Handwritten Digit Classification

Sucharita Jayanti

Original Problem:

My original goal was to obtain a proper data set and narrow down my potential set of classification methods to one specific method.

Problems with that Problem:

I tried numerous methods to try to obtain a data set for the Telugu Alphabet and ran into the following problems (essentially in order):

- The Telugu Alphabet has 58 (18 vowels and 40 consonants) letters, and each of these letters can be combined in a number of different ways, resulting in hundreds of different symbols that need to be distinguished. Therefore, any data set that contained all of these letters would have to be huge. I therefore, knew that I would have to work with a subset of these letters if I was to try to create my own data set.
- To create a data set, the simplest method would have been to get several people to write out the letters and then scan in the results. However, this method would case several problems.
    - Finding more than 15 different people in the Hanover area who actually read and write Telugu would be difficult. While I could potentially get non-Telugu speakers to write out the alphabet, this practice would defeat the purpose of the exercise since they would show little to no inconsistency.
    - In order for the data set to be usable, the letters would have to all be approximately the same size and position on the paper. Size, to maintain resolution and image size, and position, because many Telugu letters are very similar in shape and various letters can only be told apart due to the varying positions of different symbols.
- When my first idea failed, I attempted to get sample letters out of scanned texts on the internet. However, as I stated above, there are hundreds of different symbols in the Telugu language, since each consonant can be combined with up to 1 vowel and up to 1 consonant:

    So the number of symbols:
    18 vowels + 40 * 19 * 41 consonants = 31, 178 symbols
    Not all of these are practically used… but that still leaves nearly 25, 000 symbols

  It was therefore, difficult to consistently find the same symbols in different handwritings. I also once again, ran into problems with the image resolution.

- Finally I attempted to contact professors who were researching this particular problem. My quest for a data set still failed, however, in doing this I learnt that my inability to find a data set was not due to a lack of persistence on my part, but was rather, because there were no publically available data sets for me to use.
- Ultimately, I ended up understanding that sometimes, you fail, and you need to make the best of it and move on.

The New Problem – Classifying Handwritten Digits

Having put all this thought into this problem, I decided that it would be best for me to stick to a similar problem, and I have therefore moved onto trying to classify handwritten digits.

- **The Data Set:**

  The data set I'm using can be found at:
  And belongs to Semeion Research Center of Sciences Communication, Rome, Italy.
  It consists of 1593 images, each represented as a 1 by 256 matrix (to be reshaped into a 16 by 16 matrix).
  The images have been converted into black and white by rounding the value of each pixel into a 1 or a 0.

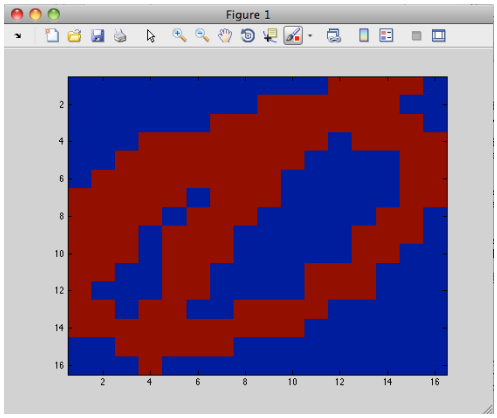

Figure 1: *An image from the data set representing the digit 0*

  I then split this data set into a training set and a testing set.

- **The Algorithm:**

  I have implemented the following algorithms:
  - Principal Component Analysis (PCA) to explore the data.
  - Fisher Linear Discriminant (FLD) followed by a simple nearest neighbor classifier to establish a baseline accuracy.
  - K – nearest neighbor algorithm on the original data set with the original dimensionality.

- Large Margin Component Analysis (the linear version) as explained in “Large Margin Component Analysis by Torresani and Kuang-chih (2006).

Large Margin Component Analysis:

The Goal: To reduce the dimension of the training data to the most relevant components and simultaneously cluster the data based on it’s labels.

Method:
Let D be the original dimension of the data.
Let d be the dimension you wish to reduce the data to.

L is a d by D matrix

We want to optimize:

$$\epsilon(\mathbf{L}) = \sum_{ij} \eta_{ij} ||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)||^2 + c \sum_{ijl} \eta_{ij}(1 - y_{il}) h(||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)||^2 - ||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_l)||^2 + 1),$$

using gradient descent to optimize the function and using

$$\begin{aligned} \frac{\partial \epsilon(\mathbf{L})}{\partial \mathbf{L}} = \; & 2\mathbf{L} \sum_{ij} \eta_{ij} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T + \\ & 2c\mathbf{L} \sum_{ijl} \eta_{ij}(1 - y_{il}) \left[(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T - (\mathbf{x}_i - \mathbf{x}_l)(\mathbf{x}_i - \mathbf{x}_l)^T \right] \\ & h'(||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)||^2 - ||\mathbf{L}(\mathbf{x}_i - \mathbf{x}_l)||^2 + 1) \end{aligned}$$

as the update function

Results:

Note: there are 10 classes, therefore random accuracy = 10%

- **The Results of PCA:**

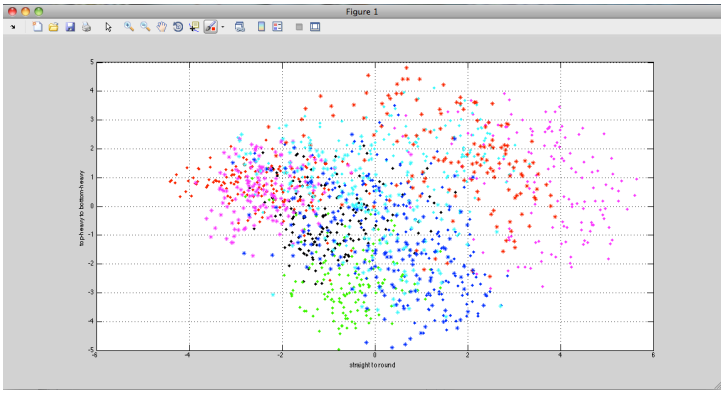

Figure 2: *PCA to 2 dimensions on the train dataset.*

The x-axis is roundess of the symbol and the y-axis is bottom-heaviness

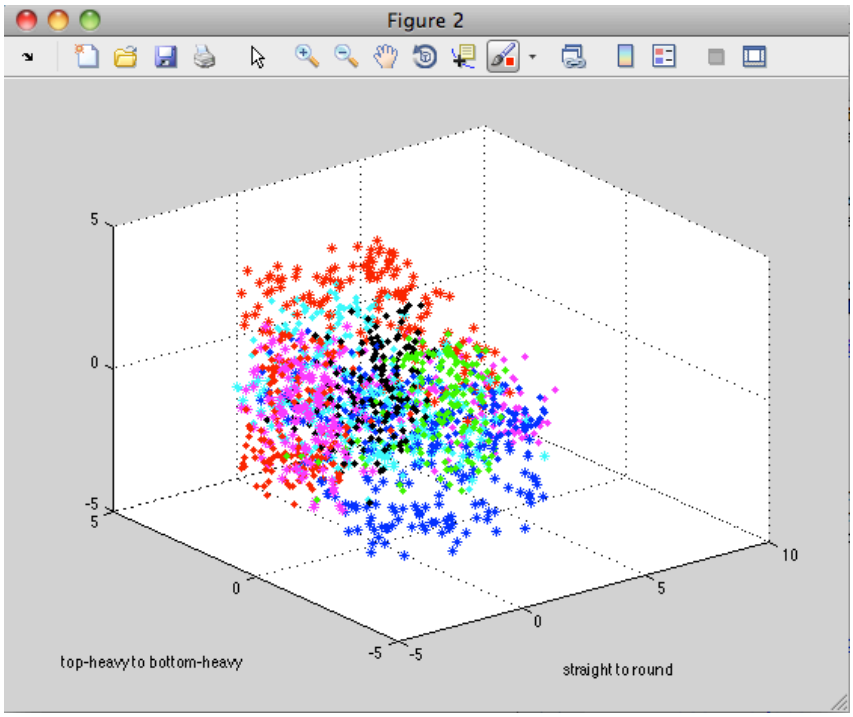

Figure 3: *PCA to 3 dimensions on the train dataset*

- The Results of FLD (Fisher Linear Discriminant):

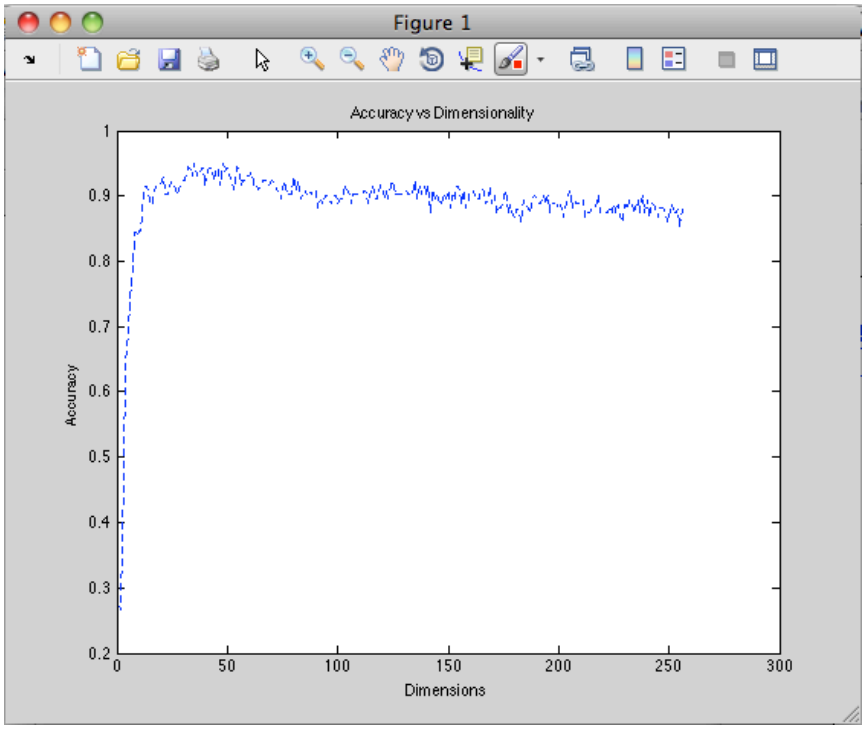

Figure 4: *Dimensions to which the data is reduced, vs the Accuracy of the FLD classifier.*

As you can see, after reducing the data to around 50 dimensions, the accuracy simply goes down.

- The Results of knn:

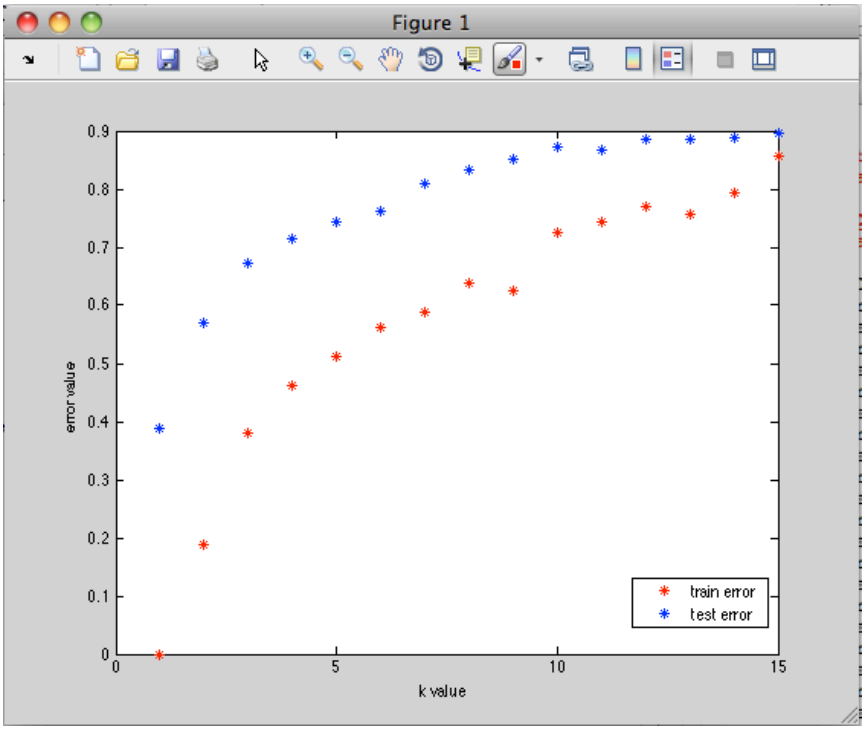

Figure 5: *k-value vs error value for knn algorithm*

- The Results of LMCA:

K- Value	LMCA Train	KNN Train	LMCA Test	KNN Test
1	0.206250	0.000000	0.192603	0.387997
2	0.393750	0.187500	0.344033	0.568737
5	0.587500	0.512500	0.593161	0.744592
10	0.756250	0.725000	0.769714	0.872994
15	0.893750	0.856250	0.844382	0.896022

<u>Observations and Trends:</u>

LMCA has 4 variables:

- alpha for the gradient descent
- N = number of dimensions to which you are reducing the data
- K of the knn at the end

- c for the relative weight of the factor that distances the various clusters.

1. As k increases, c must also increase to achieve the best results.
2. Lower ks tend to have the best results, however, that could be due to the small training set used (150 samples -> 15 samples for each class).
3. Alpha had to be incredibly low (10^-7 or smaller)
4. The update function contains a nested loop that runs |training set|^3 times.
5. For the comparison between knn and LMCA:
    a. Knn consistently does better on the training set. However a large part of this is most likely the bias caused by the original point being one of the k nearest neighbors.
    b. LMCA consistently does better on the testing set, and is therefore actually useful.

References:

Belhumeur, Peter N, Joao P Hespanha, and David J Kriegman. "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection." *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* 19.7 (1997): n. pag. *http://www.cs.columbia.edu/*. Web. 28 May 2012.

Torresani, Lorenzo, and Kuang-chih Lee. "Large Margin Component Analysis ." *Advances in Neural Information Processing Systems* 19 (2006): n. pag. *http://books.nips.cc/*. Web. 28 May 2012.