Feature Extraction and Inference for Handwriting Recognition

Yo Sup "Joseph" Moon, Dana Modzelewski, William Chambers, Claudia Friedsam

Harvard University: CS51
Final Project Writeup
Due 11:59PM, April 29th, 2012.

§ **Demonstration Video:** `http://www.youtube.com/watch?v=JEPiBFwcsiY`

1 Overview

We implemented two different machine learning algorithms in Python to apply to the problem of handwriting recognition, specifically on the digits 0–9. The two algorithms we chose are k-means clustering and principal component analysis (PCA). The k-means algorithm assigns each of the different data points to one of the k clusters in such a way that the distance between each point and its cluster's mean, or centroid, is minimized. The PCA algorithm serves to reduce the dimensionality of the data by linearly transforming them onto a new coordinate system in which each basis vector is chosen to maximize the variance in the data. PCA can thus help better separate the data by eliminating unimportant variations while preserving the most important ones. To identify the labels to handwritten digits in the test set, we implemented both simple unsupervised k-means clustering as well as supervised learning which incorporates both k-means and PCA.

2 Algorithms

2.1 K-means Algorithm

The k-means algorithm is a specific instance of the Expectation Maximization algorithm (EM), an iterative procedure to find a maximum a posteriori estimate for parameters in a statistical model. The following is a basic description of the algorithm:

- Create initial random partition of n data points in k clusters
- Calculate centroids C of k clusters
- Initialize C_{old} with empty list
- While $C \neq C_{old}$ **do**:
  - $C_{old} \leftarrow C$
  - Reassign data points to closest cluster centroid
  - Calculate C for new clusters
- Return C and k.

2.2 Principal Component Analysis

Principal Component Analysis (PCA) is a procedure widely used for dimensionality reduction to construct a feasible feature space of very high-dimensional data. We adapt the maximum variance variant of PCA as described in Bishop[1].

We wish to find unit vectors $\{u_i\}_{i=1}^{D}$ that describe each coordinate in the projection $\mathbb{R}^M \rightarrow \mathbb{R}^D,\ M \geq D$. The idea is to find the u_i's so that in the new vector space, the variance of the data points are maximized. Now, the mean of the projected data is $u_i^T \overline{x}$ where $\overline{x}$ is given by

$$\overline{x} = \frac{1}{N} \sum_{n=1}^{N} x_n,$$

and the variance of the projected data is given by

$$\frac{1}{N} \sum_{n=1}^{N} \{u_i^T x_n - u_1^T \overline{x}\}^2 = u_i^T \mathbf{S} u_i,$$

where $\mathbf{S}$ is the covariance matrix:

$$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \overline{x})(x_n - \overline{x})^T.$$

Maximizing the projected variance with respect to u_i by Lagrange multipliers, we have that the the stationary points of the Lagrangian is where $\mathbf{S}u_i = \lambda_i u_i$, and that the variance is precisely the eigenvalue λ of $\mathbf{S}$. Hence, our problem of finding the linear transformation $T \in \mathcal{L}(\mathbb{R}^M, \mathbb{R}^D)$ reduces to finding the D largest eigenvalues of the covariance matrix $\mathbf{S}$ of the data.

3 Planning

We began planning in March with the intention of working with the k-means and PCA algorithms in Python. Handwriting recognition was, at the time, one of several problems were were looking to address. By the first week of April we had setup a repository using GitHub, gotten Python and relevant modules working and, with Mike's help, found the CS 181 data set and began parsing it. We also had a large amount of the k-means and PCA code implemented, though the code was not completely workable or bug free.

From here we hoped to finish the main algorithms, finish parsing the data into a workable format, and investigate different techniques on handwriting recognition within the first week. By the end of the second week, we hoped to implement basic character recognition on our data set and investigate ways to visualize the results. Our plan for the final week was to finish streamlining the implementation and work on our GUI and other extensions.

In reality, bringing all of our code together into one main function took a considerable amount of time – much more than we had anticipated. We were able to meet our first week milestones, but by the end of the second week, we only had modular implementations of different parts of our future **main.py** function. PCA had a few last bugs to be ironed out when it was applied to our real data set. Some basic functionality for image the data set had already emerged. Writing one main function to handle all of the algorithms was an important and difficult step in creating a working implementation and was worked on through the last week in addition to working on the GUI and improving our image display functions to include color-coding capabilities.

4 Design and Implementation

We downloaded a subset of the MNIST database from Harvard's CS 181 website. This dataset comprises of a training set of 9000 digits and a test set of 1000 digits. Each digit is represented as a 14×14 array, where each entry is a value between 0 and 255 signifying the hardness of stroke. Each array was subsequently parsed as a vector $v_i \in \mathbb{Z}/256\mathbb{Z}^{\oplus 196}$ and concatenated to form a training matrix $\mathcal{M}_{196\times 9000}^{train}$ and a test matrix $\mathcal{M}_{196\times 1000}^{test}$. The two matrices were subsequently stored as a python pickle, which is python's native way of storing files preserving algebraic structure. This greatly facilitated the testing, evaluation, and reproducibility of our pipeline.

4.1 Naive Implementation

Our simple, unsupervised clustering implementation makes use of k-means only and takes a set of data points and a desired number of clusters, k. A naive application of this implementation is to choose ten clusters in hopes that any one cluster will contain all instances of a digit and no occurrences of any other digits. Choosing a larger number of clusters might further reduce the number of different digits within one cluster. To our surprise, even k-means alone produced reasonable results, with the algorithm capable of distinguishing between well-defined digits such as 0's and 1's. However, running k-means on a 196×1000 array is very time-consuming and circumstantial at best; we strove for a more deterministic and time-efficient procedure for digit classification.

4.2 Supervised Learning with Sub-clustering

To improve upon the accuracy of such a simple method, we implemented supervised learning. This method requires a learning data set to train our model, which can then be applied to trial data to obtain classifications. To this end, we combined k-means with PCA for better results. We start by finding the covariance matrix of the data set as well as the D eigenvectors corresponding to the D largest eigenvalues of the covariance matrix. By projecting the data onto $\mathbb{R}^D$ by the linear transformation T defined by those eigenvectors we reduce the dimensionality of the data while maximizing the variance of the data in the new coordinate system. We proceeded as follows:

- Calculate covariance matrix $\mathbf{S}$ on the training data given by $\mathcal{M}^{\text{train}}_{196\times 9000}$. We obtain the linear projection $\mathrm{T} \in \mathcal{L}(\mathbb{R}^{196}, \mathbb{R}^D)$.
- Subdivide each of the 10 classes of the digits (0-9) into k clusters, resulting in $10 \cdot k$ subclasses in total. Acquire the centroids $\{c_i\}_{i=1}^{10k}$.
- Project the test set given by $\mathcal{M}^{\text{test}}_{196\times 1000}$ by the linear projection T.
- For $v_i \in \mathrm{T}\mathcal{M}^{\text{test}}_{196\times 1000}$ find $\min_{j\in\{1,\ldots,10k\}}\{\mathrm{d}(v_i, \mathrm{T}c_j)\}$ and assign v_i to subclass defined by c_i, which gives a corresponding classification to one of the equivalence classes $0-9$. Here $\mathrm{d}(\cdot)$ is the Euclidean distance.

The motivation for such an approach comes from the linearity of our projection T. Firstly, it is our assumption that the centroid of a subspace in $\mathbb{R}^{196}$ given by the representation of the digits $0-9$ is an appropriate identifier for itself. To account for the loss of information in letting a single centroid define the subspace, we let multiple centroids define the subspace by implementing k-means. Secondly, we note that implementing k-means on $\mathbb{R}^{196}$ is computationally infeasible due to its large dimensionality. We wish to implement k-means on $\mathbb{R}^D,\ D \leq 196$. This will only be a valid procedure if the centroids were preserved under T. To our advantage, by the linearity of T we know:

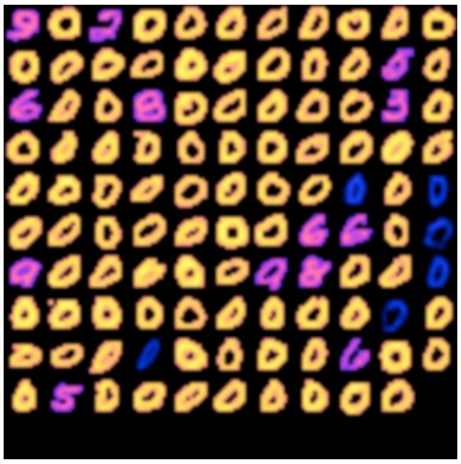
$$\frac{1}{N}\sum_{i=1}^{N}\mathrm{T}(v_i) = \frac{1}{N}\mathrm{T}\left(\sum_{i=1}^{N} v_i\right) = \mathrm{T}\left(\frac{1}{N}\sum_{i=1}^{N} v_i\right)$$

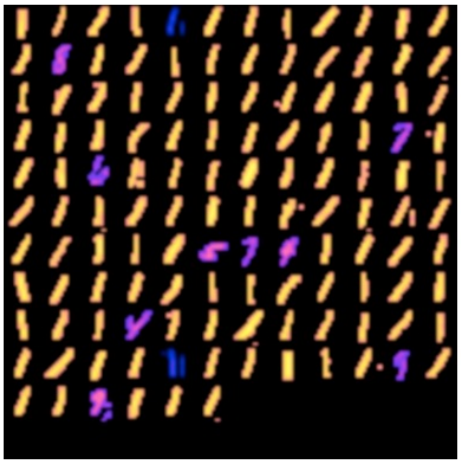
and thus the centroids are preserved under the mapping. Hence we can reduce the computational cost of clustering on $\mathbb{R}^{196}$ by instead clustering on $\mathbb{R}^D$ and identifying the corresponding clusters in $\mathbb{R}^{196}$. We found that running PCA with even 10 principal components and 3 sub-clusters gave a very reasonable assignment to the test digits, with the recall of certain digits such as 0 and 1 to be near 90 percent. We found that running PCA is cheap in computational cost; upwards of 50 principal components can be calculated within seconds. However, the running time of k-means grows exponentially with the number of principal components D, so there was always a trade-off between the two algorithms. We experimentally verified that $k = 3; D = 10$ is a very reasonable parameter for our model; the algorithm is implemented very quickly and the recall and precision is very reasonable (> 0.6) with regards to its run-time.

4.3 Output

We display the outputs of our implementations of the algorithms as image files. For the simple, unsupervised k-means only implementation, k images are generated, each containing all the digits

that were assigned to a given cluster. The supervised implementations generate 10 images, one for each digit, which contain all data points that were either assigned the corresponding digit or missed by our algorithm. The confusion matrices of our procedure is subsequently computed for each digit, and relevant metrics for the success of our procedure, such as the *recall* and *precision* is calculated. The numbers are displayed in a montage and are color coded accordingly for true positives (yellow), false positives (pink), and false negatives (blue)[1].

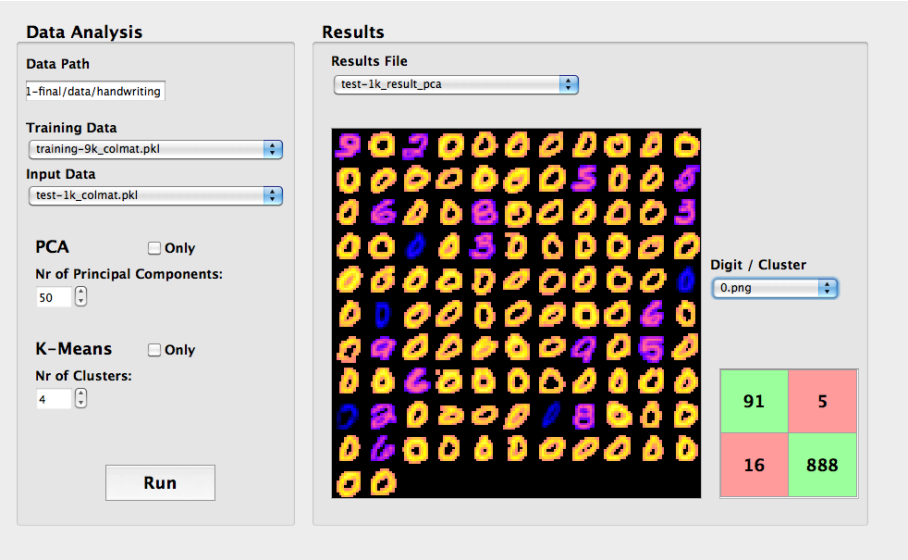




4.4 GUI

The GUI was written in PyQt it allowed us to test the parameters of our model and visualize the output in a pain-free way.

[1]Our code was adapted from an example from the web: `http://www.datawrangling.com/python-montage-code-for-displaying-arrays`



5 Reflection

There was some initial challenge for some of our group members in getting started working in Python, but the power of a scripting language and the functions of the NumPy module proved invaluable for implementing the algorithms simply. Evaluating the results of our algorithms was difficult to do at intermediate steps, but became easier once we had an easy to read image output. As mentioned above, bringing the different components of the project together was one of the more difficult aspects, in part because of the need for extra thought on data formats across modules.

After completing this project we are all walking away with a good understanding of how both k-means and PCA work and how we can use the two in tandem to improve the accuracy of our results. Coming into the project most of us had no experience coding in Python and were faced with the challenge of learning the syntax of a new language or even programming in a scripting language in general. In that respect such a project was doubly beneficial as it forced us to learn not only the algorithms but the language as well.

One thing that we made sure to take time to do at the end of each meeting was make a plan for what we hoped to accomplish in the coming week as well as who would be in charge of doing what. In general, Joseph and Dana were in charge of the PCA algorithm and Claudia and Will were in charge of k-means. In terms of more specialized tasks, Claudia oversaw the developments of the GUI, Joseph oversaw the general construction of the pipeline with respect to the application of the feature extraction in the learning step, Dana oversaw the generating of visual output to evaluate our results, and Will oversaw the development of our unsupervised learning methods.

If we had more time with this project we would apply our algorithms to other sorts of problems such as neural networks or even shape recognition on maps. It would have been interesting to find a larger data set that included letters, or, more difficultly, develop a method to use samples of our own handwriting. Lastly, with extra time we would have worked to develop our GUI into an executable program. We wish to note that our current representation is not bound to only handwriting data, but for arbitrary data in array form as the **parse_input.py** function is written in a general fashion to accommodate most types of data.

6 Advice for Future Students

The most important thing we learned from this project and would advise to future students is to take extra time in the beginning to plan and solidify your invariants. One of the largest challenges we ran into was in attempting to merge the different pieces of code each of us had written individually together into one working unit. We had taken time to go over how we intended to pass data from one function to another, but we never explicitly wrote it out. As a result, our functions would break

when one person had written them such that a data point was represented by a column in a matrix and another assumed it would be a row.

References

[1] Bishop , Christopher. *Pattern Recognition and Machine Learning.*Cambridge: Springer, 2006. Print.