Constructing Optimal Subspaces for Pattern Classification

Avinash Kak

Purdue University

November 29, 2022

6:08pm

An RVL Tutorial Presentation

Originally presented in Summer 2008
Updated most recently in November 2022

CONTENTS

Back to TOC

1: Introduction

- Measurements meant to be used for machine learning applications are often made in high-dimensional spaces. Unfortunately, the higher the dimensionality of the space in which the information is being collected, the greater the likelihood that your final results will not be so reliable.

- Consider a non-deep-learning [see comment at the end of this section] based face-recognition experiment in which we want to compare a test photo with the images stored in a database of mugshots, these being photos typically taken by police after someone is arrested. [A mugshot is a frontal image of a face, with the human subject looking straight at the camera. Before they are entered in a database, the images are "normalized" — these days with software tools that can do the job automatically — so that only a rectangular region bounded by the lines just outside the ears, the line just above the head, and the line just below the chin is retained.]

- Let us say that each mugshot is of size $m \times n$, with m and n being typically a number between 480 and 600. Each mugshot can be thought of as a vector in an mn dimensional measurement space. At the lower end of the range, if $m = n = 480$, the dimensionality of this space 230400.

- In principle, one could argue that given a database that consists of N different mugshot vectors (for N different individuals) in a 230400 dimensional space, we could test whether the photo of a new individual corresponds to any already in the database by measuring the Euclidean distance between the vector for the new photo and each of the N vectors already in the database.

- Unfortunately, that is not likely to work because of the very high dimensionality — 230400 — of the space in which the photos are represented.

- You can expect weird things to happen when the dimensionality of a space goes up and becomes arbitrarily large. [Consider a square $[0,1] \times [0,1]$ in a 2D plane and, say, we want to populate it with randomly placed points. For each such point, we make *two* calls to a function $uniform(0,1)$ that returns a real number between 0 and 1 with a uniform distribution. The two calls return two values for the two coordinates of a point in the square. After you have placed a decent number of points in this manner in the square, what you will see in the square is what you would expect — a square with points that look more or less uniformly distributed inside it. Now let us consider a 3-dimension case — we will place points randomly inside a cube given by $[0,1] \times [0,1] \times [0,1]$. For each point to be placed inside the cube, this will require three calls to the function $uniform(0,1)$. If we try to visualize the point cloud inside the cube, it will look as expected — a more-or-less uniformly distributed collection of points inside the cube. Let's now generalize the experiment to a d-dimensional hypercube whose each side is of size 1. Remember, one of the corners of the hypercube would be at the origin and you will have a corner along each of the d axes at a unit distance from the origin. Those and other corners would make for a total of 2^d corners for the cube. To place a point randomly inside the hypercube, we would need to make d calls to the function $uniform(0,1)$. What is interesting is that even if just one of these d calls to $uniform(0,1)$ yields a value close to 0 or close to 1, the resulting point will be within a small distance δ of the surface of the cube. When d

approaches infinity, we can expect with probability 1 that at least one of the d calls to $uniform(0,1)$ will yield a value either close to 0 or close to 1. What that implies is that as the dimensionality d approaches infinity, *every one of the points* placed inside the hypercube will be inside a thin shell *just inside* the surface of the cube. That is, there will be *no points* in the main interior of the hypercube. If that's not weird, what is?]

- For another effect, it was shown by Beyer, Goldstein, Ramakrishnan, and Shaft in a report titled "When is Nearest Neighbor Meaningful?" that, under some pretty general conditions placed on a probability distribution, as the dimensionality of a space becomes unboundedly large, all of the points drawn from the distribution would appear to be at roughly the same distance from any given point in the space.

- Another major headache when working with high-dimensional spaces is that for a machine learning algorithm to generalize properly from the training data, the higher the dimensionality, the larger the number of training data samples you need. [It is generally recommended that if d is the dimensionality of the space in which you are representing the entities you want recognized with a machine learning algorithm, you need at least $10 \times d$ training samples for each class. It goes without saying that you would want these training samples to capture all of the diversity exhibited by the instances of a class.]

- This tutorial is about some of the more popular techniques out there that are used for dimensionality reduction.

- The arguments presented in this section were based on a

non-deep-learning based solution to a face recognition problem. It is not too difficult to think of application scenarios where a more traditional computer vision solution such as the one considered in this section would be more appropriate than the data intensive solutions based on deep learning:

Consider a police department that has a file on around 1000 hard-core criminals in the geographic area under its jurisdiction. The authorities are NOT interested in solving the general problem of face recognition. The authorities have no interest in a system that scrapes the internet for all the photos likely to be of a given individual. All that the authorities want to know is if a the face image of a suspect corresponds to one of those 1000 images in its files. It is possible that a traditional computer vision solution would be more effective in this case. The same arguments would apply to several cases of industrial and robotic object detection and recognition.

Back to TOC

2: Principal Components Analysis (PCA)

- Let's say we are given N vectorized images

$$\vec{\mathbf{x}}_i \qquad i = 1, 2, ..., N \tag{1}$$

In face recognition, if each face image is represented by a 64×64 array of pixels, the vectorized form of each such image will be a 4096-element vector consisting of all the pixels in a left-to-right and top-to-bottom scan of the face. We consider $\vec{\mathbf{x}}_i$ to be a *column* vector. [**In practice, the face images will be of size much larger than 64×64. We will use this size just to illustrate some of the concepts**]

- We can associate the following mean image vector with the N vectorized images:

$$\vec{\mathbf{m}} = \frac{1}{N} \sum_{i=1}^{N} \vec{\mathbf{x}}_i \tag{2}$$

- We can estimate the covariance C of the image set by

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^{N} \{(\vec{\mathbf{x}}_i - \vec{\mathbf{m}})(\vec{\mathbf{x}}_i - \vec{\mathbf{m}})^T\} \tag{3}$$

[Note: **Given an n-dimensional random vector $\vec{z} = (z_1, z_2, ..., z_n)^T$, its covariance by definition is an $n \times n$ matrix C whose $(i, j)^{th}$ element is given by**

$C_{ij} = E\{(z_i - E\{z_i\})(z_j - E\{z_j\})\}$ **where** $E\{.\}$ **is the expectation operator. The form shown above is a numerical approximation to this covariance in which we implement the expectation operator as an average over the** N **samples drawn from a random process.**]

- The diagonal elements of the covariance matrix $\mathbf{C}$ will be the averages of the squares of the individual pixel values (after you have subtracted the means) and the off-diagonal elements will be the averages of the pairwise products of different pixels (again after the mean is subtracted out). If the images were just white noise, $\mathbf{C}$ would be a diagonal matrix.

- In order to achieve some invariance to illumination, each image vector $\vec{\mathbf{x}}_i$ may first be normalized by subjecting it to the constraint $\vec{\mathbf{x}}_i^T \vec{\mathbf{x}}_i = 1$.

- **Dimensionality reduction by PCA consists of calculating the eigenvectors of the covariance matrix C and retaining for features the eigenvectors corresponding to the K largest eigenvalues. This constitutes the orthogonal PCA feature set.**

- Let's denote our orthogonal PCA feature set by $\mathbf{W}_K$:

$$\mathbf{W}_K \quad = \quad \left[\vec{\mathbf{w}}_1 \mid \vec{\mathbf{w}}_2 \mid .... \mid \vec{\mathbf{w}}_K\right] \tag{4}$$

  where $\vec{\mathbf{w}}_i$ denotes the i^{th} eigenvector of the covariance matrix $\mathbf{C}$. $\mathbf{W}_K$ is an $4096 \times K$ matrix for 64×64 face images.

- **Subsequently, each vector $\mathbf{w}_i$ will play the role of a feature for classification. The classification will be carried out in the K dimensional subspace corresponding to the retained eigenvectors.**

- The **feature values** for an unknown image will correspond to its projections on the eigenvectors. That is, if $\vec{\mathbf{x}}$ is an unknown vectorized image, the feature values corresponding to this image would be given by

$$\vec{\mathbf{y}} \quad = \quad \mathbf{W}_K^T(\vec{\mathbf{x}} \; - \; \vec{\mathbf{m}}) \tag{5}$$

  Note that $\vec{\mathbf{y}}$ is just K dimensional and K is usually a very small number compared to the dimensionality of the image vectors $\vec{\mathbf{x}}$. [**$\vec{y}$ is NOT a vector in the original high-dimensional space. It is merely a sequence of coefficients formed by taking the dot-product of $\vec{x} - \vec{m}$ with each of the leading eigenvectors.** The point being made here would be the same as calling a coordinate pair $(3, 4)$ a *vector* in $\mathcal{R}^2$. For a vector representation of the point $(3, 4)$, you'll need to express it as $3\hat{x} + 4\hat{y}$. See Section 2.6 for further details regarding this point.]

- The space spanned by the K column vectors in $\mathbf{W}_K$ is a K-dimensional subspace of the original high-dimensional space in which the image vectors $\vec{\mathbf{x}}$ reside. [**This subspace is commonly called the PCA space.**] This subspace is a *hyperplane* or a *linear manifold* in the original high-dimensional space.

- While the projection of $\vec{\mathbf{x}} - \vec{\mathbf{m}}$ into the hyperplane spanned by

the column vector of $\mathbf{W}_K$ gives us the desired low-dimensional representation of $\vec{\mathbf{x}}$, the projection of $\vec{\mathbf{x}} - \vec{\mathbf{m}}$ into the space spanned by the remaining eigenvectors of C tells us how much information was lost in generating the low-dimensional representation.

- Let the matrix $\mathbf{W}'_K$ consist of the trailing eigenvectors of C, meaning the eigenvectors that did not get used in $\mathbf{W}_K$.

- The product $\mathbf{W}'^T_K(\vec{\mathbf{x}} - \vec{\mathbf{m}})$ then gives us the dot-products of $\vec{\mathbf{x}} - \vec{\mathbf{m}}$ with each of the trailing eigenvectors of C that are in $\mathbf{W}'_K$. **These correspond to the projections of $\vec{\mathbf{x}} - \vec{\mathbf{m}}$ on the hyperplane that is perpendicular to the one defined by W_K.** The projections on the trailing eigenvectors can be represented by the error vector $\vec{\mathbf{e}}$:

$$\vec{\mathbf{e}} \;=\; \mathbf{W}'^T_K(\vec{\mathbf{x}} \;-\; \vec{\mathbf{m}}) \tag{6}$$

As with $\vec{\mathbf{y}}$, $\vec{\mathbf{e}}$ is NOT a vector in the original high-dimensional space, but merely a sequence of dot products with the trailing eigenvectors of C.

- We can therefore write for the square of the norm of the error vector when representing $\vec{\mathbf{x}}$ by $\vec{\mathbf{y}}$:

$$\begin{aligned} d_{err}^2 &= ||\vec{\mathbf{e}}||^2 = \vec{\mathbf{e}}^T\vec{\mathbf{e}} \\ &= \left(\mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}})\right)^T \mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}}) \\ &= (\vec{\mathbf{x}} - \vec{\mathbf{m}})^T \mathbf{W}_K \mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}}) \end{aligned} \tag{7}$$

- **For any user-specified choice for K, the PCA algorithm gives you the best hyperplane in the original data space that minimizes the value of d_{err}^2 as averaged over all the data samples.**

- **When derived in a probabilistic framework, for a user-specified K, the PCA algorithm returns a K-dimensional hyperplane that minimizes the expectation $E\{d_{err}^2\}$. Furthermore, the algorithm also guarantees that every pair of the eigenvectors retained in the low-dimensional space will be statistically uncorrelated.** That is $E\{\vec{\mathbf{w}}_i\vec{\mathbf{w}}_j\} = E\{\vec{\mathbf{w}}_i\}E\{\vec{\mathbf{w}}_j\}$.

- **Note, however, that it is only for Gaussian data that statistical uncorrelatedness implies statistical independence.**

- In real applications of PCA, you are almost always faced with non-Gaussian data that resides on **highly curved (the same thing as nonlinear) manifolds** (See Section 2.3 of this tutorial). So even though the features as yielded by PCA are statistically

uncorrelated, they may yet be highly statistically dependent.

- In and of itself, PCA is not designed to retain class discriminant information. Nonetheless, it has been found to be very effective for classification. (See, for example, the paper "PCA versus LDA" by Martinez and Kak, IEEE Transactions on PAMI, 2001) [A general rule of thumb is that, if d is the dimensionality of the space in which you are attempting classification, you need at least $10 \times d$ training samples for each class. This can make for unrealistic demands on the amount of training data needed when d is large. Hence the need for PCA.]

Back to TOC

2.1: Computation of the Eigenvectors for PCA

- Consider the problem of recognizing faces from the frontal images of the faces in a database of such images (such images are also called mugshots).

- It is fairly typical to use what are known as **normalized faces** for such recognition. Either automatically or, as is often the case, manually, you crop each training image so as to retain a rectangular region that is bounded by the lines just outside of the ears, the line just above the head and the line just below the chin.

- When a query image comes in, it can be cropped in a similar manner before it is compared with the database images.

- Let's assume that each such normalized image is an array of 64×64 pixels. This means that image vector $\mathbf{x}_i$ will be 4096 elements long and that the covariance matrix $\mathbf{C}$ will be 4096×4096. Let's now focus on estimating this covariance matrix

using

$$\mathbf{C} \;=\; \frac{1}{N}\sum_{i=1}^{N}\{(\vec{\mathbf{x}}_i \;-\; \vec{\mathbf{m}})(\vec{\mathbf{x}}_i \;-\; \vec{\mathbf{m}})^T\} \tag{8}$$

- While the large size of $\mathbf{C}$ may or may not be daunting in these modern times, a direct eigendecomposition of large matrices can eat up significant computational resources. There is also the issue of numerical stability of eigendecomposition of very large matrices that are of very low rank.

- **Fortunately, there is a computational trick available to make the job of eigendecomposition C very easy and numerically stable.**

- To see this computational trick, let's arrange all of the zero-mean image vectors in the form of a matrix:

$$\mathbf{X} \;=\; \Big[\vec{\mathbf{x}}_1 - \vec{\mathbf{m}} \;|\; \vec{\mathbf{x}}_2 - \vec{\mathbf{m}} \;|\; \ldots \;|\; \vec{\mathbf{x}}_N - \vec{\mathbf{m}}\Big] \tag{9}$$

  Assuming that we have $N = 100$ training images available, the size of the matrix X is 4096×100.

- Suppressing the division by N, we can now show that

$$\mathbf{C} = \mathbf{X}\mathbf{X}^T \tag{10}$$

- As is to be expected, $\mathbf{C}$ will be of size 4096×4096 for our example. [To see the equivalence between this form for $\mathbf{C}$ and the form shown earlier, imagine that $\mathbf{X} = [\vec{a}\ \vec{b}\ \vec{c}]$. $\mathbf{X}\mathbf{X}^T = [\vec{a}\ \vec{b}\ \vec{c}]\begin{bmatrix} \vec{a}^T \\ \vec{b}^T \\ \vec{c}^T \end{bmatrix} = \vec{a}\vec{a}^T + \vec{b}\vec{b}^T + \vec{c}\vec{c}^T$]

- If $\vec{\mathbf{w}}$ represents an eigenvector of $\mathbf{C}$, it must satisfy

$$\mathbf{X}\mathbf{X}^T\vec{\mathbf{w}} = \lambda\vec{\mathbf{w}} \tag{11}$$

- While our goal remains to find the eigenvectors $\vec{\mathbf{w}}$ of the composite matrix $\mathbf{X}\mathbf{X}^T$, let's go ahead and find the eigenvectors of $\mathbf{X}^T\mathbf{X}$. Denoting the eigenvectors of $\mathbf{X}^T\mathbf{X}$ by $\vec{\mathbf{u}}$, we can obviously write

$$\mathbf{X}^T\mathbf{X}\vec{\mathbf{u}} = \lambda\vec{\mathbf{u}} \tag{12}$$

- Since, for the case of $N = 100$ training images, the composite matrix $\mathbf{X}^T\mathbf{X}$ will only be of size 100×100, finding the eigenvectors $\vec{\mathbf{u}}$ is easy.

- To see how we can obtain $\vec{\mathbf{w}}$ from $\vec{\mathbf{u}}$, let's pre-multiply both sides of the above equation by $\mathbf{X}$. We get

$$\mathbf{X}\mathbf{X}^T\mathbf{X}\vec{\mathbf{u}} = \lambda\mathbf{X}\vec{\mathbf{u}} \tag{13}$$

which is better displayed as

$$(\mathbf{X}\mathbf{X}^T)\mathbf{X}\vec{\mathbf{u}} = \lambda\mathbf{X}\vec{\mathbf{u}} \tag{14}$$

or, equivalently as

$$\mathbf{C}(\mathbf{X}\vec{\mathbf{u}}) = \lambda(\mathbf{X}\vec{\mathbf{u}}) \tag{15}$$

- The above equation implies that we can obtain $\vec{\mathbf{w}}$ from $\vec{\mathbf{u}}$ by

$$\vec{\mathbf{w}} = \mathbf{X}\vec{\mathbf{u}} \tag{16}$$

- For the example we are considering, we have only 100 eigenvectors for $\vec{\mathbf{u}}$ since $\mathbf{X}^T\mathbf{X}$ will only be of size 100×100.

- IMPORTANT: The eigenvectors calculated with the trick described on the previous page will **NOT** be of unit magnitude. Therefore, you must normalize them before projecting either the training images or the test images into the eigenspace formed by the eigenvectors that you choose to retain.

- It may seem at first sight that whereas $\mathbf{C}$ may possess as many as 4096 eigenvectors for the example under consideration, the trick described on the previous page gives us only 100 eigenvectors. So you might ask: What about the other eigenvectors? As it turns

out, the other eigenvectors do not really exist — as I argue on the next page.

- The total number of eigenvectors of $\mathbf{C}$ will never exceed the number of images used for its computation. The reason for that, as shown by the Eq. (8) for $\mathbf{C}$, is that if you use 100 images for its computation, $\mathbf{C}$ will be a sum of 100 unit-rank outer-product matrices. The rank of $\mathbf{C}$ will therefore never exceed the rank of $\mathbf{X}^T\mathbf{X}$.

Back to TOC

2.2: A Potential Shortcoming of PCA

- It is relatively easy to lose class discriminant information with PCA. This is illustrated in Figure 1:

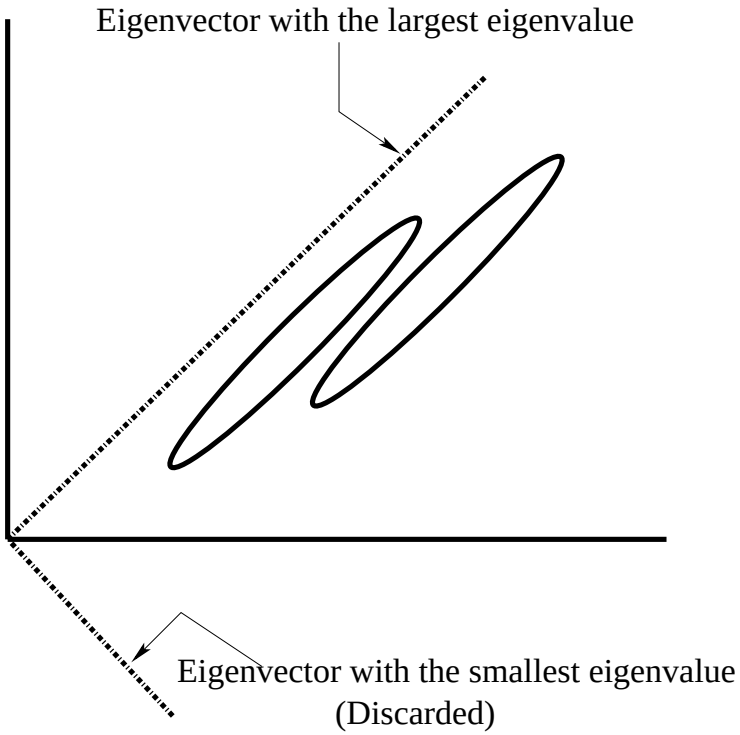

Figure 1: *An example to illustrate how it would be possible for a PCA based classifier to lose the inter-class discriminatory information.*

- Whether or not losing class discriminatory information by

discarding the eigenvectors corresponding to the smallest eigenvalues becomes an issue for real-life applications depends obviously on the shapes of the class distributions.

- For the example shown on the previous page, if we discard the eigenvector corresponding to the smaller of the two eigenvalues, we would need to project all of our training data on the sole retained eigenvector. Given the large overlap between the projections of the two class distributions, it would be impossible to construct a classifier with a low misclassification error.

Back to TOC

2.3: An Even More Serious Shortcoming of PCA

- **The goal of PCA is to give you the best possible K-dimensional hyperplane approximation to all your training data.** (That is, your data will be represented by its projections on the best fitting hyperplane.) [**For example, if the original data space is 3-dimensional and $K = 2$, the PCA algorithm will give you the best possible plane that passes through your 3-dimensional data.**]

- What that implies is that if most of the variation in your data vectors $\vec{\mathbf{x}}$ is "linear" — in other words, if most of the variation in the data can be explained by forming vectors that are linear combinations of other vectors — then PCA should work well for you. Remember, every point on a hyperplane is a linear vector sum of other points on the hyperplane.

- **Unfortunately, the above assumption is often not satisfied in important applications. Face recognition being a classic case in which the assumption is grossly violated.**

- What you see at left in Figure 2 are the face vectors $\vec{\mathbf{x}}$ for three different human subjects as the angle of view is varied from $-90°$

to +90° in yaw and −60° to +60° in pitch. The data points (with one color for each human subject) are shown in the 3D subspace formed by the largest three eigenvectors of the covariance matrix. **The fact that the face vectors fall on highly structured surfaces even in a 3D space means that all of the data in the original high-dimensional space resides on a very low-dimensional manifold that cannot be approximated by a hyperplane.**

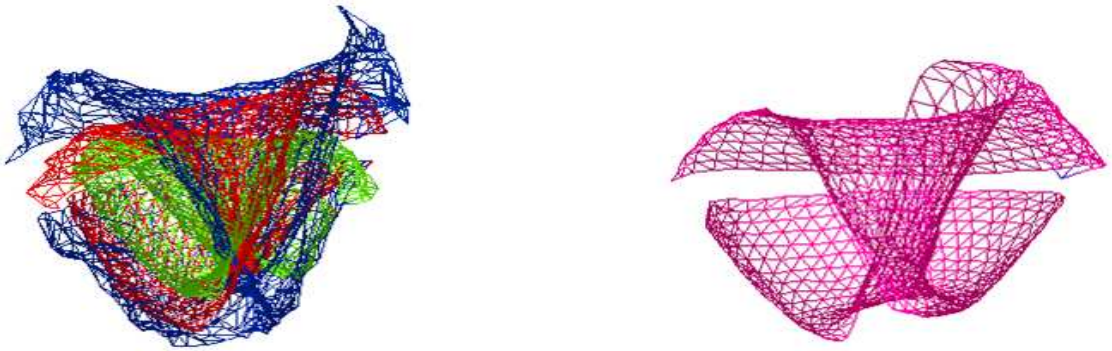

Figure 2: *Each color in the depiction at left is a surface formed by the three largest eigenvectors computed from the face photo of a real human subject. The depiction at right is the average of the three surfaces at left.*

- The manifold depictions in Figure 2 are from the following publication on face recognition from the Robot Vision Lab at Purdue:

  `https://engineering.purdue.edu/RVL/Publications/FaceRecognitionUnconstrainedPurdueRVL.pdf`

- What you see on the right in Figure 2 is the mean manifold for

the three human subjects.

- Obviously, if the data resides predominantly on a highly structured nonlinear surface in some subspace of the original high-dimensional measurement space, it is unlikely to lend itself to PCA type of modeling.

- That leads to the question of how to carry out dimensionality reduction when the original data resides on a low-dimensional manifold that cannot be approximated by a hyperplane. See my tutorial "*Clustering Data That Resides on a Low-Dimensional Manifold in a High-Dimensional Measurement Space*" for answers to this very important question. Here is a clickable URL that takes you directly to that tutorial:

  `https://engineering.purdue.edu/kak/Tutorials/ClusteringDataOnManifolds.pdf`

Back to TOC

2.4: PCA + NN for Classification

- A common approach to classification goes as follows: Let's say we have **N** total images depicting $\mathcal{C}$ classes. We assume that each image contains only one object (a typical situation in face recognition experiments) and the class label of each object is one of $\mathcal{C}$ labels.

- We assume that the **N** images constitute our human-labeled data. That is, the class label of the object in each image is known.

- We now apply PCA to all of the **N** images and retain just a handful of the eigenvectors with the largest eigenvalues for a subspace in which we carry out classification.

- In the subspace thus constructed, we can now use the k-NN (k Nearest Neighbors) classifier to find the class label for a new query image.

- Often, people just use the one-nearest-neighbor classifier.

- Nearest neighbor classification requires that we be able to measure

the distance between data points in the subspace.

- Most application use the Euclidean distance for finding the nearest neighbor. Of course, one can also use the various $\mathbf{L}_p$ norms.

- Searching for the nearest neighbor can be speeded up by using a data structure such as a KD-tree for storing all the samples.

Back to TOC

2.5: PCA versus the Other Approaches for Dimensionality Reduction

- For reasons already explained, PCA is not always the best approach for dimensionality reduction. As illustrated previously, you can easily lose class discriminatory information with PCA.

- Another common approach to dimensionality reduction is a combinatorial search for the best feature subset using some sort of a greedy algorithm.

- Two examples of this method are: (1) the forward selection method; and (2) the backward elimination method.

- In the forward selection method, you search through all possible features and choose one that optimizes some feature selection criterion. [The criterion can be the minimization of the class entropy along that feature direction, as explained in the paper: Lynne Grewe and A. C. Kak, " *Interactive Learning of a Multi-Attribute Hash Table Classifier for Fast Object Recognition*," Computer Vision and Image Understanding, pp. 387-416, Vol. 61, No. 3, 1995. You can download it from `https://engineering.purdue.edu/RVL/Publications/Grewe95Interactive.pdf`.] After selecting the first feature, you again search through all remaining features and find the best choice for a subspace consisting of two features;

and so on.

- The backward elimination method starts with the full features space and eliminates one feature at a time. Elimination of a feature is carried out on the basis of criteria similar to what is used in forward selection.

- Most folks who use combinatorial approaches for feature selection seem to prefer the forward selection method.

Back to TOC

2.6: Evaluate Yourself on Your Understanding of PCA

- Considering that PCA is one of the most commonly used techniques for dimensionality reduction, the more deeply you understand it, the better you'll be at figuring out when to use it and when to resort to more sophisticated approaches. With that goal in mind, ask yourself the following questions:

- Why can we not express a data point $\vec{\mathbf{x}}$ in the original high-dimensional space as a vector sum of its projection on the hyperplane spanned by the leading K eigenvectors of the covariance matrix C and the projection of $\vec{\mathbf{x}}$ into the subspace that is perpendicular to the hyperplane? That is, why is the following vector addition wrong?

$$\mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}}) + \boldsymbol{\mathcal{W}}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}}) \tag{17}$$

  where $\boldsymbol{\mathcal{W}}_K$ is made up of the trailing eigenvectors of C — these are the eigenvectors that are left over after you have used the K leading eigenvectors as the column vectors of $\mathbf{W}_K$. [**Hint:** The product $\mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}})$ merely gives you the "dot products" of $\vec{\mathbf{x}} - \vec{\mathbf{m}}$ with the different eigenvectors. To represent this projection into the subspace spanned by the K eigenvectors in W_K as a vector in the original space, you will need to express the projection as $\mathbf{W}_K\mathbf{W}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}})$. The same goes for representing as a vector in the original space the perpendicular to the hyperplane as given by $\boldsymbol{\mathcal{W}}_K^T(\vec{\mathbf{x}} - \vec{\mathbf{m}})$.]

- Just as a mental exercise, imagine that all your data resides on two different planes of a 3D feature space. Assume that these planes are described by $z = 1$ and $z = 2$ and that each plane corresponds to a different class. This data is obviously locally 2D even though it resides in a 3D space. Let's say we reduce the dimensionality of the data from 3 to 2 by applying the PCA algorithm to all your data.

- Where would the plane yielded by PCA lie in relation to the two data planes? (Assume that even after you have subtracted the global mean from the data, it still resides predominantly on two different planes.) And under what conditions would you be able to use the reduced-dimensionality representation of the 3D data for classification?

- Finally, let us imagine that your original data resides on the 8 sides of a cube in 3D space and that each side of the cube corresponds to a different class. Obviously, the data is again locally 2-dimensional although overall it is 3-dimensional. Given the intrinsically 2D nature of the data, you may be tempted to apply the PCA algorithm to it for dimensionality reduction. Would that work? [What works in this case is known as the "local PCA" algorithm, as explained in my tutorial "*Clustering Data That Resides on a Low-Dimensional Manifold in a High-Dimensional Measurement Space.*" Here is a clickable URL for this tutorial: `https://engineering.purdue.edu/kak/Tutorials/ClusteringDataOnManifolds.pdf`]

Back to TOC

3: Linear Discriminant Analysis (LDA)

- Whereas PCA seeks to find an orthogonal set of maximal-variance directions in the underlying vector space for all available training data, the goal of LDA is to find the directions in the underlying vector space that are maximally discriminating between the classes.

- **A vector direction is maximally discriminating between the classes if it simultaneously maximizes the between-class scatter and minimizes the within-class scatter along that direction.** [**More precisely, when we *project* all of the training data on to a vector that is along a maximally discriminatory direction, the ratio of between-class scatter to within-class scatter will be the largest.**]

- The subset of *images* that correspond to class i will be denoted $\mathcal{C}_i$ and we will use $\mathcal{C}$ to denote the set of all *classes*. The mean image vector for $\mathcal{C}_i$ will be denoted $\vec{\mathbf{m}}_i$. We will refer to $\vec{\mathbf{m}}_i$ as the **class mean**. The number of all classes is given by the cardinality $|\mathcal{C}|$, and the number of images in class i by the cardinality $|\mathcal{C}_i|$.

- Let $\vec{\mathbf{m}}$ denote the overall mean image vector for all images. We

will refer to $\vec{\mathbf{m}}$ as the **global mean**.

- We now define **between-class scatter** by

$$S_B = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{|\mathcal{C}|}(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})^T \tag{18}$$

If each face image is represented by a 4096-element vector, S_B is a 4096×4096 sized matrix.

- And we define the **within-class scatter** by

$$S_W = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{|\mathcal{C}|}\frac{1}{|\mathcal{C}_i|}\sum_{k=1}^{|\mathcal{C}_i|}(\vec{\mathbf{x}}_k^i - \vec{\mathbf{m}}_i)(\vec{\mathbf{x}}_k^i - \vec{\mathbf{m}}_i)^T \tag{19}$$

where $\vec{\mathbf{x}}_k^i$ is the k^{th} image vector in the i^{th} class. As with S_B, S_W will also be a 4096×4096 sized matrix when each image is represented by a 4096-element vector.

- In the formula for S_W, the inner summation estimates the covariance for each class separately and the outer summation averages these covariances over all the classes.

- If we choose some vector $\vec{\mathbf{w}}$ in the underlying vector space (the space in which each image is represented by a vector), then

$$\vec{\mathbf{w}}^T S_B \vec{\mathbf{w}} \tag{20}$$

gives us a projection of the between-class scatter on the vector $\vec{\mathbf{w}}$. [**This result can be derived trivially from the definition of S_B itself in Eq. (18). Calculate the right hand side in that definition for the projection $\vec{\mathbf{w}}^T(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})$ instead of for $(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})$. The new argument to the summation will become $\vec{\mathbf{w}}^T(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})^T\vec{\mathbf{w}}$.**]

- Similarly,

$$\vec{\mathbf{w}}^T S_W \vec{\mathbf{w}} \tag{21}$$

gives us a projection of the within-class scatter on the vector $\vec{\mathbf{w}}$.

- The above geometrical interpretation can be used to search for the directions $\vec{\mathbf{w}}$ that maximize the ratio of between-class scatter to within-class scatter. These directions would constitute the best feature vectors from the standpoint of achieving the best class discriminations.

- The above mentioned ratio, known as **the Fisher Discriminant Function**, is given by:

$$J(\vec{\mathbf{w}}) \quad = \quad \frac{\vec{\mathbf{w}}^T S_B \vec{\mathbf{w}}}{\vec{\mathbf{w}}^T S_W \vec{\mathbf{w}}} \tag{22}$$

- We can show that a vector $\vec{\mathbf{w}}$ that maximizes the Fisher

Discriminant Function must satisfy

$$S_B\vec{\mathbf{w}} \quad = \quad \lambda S_W\vec{\mathbf{w}} \tag{23}$$

for some constant λ. By the way, this is referred to as the *generalized eigenvalue problem*. [**This result can be derived by solving the following maximization problem:** *maximize* $\vec{\mathbf{w}}^T S_B\vec{\mathbf{w}}$ *subject to* $\vec{\mathbf{w}}^T S_W\vec{\mathbf{w}} = 1$**. Using the method of Lagrange multipliers, we therefore maximize** $L(\vec{\mathbf{w}}) = \vec{\mathbf{w}}^T S_B\vec{\mathbf{w}} - \lambda(\vec{\mathbf{w}}^T S_W\vec{\mathbf{w}} - 1)$ **by setting** $\frac{dL(\vec{\mathbf{w}})}{d\vec{\mathbf{w}}} = 0$**. This gives us the equation** $S_B\vec{\mathbf{w}} - \lambda S_W\vec{\mathbf{w}} = 0$ **and that takes us directly to the result shown.**]

- If S_W could be assumed to be nonsingular, the above problem can be translated into the following more conventional eigendecomposition problem:

$$S_W^{-1}S_B\vec{\mathbf{w}} \quad = \quad \lambda\vec{\mathbf{w}} \tag{24}$$

Back to TOC

3.1: LDA for the Two-Class Problem

- Since the global mean for the two-class problem is given by $(\vec{\mathbf{m}}_1 + \vec{\mathbf{m}}_2)/2$, the between-class scatter matrix for this case is given by

$$S_B = (\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2)(\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2)^T \tag{25}$$

- This implies that $S_B\vec{\mathbf{w}}$ will always be in the direction of $\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2$ since the scalar product $(\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2)^T\vec{\mathbf{w}}$ will always reduce to just a scalar.

- Therefore, regardless of what $\vec{\mathbf{w}}$ turns out to be, for the two class case the following will always be true for some scalar c: [Just multiply from the right the two sides in Eq. (25) with any arbitrary vector $\vec{\mathbf{w}}$. You will see that $c = (\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2)^T \cdot \vec{\mathbf{w}}$]

$$S_B\vec{\mathbf{w}} = c(\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2) \tag{26}$$

- If we substitute the above result in the fundamental equation to solve $S_W^{-1}S_B\vec{\mathbf{w}} = \lambda\vec{\mathbf{w}}$, we get the following solution for the maximally class discriminating direction $\vec{\mathbf{w}}$ for the two-class case:

$$\vec{\mathbf{w}} = S_W^{-1}(\vec{\mathbf{m}}_1 - \vec{\mathbf{m}}_2) \tag{27}$$

- Note that there can exist only one solution vector $\vec{\mathbf{w}}$ for $S_W^{-1} S_B \vec{\mathbf{w}} = \lambda \vec{\mathbf{w}}$ for the two-class case. [**That is because S_B is a rank 1 matrix — since it is formed by a vector outer product. Additionally, the rank of a product of two matrices cannot exceed the smaller of the rank for the two matrices.**] Therefore, the rank of $S_W^{-1} S_B$ will also be equal to 1 (assuming S_W^{-1} exists).

- To classify an unknown query image, all we need to do is to set up a decision threshold along this maximally discriminating direction.

- To summarize, with LDA, for the two-class problem, you get only one LDA vector and it is the difference vector between the two means, the difference vector being modified by the matrix S_W^{-1}, assuming that the inverse of the within-class scatter matrix exists.

- That leads to the question of what to do when S_W is of reduced rank and its inverse does not exist. **When S_W^{-1} does not exist, we resort to the algorithm described in Section 3.5.**

- Whereas the two-class problem gives us only one LDA feature, in general, though, as we will see later, for $|\mathcal{C}|$ classes, you will get **at most** $|\mathcal{C}| - 1$ LDA vectors.

- Next we talk about LDA for discriminating simultaneously between more than two classes.

Back to TOC

3.2: LDA for Multiple Classes (Multiple Discriminant Analysis)

- Recall that for the $\mathcal{C}$ class problem our goal is to solve the **generalized eigendecomposition problem** given by $S_B\vec{\mathbf{w}}_k = \lambda_k S_W\vec{\mathbf{w}}_k$ for its eigenvectors $\mathbf{w}_k$, $k = 1, 2, ...$ and retain a small number of the eigenvectors that correspond to the largest eigenvalues λ_k. [λ_k can be shown to be the ratio of the projected between-class scatter to the within-class scatter along the k^{th} eigenvector.]

- From the definition of S_B:

$$S_B = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{|\mathcal{C}|}(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})(\vec{\mathbf{m}}_i - \vec{\mathbf{m}})^T \tag{28}$$

  note that S_B is a sum of $|\mathcal{C}|$ matrices of rank at most 1. When you construct a matrix by taking the outer product of two vectors, every row of the matrix is some multiple of the first row.

- Additionally, since the global mean $\vec{\mathbf{m}}$ is formed from a linear sum of the class means $\vec{\mathbf{m}}_i$, only $|\mathcal{C}| - 1$ of the $|\mathcal{C}|$ matrices that go into S_B are linearly independent.

- Therefore, the rank of S_B is at most $|\mathcal{C}| - 1$. As a result, the

generalized eigenvector decomposition problem has at most $|\mathcal{C}| - 1$ non-zero eigenvalues.

- Hence, we can extract at most $|\mathcal{C}| - 1$ LDA eigenvectors.

- It can be shown that when S_W is isotropic (all classes have identical variances in all of the same principal directions), the LDA eigenvectors are the eigenvectors of the S_B matrix. These correspond to the space spanned by the $|\mathcal{C}| - 1$ mean difference vectors $\vec{\mathbf{m}}_i - \vec{\mathbf{m}}$.

- Let the retained LDA eigenvectors be represented by $\mathbf{W}$: [**My apologies to the reader who is confused by the same symbol W being used in S_W for "within-class scatter"and in what's shown below for the set of LDA eigenvectors.**]

$$\mathbf{W} = \Big[\vec{\mathbf{w}}_1 \mid \vec{\mathbf{w}}_2 \mid .... \mid \vec{\mathbf{w}}_K\Big] \tag{29}$$

- It can now be argued that $\vec{\mathbf{W}}^T S_B \vec{\mathbf{W}}$ is the projection of the overall between-class scatter S_B into the subspace spanned by $\mathbf{W}$.

- We can similarly argue that $\vec{\mathbf{W}}^T S_W \vec{\mathbf{W}}$ is the projection of the overall within-class scatter S_W into the subspace spanned by $\mathbf{W}$.

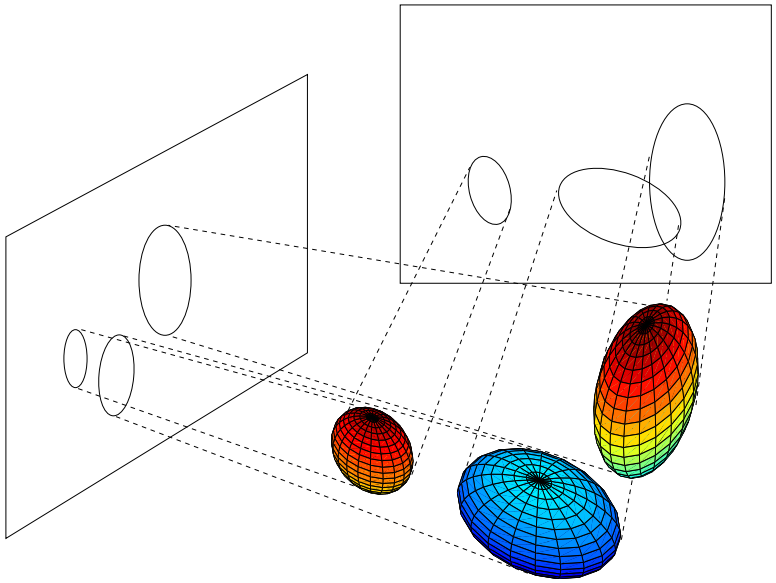

Figure 3: *If LDA were required to give us the best 2D space for the 3D covariances shown in the figure, it's likely to choose the one on the left because of the separations between the classes on that 2D plane.*

- Recall that each eigenvector $\vec{\mathbf{w}}_k$ is chosen to maximize the ratio of between-class scatter to within-class scatter along that direction.

- We can therefore conclude that the retained LDA eigenvectors in $\mathbf{W}$ will maximize the same ratio in the subspace spanned by $\mathbf{W}$.

- Figure 3, supplied by Hyukseong Kwon, shows us projecting the three different class covariances, defined originally in a 3D space, into two different 2D subspaces. LDA will use the 2D subspace that yields the best ratio of the projected between-class scatter to within-class scatter. [Hyukseong Kwon produced a most impressive Ph.D dissertation in Purdue RVL that showed how an indoor mobile robot could be made to produce architectural-quality maps of building

interiors. Check out his publications at the RVL publications page.]

Back to TOC

3.3: Can PCA Outperform LDA for Classification?

- By PCA based classification we mean a classifier that works as explained in Section 2.1.

- By LDA based classification we mean choosing a set of LDA eigenvectors and carrying out nearest-neighbor classification in the subspace spanned by the eigenvectors.

- The question posed by the title of this section is addressed by the "PCA versus LDA" paper by Martinez and Kak, which is one of the highly cited papers in face recognition (over 4140 citations on Google Scholar as of November 2022).

- The Martinez and Kak paper makes the following claim:

  > "When the training data inadequately represents the underlying class distributions, a PCA based classifier can outperform an LDA based classifier."

  This claim is very relevant to face recognition research because many face databases contain only a small number of face images (sometimes just one or two) for each human. Our claim says that

when such a database is used for training a classifier, PCA may do just as well as LDA.

- Figure 4 illustrates this point clearly. The LDA decision threshold cuts across the underlying class distributions and will therefore result in high misclassification error.

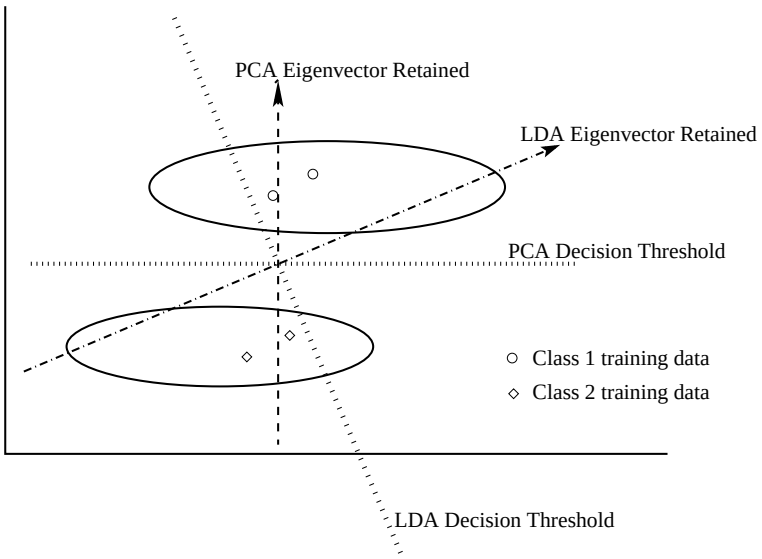

Figure 4: *As illustrated here, when the available training data grossly undersamples the class distributions (highly likely to happen in real-world scenarios), an LDA based decision threshold may give worst results than the one based on PCA.*

Back to TOC

3.4: LDA's Peaking Problem

- On the good side, the LDA eigenvectors capture the most class-discriminant directions in the underlying vector space.

- On the not so good side, **the LDA eigenvectors tend not to be orthogonal** (although they can be orthogonalized with, say, Gram-Schmidt orthogonalization).

- Also, LDA eigenvectors with $\lambda_k << 1$ may have excessively large within-class scatter in relation to between-class scatter along those directions.

- In practice, LDA often suffers from what is frequently referred to as "small sample size (SSS)" problem and the "peaking phenomenon."

- The SSS problem was illustrated Figure 4. The peaking phenomenon exhibited by LDA means that as you add more and more eigenvectors (obviously, up to a max of $|\mathcal{C}| - 1$ eigenvectors), at some point the performance of the classifier starts to decrease — contrary to what the intuition would say. This is illustrated in

the figure below that was generated by Chad Aeschliman in our laboratory.

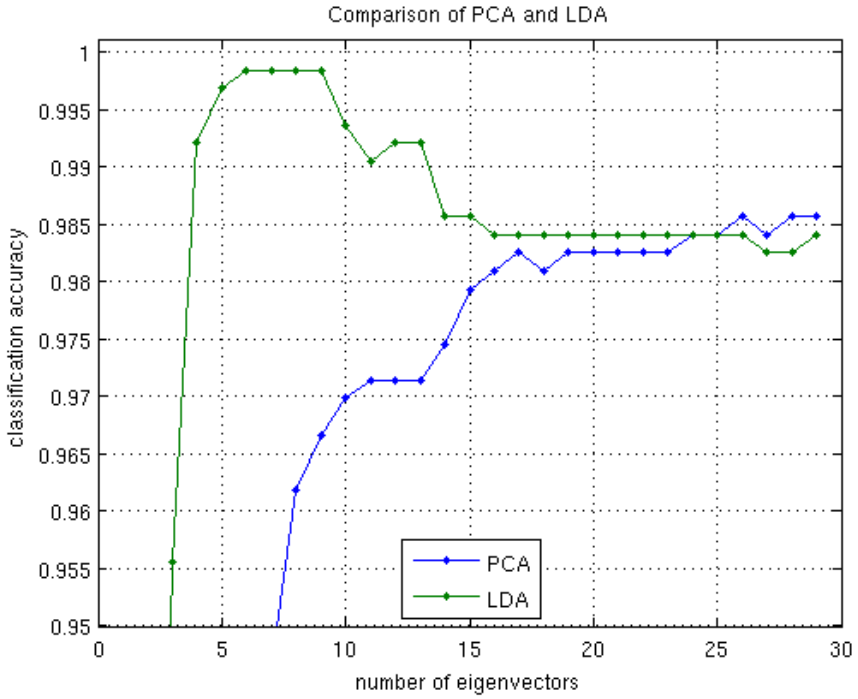

Figure 5: *An illustration of LDA's peaking problem. Contrary to intuition, the performance of an LDA based classifier may actually degrade as you add more and more eigenvectors.*

- Figure 5 was generated for a dataset of normalized 128×128 face images of 30 individuals. Overall, the dataset consisted of 1260 images, divided into two sets, one used for training and the other for testing. Of the 42 images of each individual, 21 were taken at 1° pose increments in the azimuth angle and the other 21 taken with fixed pose but with the direction of illumination changing in

1° intervals. This dataset is a subset of the CUbiC FacePix(30) face database of images that is made available by Arizona State University.

- The LDA implementation used for the results shown in Figure 5 were obtained with the Yu and Yang algorithm described in Section 3.5 of this tutorial using the Nearest Neighbor (1-NN) classifier.

- For the results shown in Figure 5, out of the 42 images available for each individual, 10 randomly selected images were used for training and the rest for testing.

- I believe the peaking phenomenon is caused by the fact that, for small eigenvalues, the within-class scatter may exceed the between-class scatter along the direction represented by the corresponding eigenvectors. In fact, since the eigenvalue is the ratio of the two, that is guaranteed to be the case when $\lambda_k < 1$. So when such eigenvectors are added to the subspace, we can expect the classifier performance to deteriorate.

- But many folks believe that the peaking phenomenon is caused by what is known as **overfitting**.

- Overfitting means that your decision surfaces are hewing too close

to the training data. In other words, your classifier may suffer from overfitting when it is trained to work too perfectly on the training data and tested too insufficiently on the test data.

- Said another way, you have overfitting when you have insufficient generalization from the training data.

- Overfitting may occur for several reasons, a common one being that you have insufficient training data for the dimensionality of the feature space.

- The greater the dimensionality of the feature space, the larger the number of training samples you need for the classifier to work well. **This is called the *curse of dimensionality.***

- As a very, very approximate rule of thumb, the number of training samples you need for each class must be at least ten times the dimensionality of the feature space.

- Shown in Fig. 6 are the results obtained when we increase the number of training images to 21 for each individual and use the rest for testing. You will notice that the peaking phenomenon seems to have disappeared — at least for the case of the current dataset.

- Note that one does not always have the luxury of increasing the size of the training set in the manner we have for the results shown on in Fig. 6. In many applications, you may only have one or two training images available for each human in the database.

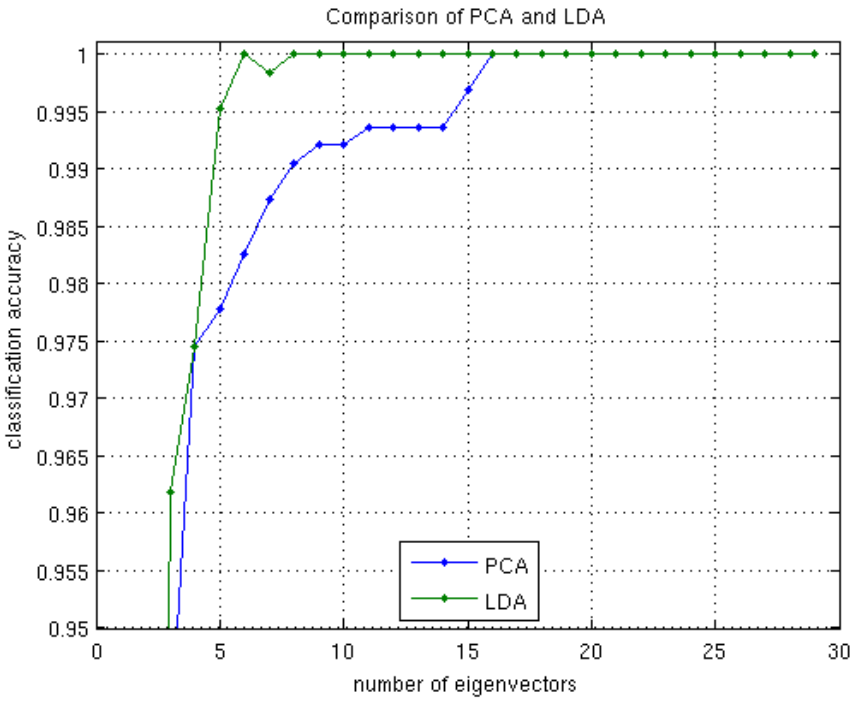

Figure 6: *As shown here, it may be possible to mitigate the peaking problem with more training data.*

- Because of the peaking issues related to its performance, LDA is sometimes used in a two-stage mode in which PCA is used first to reduce the overall dimensionality of the feature space and LDA used subsequently for establishing a subspace in which the classification is carried out by, say, the nearest neighbor rule.

Back to TOC

3.5: Computational Issues Related to LDA

- As mentioned previously, computing the LDA eigenvectors requires that we solve the generalized eigendecomposition problem

$$S_B\vec{\mathbf{w}} \quad = \quad \lambda S_W\vec{\mathbf{w}} \tag{30}$$

  which we could solve as a regular eigendecomposition problem

$$S_W^{-1}S_B\vec{\mathbf{w}} \quad = \quad \lambda\vec{\mathbf{w}} \tag{31}$$

  provided S_W is non-singular.

- Whether or not S_W is nonsingular depends both on the shapes of the individual class distributions and on how well the distributions are sampled by the training images. [Consider the case when the class distributions look like co-planar 2D disks in an underlying 3D space. In this case, even if there is no shortage of training samples, S_W will be singular.]

- In most practical situations involving face recognition (where you think of each individual's face as a separate class), you are likely to have an insufficient sampling of the class distributions. Therefore, you can count on S_W to be singular and for S_W^{-1} to not exist.

- These problems caused by S_W being singular become exacerbated when the data dimensionality is large. Recall, in face recognition, if we retain just 64×64 array of pixels for normalized faces, you will be computing S_W in a 4096 dimensional space.

- Because of these difficulties associated with solving the generalized eigendecomposition problem, people sometimes first use PCA to reduce the dimensionality of the data and then apply the LDA algorithm to the reduced dimensionality data.

- But, as we mentioned previously, applying PCA before LDA is a dangerous thing to do since you could lose class discriminatory information through PCA.

- **Getting back to S_W being singular, note that the null space of this scatter matrix can potentially contain much class discriminatory information.** This fact becomes intuitive if you realize that **the best LDA direction is one in which the projection of the between-class scatter matrix S_B is maximum and the projection of the within-class scatter matrix S_W is minimum.**

- Obviously, if there exists a direction $\vec{\mathbf{w}}$ on which the projection of S_W is zero, it's that much the better. **But such a desirable $\vec{\mathbf{w}}$ will be in the null space of S_W.**

- When S_W^{-1} does not exist, one can take recourse to the alternative of **directly** maximizing the Fisher discriminant function:

$$J(\vec{\mathbf{w}}) \quad = \quad \frac{\vec{\mathbf{w}}^T S_B \vec{\mathbf{w}}}{\vec{\mathbf{w}}^T S_W \vec{\mathbf{w}}} \tag{32}$$

by heuristically discarding those subspaces in the original N-dimensional measurement space that are not likely to contain class discriminant directions.

- As to how to invoke such heuristics can be seen in an algorithm proposed by Hua Yu and Jie Yang ("A Direct LDA Algorithm for High-Dimensional Data...." in Pattern Recognition, 2001).

- Yu and Yang's algorithm involves the following steps:
  - Use regular eigendecomposition to diagonalize S_B. This will yield a matrix $\mathbf{V}$ of eigenvectors such that

$$\mathbf{V}^T S_B \mathbf{V} \quad = \quad \Lambda \tag{33}$$

    where $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and Λ is a diagonal matrix of eigenvalues in a descending order.

  - Now discard those eigenvalues in Λ that are close to 0 since such directions carry no inter-class discriminations. Let's say you retained only M eigenvalues. Let $\mathbf{Y}$ be the matrix formed by the first M eigenvectors in $\mathbf{V}$. $\mathbf{Y}$ will be of size $N \times M$ where, for our example, $N = 4096$ for the case of 64×64

pixels retained in the normalized faces. M could be as small as, say, 20.

– Obviously,

$$\mathbf{Y}^T S_B \mathbf{Y} \quad = \quad \mathbf{D}_B \tag{34}$$

where $\mathbf{D}_B$ is the upper-left $M \times M$ submatrix of Λ.

– Now construct a matrix $\mathbf{Z}$ as follows:

$$\mathbf{Z} \quad = \quad \mathbf{Y}\mathbf{D}_B^{-1/2} \tag{35}$$

We can show that $\mathbf{Z}$ unitizes $\mathbf{S}_B$, meaning that $Z^T S_B Z = I_{M \times M}$, in a space whose intrinsic dimensionality has been reduced from N to M by discarding the subspace in which the between-class scatter is minimal. Note that the smaller the between-class scatter, the worst it is. The size of Z is also $N \times M$.

– Now diagonalize $\mathbf{Z}^T S_W \mathbf{Z}$ by regular eigendecomposition. This will yield a matrix $\mathbf{U}$ of eigenvectors such that

$$\mathbf{U}^T \mathbf{Z}^T S_W \mathbf{Z}\mathbf{U} \quad = \quad \mathbf{D}_W \tag{36}$$

– Since the goal is to maximize the ratio of between-class scatter to within-class scatter and since much inter-class discriminatory information is contained in the smallest

eigenvectors of S_W, we now discard the largest eigenvalues of $\mathbf{D}_W$ and drop the corresponding eigenvectors.

– As opposed to discarding the largest eigenvectors of S_W at this point, an alternative strategy consists of retaining all of the eigenvectors for now and then discarding the largest of the eigenvectors at the end.

– Let $\hat{\mathbf{U}}$ denote the matrix of the eigenvectors retained from $\mathbf{U}$.

– Yu and Yang claim that the matrix of the LDA eigenvectors that maximize the Fisher discriminant function is given by

$$\mathbf{W}^T \quad = \quad \hat{\mathbf{U}}^T \mathbf{Z}^T \tag{37}$$

- With regard to actual implementation of the Yu and Yang algorithm, special attention needs to be paid to the eigendecompositions shown earlier for S_B and for $\mathbf{Z}^T S_W \mathbf{Z}$. The computational trick required for the implementation of the first eigendecomposition is the same as in Section 2.1, except for the fact that now you must also retain the eigenvalues. But note that the eigenvalues for the eigenvectors of the decomposition expressed by the form $\mathbf{G}\mathbf{G}^T$ are the same as for the eigenvectors of the decomposition $\mathbf{G}^T\mathbf{G}$.

- With regard to the eigendecomposition called for in Eq. (36), you have to first express $Z^T S_W Z$ as a composite $\mathbf{G}\mathbf{G}^T$ before you can use the computational trick in Section 2.1. This requires that the matrix S_W be expressed in the composite form $\mathbf{G}\mathbf{G}^T$. That is easy to do if you stare long enough on the summation form for this matrix.

- When retaining only a subset of the eigenvectors from the set $\mathbf{W}$, note that you choose those eigenvectors that correspond to the smallest eigenvalues in D_W.

Back to TOC

4: Focusing on Class Separability: Why it Matters

- **Let's now reflect on a basic shortcoming of LDA:** The between-class scatter is measured by the distances between the class means without regard to the spread of the classes themselves.

- As Figure 7 illustrates, depending on the underlying class distributions and the number of eigenvectors retained, this can easily result in misclassification errors even in situations when such errors can be avoided because the classes are clearly separable.

- We therefore need classifiers that create decision boundaries not just on the basis of gross statistical properties such as the mean and the covariance, but also on the basis of the overall separability of the class distributions.

- With regard to class separability, the following question is extremely important: Are the classes linearly separable? In other words, considering for a moment only two classes, can we separate them by a hyperplane?

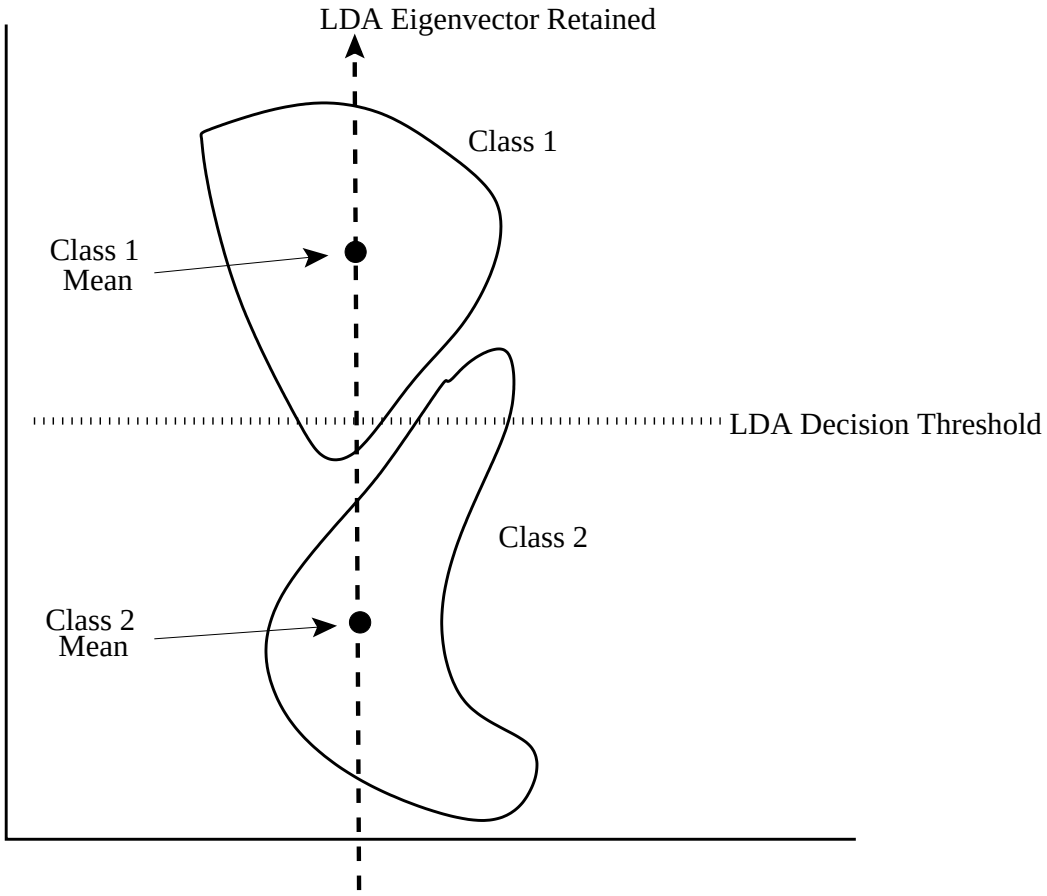

Figure 7: *This figure illustrates that measuring class separability through just the difference between the means can be "dangerous".*

- If the answer to the above question is yes, the decision boundary can be learned by a variety of approaches, the simplest being by induction from the labeled data. (For example, one could use the Quinlan's C4.5 algorithm to induce a decision tree whose decision thresholds would approximate the linear separating surface. See Grewe and Kak (CVIU 1995) for an application of this to computer vision.)

- If the separating surface between two classes is nonlinear, one can use a support vector machine (SVM) for classification.

- SVM nonlinearly transforms a feature space into a higher dimensional space in which the classes can become separable by linear or piecewise linear surfaces.

- A regular classifier obtains the optimum decision surface by minimizing the misclassification error. In the SVM literature, this is referred to as the minimization of **empirical risk**. Minimizing empirical risk using a small training set can result in a complex classifier that suffers from overfitting, as shown in Figure 8. **Such a classifier will not be able to generalize from the training set to unseen future data.**

- Instead of just minimizing empirical risk when learning from finite

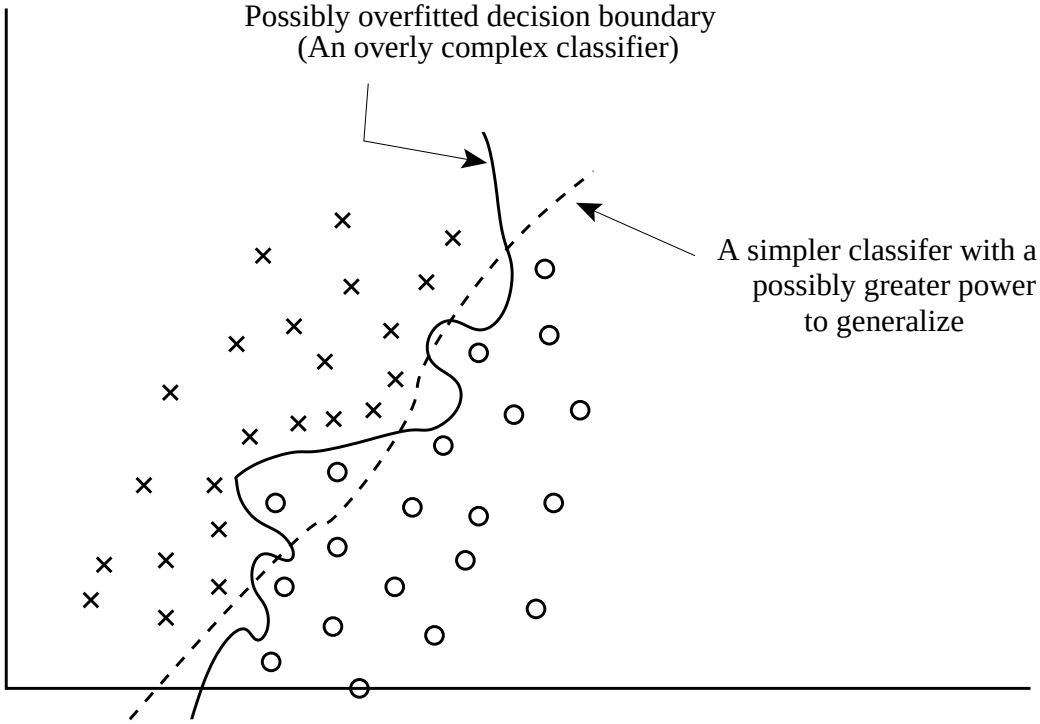

Figure 8: *Overfitting means creating a decision boundary that follows the training samples much too closely.*

training data, an SVM classifier minimizes what is known as the **structural risk**. Minimizing structural risk means minimizing the empirical risk while also minimizing the complexity of the decision boundary in order to increase the generalization ability of the classifier.

- The complexity of a classifier can be measured with the Vapnik-Chervonenkis (VC) dimension. (More of that in a later seminar.) Maximizing the margin of separation between the classes while minimizing the empirical risk amounts to finding a classifier with the low VC dimension.

- An SVM solution usually involves a **kernel trick** that allows us to train a classifier with only the inner products of the training samples.

- For classes with nonlinear separating surfaces, SVM involves using kernel methods to hopefully transform the feature space into a higher dimensional kernel space in which the decision surface would be linearly separable.

- For classes that are not at all separable, kernel transformations may be of no help. In other words, SVM classifiers do not exist when the underlying class distributions have significant overlap.

- Note that in its basic classification, SVM is only a binary classifier. It may however be used used for multi-class problems in a one-versus-the-rest mode.

Back to TOC

5: Margin Maximization Discriminant Analysis (MMDA)

- As mentioned, LDA features are vectors directions given by $\vec{\mathbf{w}}$ that maximize the Fisher discriminant

$$J(\vec{\mathbf{w}}) \quad = \quad \frac{\vec{\mathbf{w}}^T S_B \vec{\mathbf{w}}}{\vec{\mathbf{w}}^T S_W \vec{\mathbf{w}}} \tag{38}$$

  where S_B and S_W are the between-class and within-class scatter matrices.

- As mentioned already, a major shortcoming of this criterion is that it measures the between-class scatter through the distance between the class means **as opposed to by the separating margins between the classes**.

- Additionally, to find the eigenvectors, one must solve the generalized eigendecomposition problem $S_B \vec{\mathbf{w}}_k = \lambda_k S_W \vec{\mathbf{w}}_k$, or, equivalently, $S_W^{-1} S_B \vec{\mathbf{w}}_k = \lambda_k \vec{\mathbf{w}}_k$.

- Existence of the term S_W^{-1} in the Fisher discriminant function causes the LDA solution to become unstable when the number of training samples available is much less than the dimensionality of the training data.

- When training data per class is insufficient, S_W becomes singular or near-singular. (Obviously, one way to solve this problem is to first reduce the dimensionality with, say, PCA. But then you run the risk of losing the class discriminant information.)

- Since several of the computational, stability, and classification performance problems associated with LDA can be attributed to the quotient structure of $J(\vec{\mathbf{w}})$, that raises the question whether it is possible to formulate an alternative form of the criterion. Our desire would be to formulate a criterion that increases as the classes become more and more separated, and that at the same does not require us to invert matrices that may be near singular.

- Zhu Yan has proposed the maximization of

$$J(\vec{\mathbf{w}}) \quad = \quad \vec{\mathbf{w}}^T(S_B \quad - \quad \beta S_W)\vec{\mathbf{w}} \tag{39}$$

  This then is MMDA criterion. Zhu Yan also refers to this form as the additive form of LDA. The constant β is referred to as the **class-spread regulator**.

- For Gaussian class distributions, when $\beta = -1$, MMDA reduces to PCA because $C = S_B - S_W$ for such distributions. And when $\beta = 1$, MMDA reduces to MMC that stands for **Margin Maximization Criterion**. The MMC criterion was first put forward by Li, Jiang, and Zhang in 2003.

- Zhu Yan recommends using MMDA with $\beta = 9$.

Back to TOC

5.1: Derivation of the MMDA Criterion

- The MMDA criterion is based on finding those directions in the feature space that maximize the margin shown in Figure 9.

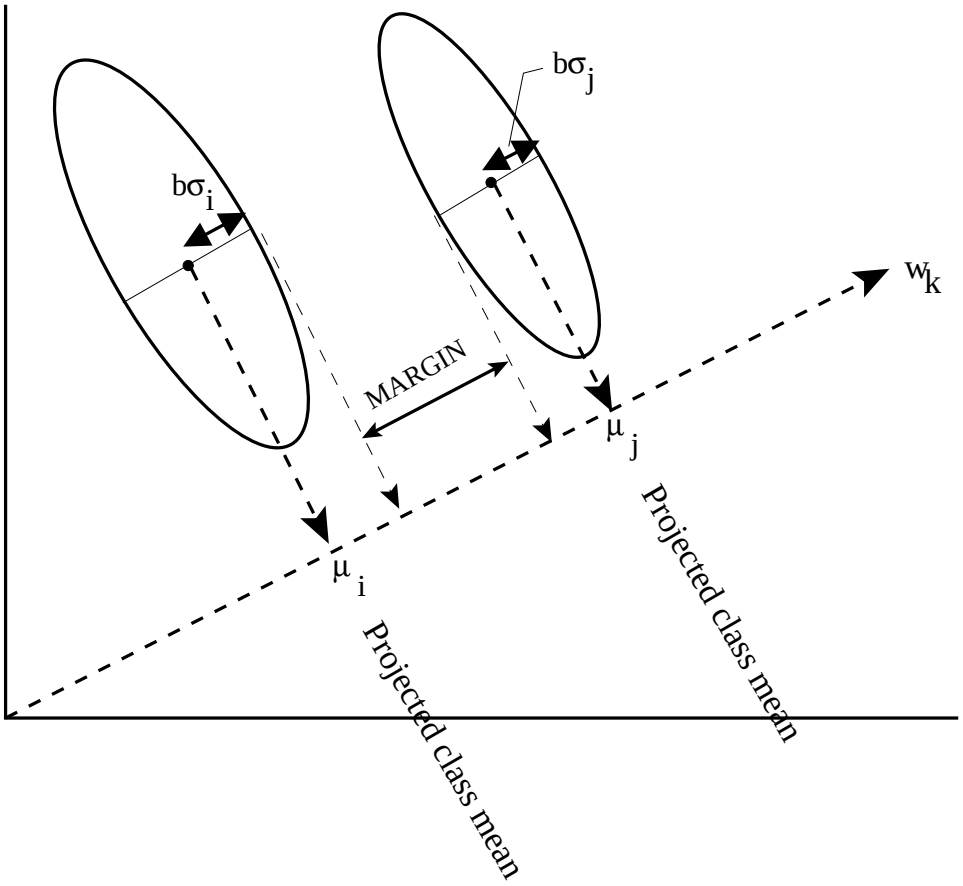

Figure 9: *This figure illustrates the concept of a "margin" between a pair of distributions: It is the distance between the two closest points belonging to the two different classes.*

- As shown in the figure, for each pair of classes i and j and for any direction $\vec{\mathrm{w}}$ in the feature space, we can construct a scalar M_{ij} that measures the size of the margin between the projections of those classes on the direction vector:

$$M_{ij} \quad = \quad |\mu_i \; - \; \mu_j| \quad - \quad b(\sigma_i \; + \; \sigma_j) \tag{40}$$

  Note that b controls how far we go out from the class mean to capture a class distribution.

- Suppose $b = 3$, then we assume implicitly that we consider a class to be no wider than 3σ from the projected mean along that direction, where σ is the projected standard deviation along the vector $\vec{\mathrm{w}}$.

- With regard to choosing b, recall the Chebyshev's inequality that says that for **all** probability distributions:

$$Prob(|X - \mu| \geq b\sigma) \quad \leq \quad \frac{1}{b^2} \tag{41}$$

  for any random variable X with expected value μ and variance σ^2 and for any $b > 0$. That is, no more than $\frac{1}{b^2}$ of a well-constructed set of training samples can be expected to be outside b standard deviations away from the mean regardless of the underlying probability distribution.

- The search for the directions that are best with respect to the M_{ij} criterion shown previously for all possible values for i and j is

made complicated by the difficult-to-handle nonlinearity introduced by the absolute magnitude operator.

- We therefore use the following closely-related alternative formulation:

$$\hat{M}_{ij} \quad = \quad |\mu_i \; - \; \mu_j|^2 \quad - \quad \beta(\sigma_i^2 \; + \; \sigma_j^2), \qquad \beta = b^2 \tag{42}$$

- We now construct a discriminant function by averaging the above over all pairs of classes:

$$\begin{aligned} J \quad &= \quad E_{ij}\{\hat{M}_{ij}\} \\ &= \quad E_{ij}\{|\mu_i - \mu_j|^2\} \; - \; \beta E_{ij}\{\sigma_i^2 + \sigma_j^2\} \end{aligned}$$

where E_{ij} is the expectation operator that forms the average with respect to all pairs of classes.

- Note that

$$
\begin{aligned}
E_{ij}|\mu_i - \mu_j|^2 &= E_{ij}\left\{\vec{\mathrm{w}}^T(\vec{m}_i - \vec{m}_j)(\vec{m}_i - \vec{m}_j)^T\vec{\mathrm{w}}\right\} \\
&= E_{ij}\left\{\vec{\mathrm{w}}^T\left[(\vec{m}_i - \vec{m}) - (\vec{m}_j - \vec{m})\right] \cdot \right. \\
&\qquad \left. \left[(\vec{m}_i - \vec{m}) - (\vec{m}_j - \vec{m})\right]^T \vec{\mathrm{w}}\right\} \\
&= 2\vec{\mathrm{w}}^T E_i\left\{(\vec{m}_i - \vec{m})(\vec{m}_i - \vec{m})^T\right\}\vec{\mathrm{w}} \\
&= 2\vec{\mathrm{w}}^T S_B \vec{\mathrm{w}}
\end{aligned}
$$

$$
\begin{aligned}
\beta E_{ij}\{\sigma_i^2 + \sigma_j^2\} &= 2\beta E_i\{\sigma_i^2\} \\
&= 2\beta E_i\left\{E_k\left\{\vec{\mathrm{w}}^T(\vec{\mathrm{x}}_k - \vec{\mathrm{m}}_i) \cdot \right.\right. \\
&\qquad \left.\left.(\vec{\mathrm{x}}_k - \vec{\mathrm{m}}_i)^T\vec{\mathrm{w}}\right\}\right\} \\
&= 2\beta\vec{\mathrm{w}}^T S_W \vec{\mathrm{w}}
\end{aligned}
$$

where the expectation with respect to the index k is over the training samples for class i.

- Substituting the above in the expression for J, we get the margin-based discriminant function:

$$J(\vec{\mathbf{w}}) = \vec{\mathbf{w}}^T(S_B - \beta S_W)\vec{\mathbf{w}} \tag{43}$$

- We obviously want a set of unit vectors $\vec{\mathbf{w}}$ that maximize J. That

is,

$$\vec{\mathbf{w}} \quad = \quad \text{argmax}_{\vec{\mathbf{w}}}\ J(\vec{\mathbf{w}}) \tag{44}$$

subject to the constraint $\vec{\mathbf{w}}^T\vec{\mathbf{w}} = 1$ since we are only interested in the directions that optimize the MMDA criterion. To incorporate the constraint, we use the method of Lagrange multipliers.

- We therefore maximize

$$L(\vec{\mathbf{w}}) \ = \ J(\vec{\mathbf{w}}) \ - \ \lambda(\vec{\mathbf{w}}^T\vec{\mathbf{w}} \ - \ 1) \tag{45}$$

by setting

$$\frac{dL(\vec{\mathbf{w}})}{d\vec{\mathbf{w}}} \quad = \quad 2(S_B - S_W)\vec{\mathbf{w}} \ - \ 2\lambda\vec{\mathbf{w}} \quad = \quad 0 \tag{46}$$

- This leads to the standard eigenvector problem for MMDA:

$$(S_B \ - \ \beta S_W)\vec{\mathbf{w}} \quad = \quad \lambda\vec{\mathbf{w}} \tag{47}$$

as opposed to the generalized eigenvector problem we ran into for LDA.

- But note that, depending on how the between-class scatter compares with the within class scatter, the matrix $S_B \ - \ \beta S_W$ may not be positive semidefinite — unlike the covariance matrix C for PCA. We could therefore have complex eigenvalues for MMDA.

- The smaller the magnitude of a complex eigenvalue for MMDA, the larger the overlap between the classes along that eigenvector.

- Because MMDA leads to a regular eigendecomposition, we are guaranteed to have an orthogonal set of eigenvectors, the number of eigenvectors depending on the rank of $S_B - \beta S_W$. This is unlike LDA where we cannot have more than $\mathcal{C} - 1$ eigenvectors.

Back to TOC

6: Relevance-Weighted Discriminant Analysis (RWDA)

- Earlier I pointed to two problems with LDA: (1) Problems caused by insufficient training data; and (2) LDA's peaking phenomenon. Insufficient training data causes stability problems in the direct or indirect estimation of S_W^{-1}. I also showed how this problem, caused by the quotient structure of the Fisher discriminant function, can be gotten rid of in MMDA. Now we will focus on the second problem — LDA's peaking phenomenon.

- LDA's peaking problem apparently disappears when the features are weighted to reflect their relevance to classification.

- Zhu Yan claims that the generalization ability of a classifier is improved if the features are weighted by their relevance.

- Zhu Yan's definition of feature relevance: *A feature is relevant if there exists at least one non-overlapping region of projected data between two or more classes along that feature direction.* **The greater this non-overlap, the more relevant the feature.**

- In other words, the non-overlapping regions of a feature direction constitute the "working part" of that feature, as Yan puts it.

- Note that the non-overlapping region along a feature direction is NOT the same thing as the margin along that direction.

- For the two-class case, whereas the margin means the distance between the closest points belonging to the two different classes when they are projected onto a direction vector, the total non-overlapping region means how much of a direction vector is not simultaneously covered by both classes at the same time.

- Figure 10 shows the non-overlapping regions on a feature direction. If we assume that on the average the standard deviation of each class distribution as projected onto a feature direction is σ_C, and that we believe that most training samples would be within a projected standard-deviation of $M\sigma_C$ for some constant M, the total extent of the non-overlapping regions on a feature direction will be $4M\sigma_C$ when the class projections are completely non-overlapping.

- However, when the class projections actually overlap, then it is not so straightforward to measure the size of the non-overlapping regions along a feature direction.

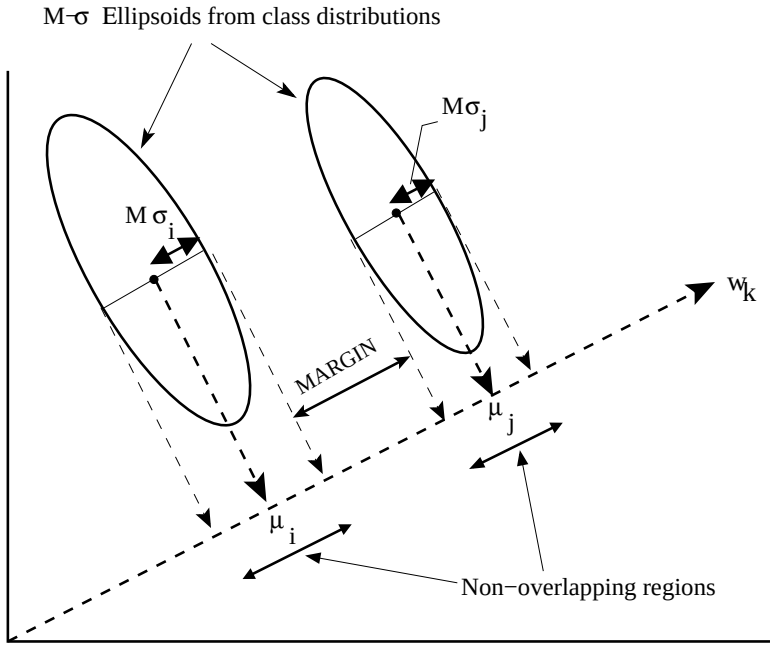

Figure 10: *For the example shown, when the two clearly separable classes are projected on to the feature direction w_k, it is easy to estimate the sizes of the non-overlapping regions and the margin along the feature vector.*

- By examining the "geometry" of projections of class distributions on vectors, Zhu Yan has come up with formulas for measuring the size of the non-overlapping regions in terms of the statistical parameters of the projections.

- Zhu Yan has proposed the following formula to associate relevance with a given feature direction $\vec{\mathrm{w}}_k$:

$$\Gamma_k \quad = \quad \begin{cases} 1 & \lambda_k \geq M^2 \\ \\ \sqrt{\frac{\lambda_k}{\lambda_T}} & \lambda_k < M^2 \end{cases} \tag{48}$$

  where λ_k is the eigenvalue associated with the eigenvector $\vec{\mathrm{w}}_k$ as produced by LDA. Since the eigenvalue measures the ration of between-class scatter to within-class scatter along that eigenvector, we use λ_T as a threshold that controls the minimum non-overlap that must exist along the eigenvector. M is the same as in the previous figure — we call it the spread regulator.

- Yan has shown that when the LDA features are prioritized by their relevance weights, the LDA's peaking problem disappears.

Back to TOC

7: Deep Learning Based Methods for Dimensionality Reduction — Autoencoders

- This section presents deep-learning based approaches for dimensionality reduction. In DL, a reduced-dimensionality representation for an input data object is referred to as its *embedding*. The goal in our case would be to train a neural network to learn to return the embeddings for a class of images. After training, the network would hopefully do a good job of returning the embeddings even for previously unseen images as long as they are from the same distribution as the training images.

- For the sort of applications I discuss in my computer vision class, the embeddings for images would be generated with an Autoencoder.

- The basic idea of an Autoencoder for creating a reduced-dimensionality representation of in input data is quite simple: You create an Encoder-Decoder paper as shown in Figure 11. The Encoder's job is to create a "compressed" representation of the input and the Decoder's job is to convert the compressed representation back to its original form — to the best extent possible.

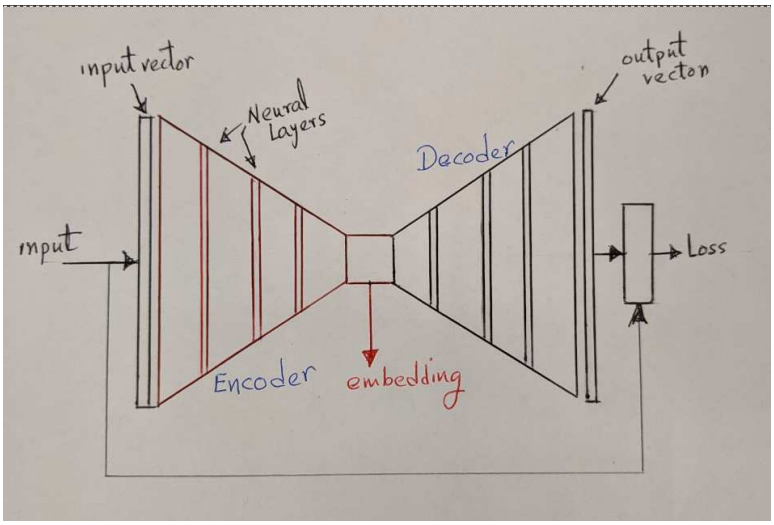

Figure 11: *A simple Autoencoder*

- Obviously, since the Encoder would have lost some information during the encoding process, the Decoder would not be able to fully recover the original. However, we would like to reduce the error between the original and the reconstruction to its least possible value.

- If we measure the error at the output with a difference function like the "Mean-Squared Error", the problem of finding the best reduced-dimensionality representation of the input is set up ideally for neural learning. You estimate the errors and backpropagate them as you update the learnable weights in the network until the errors have reached their least possible values.

- Training a network in the manner described above would be an

example of self-supervised learning.

- As you can see, the basic idea of an Autoencoder is simple and therefore easy to program. **However, it is unlikely to work well in applications where you require a significant reduction in dimensionality.** Here is the reason for that: Let's say that your input data is of dimensions 100,000 and you want the embedding for each input input image to be 10 dimensional — not an unlikely scenario for a PCA-based classifier when you retain just the 10 largest eigenvectors for the PCA-space. That would require the Encoder in Figure 11 to discard a lot of information. However, for neural learning to take place, the Decoder would now have to conjure up a significant part of what was discarded by the Encoder so that the final output would bear some semblance to the input. The greater the reduction in dimensionality, the greater the difficulty for the Decoder to fulfill this requirement.

- The Variational Autoencoder (VAE) that I'll now present overcomes the difficulty described above by using a more principled approach in how the Autoencoder is designed.

- VAE is based on a probabilistic model that jointly considers the input image, the part of the input that is discarded by the Encoder during dimensionality reduction, and the final embedding produced. Let $\mathbf{x}_{in}$ be an input image, and let $\mathbf{x}_{enc}$ stand for a reduced-dimensional representation of the input. The process of

losing information in the Encoder can be captured by the statement $\mathbf{x}_{enc} \sim \mathcal{N}(\mu, \sigma)$, which says that the encoded representation can be imagined to be drawn from a normal distribution with mean μ and standard-deviation σ.

- With a probabilistic interpretation of the encoding process as described above, we are faced with the following question: **What exactly do we want to see at the output of the Encoder? Should it be some instance $\mathbf{x}_{enc}$ drawn from the distribution $\mathcal{N}(\mu, \sigma)$?** Not really. **For practical reasons, we want each input image to go into a unique output from the Encoder.** All we know at this time is that, at a purely conceptual level, the process of encoding results in the transformation of the input image into the distribution $\mathcal{N}(\mu, \sigma)$. This distribution has two unique entities associated with it: μ and σ. **So it makes sense to train the Encoder the output the values μ and σ for an input image.**

- One could argue that by training the Encoder to output (μ, σ), you are asking the Encoder to give you the entire distribution $\mathcal{N}(\mu, \sigma)$ corresponding to a given input image.

- Subsequently, you can use the unique μ as the embedding for the input image. But what happens to σ that was also produced by the Encoder? What follows shows that the calculation of σ is critical for the learning required by the Autoencoder.

- That brings me to the learning required by the sort of Encoder described above. If we wish to use self-supervised learning — since we have no other options anyway — we must get the Decoder to generate from the output of the Encoder an image that, to the best extent possible, looks like the input image.

- Considering that the Encoder has provided us with the distribution $\mathcal{N}(\mu, \sigma)$ that describes the encoding process, we could sample the distribution and feed it into the Decoder for the production of the desired output. Doing so would amount to using the network architecture shown in Figure 12.

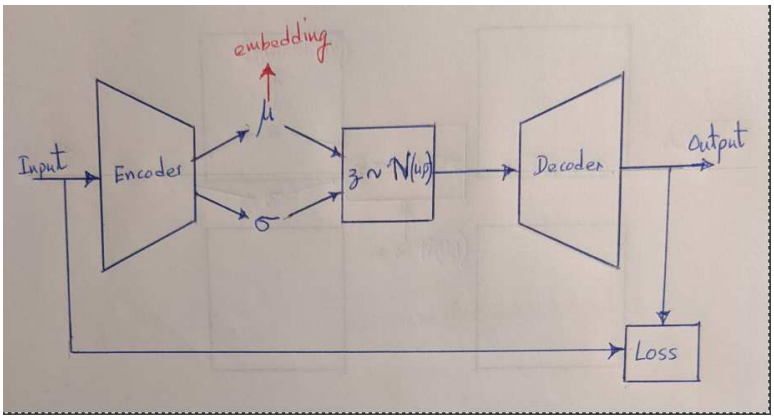

Figure 12: *Variational Autoencoder — the main idea*

- Unfortunately, drawing a random sample from a distribution is not a differentiable step — differentiability being a fundamental requirement for any neural learning. Said another way, there is no way to backpropagate the loss through the computational step

$$z \sim \mathcal{N}(\mu, \sigma).$$

- This implementation difficulty is resolved by using the approximation:

$$z \;=\; \mu \;+\; \sigma * \epsilon \tag{49}$$

where $\epsilon \sim \mathcal{N}(0, 1)$ as shown in Figure 13.

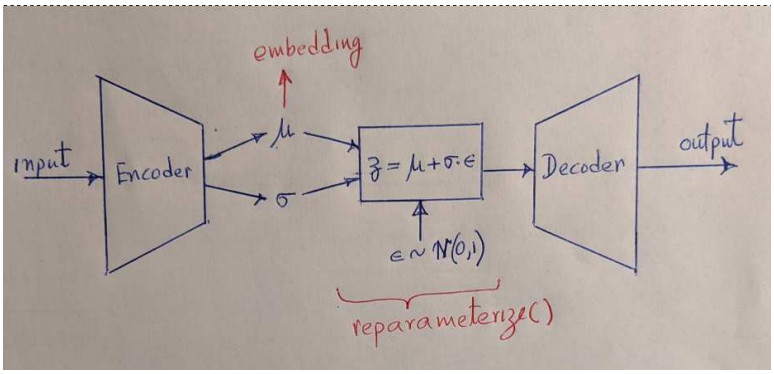

Figure 13: *Variational Autoencoder — What's actually implemented*

- Note that the parameter ϵ in Eq. (49) is not meant to be a learnable parameter — it cannot be because of the reasons already stated. Drawing a sample from from a distribution is not a differentiable step and therefore loss cannot be propagated through it. However, the calculation of the latent vector z *is* differentiable — despite the fact that it includes a non-learnable parameter ϵ.

- Therefore, for any input x, the job of learning is to figure out the best values to use for (μ, σ) for that input. The role of ϵ is to give a differentiable degree of freedom to the Decoder when it seeks to generate an output image from the latent vector z that is a good approximation to the input.

- The approximation represented by Eq. (49) is frequently referred to as the "reparameterization trick" — although there is nothing terribly tricky about it — and implemented in a function whose name is usually $reparameterize()$.

Back to TOC

8: Acknowledgments

The Introduction section talked about two examples of how things can go wrong in very high dimensional spaces. I first became aware of these effects in a presentation given by Tanmay Prakash in an RVL talk. Tanmay produced a truly outstanding Ph.D. dissertation in Purdue RVL. He is now with Google.

Sections 5 and 6 of this tutorial are based on Zhu Yan's Ph.D. dissertation entitled "Further Insights into Subspace Methods for Face Recognition Applications" that was submitted to Nanyang Technological University, Singapore, in 2008. That material is included here with her permission and with the permission of her dissertation supervisor Professor Eric Sung.

Thanks are also owed to Chad Aeschliman, formerly at Purdue Robot Vision Lab, for implementing the PCA and the LDA algorithms on the FacePix database and for supplying the results shown in this tutorial. Chad produced a very impressive Ph.D dissertation in Purdue RVL on the subject of tracking multiple moving objects in aerial videos.

I must also thank Donghun Kim for supplying the manifold representations of the face data that you saw in Section 2.3 of this presentation. Donghun also received his Ph.D. from Robot Vision Lab at Purdue with a very impressive dissertation on "Recognizing Faces in the Wild." The exact title of this dissertation is "*Pose and Appearance Based Clustering of Face Image on Manifolds and Face Recognition Applications Thereof.*" I believe Donghun is now with Samsung.