Proposing a New Framework for Biometric Optical Recognition for Handwritten Digits Data Set

Author: Marius Iulian Mihaiescu, Computer Science Department, Titu Maiorescu University, Bucharest, Romania, mihmariusiulian@gmail.com

In this article we examine and demonstrate how we can recognize the handwritten digits using Kernel Discriminant Analysis and we make investigation on the optical recognition of the handwritten digits data set. The framework is based on linear discriminant analysis and Fisher's linear discriminant. The output of the framework represents a combination which it can be used as a linear classifier.

Introduction

Linear discriminant analysis (LDA) and Fisher’s linear discriminant are two important methods used in statistics, pattern recognition and machine learning for finding a linear combination of features which characterizes two or more classes of objects or events. The output results which is a combination it can be used as a linear classifier or, more commonly, for reduction in dimensionality before the later classification.

LDA is very similar to principal component analysis (PCA) in the sense that both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data, maximizing the following objective function:

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

Where:
are the Between-Class Scatter Matrix and Within-Class Scatter Matrix. The most optimal solution can be found by calculating the eigen value of $S_B^{-1}S_w$ and taking the eigen vectors corresponding to the largest eigen values to form a new basis for the data.

### Kernel Discriminant Analysis (KDA)

Kernel Discriminant Analysis is one of the components from LDA which is used in non-linear distribution, such as KPCA is for PCA. The primary objective of kernel discriminant analysis is used to find a transformation maximizing for between-class variance and minimizing the within-class variance. This aspect can be demonstrating with Kernels, and can be represented as:

$$j(w) = \frac{w^T S_B w}{w^T S_w w}$$  \hspace{1cm} (1)$$

where

$$S_B = \sum_{c=1}^{C} (\mu_c - \bar{x})(\mu_c - \bar{x})^T$$  \hspace{1cm} (2)$$

$$S_w = \sum_{c=1}^{C} \Sigma_{x \epsilon c}(x_i - \mu_c)(x_i - \mu_c)^T$$  \hspace{1cm} (3)$$

\begin{align*}
K_c & \text{ represents the kernel matrix for the classes } c, \mu_c \\
& \text{ which represent column} \\
& \text{ means vector for } K_c, I \text{ represent the identity matric, } I_c \\
& \text{ represent the number of samples in class } c, \text{ and } I_{lc} \\
& \text{ represent the } l_c \times l_c \text{ matrix with all entries } 1/l_{c}. \text{ The complete explanation for KDA is situated beyond the goal of this article.}
\end{align*}

A powerful and interesting tool is represented by the Kernel trick. It is very useful and powerful because it provides a bridge from linearity to non-linearity for any algorithm which entirely depends on the dot product between two vectors. These result from the fact that, if we first embed our input data into an infinite-dimensional space, a linear algorithm operating in this space will behave non-linearly in the original input space. In this moment, the Kernel trick is really attractive because that embedding does not need to be ever computed. If our algorithm can be expressed only in terms of an inner product

between two vectors, all we need is to replace this inner product with the inner product from some other suitable space. That is where the “trick” takes place: whenever a dot product is used, it is replaced with a Kernel function. The Kernel function represents an inner product in a specific space, and is usually denoted as:

\[ K(x, y) = \langle \varphi(x), \varphi(y) \rangle \]  \hspace{1cm} (7)

In the case of using the kernel function, the algorithm can be carried into a higher-dimension space without explicitly embedding the input points into the respective space. To see why this is highly important to be used, it’s recommended that you may check the input text in Kernel Principal Component Analysis provided in the C# application and Kernel Support Vector Machines for Classification and Regression also provided in C# application which includes a video proof of the Kernel trick.

Some common Kernel functions include the linear kernel, the Polynomial kernel, and the Gaussian kernel. Below is a simple list with their most interesting characteristics.

<table>
<thead>
<tr>
<th>Kernel Type</th>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Kernel</td>
<td>The Linear kernel is the simplest Kernel function. It is given by the common inner product ( &lt;x, y&gt; ) plus an optional constant ( c ). Kernel algorithms using a linear kernel are often equivalent to their non-kernel counterparts, i.e., KPCA with a Linear kernel is equivalent to standard PCA.</td>
<td>( k(x, y) = x^T y + c )</td>
</tr>
<tr>
<td>Polynomial Kernel</td>
<td>The Polynomial kernel is a non-stationary kernel. It is well suited for problems where all data is normalized.</td>
<td>( k(x, y) = (\alpha x^T y + c)^d )</td>
</tr>
<tr>
<td>Gaussian Kernel</td>
<td>The Gaussian kernel is by far one of the most versatile kernels. It is a radial basis function kernel, and is the preferred kernel when we don’t know much about the structure of the data we are trying to model.</td>
<td>( k(x, y) = \exp \left(- \frac{</td>
</tr>
</tbody>
</table>
Analyzing the code source

The KDA code provided here is part of a bigger framework build by Cesar Roberto de Souza (see acknowledgment section). It combines various topics on statistics and machine learning which was needed through different researches. In this moment, the framework supports PCA, KPCA, LDA, KDA, PLS, SVMs, HMMs, LM-ANN, and a lot of other algorithms. It is called Accord.NET, and is an extension framework to AForge.NET, a popular framework for computer vision and machine learning. It is currently being hosted at http://accord-net.origo.ethz.ch. A part of the main goal of this article beside to build some documentation and sample applications for it is to provide an idea of how the framework can be applied in biometric handwritten recognition process. The source code available in this article contains only a small code snippet from the framework necessary for kernel discriminant analysis. The classes that are needed are illustrated in the picture below.

![Figure 2: The Accord.NET classes](image)

Digit Recognition

The UCI Machine Learning Repository² represent an impressive collection of databases, domain theories, and data generators which can

---

be used by the machine learning community for the empirical analysis of different machine learning algorithms. One of the available datasets is the Optical Recognition of the Handwritten Digits Data Set which we have used for biometric handwritten recognition process.

In Optical digits data, digits are represented as 32x32 matrices. They are also disposable in a pre-processed form in which the digits are divided into blocks of 4x4 and the number of on pixels has been counted in each block. This generates a matrix of 8 rows and 8 columns as input matrices where each element is an integer between 0 and 16.

Dimensionality reduction represents one of the most important steps if we are going to use these classifiers which suffer from the Curse of Dimensionality. The kernel methods have no issues in processing large dimensionality problems because they do not suffer modifications from this kind of limitations.

![Sample digits extracted from the raw Optical digits dataset](image)

**Figure 3:** Sample digits extracted from the raw Optical digits dataset

Kernel methods are very attractive because they can be applied directly to problems which would require significant thinking process on data pre-processing and extensive knowledge background about the structure of the data being modeled. Even if we have knowledge about the data, a blind application with kernel methods often brings as an output different interesting results. Obtaining optimality using different kernel methods can become an extremely difficult task because we have an infinite space of choices of different kernel functions from where to choose - and for each kernel, an undefined parameter space from where we can tweak.

The code presented below present how a Kernel Discriminant Analysis
is obtained. To have in consideration the fact that the inputs are given as entire vectors with 1024 positions. This would be useless if we use Neural Networks, for example.

```csharp
// Extract inputs and outputs
int samples = 500;
double[,] input = new double[samples, 1024];
int[] output = new int[samples];

// Create the chosen Kernel with given parameters
IKernel kernel = new Gaussian((double)numSigma.Value);

// Create the Kernel Discriminant Analysis using the selected Kernel
KDA = new KernelDiscriminantAnalysis(input, output, kernel);

// Set the threshold of minimal proportion to keep components in the analysis
kda.Threshold = (double)numThreshold.Value;

// Set the regularization constant to avoid singularities at the solution
kda.Regularization = (double)numRegularization.Value;

// Compute the analysis.
kda.Compute();

// Show information about the analysis in the form
// (most properties can be databound directly to visual controls)
dgvPrincipalComponents.DataSource = kda.Discriminants;
dgvFeatureVectors.DataSource = new ArrayDataView(kda.DiscriminantMatrix);
dgvClasses.DataSource = kda.Classes;
```

**Classification of digits by KDA**

Nevertheless, as we have said, the discriminant analysis of Kernel doesn't suffer because of the curse of dimensionality, meaning it may be applied without any processing to the raw 32x32 digit dataset. We only need to transpose the 32x32 matrices into vectors of 1024 length. Anyway, considering the not so efficiency of KDA method, this is not very fit to use in a large
dataset. KDA scales as $O(n^3)$ to the number of training samples, although the increase in dimensions is nearly unnoticeable in the time of running the analysis. Moreover, its solutions don’t compare to Support Vector Machines (SVMs), which means the Kernel matrix will require a lot of memory space to accommodate during classification. We only require a subset of the training dataset from the Optical digits’ set in order to avoid the long processing times in this demo. First five hundred entries are used for training, while the rest is allocated to testing. The classification using kernel discriminant analysis is often considered as the minimal distance between a data point which is projected into the feature space and the class mean in the respectively feature space. In the next example we can see the motivation which stands behind this approach.

**Figure 2:** The image represents the problem of Yin Yang example. Left image illustrate the original problem which is provided in the input space, the right image illustrate the projection resultant of a KDA which is done with a Gaussian kernel which have the sigma parameter initialized with 1.0.

The red dot from the right image represents the projection of the red dot from the left image. Observe how the dot is near to the blue class in the both images. In the analysis characteristic space, the classes have been placed more than one linear representation. In this case, we have the Euclidean distance (or, equivalently, the Malahanobis distance) to the class mean in characteristic of space which becomes a very suitable measure of class closeness.

The following code demonstrates the classification of new instances using an already computed Kernel Discriminant Analysis.

```java
// Get the input vector drawn
double[] input = canvas.GetDigit();
```

// Classify the input vector
int num = kda.Classify(input);

// Set the actual classification answer
lbClassification.Text = num.ToString();

The Application

The application which have been developed for this article, it performs the recognition of the handwritten digits using kernel discriminant analysis. The application it’s developed using C# as programming language using as developing environment Microsoft Visual Studio 2010 Ultimate Edition. For those who are interested in the code source please send an e-mail to cesarsouza@gmail.com; mihmariusiulian@gmail.com or radu.boriga@utm.ro.

To run the application, you must execute first the executable, and then click on the menu File and select Open. This option will load some data entries from the Optical Digits dataset into the application.

![Image](image_url)

Figure 3: The left image represents the data entries which are loaded in the application. The right image present some detailed information about the different distinct classes of hand written digits.

To perform an analysis process, we click on the Run analysis button. When the analysis will be complete, the rest of the tabs (KDA, Classes, and
Classification) will be populated with the corresponding analysis’ information. Each factor founded during the discriminant analysis is represented in a pie graph, which helps you to have an easy visual inspection.

![Pie chart showing component proportion](image1)

**Figure 4:** Factors which have been found using the Discriminant analysis with their relative importance. Observe that the initial input space of 1024 dimensions, only 9 is selected as important to the classification problem.

When the analysis is complete, we can go through testing its classification ability in the data set. The rows marks with green color illustrate the correctly identified by the space Euclidean distance classifier which is represented as a discriminant. We can observe that this is correctly identified from the testing data. The training data set are disjoint and independent.

![Classification results table](image2)

**Figure 5:** The default values in the application. In the status bar we can observe a 92% accuracy.
We can now use classifies the new digits which are represented directly in the application. This step can be accomplished when the analysis process has been completed and validated. In the right part of window application we have the response which is relative for each of the discriminant functions that have been used and implemented.

![Figure 6: The recognition process of the digit](image)

**Conclusions**

In this article, we explained in details how Kernel Discriminant Analysis can be used in the problem of biometric handwritten recognition with positive results. KDA, has some limitations, but it does not represent a threshold in our way. By comparison with Support Vector Machines (SVMs), its solutions are not rare. We have a poorly scales specific for the input sample size as $O(n^3)$ despite having no problems which are dealing with high-dimensionality input vectors.

The choice of the kernel represents one of the most important problems. This problem is often flexible with grid search and cross-validation, which are very expensive operations when it’s coming up about the terms of processing power and training data available.

Most Kernel-based methods have a big advantage in comparison with
more traditional methods such as Neural Networks: they are not affected from local minima. This means that any solution which can be found will also be the best solution possible for the given values of the learning parameters.

Acknowledgements

I would like to thank to Cesar Roberto de Souza⁴ (Universidad Federal de Sao Carlos) for his great idea and his support in this direction. The framework Accord.NET on which the application is builds, it is developed by Cesar Roberto de Souza and the information provided in this article about the framework structure is provided with his acceptance.

References


⁴ http://crsouza.blogspot.ro/2012/05/accordnet-has-new-home.html