a background section describing the historical significance of LDA, and how it arose. It then describes the conceptual underpinnings of LDA, and how it compares to PCA. Next, it explains the mathematical derivation of LDA, building off of the conceptual underpinnings previously presented. Then, it clearly and concisely outlines the 7 step algorithm to perform LDA on a data set. The next section is the warm up section, which poses questions about the iris data set to familiarize the reader with the mechanics of the LDA algorithm, and to further solidify the conceptual understanding of LDA through visual graphs. The section after this is the lab section. In it, a real world bank fraud data set is set up as a classification task, and the reader is asked to implement both LDA and PCA to reduce the data set's dimensionality. The reader is then asked to apply least squares classification on both reduced data sets, and to discuss the results. Finally, the reader learns how to extend LDA to perform classification on this data set. The lab document concludes with a page of cited references, and an appendix with laboratory solutions, and the code to generate them.

BACKGROUND

THE DIMENSIONALITY REDUCTION PROBLEM AND ITS HISTORY

Within machine learning, there exists a problem called 'the curse of dimensionality'. This problem, roughly stated, says that the ability of a predictive model to learn from data becomes exponentially more complicated as the number of features of a data set increases. To illustrate, say that we are given a data set of flowers with the features 'color' and 'shape'. Most predictive models would be able to learn to predict flower type fairly easily given only these two features. Now say that we are given a data set of flowers with 20,000 features. For most models, it is almost impossible to learn a predictive function. There are just too many features to choose from when creating a predictive function. What often happens is that a model will use a relatively small amount of information from each feature to build a predictive function. Usually, this small amount of information is a random idiosyncrasy of the data set with no real relationship to a class label. The result is a predictive model that works well on its training set, but is completely incapable of predicting labels over new data sets.

To overcome this problem, methods to reduce the number of redundant features, or create more meaningful features in a data set have been proposed. The group of methods we introduce in this lab are referred to as 'dimensionality reduction' methods.

The idea of finding the best features of a data set has a long history, originating with the first developments of statistical inference dating back to the works of Charles Peirce in 1877. The major focus in those days was on the predictive capability of the models, rather than the relevance of individual features. It was in 1936 that Dr. Fisher developed a method for determining feature subsets that can enhance the predictive capabilities of a model. His original paper can be found in the lab references section. The method he proposed is called Linear Discriminant Analysis (LDA) today, and is widely used for dimensionality reduction in classification problems. Using LDA, a lower dimensional feature subspace of a data set can be found that often increases the predictive capability of a classification function.

THE CONCEPTUAL MATHEMATICAL UNDERSTANDING OF DIMENSIONALITY REDUCTION

The key goal of dimensionality reduction methods is to find a small subset of the features that still represent a majority of the data set's information. Conceptually, these methods work by removing redundant features, and combining features into more meaningful ones. This removal and combination process is performed by a mathematical method called projection. The key idea is that, given data with a high number of dimensions, find the data point that exists in a lower dimension that is closest to the data point in the higher dimension space. The goal of dimensionality reduction methods is to find the 'best' lower dimensional space to project data onto. The assumptions about how to quantify 'redundant' and 'meaningful' features distinguish dimensionality reduction methods, and dictate how to define a projection subspace. Many of the metrics used to quantify subspace quality are defined either in terms of minimum distances, or in terms of minimizing statistical properties of the projected data set. In the next section we describe the conceptual underpinning of the metrics used for LDA, and compare them with PCA.

PCA AND LDA CONCEPTUALLY

PCA is a dimensionality reduction method that works by finding a projection subspace that minimizes the sum of distances from every point to the projection line $\boldsymbol{w}$. Figure 1 shows this process, where the o's and x's are the original data points, the blue line is the optimal PCA projection subspace, and the green line represents the distance from one of the original data points to the projection line. The only consideration in quantifying a 'good' projection is the distance between the projection line $\boldsymbol{w}$ and the data point. No class label information is used by PCA to determine the projection line. In the scenario provided below, it is clear that PCA learns a lower dimensional subspace that would be very difficult to classify data in. There are many overlaps between the 'o' class and 'x' class in the projected data space, with no clear separating boundary between classes. So, while the dimensionality has been reduced, classification has been made more difficult.

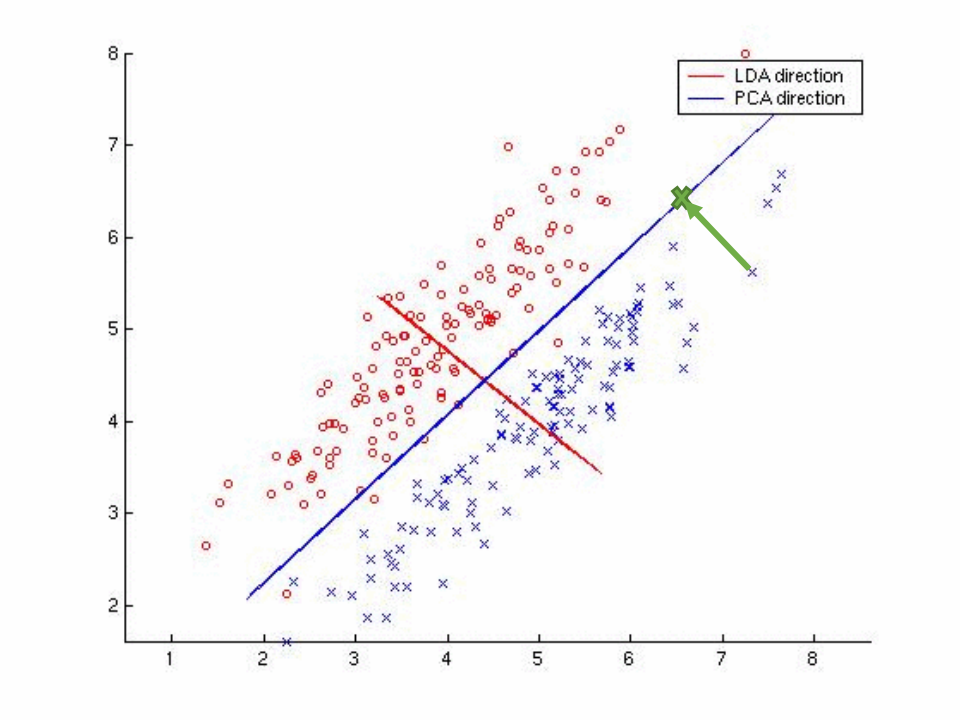


Figure 1

As previously stated, PCA performs this quantification by minimizing the sum of the distances of each initial data point to the data point resulting from a projection onto the subspace (the green line). This is equivalent to solving the following optimization problem.

$$\min_{\boldsymbol{w}} \sum_{\boldsymbol{i=1}}^{|\boldsymbol{D}|} d_i^2 \, , where \; d_i^2 = ||\boldsymbol{a}_i - proj_{\boldsymbol{w}} \boldsymbol{a}_i||^2$$

As shown previously in this course, the solution to this optimization problem is the subspace spanned by the k-largest singular vectors found from the singular vector decomposition of data matrix A. A review of the derivation and proof can be found in the paper *PCA versus LDA* by Martinez et. al. in the references section.

$$\boldsymbol{A = U \sum V^T}$$

$$\boldsymbol{w = v_1} \; in \; the \; 1 \; dimension \; case, and \; \boldsymbol{W = [v_1 \ldots v_k]} \; in \; the \; k \; dimension \; case.$$

LDA works in a way very similar to PCA, but uses different metrics for optimization. Rather than optimizing data point distances to the projection space, LDA works by calculating metrics using the class labels of data points, and optimizes those. Specifically, LDA works by optimizing relationships between the mean data point positions of the two classes. First, LDA tries to maximize the distance between the projected means of both classes. Second, LDA tries to minimize the sum of distances of each projected data point to its projected class mean, also known as the 'scatter' of the classes. As shown in Figure 2 below, finding the LDA optimal subspace amounts to finding a subspace that will maintain good projected class separation. In the case for the data set shown in Figure 2, while classifying data projected onto the PCA subspace will result in extremely poor performance, classifying data projected onto the LDA subspace will result in very high performance.

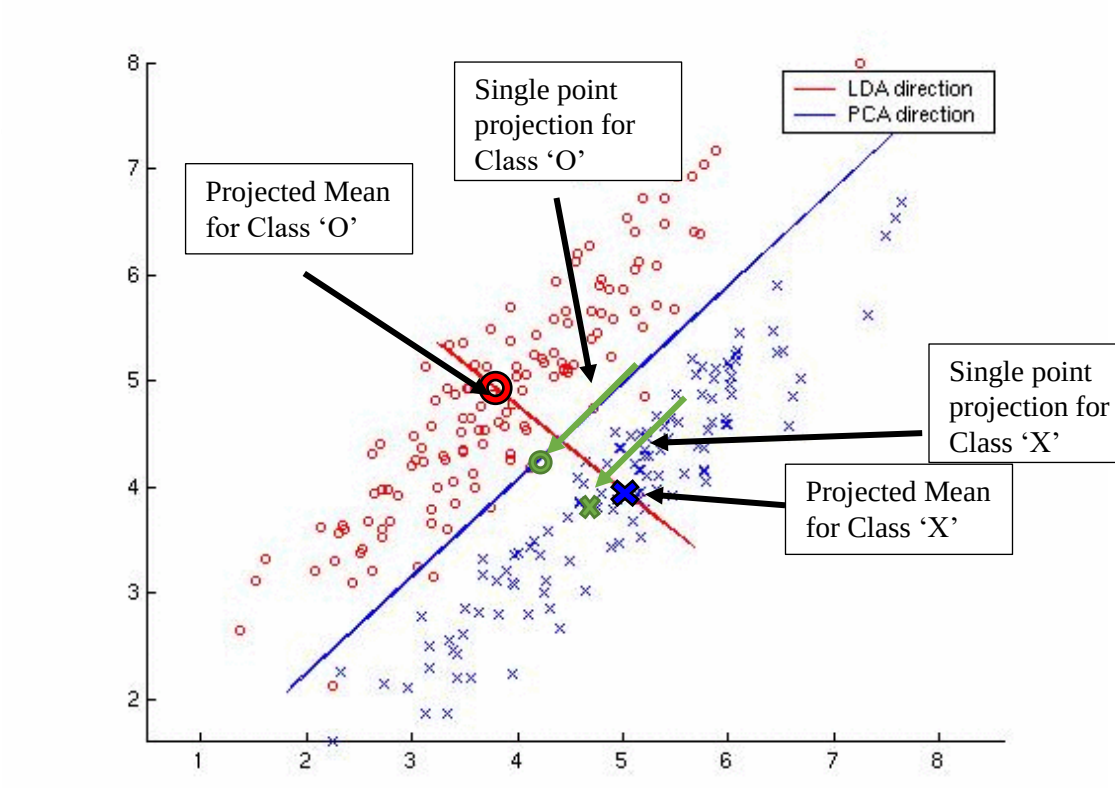


Figure 2

LDA MATHEMATICAL DERIVATION

In this section, we work through a linear algebra based mathematical derivation of LDA. We start by defining class means, scatter, and the Fisher LDA Objective function. We then derive the objective function terms in relation to the input data matrix X. Next, we derive how to optimize the objective function using vector calculus. Finally, we show how the optimization problem reduces to a generalized eigenvalue problem. This derivation closely follows the one found in *Data Mining and Analysis* by Mohammed Zaki and Wagner Meira, which can be found in the lab references section.

REVIEW OF EIGENVALUES AND EIGENVECTORS

In linear algebra, an **eigenvector** or **characteristic vector** of a linear transformation is a non-zero vector that does not change its direction when that linear transformation is applied to it. More formally, if T is a linear transformation from a vector space V over a field F into itself and $\mathbf{v}$ is a vector in V that is not the zero vector, then $\mathbf{v}$ is an eigenvector of T if $T(\mathbf{v})$ is a scalar multiple of $\mathbf{v}$. This condition can be written as the equation

$$T(\mathbf{v}) = \lambda\mathbf{v}$$

where λ is a scalar in the field F, known as the **eigenvalue**, **characteristic value**, or **characteristic root** associated with the eigenvector $\mathbf{v}$.

If the vector space V is finite-dimensional, then the linear transformation T can be represented as a square matrix A, and the vector $\mathbf{v}$ by a column vector, rendering the above mapping as a matrix multiplication on the left hand side and a scaling of the column vector on the right hand side in the equation

$$A\mathbf{v} = \lambda\mathbf{v}$$

There is a correspondence between n by n square matrices and linear transformations from an n-dimensional vector space to itself. For this reason, it is equivalent to define eigenvalues and eigenvectors using either the language of matrices or the language of linear transformations.

Geometrically an eigenvector, corresponding to a real nonzero eigenvalue, points in a direction that is stretched by the transformation and the eigenvalue is the factor by which it is stretched. If the eigenvalue is negative, the direction is reversed.

DEFINING THE MEANS

Let $\boldsymbol{D_i}$ represent the subset of data points labeled with class c_i, $|\boldsymbol{D_i}| = n_i$ be the number of points with class c_i, $\boldsymbol{w}$ be the weight vector for the projection line, and $a_i = \boldsymbol{w^T x_i}$ be the coordinate of the projection of $\boldsymbol{x_i}$ onto $\boldsymbol{w}$. Each projected data point a_i has an associated class label y_i. Therefore, we can compute the mean of the projected data points m_i as shown below:

$$\begin{aligned} m_1 &= \frac{1}{n_1}\sum_{\boldsymbol{x_i}\in\boldsymbol{D_1}} a_i \\ &= \frac{1}{n_1}\sum_{\boldsymbol{x_i}\in\boldsymbol{D_1}} \boldsymbol{w^T x_i} \\ &= \boldsymbol{w^T}\left(\frac{1}{n_1}\sum_{\boldsymbol{x_i}\in\boldsymbol{D_1}} \boldsymbol{x_i}\right) \\ &= \boldsymbol{w^T \mu_1} \end{aligned}$$

Similarly, it can also be shown that $m_2 = \boldsymbol{w^T \mu_2}$.

The above equations show that the mean of the projected data points m_i can be written in terms of the mean of the data points in the original data space $\boldsymbol{\mu_i}$. The intuitive objective is that classification in the lower dimensional subspace would be easier if the means m_i of classes in the projected space are well separated. Thus, our first objective is to maximize $|m_1 - m_2|$.

DEFINING THE SCATTER MATRIX

While maximizing the differences between class means is important for good separation, large variances in the projected data points could lead to overlaps between the two classes. So, for the second objective, LDA tries to minimize the spread of the projected data points around their respective projected means m_i. To do this, LDA defines a 'scatter' term for class c_i as s_i^2. The equation for scatter is:

$$s_i^2 = \sum_{\boldsymbol{x_j \in D_i}} (a_j - m_i)^2$$

The above equation is the total squared deviation from the mean. It is similar to the equation for statistical variance, except that variance is the average deviation from the mean.

DEFINING THE FISHER LDA OBJECTIVE

So, the objective of LDA is to maximize $|m_1 - m_2|$, while also minimizing the total scatter $s_1^2 + s_2^2$. To do this, we define the error function and optimization problem as follows:

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$\max_{\boldsymbol{w}} J(\boldsymbol{w})$$

The above error ratio objective function is often referred to as the 'Fisher LDA Objective'. It is maximized when the difference between projected means is high, and the total scatter is low.

DEFINING THE FISHER LDA OBJECTIVE IN TERMS OF THE DATA

We now have an optimization problem that we're trying to solve. However, all of the terms are defined in the projected data space. To optimize the loss function using our data, we must rework the terms such that they are defined in relation to our original data set. To start, we rewrite the numerator term as shown below:

$$\begin{aligned}(m_1 - m_2)^2 &= \left(\boldsymbol{w^T(\mu_1 - \mu_2)}\right)^2 \\ &= \boldsymbol{w^T\left((\mu_1 - \mu_2)(\mu_1 - \mu_2)^T\right)w} \\ &= \boldsymbol{w^T B w}\end{aligned}$$

Where $\boldsymbol{B} = \boldsymbol{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}$ is a dxd, rank-one matrix called the 'between-class scatter matrix'. Next, we rewrite the denominator term as follows:

$$s_1^2 = \sum_{\boldsymbol{x_i \in D_1}} (a_i - m_1)^2$$

$$
\begin{aligned}
&= \sum_{\boldsymbol{x_i \in D_1}} \left(\boldsymbol{w^T x_i - w^T \mu_1}\right)^2 \\
&= \sum_{\boldsymbol{x_i \in D_1}} \left(\boldsymbol{w^T (x_i - \mu_1)}\right)^2 \\
&= \boldsymbol{w^T} \left(\sum_{\boldsymbol{x_i \in D_1}} \boldsymbol{(x_i - \mu_1)(x_i - \mu_1)^T} \right) \boldsymbol{w} \\
&= \boldsymbol{w^T S_1 w}
\end{aligned}
$$

Similarly, it can also be shown that $s_2^2 = \boldsymbol{w^T S_2 w}$.

The above equation is used to compute both s_1^2 and s_2^2. These two equations can be recombined as follows:

$$s_1^2 + s_2^2 = \boldsymbol{w^T S_1 w} + \boldsymbol{w^T S_2 w} = \boldsymbol{w^T (S_1 + S_2) w} = \boldsymbol{w^T S w}$$

Where $\boldsymbol{S = S_1 + S_2}$ denotes the 'within-class scatter matrix'. Our reformulated optimization in terms of the original data space can be written as follows:

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

OPTIMIZING THE FISHER LDA OBJECTIVE

Now, we must find the maximum of the loss function. To do this, we take the derivative of the loss function with respect to w, and set it equal to 0. Recall that if $f(x)$ and $g(x)$ are two functions that:

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - g'(x)f(x)}{g(x)^2}$$

Taking the derivative of the loss function with respect to w gives:

$$\frac{d}{d\boldsymbol{w}}\big(J(\boldsymbol{w})\big) = \frac{2\boldsymbol{Bw}\big(\boldsymbol{w^T S w}\big) - 2\boldsymbol{Sw}(\boldsymbol{w^T B w})}{(\boldsymbol{w^T S w})^2}$$

Solving this equation yields:

$$
\begin{aligned}
\boldsymbol{Bw}\big(\boldsymbol{w^T S w}\big) &= \boldsymbol{Sw}\big(\boldsymbol{w^T B w}\big) \\
\boldsymbol{Bw} &= \boldsymbol{Sw}\left(\frac{\boldsymbol{w^T B w}}{\boldsymbol{w^T S w}}\right) \\
\boldsymbol{Bw} &= J(\boldsymbol{w})\boldsymbol{Sw} \\
\boldsymbol{Bw} &= \lambda \boldsymbol{Sw}
\end{aligned}
$$

Where $\lambda = J(\boldsymbol{w})$. This represents what's called a 'generalized eigenvalue problem' where lambda is an eigenvalue. Thus, $J(w)$ is maximized when $\lambda = J(\boldsymbol{w})$ is maximized. Thus, λ is the largest eigenvalue, and $\boldsymbol{w}$ is the largest eigenvector that produces that eigenvalue. Thus, the dominant eigenvector specifies the best linear discriminant vector $\boldsymbol{w}$.

SOLVING THE LDA EIGENVALUE PROBLEM

If S is invertible, then the generalized eigenvalue problem can be reduced to a regular eigenvalue problem as follows:

$$\boldsymbol{Bw} = \lambda \boldsymbol{Sw}$$
$$\boldsymbol{S^{-1}Bw} = \lambda \boldsymbol{S^{-1}Sw}$$
$$\boldsymbol{S^{-1}Bw} = \lambda \boldsymbol{w}$$

THE LDA ALGORITHM

The pseudo-code for the algorithm is summarized in the 7 following lines:

LDA($\boldsymbol{D} = \{(\boldsymbol{x_i}, y_i)\}, i = 1, \dots, n$):

1. $\boldsymbol{D_i} = \{\boldsymbol{x_j} \mid y_i = c_i, j = 1, \dots, n\}, i = 1, 2$ //Form the class specific subsets.
2. $\boldsymbol{\mu_i} = mean(\boldsymbol{D_i}), i = 1, 2$ //Find class means
3. $\boldsymbol{B} = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^{\boldsymbol{T}}, i = 1, 2$ //Find the between class scatter
4. $\boldsymbol{Z_i} = \boldsymbol{D_i} - \boldsymbol{1_{n_i}\mu_i^T}, i = 1, 2$ //Center class matrix
5. $\boldsymbol{S_i} = \boldsymbol{Z_i^T Z_i},\ i = 1, 2$ //Class scatter matrices
6. $\boldsymbol{S} = \boldsymbol{S_1} + \boldsymbol{S_2}$ //Within class scatter matrices
7. $\lambda_1, \boldsymbol{w} = eigen(\boldsymbol{S^{-1}B})$ //Compute dominant eigenvalue and eigenvector

The derivation we've shown describes the case for finding the best 1-dimensional linear discriminant projection. The best k-dimensional LDA subspaces uses an analogous proof, and results in being the k eigenvectors with the k largest eigenvalues. This is very similar to how PCA's k-dimensional subspace is the k singular vectors with k largest singular values. To produce a lower dimensional subspace for LDA, simply project each data point onto the matrix formed by the k largest eigenvectors.

WARM UP

The following warm up questions serve to provide experience with both an eigenvector and SVD solver, and a step by step work through of applying. The data set to use for questions 3 and 4 is a reduced version of the fisher iris data set. The original data set can be found here: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data. The reduced fisher iris data set contains only the first two numeric features, and numeric class labels of -1 for the first 50 instances, and 1 for the last 100 instances.

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{vmatrix}$$

1. Given the above matrix A, use an eigenvector solver to find the eigenvectors.
2. Given the same matrix A, use SVD to find the right singular vectors.
3. Find the class means, between class scatter matrix, and within class scatter matrix for the reduced fisher data set.
4. Find the optimal LDA projection line $\boldsymbol{w}$ for the reduced iris data set (We provide this pre-labeled as binary classes), and plot both the original data points and $\boldsymbol{w}$ shifted by the data mean in a 2D plot. Use the points $(cw[0] + mu[0], cw[1] + mu[1])$ where $\boldsymbol{c}$ ranges from -4 to 4, and $\boldsymbol{mu}$ is the mean data point vector of the entire data set. In the plot, $\boldsymbol{w}$ should be shown passing through the data points.

LAB

The following lab questions use a real world data set for a bank note authentication task. The data set specifies 4 features of banknotes, with a label specifying whether the banknote was authentic or a fake. The using this link: https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt

1. Using 'data_banknote_authentication.txt', perform PCA using the SVD and compute the projected 1-D coordinates. The first 4 columns of the data set are features, and the last column is the class label. List the 1-D coordinates for the first 10 data points in the PCA projected data set.
2. Perform LDA and compute the projected 1-D coordinates. You may solve either the generalized eigenvalue problem, or the regular eigenvalue problem. List the 1-D coordinates for the first 10 data points in the LDA projected data set.
3. Perform LS classification on the PCA reduced data, and LDA reduced data, and calculate accuracy for both. List the accuracy for the LS classifier trained on the PCA data set, and the accuracy for the LS classifier trained on the LDA data set.
4. Plot the projected 1-D PCA coordinates, and the 1-D LDA coordinates. In these graphs, plot the projected data points with class = -1 at height y=0, and points with class = 1 at height y=1 for both the LDA and PCA points respectively. The x coordinate should be the 1-D projection value for each point. Using the graph, explain why it makes sense that LS on the LDA data set would result in better accuracy than PCA.
5. LDA can be used as both a dimensionality reduction, and classification method. LDA is first used to reduce dimensionality. Then, a threshold 'c' is chosen for the classification rule $\boldsymbol{w^T x} < c$. Choose a threshold 'c' in the 1-D LDA space to best discriminate classes for our data set using the rule $\boldsymbol{w^T x}$, where $\boldsymbol{x}$ such that $\boldsymbol{w^T x} > c$ belongs to class label -1, and $\boldsymbol{x}$ such that $\boldsymbol{w^T x} \leq c$ belongs to class label 1. Calculate the accuracy of your classifier. How does this compare to the accuracy of LS applied to the 1-D LDA space?

REFERENCES

Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, *7*(2), 179–188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x

Osuna, R., Linear Discriminant analysis ppt, retrieved from: https://datajobs.com/data-science-repo/LDA-Primer-[Gutierrez-Osuna].pdf

Cunningham, J., Linear Discriminant Reduction, Journal of Machine Learning Research, 16, retrieved from: http://jmlr.org/papers/volume16/cunningham15a/cunningham15a.pdf

Pierce, C., (1877–1878), Origins of statistical inference, and of meaningful data for prediction, Illustrations of the Logic of Science, *Popular Science Monthly*, 12–13

Data Mining and Analysis Main/Home Page. (n.d.). Retrieved December 9, 2016, from http://www.dataminingbook.info/pmwiki.php/Main/HomePage

Wall, Michael E., Andreas Rechtsteiner, Luis M. Rocha."Singular value decomposition and principal component analysis". in A Practical Approach to Microarray Data Analysis. D.P. Berrar, W. Dubitzky, M. Granzow, eds. pp. 91-109, Kluwer: Norwell, MA (2003). LANL LA-UR-02-4001.

Examples: Statistical Pattern Recognition Toolbox for Matlab. (n.d.). Retrieved December 9, 2016, from https://cmp.felk.cvut.cz/cmp/software/stprtool/examples.html#ldapca_example

Martinez, A. M., & Kak, A. C. (2001). PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(2), 228–233. https://doi.org/10.1109/34.908974

UCI Machine Learning Repository: banknote authentication Data Set. (n.d.). Retrieved December 9, 2016, from https://archive.ics.uci.edu/ml/datasets/banknote+authentication

“Eigenvalues and Eigenvectors.” *Wikipedia*, December 9, 2016. https://en.wikipedia.org/w/index.php?title=Eigenvalues_and_eigenvectors&oldid=753796634.

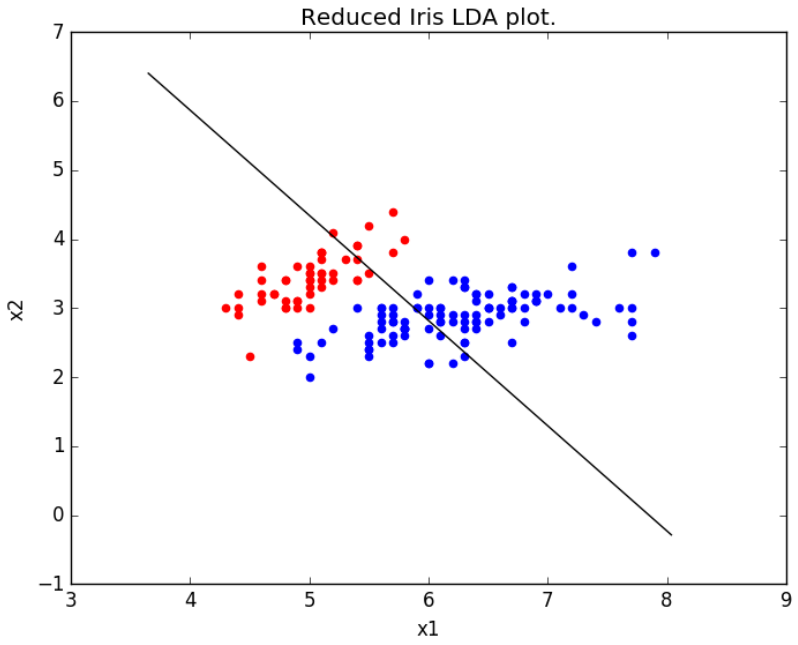
APPENDIX

WARM UP ANSWERS

WARM UP PARTS 1, 2, AND 3 ANSWERS

```
python FinalMain.py
[[1 0 0]
 [0 2 0]
 [0 0 3]]
Eigenvectors of A.
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
Eigenvalues of A.
 1.  2.  3.]
[[1 0 0]
 [0 2 0]
 [0 0 3]]
Singular vectors of A.
[[ 0.  0.  1.]
 [ 0.  1.  0.]
 [ 1.  0.  0.]]
Class 1 mean: [[ 5.006]
 [ 3.428]]
Class 2 mean: [[ 6.262]
 [ 2.872]]
Between class scatter matrix: [[ 1.577536 -0.698336]
 [-0.698336  0.309136]]
Within class scatter matrix: [[ 49.5838  16.9552]
 [ 16.9552  18.0024]]
```

WARM UP PART 4 ANSWER



LAB ANSWERS

LAB PART 1 ANSWER

```
[[ -9.25782979]
 [ -8.9755722 ]
 [  2.62497917]
 [-11.09306297]
 [  5.78927538]
 [-11.25053141]
 [ -2.58413563]
 [  9.50991944]
 [ -5.76824804]
 [ -9.16625409]]
```

LAB PART 2 ANSWER

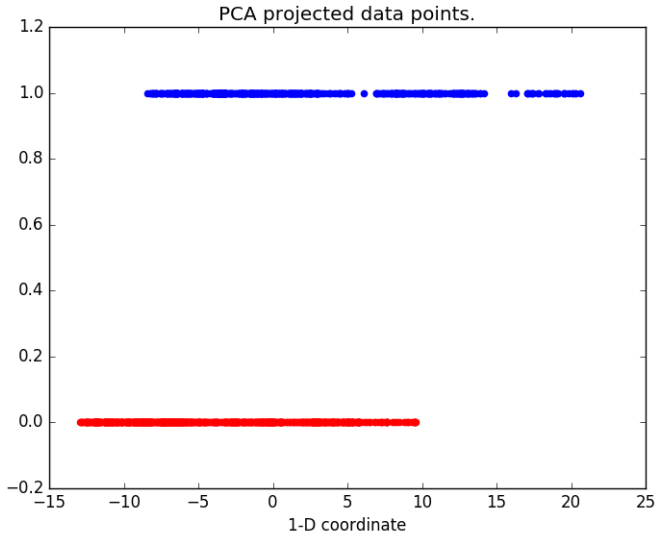
```
[[ 4.74100656]
 [ 5.40504689]
 [ 2.81674537]
 [ 4.31717522]
 [ 0.84395825]
 [ 5.08443029]
 [ 4.28817676]
 [ 3.25666827]
 [ 4.33051639]
 [ 3.68146277]]
```

LAB PART 3 ANSWER

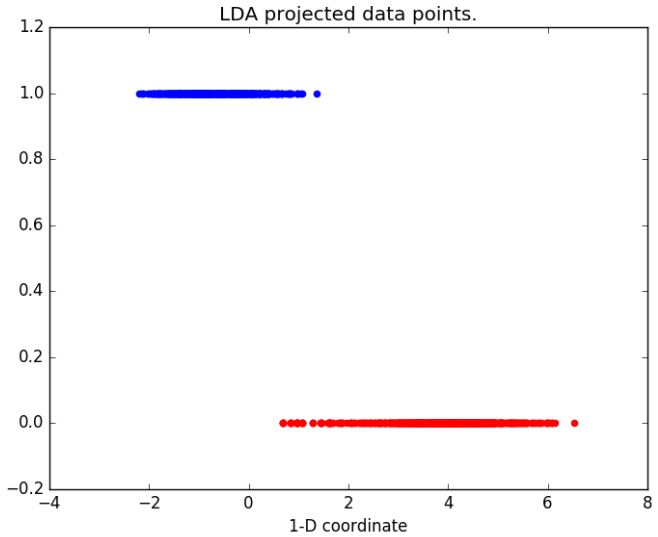
```
PCA LS classification accuracy: 0.444606413994
LDA LS classification accuracy: 0.666909620991
```

LAB PART 4 ANSWERS

PCA Plot



LDA Plot



In the above plots, it is clear that the 1-D coordinates for the LDA projected data set has very little overlap. This means that it will be easier to find a linear separator with high accuracy. In the above PCA 1-D coordinates, there is a lot of overlap between the coordinates for class 1 and class 0. This means that whatever linear model that is fit to the data set will have poor classification accuracy.

LAB PART 5 ANSWERS

We chose c = 1.5 since the 1-D LDA plot of the coordinates from part 4 indicated that a vertical separating boundary between classes is located at roughly this position.

```
LDA threshold classification accuracy: 0.976676384840
```

This accuracy is much higher than the accuracy of the LS classifier model. This higher performance is likely due to the LS classifier issue of being biased by well classified data points. The well classified data points of the class shown in red in the LDA plot from part 4 likely pulled the LS decision boundary towards the right, thus reducing its overall accuracy.

WARM UP AND LAB CODE

```
import numpy as np
import scipy.linalg as sp_linalg
import matplotlib.pyplot as plt

def main():
    warmUp()
    lab()

def warmUp():
    data = np.loadtxt('./fisher.csv', delimiter=',')

    #PART A WARM UP.
    #Find eigenvalues and eigen-vectors.
    A = np.diag((1, 2, 3))
    print(A)
    w, v = np.linalg.eig(A)
    print('Eigenvectors of A.')
    print(v)
    print('Eigenvalues of A.')
    print(w)

    # PART B WARM UP.
    # Find singular vectors of A.
    A = np.diag((1, 2, 3))
    print(A)
    U, s, V = np.linalg.svd(A)
    print('Singular vectors of A.')
    print(V)

    # PART C WARM UP.
    #Get data set for c1
    D1 = data[data[:, -1] == -1]
    D1 = D1[:, :-1]
    #Get data set for c2
    D2 = data[data[:, -1] == 1]
    D2 = D2[:, :-1]
    meanc1 = D1.mean(axis=0)
    meanc1 = meanc1.reshape((D1.shape[1], 1))
    meanc2 = D2.mean(axis=0)
    meanc2 = meanc2.reshape((D2.shape[1], 1))
    #Calculate B
    B = np.dot(meanc1 - meanc2, (meanc1 - meanc2).T)
    #Calculate centered class matricies.
    Z1 = D1 - np.dot(np.ones((D1.shape[0],1)), meanc1.T)
    Z2 = D2 - np.dot(np.ones((D2.shape[0],1)), meanc2.T)
    #Calculate scatter matricies Si
    S1 = np.dot(Z1.T, Z1)
    S2 = np.dot(Z2.T, Z2)
    #Calculate within class scatter matrix S.
    S = S1 + S2
    #Solve the reduced eigenvalue problem.
    eigVals, eigVects = sp_linalg.eig(B, S, left=True, right=False)
    w = eigVects[:, 0]
    w = w.reshape((w.size, 1))

    print('Class 1 mean: ' + str(meanc1))
    print('Class 2 mean: ' + str(meanc2))
    print('Between class scatter matrix: ' + str(B))
    print('Within class scatter matrix: ' + str(S))

    xList = []
    yList = []
    dataMean = data[:, :-1].mean(axis=0)
    for c in np.linspace(-4, 4, 80):
```

```
            xList.append(c * w[0] + dataMean[0])
            yList.append(c * w[1] + dataMean[1])

        #Create plots
        plt.scatter(data[:50, 0], data[:50, 1], color='r')
        plt.scatter(data[50:, 0], data[50:, 1], color='b')
        plt.plot(xList, yList, color='black')
        plt.xlabel('x1')
        plt.ylabel('x2')
        plt.title('Reduced Iris LDA plot.')
        plt.show()

def lab():
    print('Running lab...')
    # data = np.loadtxt('./bcdata.csv', delimiter=',')
    data = np.loadtxt('./data_banknote_authentication.csv', delimiter=',')
    X = data[:, :-1]

    '''
    PART A
    '''
    #Perform SVD and get first singular vector.
    U, s, V = np.linalg.svd(X)
    v1 = V[0, :]
    v1 = v1.reshape((v1.size, 1))
    pcaData = np.dot(v1.T, X.T).T
    #Print first 10 projected data coordinates.
    print(pcaData[:10])

    '''
    PART B
    '''
    D1 = data[data[:, -1] == 0]
    D1 = D1[:, :-1]
    # Get data set for c2
    D2 = data[data[:, -1] == 1]
    D2 = D2[:, :-1]
    meanc1 = D1.mean(axis=0)
    meanc1 = meanc1.reshape((D1.shape[1], 1))
    meanc2 = D2.mean(axis=0)
    meanc2 = meanc2.reshape((D2.shape[1], 1))
    # Calculate B
    B = np.dot(meanc1 - meanc2, (meanc1 - meanc2).T)
    # Calculate centered class matricies.
    Z1 = D1 - np.dot(np.ones((D1.shape[0], 1)), meanc1.T)
    Z2 = D2 - np.dot(np.ones((D2.shape[0], 1)), meanc2.T)
    # Calculate scatter matricies Si
    S1 = np.dot(Z1.T, Z1)
    S2 = np.dot(Z2.T, Z2)
    # Calculate within class scatter matrix S.
    S = S1 + S2
    # Solve the reduced eigenvalue problem.
    eigVals, eigVects = sp_linalg.eig(B, S, left=True, right=False)
    w = eigVects[:, 0]
    w = w.reshape((w.size, 1))

    #Project onto the LDA space.
    ldaData = np.dot(w.T, X.T).T
    #Print first 10 projected data coordinates.
    print(ldaData[:10])

    '''
    PART C
    '''
    #LS on PCA data.
    pcaX = np.hstack((pcaData, np.ones((pcaData.shape[0], 1))))
    pcaW = np.linalg.pinv(pcaX).dot(data[:, -1])
    pcaPredicted = []
```

```
    for row in pcaX:
        out = np.dot(pcaW, row)
        if out >= 0:
            pcaPredicted.append(1)
        else:
            pcaPredicted.append(0)
    pcaAccuracy = calculateAccuracy(data[:, -1], pcaPredicted)
    print('PCA LS classification accuracy: %.12f' % pcaAccuracy)
    #LS on LDA data.
    ldaX = np.hstack((ldaData, np.ones((ldaData.shape[0], 1))))
    ldaW = np.linalg.pinv(ldaX).dot(data[:, -1])
    ldaPredicted = []
    for row in ldaX:
        out = np.dot(ldaW, row)
        if out >= 0:
            ldaPredicted.append(1)
        else:
            ldaPredicted.append(0)
    ldaAccuracy = calculateAccuracy(data[:, -1], ldaPredicted)
    print('LDA LS classification accuracy: %.12f' % ldaAccuracy)

    '''
    PART D
    '''
    #PCA plot.
    X0 = pcaData[data[:, -1] == 0]
    X1 = pcaData[data[:, -1] == 1]
    plt.scatter(X0 , np.zeros(X0.size),color='r')
    plt.scatter(X1, np.ones(X1.size),color='b')
    plt.xlabel('1-D coordinate')
    plt.title('PCA projected data points.')
    plt.show()

    #LDA plot.
    X0 = ldaData[data[:, -1] == 0]
    X1 = ldaData[data[:, -1] == 1]
    plt.scatter(X0 , np.zeros(X0.size),color='r')
    plt.scatter(X1, np.ones(X1.size),color='b')
    plt.title('LDA projected data points.')
    plt.xlabel('1-D coordinate')
    plt.show()

    '''
    PART E
    '''
    c = 1.5
    thresholdPredicted = []
    for projPoint in ldaData:
        if projPoint <= c:
            thresholdPredicted.append(1)
        else:
            thresholdPredicted.append(0)
    thresholdAccuracy = calculateAccuracy(data[:, -1], thresholdPredicted)
    print('LDA threshold classification accuracy: %.12f' % thresholdAccuracy)

def calculateAccuracy(yactual, ypredicted):
   metrics = {}
   metrics["accuracy"] = 0

   for i in range(0, len(yactual)):
      if ypredicted[i] == yactual[i]:
         metrics["accuracy"] += 1

   metrics["accuracy"] = metrics["accuracy"] / float(len(yactual))
   return metrics["accuracy"]

if __name__ == '__main__':
    main()
```