AMATH 563 Clustering and Classification

Ben Mishkin

I INTRODUCTION AND OVERVIEW

One of the fundamental ways that humans recognize patterns is by picking out clumps or clusters of similarly situated objects: think how we divide our cities into neighborhoods based on demographics, history, and geography; our environment into forests, mountain ranges, and deserts based on topography, ecology, and climate; plants and animals into species based on morphology and genetics. It is therefore a natural question to ask whether we can teach a computer to do the same thing: given some data, can a computer identify clusters in the data, and then assign a new piece of data to a given cluster? In this paper, we examine some clustering methods in the context of facial recognition.

II THEORETICAL BACKGROUND

Suppose we are given a set of points in some potentially high-dimensional feature space (we'll return to what this space is and how the points are generated below). Clustering and classification algorithms attempt to answer several related questions:

- Are there clusters in the data, and if so, how many?
- Where is the center of each cluster?
- Where are the boundaries of each cluster?
- If given a new data point, which known data points does it most resemble, that is, which cluster is it in?

There are a number of algorithms that attempt to answer these questions, all of which take different approaches to the problem and all of which have variations and choices for the modeler to make. The ones considered in this paper are:

- K-Means
- K-Nearest-Neighbors (KNN)
- Naïve Bayes
- Linear Discriminant Analysis (LDA)
- Support Vector Machines (SVM)
- Decision Trees

There are some commonalities between these methods, but also substantial differences, both in the algorithmic structure of these methods and the exact questions they seek to answer. We provide a brief overview of the salient details of each:

- K-means attempts to divide the data into *k* clusters, the number *k* being provided by the modeler. The algorithm works by iteratively updating a center of mass for points close to one of the *k* centroids. The final cluster map is essentially a Voronoi partition of the the feature space based on the found centroids. It requires no pre-labeling of points on the part of the modeler, the only choice being the choice of *k*.
- KNN answers the final question of the list above by comparing a point of interest to its *k* nearest neighbor points which belong to known classes, then assigns a classification to the point of interest based on the identity and distance of these neighbors. KNN has the advantage that it can easily model substantially nonlinear decision boundaries. Whereas K-means takes into account global structures in the

data, KNN is a much more local approach.

- Naïve Bayes takes a probabilistic approach to the classification problem, by leveraging Bayes' theorem to represent conditional probabilities of a data point being in a certain class given its location in feature space. Importantly, it assumes all features are independent of each other, which allows the problem to be reduced to a series of one-dimensional problems, which can be an advantage for high-dimensional data.
- LDA works by attempting to project data onto a lower-dimensional subspace that maximizes the distance between inter-class data and minimizes the distance between intra-class data. As with Naïve Bayes, working in a lower dimensional subspace can have considerable advantages (see the discussion on the curse of dimensionality below). The core of the algorithm is the calculation of a Rayleigh quotient, a well-understood and optimized linear algebra problem.
- SVMs are at their core an optimization problem. The SVM seeks to construct some hyperplane that, like LDA, seeks to minimize the mislabeling of data and maximize the margin between the clusters and the hyperplane. The SVM is particularly useful since it is amenable to generalizations that allow for the construction of non-linear decision boundaries. It is one of the most important and widely used classification schemes.
- Decision trees are a hierarchical method for classification, and can be thought of as a sophisticated game of 20 questions. At each node in the tree, we ask a question that optimally splits the data into subsets, then proceed to the next node based on the features of the data and then ask a new question about the new subset of the data. Decision trees can produce complex, disjoint decision boundaries between clusters, at the cost of sometimes substantial computational complexity. They also enjoy easy interpretability, as the modeler can interrogate the tree for the rule at each node. They can be very sensitive to fluctuations in training data, and for this reason multiple decision trees, each trained on a different subset of the data can be grouped together into a *random forest*, with each tree voting on the classification of a new data point.

The above is only a brief outline of each of the algorithms. It is up to the modeler to choose an appropriate method or combination of methods for the problem at hand, recalling that in each case, there exist sophisticated variations of the basic sketches presented above. The first, and most important decision the modeler must make is whether to employ a *supervised* or *unsupervised* algorithm, that is, whether to use an algorithm that requires training data to be pre-labeled into classes. The first algorithm above, k-means, is unsupervised while the rest are supervised. There is another common classification scheme, Gaussian mixture models, that are also unsupervised and attempt to separate the data by reconstructing probability distributions, but which did not work well for the facial recognition task under consideration in this paper.

We turn now to a question we have left unexamined: what is the nature of the feature space our points live in, and how does it effect the behavior of the algorithms presented above? In general, it is often desirable to reduce the dimensionality of data as far as possible (kernel methods in implementation of SVMs being a notable exception) when employing these methods. There is a two-fold purpose, one being that lower-dimensional data is easier to interpret, but also because high-dimensional data begins to be effected by the so-called *curse of dimensionality*.

There are a number of related problems when working in high-dimensional spaces, but perhaps the most salient is the behavior of norms in high dimensions. As the dimension of a vector space goes to infinity, distances between points tend to become more uniform; that is, almost all points are almost exactly the same distance from each other. As most of the algorithms above hinge on differentiating clusters by measuring the distance between sets of points, this presents an obvious problem. Thus, intelligent dimensionality selection is a critical first step to all the methods above.

Luckily, there is a simple and principled way to do dimensionality reduction, the Singular Value Decomposition (SVD). Suppose each of our data points has *n* features associated with it. The SVD allows us to create and rank optimal new features that successively explain the most variance in our sample set. We can then select a rank-*r* projection of our data into a lower dimensional subspace while retaining an optimal amount of information about the original problem. We'll see in the implementation details below how this works for image data.

III Algorithm implementation and development

Thanks to the extensive built-in functionality of MATLAB, implementation of the above algorithms and their variations is extremely straightforward, all being simple one-line commands. The most code-intensive part of this facial recognition exercise is the data preparation.

The data were given as labeled collections of a number of faces, some of which had undergone pre-processing in the form of cropping, and others which had not. The script '*Face_read_in.m'* reads these images into MATLAB and applies some preliminary reshaping and vectorizing. '*ImageSVDAnalysis.m*' then concatenates the vectorized images into a single matrix, each column of which is a different image, and creates a label vector that identifies each of the images. It then implements the economy SVD on the data matrix. '*Cropped_image_analysis.m*' then takes the matrices generated by the SVD and performs an appropriate truncation, based on the distribution of the singular values (see discussion below). The singular vectors in *v* are the features we'll use to classify images, and the dimension of *v* is the dimension of the feature space. The script then generates random permutations of the data for training and validation, taking 40 of each face as training data and leaving the other ~24 faces as test data. For each instantiation of the training data, the script uses the built-in MATLAB clustering commands to generate a clustering and classification data, then tests the result against the reserved testing data. A similar approach was used to examine the SVD of the uncropped images, and similar code (*'sex_detector.m'*) attempts the clustering based on sex.

IV Results

We'll consider first the interpretation and use of the SVD of our images. Recall the form of the SVD: if $A \in \mathbb{R}^{m\times n}$ is the matrix of reshaped images, then the SVD $A = U\Sigma V^*$ has the following interpretation. The columns of U each represent an "average face" from the training set, or more specifically, the first column represents an

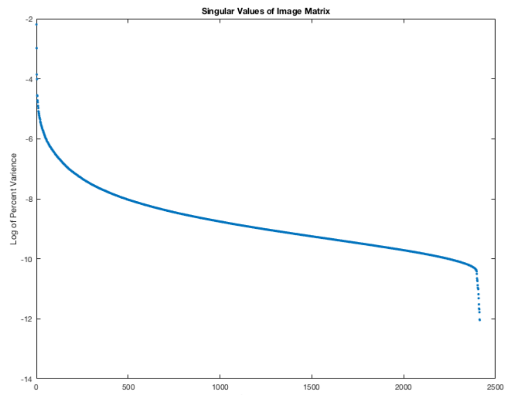

Figure 1: Log singular values of the cropped data

average face and the following columns represent successively less important modes of variation from the average face. The singular values in Σ tell us the the amount of variance in the set images that each of the modes in U captures, and the $(j,k)^{th}$ entry of V^* tells us the projection of the j^{th} image onto the k^{th} mode in U. It is this projection that gives rise to the feature space: each feature of a given image is the strength of the projection onto a given mode in U.

The next obvious question is to determine how many modes are needed to well-reconstruct the images, so we can determine what dimension feature space we should use for classification. Figure 1 is a plot of the singular values of the data matrix, and Figure 2 are various rank reconstructions of the faces. We see that even a rank five reconstruction is recognizably a human face, and that at rank 60 and above facial features are readily recognizable, with differences primarily in shading. A rank 60 reconstruction of the faces captures about 43% of the variance, while a rank 240 reconstruction captures about 65%. It takes over a rank 1500 truncation to capture 95% of the variance. In the uncropped data set, a similar percentage of singular values actually captures a similar amount of variance, but issues arise because much of that variance is in the background rather than in the facial features. Figure 3 displays the first six modes

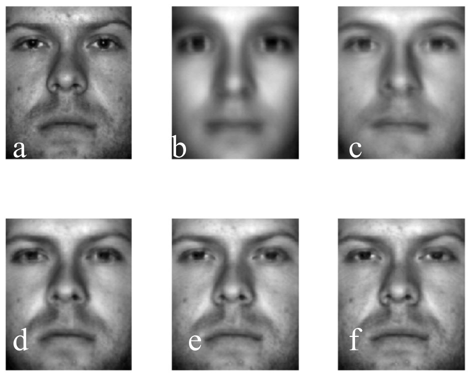

Figure 2:(a) full rank (b) rank 5 (c) rank 30 (d) rank 60 (e) rank 120 (f) rank 250 reconstructions

of the data. In the interest of exploring low-rank models, which are important for memory-limited applications, we'll take a relatively low-rank approximation of the data for the
clustering analysis and consider only the first 60 modes.

Turning now to the success of the classification schemes, KNN with 5 unweighted neighbors performed the best, identifying on average over 88% of the faces correctly. LDA and Naïve Bayes performed almost as well, at 81% and 82% accuracy respectively. The SVM got it right about 69% of the time, and the decision tree came in last at about 50% accuracy. This is all using MATLAB's off-the-rack defaults and a low-rank truncation with no image pre-processing, so it is very likely that taking

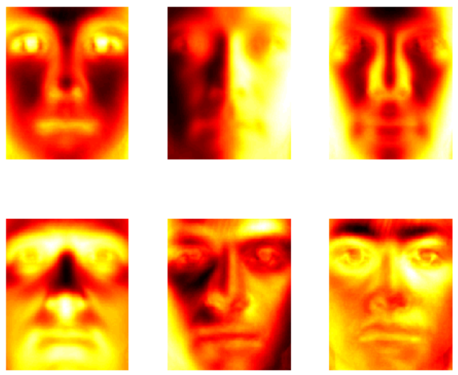
Figure 3: First Six Facial Modes

those extra steps would substantially boost accuracy, though all models performed substantially better than random guessing. In particular, we see the aforementioned sensitivity of the decision tree to the training data–it is likely that a random forest would perform substantially better, and is a future step we would like to take with these models. The success of KNN we believe arises from its joint ability to model complex decision boundaries and function well with relatively little training data for each cluster.

In the sex detection task, the rankings were almost the same, with KNN identifying the correct sex 89% of the time. Interestingly, the SVM performed much worse on this task than the face identification, getting it right only 24% of the time. This may represent the fact that the decision boundary is highly non-linear, and the unmodified SVM algorithm performs poorly with non-linear boundaries. Again, kernel methods would likely substantially boost the accuracy of this method.

The unsupervised K-means algorithm was ambivalent about the data. Running the algorithm with different K-values, on average it put faces that should belong to the same cluster together only about 45% of the time, with no clear peak at the 38 clusters we know a-priori exist in the data. We think this is because K-means needs to consider the whole set of data points when making its decisions about where the clusters lie, and the number of points in each cluster is small relative to the number of dimensions we're working in.

V Summary and Conclusions

Out-of-the-box MATLAB functionality is very effective and straightforward to use on data classification tasks, allowing the construction of remarkably accurate facial-recognition routines in just a line or two of code. Additionally, we saw that for the purposes of clustering and classification, low-rank approximations of images are often sufficient for good accuracy. We did, however, see the importance of having sufficient data to train a model on, and the sensitivity of model performance to the ability to handle non-linear decision boundaries.

References:

[1] Brunton, S. L., & Kutz, J. N. (2019). *Data-driven science and engineering machine learning, dynamical systems, and control*. Cambridge: Cambridge University Press.

[2] Kutz, J. N. (2013). *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. Oxford: OUP Oxford.

[3] Trefethen, L. N. (L. N. (1997). *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics.

Novel MATLAB Functions:

Fitcnb-Fits a Naïve Bayes model given training and test data
Classify-Fits a LDA model given training and test data
Fitcecoc-Fits a multi-feature SVM model given training and test data
Fitctree- Fits a decision tree model given training and test data
Kmmsearch-Fits a KNN model
Predict-Takes the output of the above models and classifies new data
Fitgmdist-Fits a Gaussian Mixture Model
Kmeans-Runs the Kmeans algorithm