Machine Problem 3

Background

Thus far we have worked on the MNIST data of handwritten digits with KNN and the Naive Bayes classifier. Each image of dimension 784 is relatively large when we consider the machines we are running these programs on, so in this assignment we are to compute a subspace to project the images onto using some information of the data. To project the images we will use the Principal Component Analysis (PCA) and the Fisher Discriminant Analysis (FDA) algorithms. Once we project the images, we are to run KNN and Naive Bayes on the data and compare our new results to our past results.

Assumptions

PCA and FDA are relatively simple when you consider the linear algebra and multivariate statistics, so to save time you can assume that the PCA and FDA programs that I provide follow the logic described in the "Dimensionality Reduction" lecture slides with some exceptions that I will clarify. Also because of the large amount of results I have left all the results as appropriately labeled files in a folder called Results.

Algorithm

PCA is performed exactly as specified in the slides but instead of performing operations per vector, matrices are constructed where each column is a vector, and those matrices are multiplied. To calculate the coordinate for each vector we applied the formula in *Figure 1.1,* where the Z_i is i-th coordinate of the K dimensional projected vector. In my implementation, I computed the projected vectors with the formula in *Figure 1.2 (read top to bottom),* where x_i is an original 784-dimensional image, z_i is the K-dimenstional projected image, m is the average vector and all the u_i^T are K transposed 784-dimensional eigenvectors whose eigenvalues are the largest such that as a sum they constitute 95% of the total sum of the covariance matrix's eigenvalues. If you observe the code, this implementation works by the mechanics of composite matrices, and is much easier to implement if everything is set.

$$Z = U^T(X - M)$$

$$Z = (z_1, z_2, ..., z_n)$$

$$U^T = (u_1^T, u_2^T, ..., u_n^T)$$

$$X = (x_1, x_2, ..., x_n)$$

$$M = (m, m, ..., m)$$

$$m = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Figure 1.2

$$z_i^n = \boldsymbol{u}_i^T(\mathbf{x}^n - \mathbf{m})$$

Figure 1.1

FDA like PCA is performed exactly like in the slides using the following formulas:

1. $$S_w = \sum_{i=0}^{9} \left(\frac{1}{N_i} \sum_{n \in C_i} (x_k - m_i)(x_k - m_i)^T \right)$$

2. $$S_b = \sum_{i \neq j} (m_i - m_j)(m_i - m_j)^T = \sum_{i=0}^{8} \sum_{j=i+1}^{9} (m_i - m_j)(m_i - m_j)^T$$

3. $$A = (S_w + I\epsilon)^{-1} S_b$$

Where the following are set:
- x_k is an element of C_i
- A is the matrix we want to extract the eigenvectors from.
- I is the identity matrix.
- N_i and m_i is class i's, C_i, population size and mean respectively.
- Epsilon in the definition of A is a very small value, approximately 0.001.

When applying equation 2 to calculate the projected images, I found that if instead I set the indices i and j to go from 0 to 9, with the restriction of not computing the outer product when i = j, my projected images produced the same error when ran with KNN and Naive Bayes. I believe this is because S_b still becomes a matrix that spans the space spanned by the class mean vectors which will still have a rank of at most 10 - 1 = 9 and thus only span a space of dimension 9.

In equation 3, I include the Epsilon with the identify matrix to construct an invertible version of S_w because as previously described it has only rank of at most 9 and thus is not invertible in 784-space which would make finding the eigenvalues harder. As we will see, this is works quite nicely though.

Implementation

For this assignment I implemented everything in C++. I revisited lapack linear algebra library but found that there were no functions that worked with eigenvectors nor flexible functions for matrices. I decided to switch to the GNU's Scientific Library (GSL) which contains many more mathematical tools which in this case were extremely practical and easy to use. Aside for required libraries, I wrote all the code in Apple's Xcode 6 IDE which made everything extremely fast and easy to debug. The four provided folders are actually the project folders for each algorithm. To find the source code, see the README file.

Experiment Setting

As specified, I calculated two files one of the projected images of PCA and the other for FDA. I ran KNN with the 3 test protocols and ran Naive Bayes using just MLE.

With PCA, I calculated a projected subspace of dimension 154, but as a part of my experimentation I tried different subspace dimensions and found that no tuning was needed as the generated errors were very good if not the best.

With FDA, adding the scaled identity matrix enabled me to not just produce 9 dimensional projected images, so I tried higher values for the dimensions by producing more than 9 eigenvectors; this produced some interesting results. I tried tuning the scaling of the identity matrix and found very little if not any change so I left it at a value of 0.001.

Results

For PCA, the new images of dimension 154 did not affect the error in KNN. For Naive Bayes (NB) I saw an 8% increase in accuracy. My original error for NB was around 20% and it decreased down 12% which considering the Naivety in NB is really good. This decrease makes sense because PCA de-correlates the pixels and their neighbors.

For FDA, there were some interesting results. For 9 dimensional images, KNN produced an increase in error of about 20%, so on average there was 22% error. This is not surprising considering that a lot of information was lost. In NB, the error went up by 6%, i.e. from 20% to 26 % error. This is the interesting part; I believe that this happens because there is more information that is lost compared to the decrease of within-class scatter. By a mistake that I made in my computation of S_w, missing some additions that need to be made to S_w, I discovered that if we decreased S_w we can decrease the error of the projected images in KNN and NB lower than PCA which makes sense because your creating a bases that consolidates classes much more than it normally would.

Because my matrix A defined equation 3 has rank of more than 9, I tried producing more eigenvectors than 9 and tried KNN and NB with these higher dimensional projected images and found that it produced extremely low errors. KNN: about 0.01 % error and NB: about 14.79% error. Since this was not a part of the assignment I did not keep the results for these trials.

Conclusions/Lesson

Overall, this was a long but enriching assignment. I discovered that, by the math and its effects on the data we can avoid the problem of high dimensionality and de-correlate features from data to produce data that could be processed much faster with KNN, NB, and possibly other Machine Learning algorithms. With FDA, and PCA my KNN program ran for 2 minutes, 4 minutes faster than it was with 784-dimensional data. These projected images with lower dimensions could prove useful in data processing in many fields including embedded devices and mobile development.

Most important of all, coming into this assignment I knew that the determinant of a square matrix is equal to the product of its eigenvalues; I knew that the determinant describes the volume of the span of all its column vectors, assuming the vectors are all linearly independent and span the space described by the dimensions of the square matrix. With that in mind, in PCA, I discovered that by picking the K largest eigenvalues that constitute at least 95% of the sum of all eigenvalues, with their respective eigenvectors; we were essentially picking out the vectors, or “spanning features”, that described the space or spanned the space the best and thus described the images the best.