Machine Learning for Classification and Clustering of Handwritten Persian Digits

COMP 472 - Final Project

Concordia University

April 14 2017

David Tsatsoulis ID: 25410029	Vanessa Kurt ID: 27748639	Quentin Dumoulin ID: 27734719	Lev Kokotov ID: 27758278	David Desmarais-Michaud ID: 26322891

Table of Contents

Abstract

This report details the implementation and results of several classification and clustering Machine Learning methods used to analyze a database of handwritten Persian digits. The database contains 10,000 images of black and white handwritten digits, ranging from 0 to 9 (equally weighted). The data is divided into training, validation and test sets. A number of data extraction methods are used, including Gradient, Derivative and 3 x 3 methods taken from another paper. Classification is done using a Support Vector Classifier, a K-Nearest-Neighbor Classifier, a Multi-Layer Perceptron classifier, and a Decision Tree Classifier. Clustering is done using the K-means, Affinity Propagation and Birch methods. All implementations are done in Python using the scikit-learn library. Results were evaluated and validated using Cross-Validation and Confusion Matrices. The results of the experiments conducted in the paper favour the use of the 3 x 3 data extraction method, coupled with the K-Nearest-Neighbor classifier and Affinity Propagation cluster method. The Gradient method of extraction and the Derivative method of extraction were found to be less accurate for some experiments and produced results that averaged between accurate and random guessing.

Introduction: Digit Classification and Clustering

This report details the results and implementation of our Python project for Machine Learning for Handwritten Persian digit Classification and Clustering. Using a database of 10,000 images of handwritten Persian digits ranging from 0 to 9 (equally weighted), our project implements three different feature extraction methods to convert each image into multidimensional vectors. Subsequently, each vector is fed into one of several different Machine Learning methods to classify each image into their respective categories with a high degree of accuracy. The project also implements different methods for clustering, which divides each respective numerical category into different clusters, representing the various stylistic methods in which these numbers can be written. The result is a system that is able to analyze and learn how to classify and cluster handwritten digits stored as image files.

The project implements three different methods of feature extraction: the Gradient Method, the Derivative method, and the 3x3 Pixel method (explained in more detail further in this paper). Each method results in a vector of numbers that our Machine Learning algorithms use as input for classification and clustering. Each method of feature extraction results in different levels of accuracy for results, as well as varying levels of speed and complexity. The database is divided into Learning, Validation and Test sets, which allows us to Train and Test the classification functions of our system.

Classification is implemented using four different methods: the SVM method, the Multi-Layer Perceptron method, the K-Nearest-Neighbor method, and the Decision Tree method. This report compares and evaluates the results of each classification method using a measure of straight accuracy (percentage of correct classification of the test set), a confusion matrix and cross-validation of the database. Clustering methods include K-means, Affinity Propagation and the Birch method. The report contains the results of each clustering method, as well as some visual representations of the clusters, including a centroid element to represent each category.

Databases for Training, Validating and Testing

Our database contains 10,000 images of handwritten Persian digits, ranging from 0 to 9, with equal weightings (1000 images for every number). The images contain a number of different styles and methods of writing each digit. The database was downloaded from the Concordia COMP 472 website, and each image file is in the .TIF format. All images are in black and white, and differ slightly in terms of overall image size.

Each digit category is used for learning (training and validation) and testing. Our learning set is further divided into a training and validation set. The overall breakdown is 80% learning, 20% testing, with 20% of the learning set being allocated to validation. Given that each category contains 1000 images, this results in 640 images for training, 160 for validation and 200 for testing. Our training set is used as input to our machine learning algorithms to train the system. The Validation set is then used to see if our parameters are producing results that are acceptable, before the Test set is used to finally measure the accuracy of the given method. The division of the database is done using a simple Python script to ensure the correct image files are put into their respective sets.

The test set provides the primary measure of accuracy and success of the different Machine Learning algorithms. Additionally, we employ a cross-validation method as an additional method of evaluation, which performs multiple tests using a different Learning and Test set each time (ultimately averaging out all results). This is used so that every image is at some point used both in a Learning and Test set, ensuring that our results are consistent regardless of the images chosen.

Feature Extraction Methods

Feature extraction is the method used to convert image files into multidimensional vectors that our Machine Learning algorithms understand. This project employs three different methods of feature extraction, each with different levels of complexity to run and implement. The classification and clustering methods are run and tested using each of the three feature extraction methods, and the results are compared. The feature extraction methods used in this system are the Gradient Method, the Derivative method, and the 3 x 3 method, each of which is detailed below. These methods were taken from papers cited after each section and listed in the references section.

Gradient Method

Given that each image file varies in terms of size (number of pixels), the first step in the Gradient Method involves normalizing the image by resizing all images to 42 by 42 pixels. The images are then divided into equal planes of 7 by 7 pixels to create 36 equally sized blocks. In each block, we compute the magnitude (g) and direction (θ) of each pixel using the following formulas:

$$g(i,j) = \sqrt{g_x^2 + g_y^2}$$

$$\theta(i,j) = \tan^{-1}\frac{g_y(i,j)}{g_x(i,j)}$$

These numbers are them summed, and the results stored in a 144D histogram of gradient orientation and magnitude (36 blocks x 4 orientations) (Sadri et. al., 2016).

Derivative Method

Images are normalized by resizing to 64 by 64 pixels. The images are then examined from 4 different sides (top, left, right and bottom). At each pixel, we determine the number of pixels from the edge that we find the first black pixel. This gives us 64 numbers for each of the 4 sides, providing us with a function for each side. We then compute the discrete derivative of each function. Finally, we sample every 4 pixels on each side from each derivative, to provide us with a 64D vector of numbers to represent each image (Sadri et. al., 2003).

3x3 Pixel Method

Once again, each image's size is normalized by resizing to a fixed image size of 45 by 45 pixels. The image is then divided into 15 zones of 3 by 3 pixels. For each zone, if there is at least one black pixel, the entire zone is erased and replaced with a single pixel at the center of the zone (the rest are white). Otherwise, the zone is simply erased and replaced with white pixels only. The ultimate result is a "skeleton" of the original image. This skeleton is resized to a 15 by 15 pixel image, and analyzed on a pixel by pixel basis. Each pixel is assigned a value of 0 if it is black, and 1 if it is white, and the results are put into a 225D vector (Sadri et. al., 2006).

As we will see later in this paper, each method results in different levels of accuracy for the various classification and clustering methods used. These results show that the 3 x 3 method produced both the most accurate results overall. Additionally, the 3 x 3 feature extraction method had by far the fastest execution time. Comparatively, the Gradient method resulted in the lowest overall accuracy as well as the longest running time.

Classification Methods: Results and Evaluations

Our project implements four different classification methods: The Support Vector Classifier (SVC), the Multi-Layer Perceptron (MLP) classifier, the K-nearest-neighbour classifier (KNC), and the Decision Tree (DT) classifier. In this section, we will briefly explain how each classifier works, including the parameters used, and present the results obtained by running the test sets for all digits through each classifier. Furthermore, we make use of all three data extraction methods to populate the arrays used as input for each classifier. Results are tabulated based on an overall test of accuracy in terms of correctly classified images based on the labels provided during testing.

As a further measure of evaluation, we use cross-validation and a confusion matrix to further reinforce and validate our results. The cross-validation method runs 10 separate experiments, where the data used for learning, validation and testing is alternated, and the results are then averaged. This allows us to ensure that the results are consistent regardless of what data is used for learning, validation or testing. The confusion matrix shows us the number of classifications made in each category for each label, therefore providing us with a measure of how many data points were misclassified in each individual category. For example, given number “5”, the confusion matrix shows us the number of data points labeled as “5” that were classified into each category. This again provides us with a way both evaluate our results, and better understand the misclassifications when they occur.

All our classifier functions are provided by the Python Machine Learning library “scikit-learn”. This library provides a number of customizable functions and implementations, and in many cases allows us to specify numerous parameters that help us customize our classifiers.

Support Vector Classifier

The SVC is a form of supervised learning that performs classification based on a set of training input data along with their associated labels. The arrays created by our various data extraction methods are provided as input, along with an array of labels from 0 to 9 to represent the categories of each digit. The SVC creates a hyper-plane plot of hyper-planes (or vectors) in a high dimensional space, where categories are determined by the largest distance between data points. The test data is then classified based on the category it belongs to in the hyperplane (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/svm.html). Plotting is done using a decision function.

The parameters used for our SVC were determined using experimentation to yield the best overall results, though they were primarily the default parameters of the implementation. Key parameters included a sigmoid kernel as our decision function, and a "one-vs-rest" decision function shape due to the multiple classes of digits. We also capped our iterations at 60,000 in order to allow for faster runtime. Full parameters are listed below:

Parameter	**Value**
Kernel	Sigmoid
Penalty parameter of the C term	1.0
Kernel coefficient	Automatic
Probability estimates	No
Maximum iterations	60,000
Decision function shape	One-vs-rest

K-Nearest-Neighbor

The KNC is a form of instance based learning, where instances of the training data are stored, and classification is done by a "majority vote" from the nearest neighbors of each datapoint. In other words, the classification is based on the classification of the nearest neighbors of each point (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/neighbors.html).

The KNC uses the default parameters of the implementation. This includes giving a uniform weight to each neighbor, which we felt was appropriate given the uniform size of each dataset. The k-value used depends on the number of images chosen by the user when the code is run, where n images results in a k value of n-1. We elected to use a uniform k-value for all classes due to the uniform size of the datasets.

Multi-Layer Perceptron

The MLP classifier is a supervised learning algorithm that learns a function based on training through a dataset. It accepts as input an array of features, and an array target values (labels), and is able to classify data by passing input through successive (hidden) layers, which transform the data and pass it on to the next layer. Transformations are done using weights, which change as data is processed based on the errors in output, which effectively trains the network using backpropagation (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/neural_networks_supervised.html).

The parameters used for the MLP classifier are primarily the default parameters used in our library's implementation. We did make a few small adjustments, however. Weight optimization is done using a specified solver algorithm, L-BFGS, which converges faster and performs better for relatively small datasets, such as in our experiments. Furthermore, we used a smaller regularization term parameter, which gave us better overall results, and specified a random seed for initial random generation of weights, which provides for more reliable numbers in the long term.

Parameter	**Value**
Solver	*lbfgs*
L2 penalty	0.00005
Seed	Random

Decision Tree

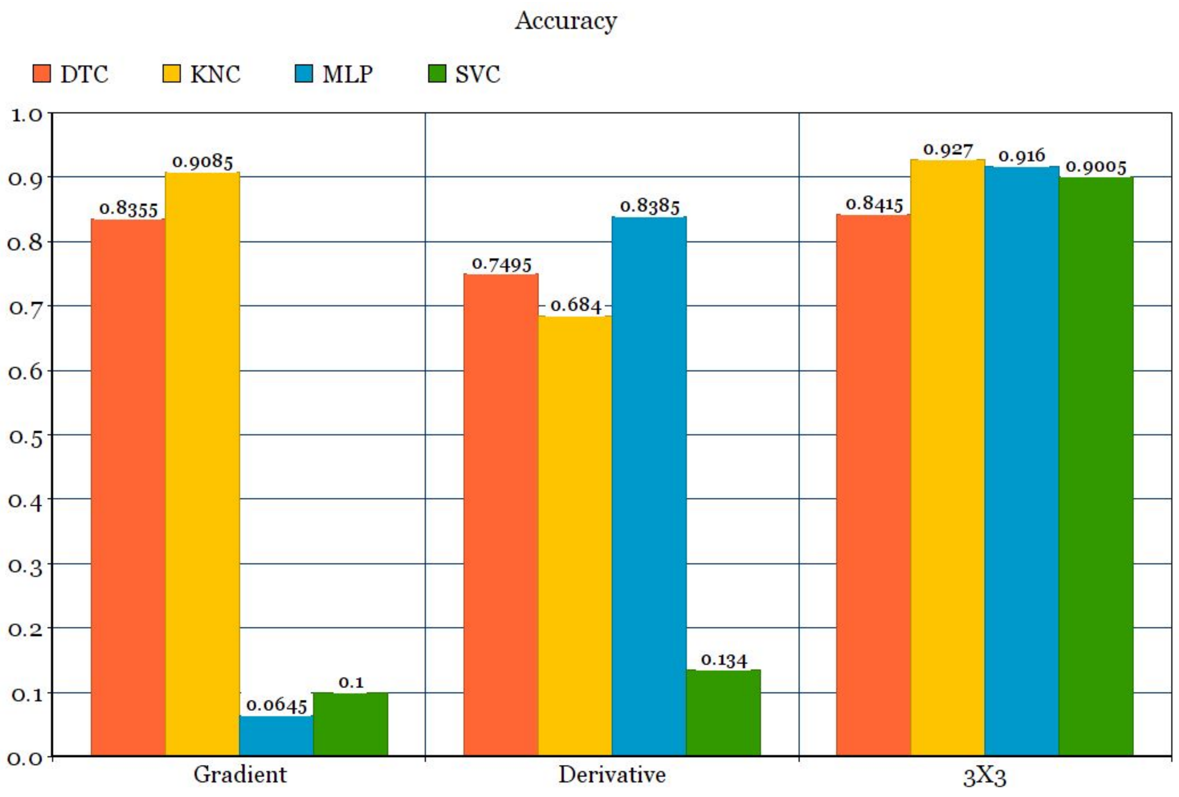
The DT classifier is a supervised learning method that creates a classification model capable of predicting the value of input using decision rules and if-then-else statements arranged in a tree-like structure (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/tree.html). The method accepts an array of input data, as well as an input of target labels, and is trained using a learning set of data similarly to the other classification methods. The implementation builds the tree using the CART algorithm, which is similar to the C4.5 algorithm (successor to ID3).

The parameters used in our classifier are the default parameters of the implementation. This includes using Gini impurity to determine on which attributes to split when building the tree (as opposed to entropy for information gain). For the most part, the parameters simply impose constraints on the size and depth of the tree and thus, little experimentation was needed.

Results

The results in the table below are the accuracy measures of the test set run after each classification algorithm used the training and validation sets for learning. Each classifier was run and tested using all three data extraction methods. All 10 digit categories were used in these tests, and they represent overall measures of accuracy for all test sets combined.

Classifier / Method of Extraction	**DTC**	**KNC**	**MLP**	**SVC**
Gradient	0.8355	0.9085	0.0645	0.1
Derivative	0.7495	0.684	0.8385	0.134
3x3	0.8415	0.927	0.916	0.9005



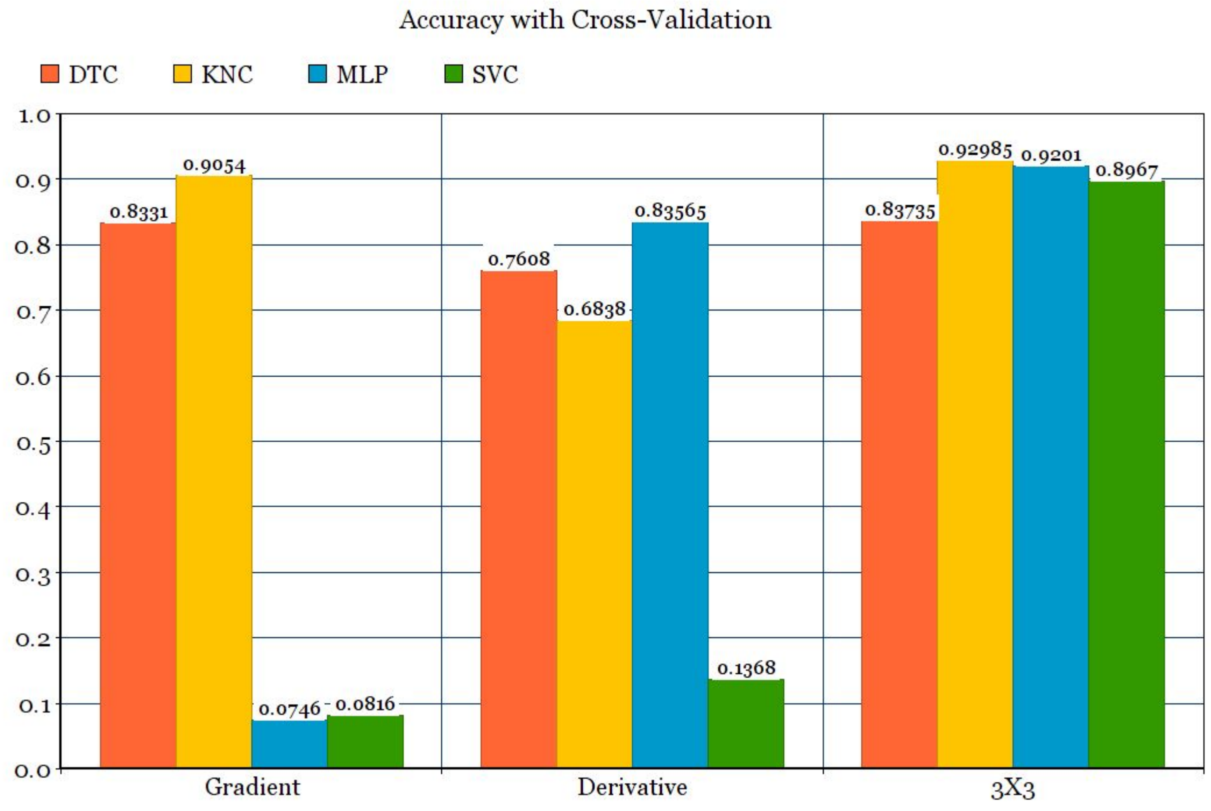
The results show that the 3 x 3 data extraction method provided the most consistently high accuracy results for all classifiers, including the highest overall measure of accuracy at 92.7% for the KNC method. The SVC method yielded high accuracy results using the 3 x 3 data extraction method, but gave extremely poor results in the Gradient and Derivative methods, with accuracy measures of 10.0% and 13.4% respectively. This amounts to essentially random guessing (given 10 categories). These results are rather interesting, given that the SVC method seems highly dependant on the data extraction method used (and thus, the type of data provided). It is worth noting that the results did not improve when changing the parameters in the function. The MLP classifier performed quite well using the 3 x 3 and Derivative data extraction methods, but had the lowest accuracy when using the Gradient method. Finally, the DT classifier had relatively strong results and remained quite consistent across all three data extraction methods.

These results suggest that both the Gradient and Derivative data extraction methods were not suitable for the SVC classification experiments conducted. Additionally, the Gradient method does not appear to be suitable for MLP classification. These results are puzzling, and resulted in us verifying that our methods of data extract were correct (which we confirmed). It appears that methods of classification can be highly dependent on the data extraction methods used.

Cross-Validation Results

As mentioned above, Cross-Validation was used as a method to evaluate the results determined in the previous section. Here, we performed the same experiments as above, but alternated the data in the sets used for learning, validation and testing, performing 10 different experiments and averaging the results. This allows us to ensure that the results initially determined were not simply a result of the images that happened to be used for learning/validation/testing, and are instead representative of the overall accuracy of our classification methods.

Classifier / Method of Extraction	**DTC**	**KNC**	**MLP**	**SVC**
Gradient	**AVG**: 0.8331	**AVG**: 0.9054	**AVG**: 0.0746	**AVG**: 0.0816
Derivative	**AVG**: 0.7608	**AVG**: 0.6838	**AVG**: 0.83565	**AVG**: 0.1368
3x3	**AVG**: 0.83735	**AVG**: 0.92985	**AVG**: 0.9201	**AVG**: 0.8967



The results of the cross-validation tests are remarkably similar to the results of the initial accuracy tests. This leads us to believe that our methods of classification are reliable and are not simply due to the choice of data used for each set. The results are consistent regardless of which data is chosen for learning and testing. In fact, in some cases the levels of accuracy increased in cross-validation, which may imply that the data used for learning in the initial test actually resulted in lower than average results.

Confusion Matrix

As an addition method to evaluate our results, we felt it would be interesting to populate a confusion matrix for the results of our initial tests. These matrices allow us to see precisely how many data points fall into each category, using every method of data extraction coupled with each classifier. Thus, it allows us to see which numbers were misclassified into which categories, and is especially interesting for the SVC method which resulted in extremely low measures of accuracy for ⅔ of the data extraction methods. As there are many tables, you may consult the Appendix to see all the data and confusion matrices. In this section, we will focus on some of the key findings.

Gradient with MLP

	0	1	2	3	4	5	6	7	8	9	Total
0	**24**	3	116	1	0	3	21	1	1	30	200
1	69	**30**	30	44	0	5	0	1	0	21	200
2	26	35	**2**	96	0	7	13	0	1	20	200
3	54	59	10	**59**	1	4	1	3	1	8	200
4	85	37	27	12	**3**	12	19	4	0	1	200
5	45	9	112	2	1	**4**	4	0	3	20	200
6	87	55	0	7	0	37	**6**	2	3	3	200
7	69	15	19	9	0	71	0	**0**	3	14	200
8	84	30	34	2	11	24	12	0	**1**	2	200
9	68	108	3	7	1	6	4	3	0	**0**	200

We begin by examining the confusion matrix for the classification experiment with the lowest overall results - the Gradient method using the MLP classifier. The confusion matrix astonishingly shows that the classifier rarely correctly classified digits, particularly for numbers 7, 8 and 9 (0, 1 and 0 correct classifications, respectively). This level of accuracy is even worse than theoretical random guessing. We can also see from the confusion matrix that the MLP classifier classified the majority of digits as either 0, 1 or 2. These results as puzzling, as the MLP classifier had relatively strong results using the other data extraction methods. Our investigations yielded no problems with either our data extraction method or our MLP classifier, both of which performed correctly in other experiments. This supports our conclusion is that this mix of classifier and data extraction method is simply not a good match.

Gradient with SVC

	0	1	2	3	4	5	6	7	8	9	Total
0	**0**	0	0	0	0	0	0	0	5	195	200
1	0	**0**	0	0	0	1	0	0	0	199	200
2	0	0	**0**	0	0	0	0	0	0	200	200
3	0	0	0	**0**	0	0	0	0	0	200	200
4	0	0	0	0	**0**	0	0	0	0	200	200
5	0	0	0	0	0	**0**	0	0	0	200	200
6	0	0	0	0	0	0	**0**	0	0	200	200
7	0	0	0	0	0	0	0	**0**	0	200	200
8	0	0	0	0	0	0	0	0	**0**	200	200
9	0	0	0	0	0	0	0	0	0	**200**	200

Derivative with SVC

	0	1	2	3	4	5	6	7	8	9	Total
0	**29**	24	1	16	21	74	15	5	2	13	200
1	2	**32**	0	4	13	70	50	5	1	23	200
2	4	7	**10**	13	41	59	16	16	20	14	200
3	3	13	5	**25**	21	23	41	39	12	18	200
4	11	36	7	10	**34**	35	17	18	8	24	200
5	12	67	2	6	23	**8**	47	9	23	3	200
6	11	35	9	13	24	43	**24**	15	14	12	200
7	15	14	4	35	28	20	36	**37**	10	1	200
8	4	7	3	5	7	55	33	13	**57**	16	200
9	5	21	4	5	25	18	72	7	31	**12**	200

The above confusion matrices detail the other poor performances in our classification experiments - the SVC using both the Gradient and Derivative methods. The confusion matrices reveal some interesting conclusions. While the performances from a raw data point of view are similar, the confusion matrices show that the misclassifications that occur are actually for very different reasons. In the case of the Gradient method, the SVC classifies virtually all digits as "9", which is obviously incorrect, but explains why our results were around 10%. In the case of the Derivative method, the results were similar (around 13%), but the confusion matrix shows a relatively even spread of classifications, with an emphases on digits 4, 5 and 6. These results again support that it is an issue of poor fit between data extraction method and classifier, as identical results in the confusion matrices would suggest an error with the classifier itself. Furthermore, these methods do perform well in other circumstances.

3X3 with KNC

	0	1	2	3	4	5	6	7	8	9	Total
0	**185**	12	0	1	0	1	0	1	0	0	200
1	9	**189**	0	0	1	0	0	0	0	1	200
2	0	15	**179**	2	0	0	2	0	0	2	200
3	0	0	23	**172**	2	0	0	3	0	0	200
4	0	4	13	14	**169**	0	0	0	0	0	200
5	2	2	0	1	0	**192**	0	0	3	0	200
6	0	5	2	0	1	0	**183**	2	0	7	200
7	0	1	0	0	0	0	0	**199**	0	0	200
8	0	1	1	0	0	0	0	0	**198**	0	200
9	0	3	0	0	0	1	6	0	2	**188**	200

If we examine the confusion matrix of the KNC using the 3 x 3 data extraction method, which yielded the most accurate results, our findings are what we would expect. The classifier correctly classifies the vast majority of digits in their respective categories, with "7" only being misclassified once (as a "1", which has a similar shape). This reinforces our findings that this was the best overall combination for our experiment.

Clustering Methods: Results and Evaluation

In addition to classification methods, our project also implements three different clustering functions that allow us to separate each individual class into a set of clusters, such that the data that belongs to a cluster shares more features with cluster members than it does with members of another cluster. In terms of our experiments, this is particularly useful because it allows us to separate the images of each individual digit into sets of clusters, such that each cluster contains digits that are written in a certain stylistic way. In other words, the clusters of the number “4” will contain various ways of writing the number “4” that may exist in our database.

Our project implements three different methods of Clustering: K-means clustering, Affinity Propagation, and the Birch method of clustering. While we used all three of our data extraction methods for classification experiments, in this case we elected only to use the 3 x 3 method. This was simply due to the fact that clustering involves performing analysis on each of the 10 number sets, and with three different clustering methods, it results in 30 different experiments per data extraction method. Given the consistently high accuracy of the 3 x 3 method in classification experiments compared to other methods, we felt it was the most appropriate to use for clustering experiments. To be able to create plottable 2-dimensional graphs, we used Principal Component Decomposition to transform the 255D arrays.

The methods for clustering were chosen based on their differing parameters, metrics and resulting graphs, as we felt it would provide us with the most interesting sets of output. Each method has certain conditions under which it performs optimally.

K-Means

K-Means clustering attempts to cluster data into a specified number of groups of equal variance. It chooses centroid elements by minimizing “inertia”, which is the within-cluster-sum-of-squares (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/clustering.html#k-means). K-means is considered to be general purpose, and works best when there are not too many clusters. It uses distances between points as a metric for clustering, and takes only the number of clusters as parameters (defaulted to 8).

Affinity Propagation

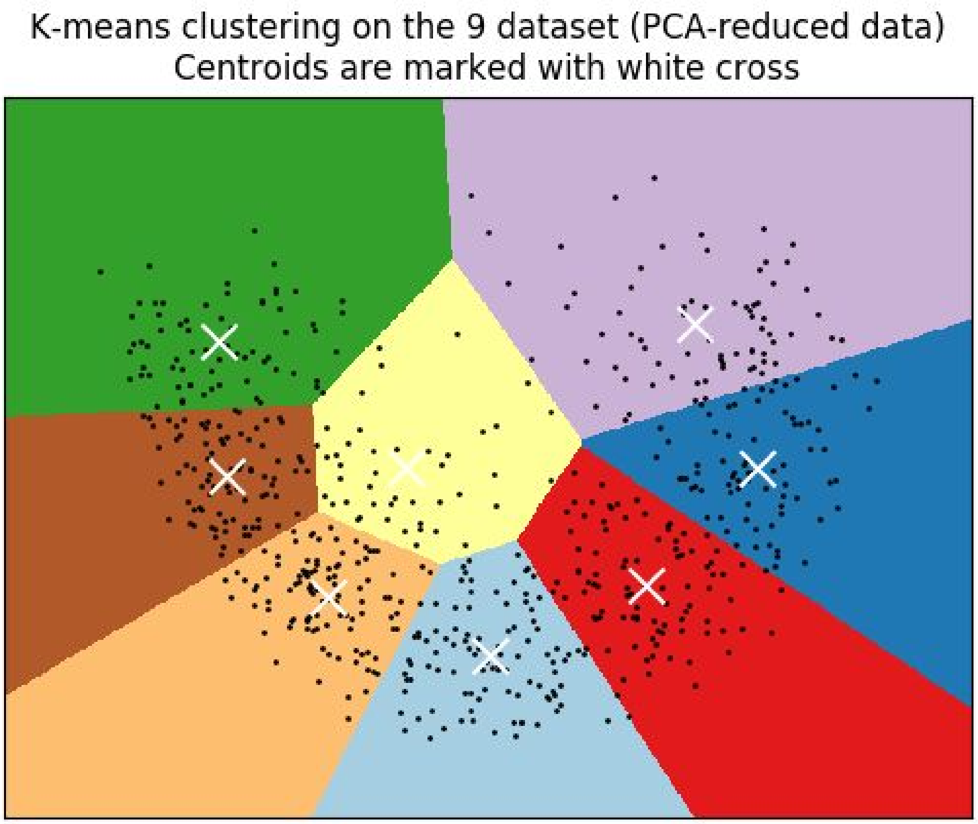
Affinity Propagation clustering sends messages between pairs of datapoints until they converge to create clusters. It then determines the most representative data points to identify each cluster, called the exemplars. When pairs communicate, they determine which of the two is a better representation of the cluster, which continues until convergence. This method of clustering does not take a specified number of clusters as a parameter, but instead determines the number of clusters from the input data. The parameters in this case are the number of exemplars to use (defaulted to the median of input similarities), and a damping factor (defaulted to 0.5). This method is most appropriate when there are many clusters of uneven size, and uses a nearest-neighbor graph as a clustering metric (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/clustering.html#affinity-propagation)

Birch

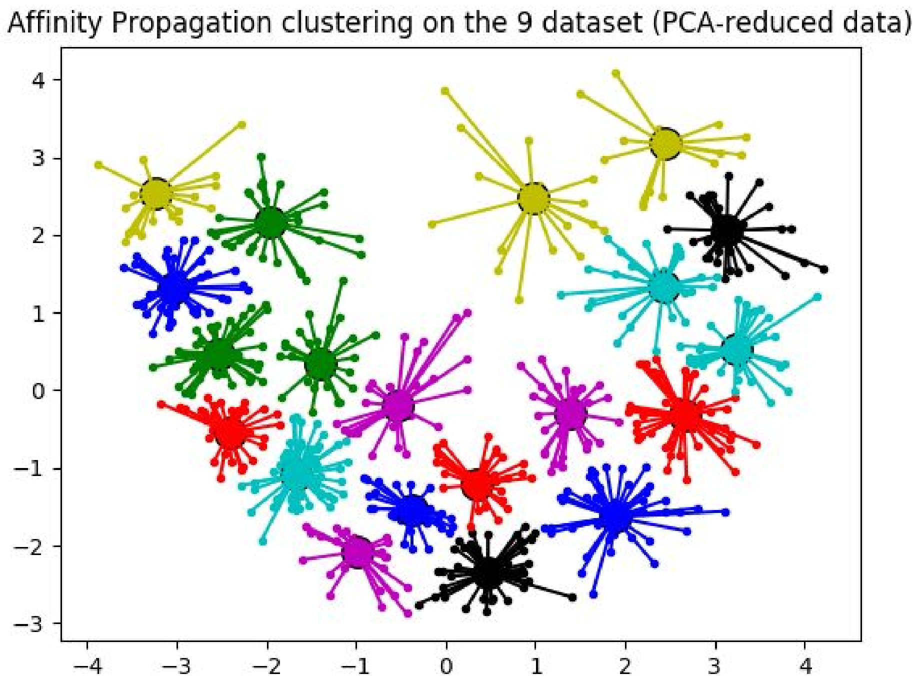
The Birch method of clustering uses a Characteristic Feature Tree (CFT), which contains nodes that each have subclusters. These subclusters contain the information needed for clustering (number of samples, centroids, etc…). It is especially useful for helping to reduce the input data, so it is ideal where there is a large dataset and we want to remove outliers. Parameters include a branching factor for the tree (defaulted to 50) and a threshold for distance between samples and subclusters (defaulted to 0.5). Euclidean distance between points is used as the clustering metric (Pedregosa et. al., 2011, http://scikit-learn.org/stable/modules/clustering.html#birch).

Results

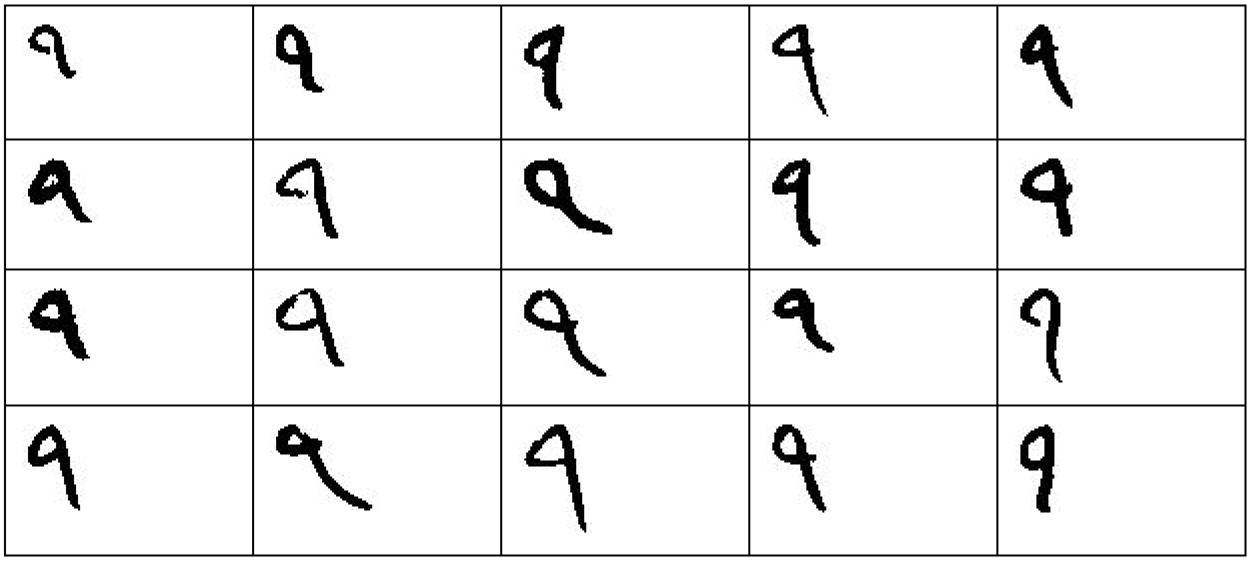
As mentioned above, in order to simplify our experiment process, we elected to only use the 3 x 3 method of data extraction to run our clustering algorithms to produce 30 different graphs. Given that these graphs look visually similar and are repetitive, we have included them in the Appendix section of this report. In this section, we will provide an example of each graph, and will focus on some key results, including some examples of representative elements for clusters determined using the Affinity Propagation method. We elected to focus on Affinity Propagation as this method yielded the most interesting results, and seems most appropriate due to the relatively high number of potential clusters and uneven cluster size that results in something as subjective as stylistic forms of writing a number.

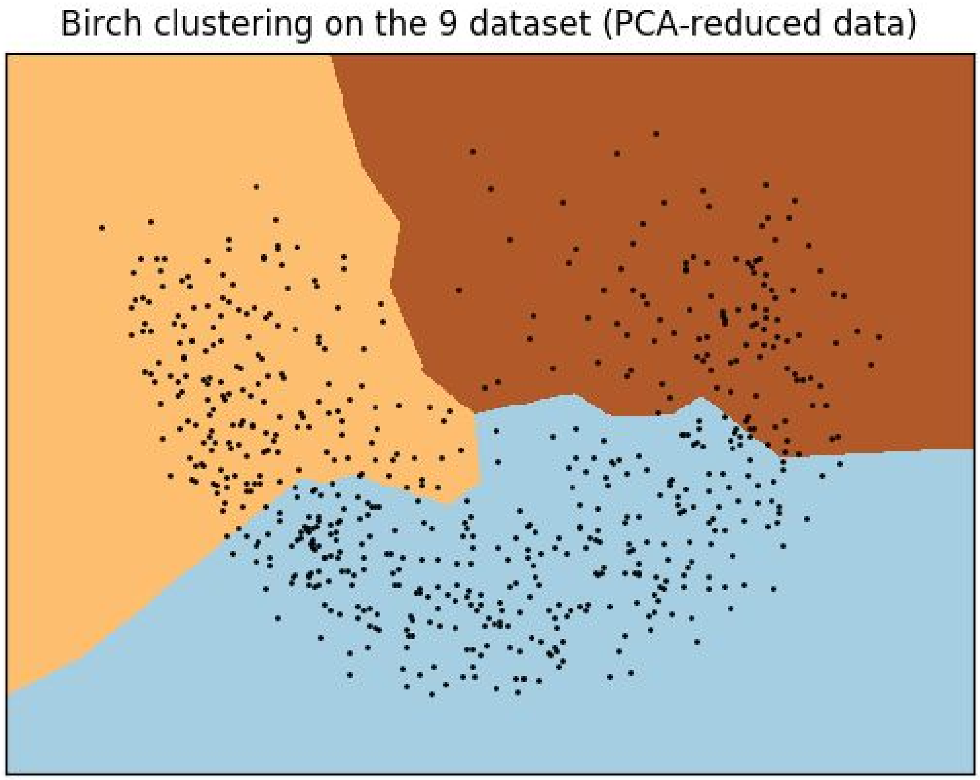


This graph contains the K-means clustering results on the “9” dataset. As we can see, this method attempts to cluster elements into one of 8 clusters of equal variance. Here, we can see that for the most part, the data points are relatively close to the identified centroid elements, though the clusters identified by yellow and purple seem to have slightly higher variance in their data points. This suggests that perhaps these points are less representative of their clusters, and that it is possible that using another method would result in them being placed in another cluster.



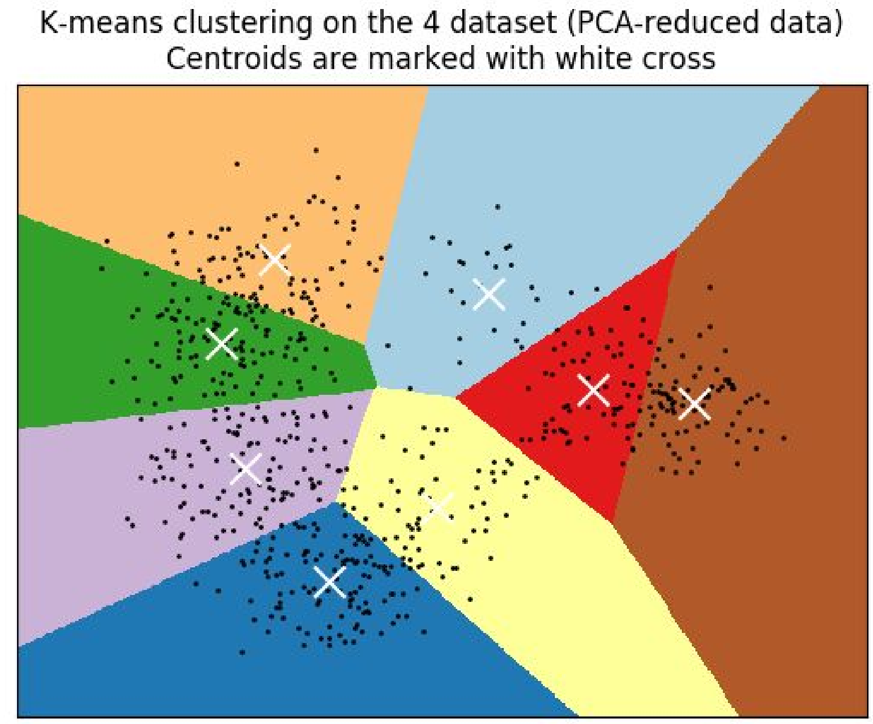
Indeed, if we examine the results on the same dataset using the Affinity Propagation method, which chooses the number of clusters based on the data, we see that the number of clusters was actually determined to be 20. This supports our earlier findings that suggested that the number of clusters could potentially be higher. Below, we have the centroid element of each cluster, which demonstrates the clear stylistic differences used when writing the number 9.

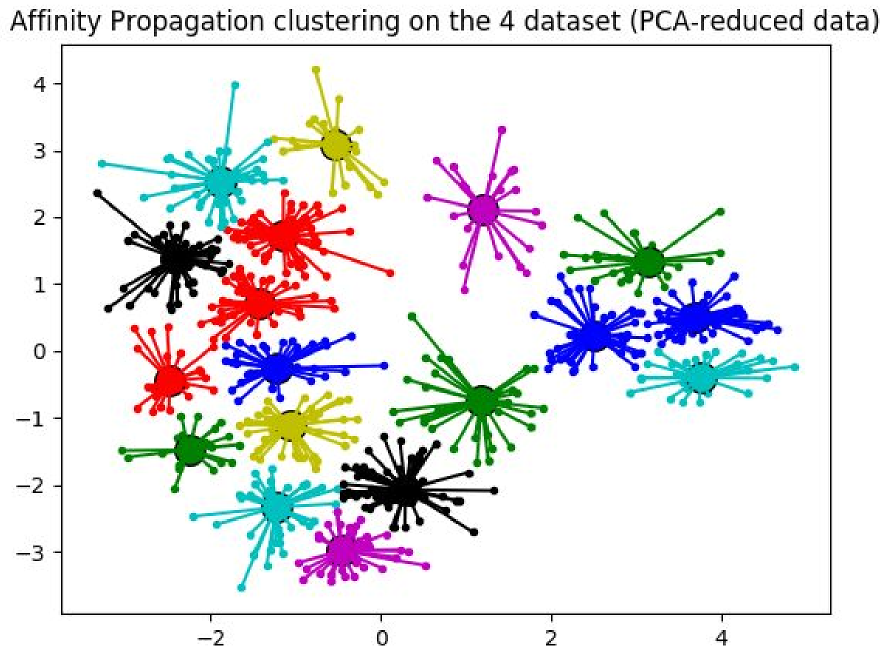


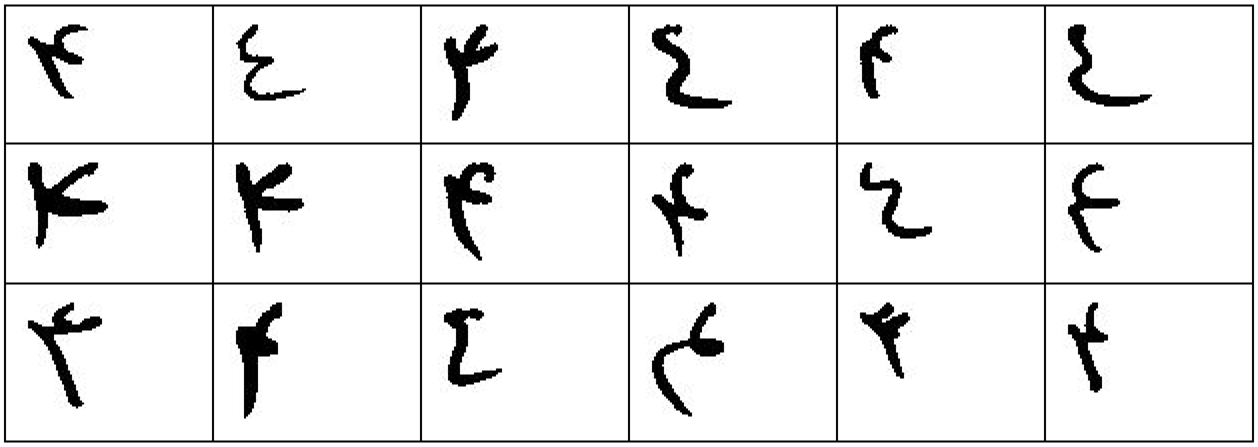


The Birch method attempts to reduce data, and only clustered into 3 different clusters. However, we can see clearly that data points are spread rather evenly across all clusters, and they actually resemble the same shape as the arrangement of data points from the K-means method. This further supports our findings, and ultimately shows us that the Affinity Propagation method yields the most interesting and accurate results, whereas the K-means imposes a constraint of too few clusters, and the Birch method attempts to reduce the data too much to be especially useful. While we could re-run the K-means method using a higher value for K clusters, it was not recommended to use this method with a high value for K.

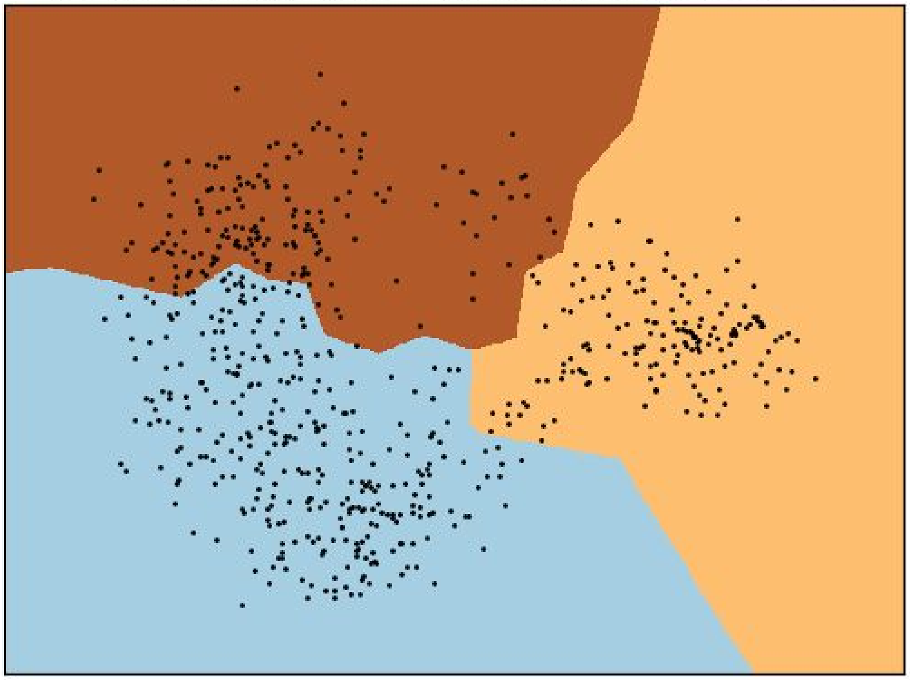
Below, we also list the same arrangement of results using the number “4” dataset:







Birch clustering on the 4 dataset (PCA-reduced data)



We can see here that the patterns observed on the “9” dataset are repeated again with the “4” dataset. The K-means cluster graphs suggest more clusters should be used, the Affinity Propagation graph identifies 18 clusters instead of 8, and the Birch cluster graph attempts to reduce the results determined in the K-means graph. The similar patterns suggest our methods of clustering are consistent across numbers.

Implementation Tools Used

The system is implemented using Python. We chose this language due to the vast availability of tools and libraries available for Machine Learning and Image Processing.

The system makes extensive use of the Python library scikit-learn (available from http://scikit-learn.org/). This library contains all the methods used for SVM, MLP, Decision Tree and KNN classification, as well as Affinity Propagation, K-means and Birch clustering. Furthermore, the library provides functions for computing Confusion matrices and measures of accuracy.

Image processing is done using the Python library CV2 - OpenCV (available from http://docs.opencv.org/3.0-beta/index.html). This tool was chosen simply due to our familiarity with the platform, having used it in other projects.

Conclusion

Our goal was to develop a system that used a number of different data extraction methods to analyze and transform data in the form of black and white image files of handwritten Persian digits into vectors of numbers that we could then use an input to a number of classification and clustering algorithms. The various methods of data extraction, classification and clustering implemented in our project have brought us to several conclusions.

In terms of classification, the K-Nearest-Neighbor classifier produced the highest overall measures of accuracy when coupled with the 3 x 3 data extraction method. As a whole, the KNC also had relatively accurate results using the other methods as well (though it suffered a bit when using the Derivative method). The Decision Tree classifier had the most consistent results across all data extraction methods, though the accuracy was not as high as some of the other methods used. The SVC had the worst overall performance, failing badly in 2 out 3 data extraction methods, though it performed very well using the 3 x 3 method. The MLP also had strong performance but suffered greatly using the Gradient method. Our conclusion is that for this kind of experiment, the 3 x 3 data extraction method is by far the ideal method to use, both due to it’s high results and the relatively fast run time and simple implementation. The Gradient method was difficult to implement, slow to run, and failed to produce adequate results for half of our classifiers, and is thus determined to be the worst overall method of data extraction for this experiment. By verifying our algorithms, and the fact that certain combinations of data extraction and classifier produce good results, we feel that in some cases, poor results are due to a bad match between methods. Our results were evaluated and validated using cross evaluation and confusion matrices, which supports our conclusion.

For clustering, we used the 3 x 3 method due to its accuracy and runtime, and found that while the K-Means and Birch methods produced some relevant findings, it was ultimately the Affinity Propagation method of clustering that yielded the most interesting results. The cluster method was able to determine how many different clusters categories are needed to accurately separate the date, rather than trying to pigeonhole our data into a defined number of clusters. By looking at the centroids of each category, we can see clear differences in how the digit is written, showing that our cluster method was successful overall.

References

Javad Sadri, Mohammad Reza Yeganehzad, Javad Saghi, "A Novel Comprehensive Database for Offline Persian Handwriting Recognition", Published in the Journal of "Pattern Recognition (Elsevier)", Volume 60, Pages 378-393, December 2016. Currently available online at: http://www.sciencedirect.com/science/article/pii/S0031320316300097

Javad Sadri, C. Y. Suen, T. D. Bui, "Application of Support Vector Machines for Recognition of Handwritten Arabic/Persian Digits", Published in the Proceedings of the Second Conference on Machine Vision and Image Processing & Applications (MVIP 2003), Vol. 1, pp. 300-307, Tehran, Iran, Feb. 2003. Currently available online at: http://www.sciencedirect.com/science/article/pii/S0031320316300097

Javad Sadri, C. Y. Suen, T. D. Bui, "A New Clustering Method for Improving Plasticity and Stability in Handwritten Character Recognition Systems", Published in the Proceedings of the International Conference on Pattern Recognition (ICPR 2006), pp. 1130-1133, Hong Kong, August 2006. Currently available online at: http://www.sciencedirect.com/science/article/pii/S0031320316300097

Javad Sadri, “COMP 472 Class Notes”, Winter 2017 Semester, Available: https://users.encs.concordia.ca/~c472_4/

Javad Sadri, “COMP 472 Project Database of Handwritten Digits”, Winter 2017 Semester, Available: https://users.encs.concordia.ca/~c472_4/

Fabian Pedregosa et. al., “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, Volume 12, pp 2825-2830, October 2011.

Appendix - Confusion Matrices

Gradient with DTC

	0	1	2	3	4	5	6	7	8	9	Total
0	**168**	5	0	2	1	9	2	2	10	1	200
1	11	**180**	0	0	0	1	1	3	2	2	200
2	0	1	**151**	22	3	1	12	6	0	4	200
3	1	0	17	**158**	8	4	6	0	1	5	200
4	4	3	12	9	**149**	6	9	1	2	5	200
5	6	1	3	3	4	**171**	2	0	8	2	200
6	3	3	7	6	4	1	**160**	5	2	9	200
7	0	4	0	0	0	1	2	**192**	1	0	200
8	7	2	1	0	1	6	1	2	**178**	2	200
9	1	4	1	3	8	2	12	2	3	**164**	200

Gradient with KNC

	0	1	2	3	4	5	6	7	8	9	Total
0	**185**	7	0	0	0	4	0	4	0	0	200
1	4	**193**	0	0	0	0	0	3	0	0	200
2	0	4	**176**	9	0	0	7	2	0	2	200
3	1	0	32	**159**	3	3	0	1	1	0	200
4	0	1	8	14	**173**	1	1	0	1	1	200
5	6	1	2	0	0	**181**	1	1	8	0	200
6	1	7	1	1	1	0	**177**	1	0	11	200
7	2	6	0	0	0	0	0	**192**	0	0	200
8	0	3	0	0	0	1	0	2	**193**	1	200
9	0	3	1	0	0	1	4	0	3	**188**	200

Gradient with MLP

	0	1	2	3	4	5	6	7	8	9	Total
0	**24**	3	116	1	0	3	21	1	1	30	200
1	69	**30**	30	44	0	5	0	1	0	21	200
2	26	35	**2**	96	0	7	13	0	1	20	200
3	54	59	10	**59**	1	4	1	3	1	8	200
4	85	37	27	12	**3**	12	19	4	0	1	200
5	45	9	112	2	1	**4**	4	0	3	20	200
6	87	55	0	7	0	37	**6**	2	3	3	200
7	69	15	19	9	0	71	0	**0**	3	14	200
8	84	30	34	2	11	24	12	0	**1**	2	200
9	68	108	3	7	1	6	4	3	0	**0**	200

Gradient with SVC

	0	1	2	3	4	5	6	7	8	9	Total
0	**0**	0	0	0	0	0	0	0	5	195	200
1	0	**0**	0	0	0	1	0	0	0	199	200
2	0	0	**0**	0	0	0	0	0	0	200	200
3	0	0	0	**0**	0	0	0	0	0	200	200
4	0	0	0	0	**0**	0	0	0	0	200	200
5	0	0	0	0	0	**0**	0	0	0	200	200
6	0	0	0	0	0	0	**0**	0	0	200	200
7	0	0	0	0	0	0	0	**0**	0	200	200
8	0	0	0	0	0	0	0	0	**0**	200	200
9	0	0	0	0	0	0	0	0	0	**200**	200

Derivative with DTC

	0	1	2	3	4	5	6	7	8	9	Total
0	**163**	12	0	2	5	6	5	2	1	4	200
1	16	**167**	2	1	2	0	3	0	3	6	200
2	1	3	**130**	13	22	1	17	7	1	5	200
3	7	2	18	**136**	16	4	6	8	0	3	200
4	2	5	29	20	**119**	7	9	4	1	4	200
5	3	1	3	3	14	**147**	10	2	7	10	200
6	4	2	11	7	11	8	**139**	8	1	9	200
7	1	0	8	8	4	2	13	**163**	0	1	200
8	2	3	0	1	1	6	5	1	**178**	3	200
9	2	8	1	5	6	3	10	0	8	**157**	200

Derivative with KNC

	0	1	2	3	4	5	6	7	8	9	Total
0	**169**	18	2	2	1	4	2	1	0	1	200
1	18	**177**	0	0	0	1	1	0	1	2	200
2	4	15	**130**	21	0	0	8	10	1	11	200
3	4	5	48	**103**	7	2	2	20	1	8	200
4	14	12	25	22	**62**	11	6	15	7	26	200
5	9	12	3	1	3	**155**	2	2	11	2	200
6	9	20	26	6	8	6	**73**	27	5	20	200
7	1	3	12	2	0	4	5	**173**	0	0	200
8	2	2	0	0	1	16	1	0	**162**	16	200
9	5	10	4	0	0	4	4	4	5	**164**	200

Derivative with MLP

	0	1	2	3	4	5	6	7	8	9	Total
0	**173**	12	0	1	2	5	3	1	0	3	200
1	11	**180**	0	0	1	2	3	0	1	2	200
2	4	0	**152**	14	7	2	13	3	1	4	200
3	1	1	24	**156**	8	0	3	5	1	1	200
4	8	6	6	14	**149**	4	6	1	0	6	200
5	1	1	4	3	3	**181**	1	1	5	0	200
6	5	9	9	4	9	1	**146**	8	0	9	200
7	0	0	1	4	1	1	4	**189**	0	0	200
8	0	4	3	0	0	5	5	0	**178**	5	200
9	1	3	3	1	5	2	9	0	3	**173**	200

Derivative with SVC

	0	1	2	3	4	5	6	7	8	9	Total
0	**29**	24	1	16	21	74	15	5	2	13	200
1	2	**32**	0	4	13	70	50	5	1	23	200
2	4	7	**10**	13	41	59	16	16	20	14	200
3	3	13	5	**25**	21	23	41	39	12	18	200
4	11	36	7	10	**34**	35	17	18	8	24	200
5	12	67	2	6	23	**8**	47	9	23	3	200
6	11	35	9	13	24	43	**24**	15	14	12	200
7	15	14	4	35	28	20	36	**37**	10	1	200
8	4	7	3	5	7	55	33	13	**57**	16	200
9	5	21	4	5	25	18	72	7	31	**12**	200

3X3 with DTC

	0	1	2	3	4	5	6	7	8	9	Total
0	**171**	14	2	0	1	3	2	4	1	2	200
1	14	**181**	0	0	1	1	1	0	0	2	200
2	0	9	**151**	22	5	0	7	2	0	4	200
3	1	2	15	**169**	7	1	3	1	0	1	200
4	1	8	9	14	**153**	4	0	2	1	8	200
5	5	0	2	1	2	**175**	2	6	5	2	200
6	1	7	9	5	5	0	**152**	15	1	5	200
7	3	0	2	1	1	2	2	**189**	0	0	200
8	2	2	0	0	4	1	2	1	**181**	7	200
9	1	7	7	2	4	2	9	3	4	**161**	200

3X3 with KNC

	0	1	2	3	4	5	6	7	8	9	Total
0	**185**	12	0	1	0	1	0	1	0	0	200
1	9	**189**	0	0	1	0	0	0	0	1	200
2	0	15	**179**	2	0	0	2	0	0	2	200
3	0	0	23	**172**	2	0	0	3	0	0	200
4	0	4	13	14	**169**	0	0	0	0	0	200
5	2	2	0	1	0	**192**	0	0	3	0	200
6	0	5	2	0	1	0	**183**	2	0	7	200
7	0	1	0	0	0	0	0	**199**	0	0	200
8	0	1	1	0	0	0	0	0	**198**	0	200
9	0	3	0	0	0	1	6	0	2	**188**	200

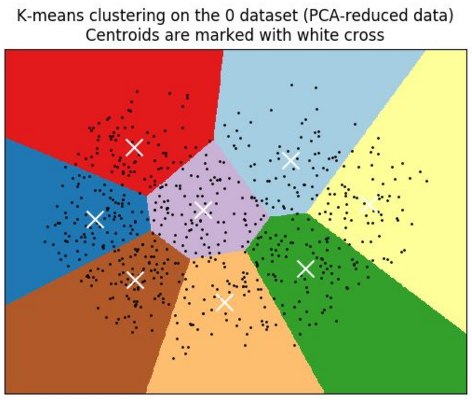
3X3 with MLP

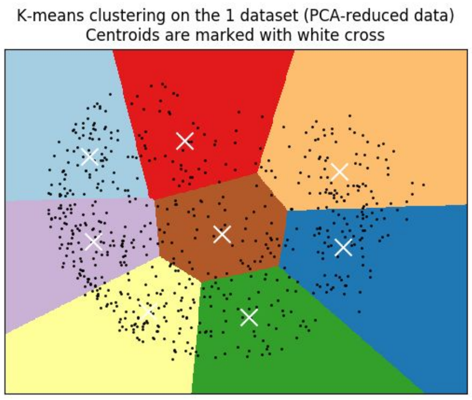
	0	1	2	3	4	5	6	7	8	9	Total
0	**190**	4	0	1	0	2	1	1	1	0	200
1	5	**190**	1	0	1	0	1	1	0	1	200
2	1	5	**178**	6	3	0	3	0	0	4	200
3	0	0	11	**177**	3	5	3	1	0	0	200
4	3	2	5	12	**175**	0	1	0	0	2	200
5	6	0	0	1	1	**182**	2	2	5	1	200
6	3	3	2	3	4	0	**176**	2	2	5	200
7	1	0	2	0	0	0	1	**195**	1	0	200
8	2	0	0	0	0	2	0	5	**186**	5	200
9	1	0	1	4	1	0	7	0	3	**183**	200

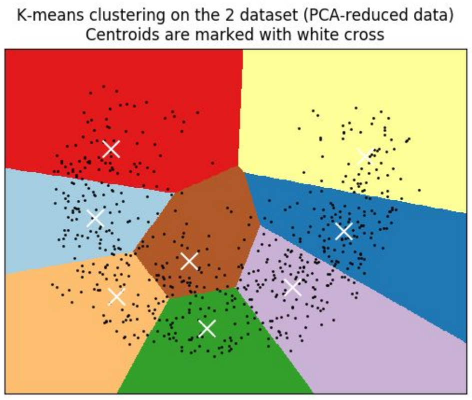
3X3 with SVC

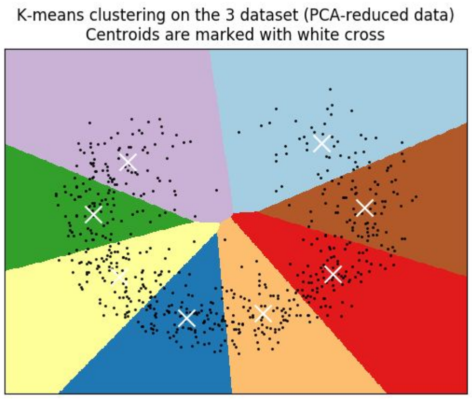
	0	1	2	3	4	5	6	7	8	9	Total
0	**182**	11	0	0	4	2	0	1	0	0	200
1	8	**185**	1	0	4	0	1	0	0	1	200
2	1	9	**169**	4	7	0	1	2	1	6	200
3	0	0	12	**176**	9	0	1	2	0	0	200
4	1	1	8	6	**181**	0	0	0	1	2	200
5	1	1	1	1	1	**189**	2	1	3	0	200
6	1	4	8	0	4	0	**162**	4	0	17	200
7	1	4	1	0	0	0	2	**191**	1	0	200
8	0	0	2	0	0	1	0	5	**191**	1	200
9	0	2	0	0	6	1	10	0	6	**175**	200

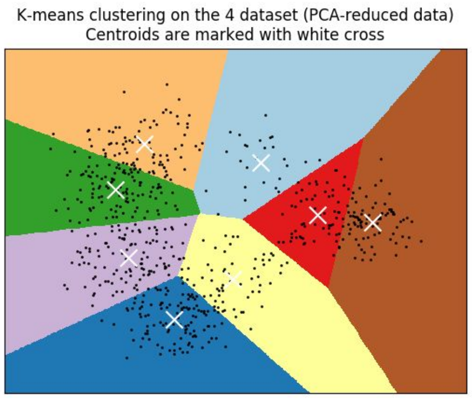
Appendix - Clustering Graphs (K-Means)

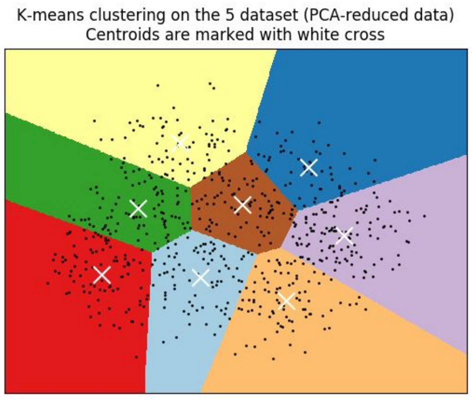


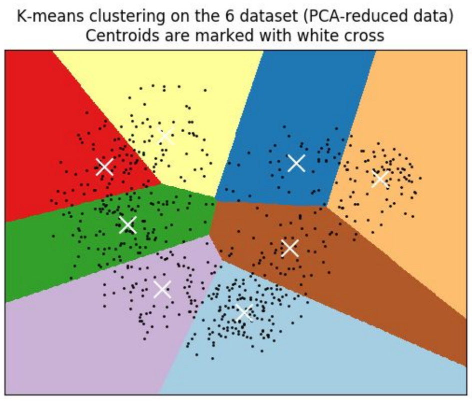


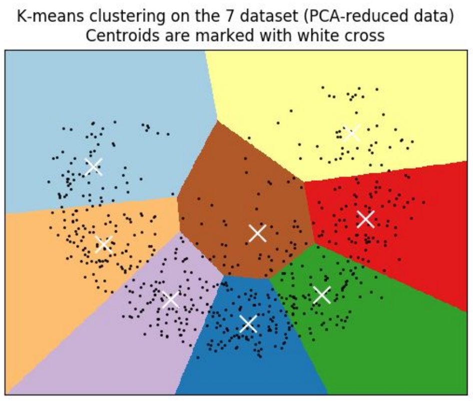


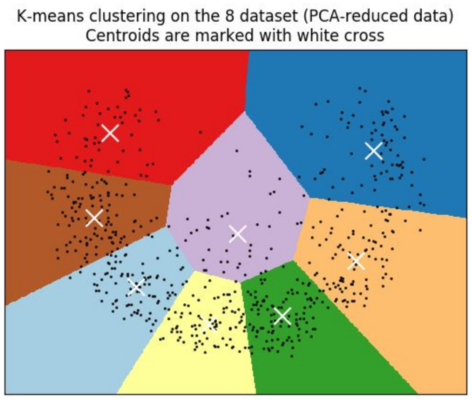


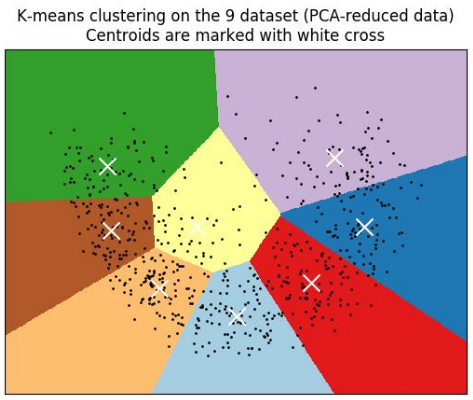




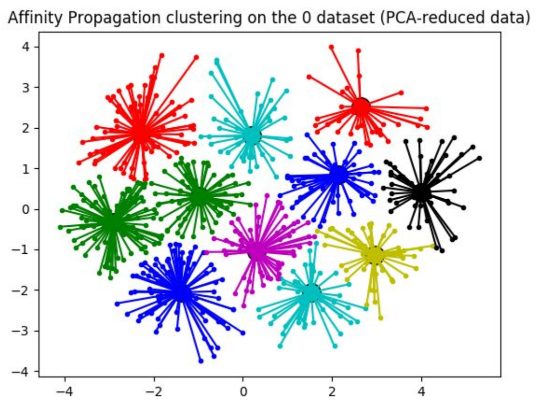

K-means clustering on the 6 dataset (PCA-reduced data)
Centroids are marked with white cross

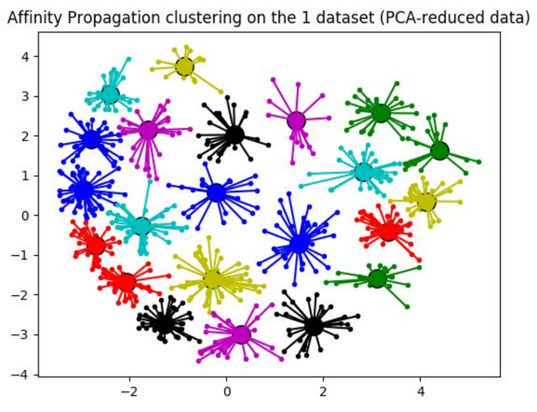

K-means clustering on the 7 dataset (PCA-reduced data)
Centroids are marked with white cross

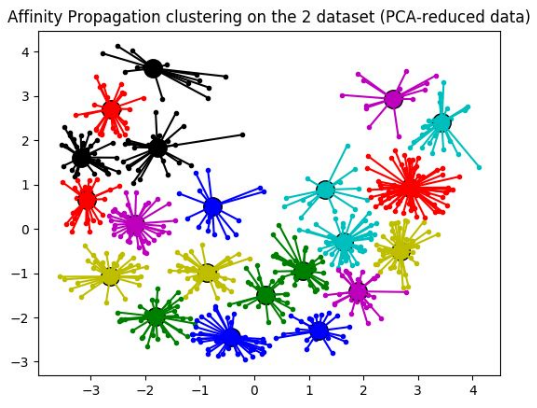

K-means clustering on the 8 dataset (PCA-reduced data)
Centroids are marked with white cross

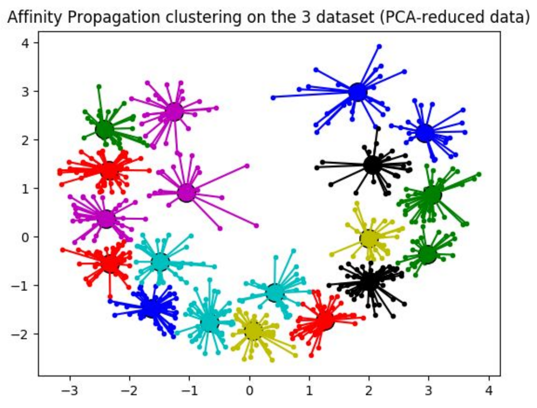

K-means clustering on the 9 dataset (PCA-reduced data)
Centroids are marked with white cross

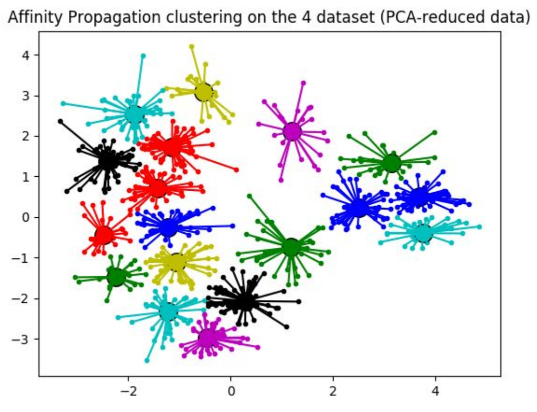
Appendix - Clustering Graphs (Affinity Propagation)

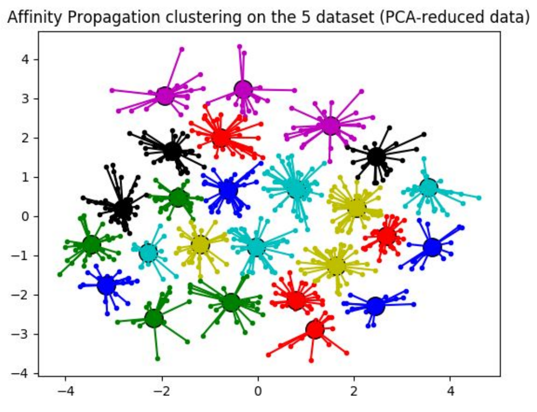


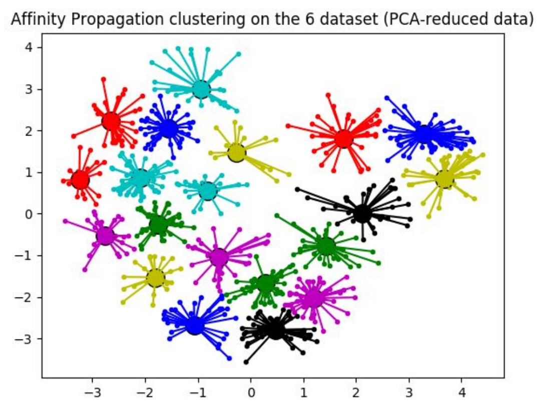


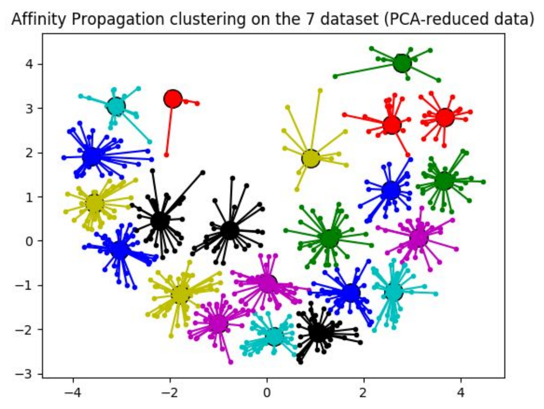


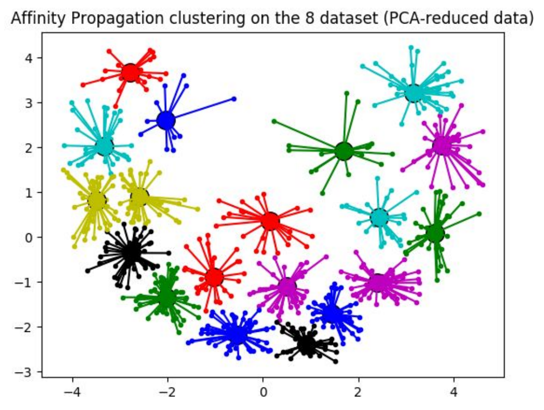


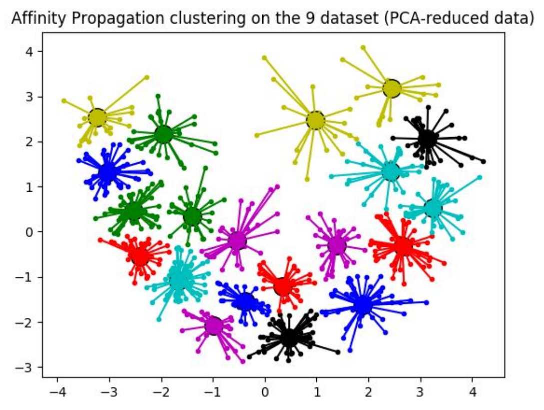




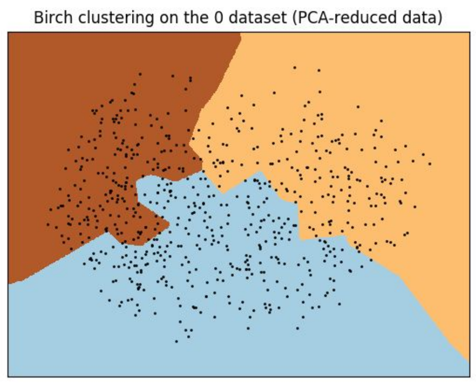

Affinity Propagation clustering on the 6 dataset (PCA-reduced data)

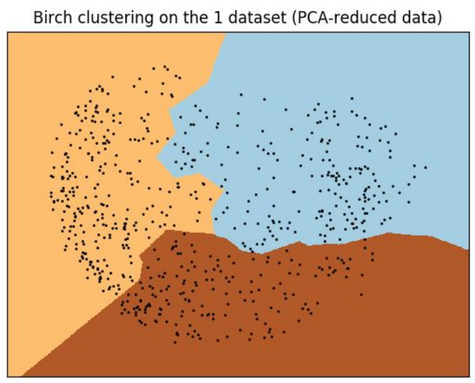

Affinity Propagation clustering on the 7 dataset (PCA-reduced data)

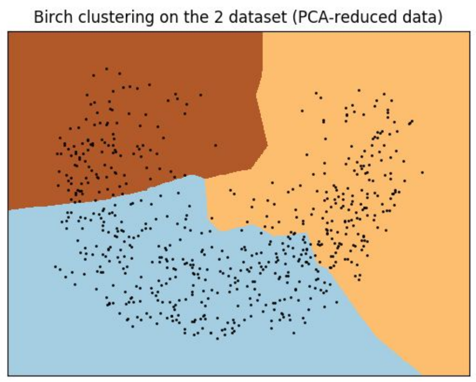

Affinity Propagation clustering on the 8 dataset (PCA-reduced data)

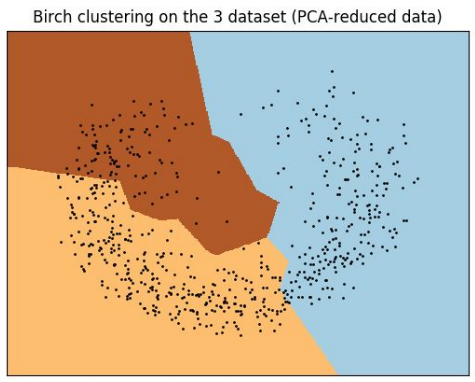

Affinity Propagation clustering on the 9 dataset (PCA-reduced data)

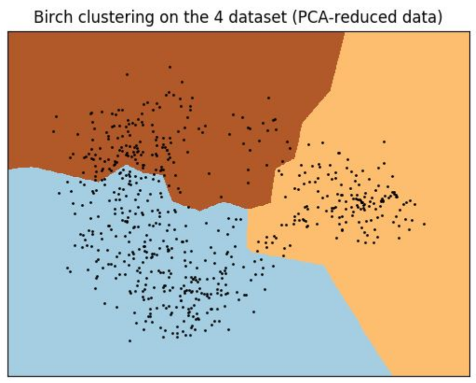
Appendix - Clustering Graphs (Birch)

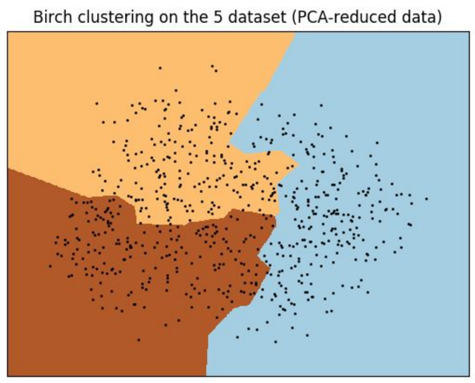




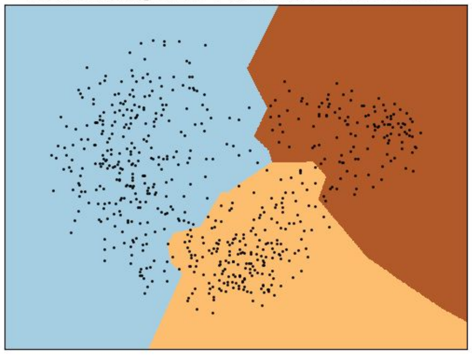




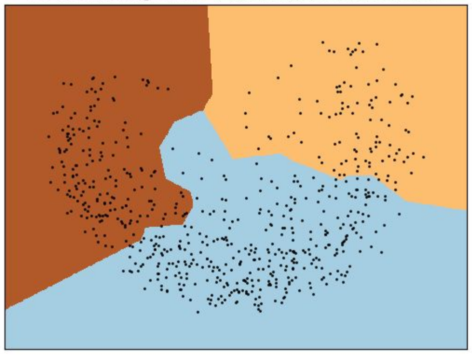




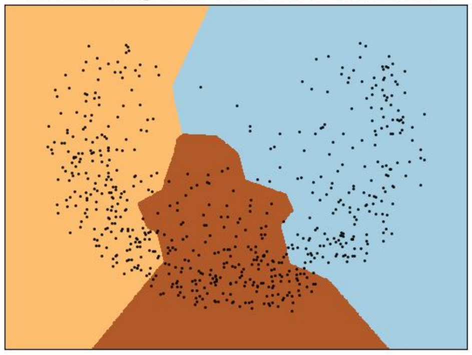
Birch clustering on the 6 dataset (PCA-reduced data)



Birch clustering on the 7 dataset (PCA-reduced data)



Birch clustering on the 8 dataset (PCA-reduced data)



Birch clustering on the 9 dataset (PCA-reduced data)