CSE 326/426 **Due on Feb 25**

Problem Set II Gustavo Grinsteins CSE 326

1 Written Exercises

Answer the following questions in 25-100 words each. Full scores will not be given if missing detailed steps for all computations.

1. (15%) Consider the case of using a kNN classifier, but with the ℓ_1 norm (e.g., $\|\vec{z}\|_1 = \sum_d |z_d|$) to measure distances rather than the ℓ_2 norm (e.g., $\|\vec{z}\|_2 = (\sum_d z_d^2)^{1/2}$). Draw a simple case of a binary classification problem for which the ℓ_1 classifier would return a different class for a test point than an ℓ_2 classifier (in two dimensions). In particular, draw ≥ 1 training points (one for each class) and a testing point that would be classified differently according to the two distance metrics.

   What properties of a data set do you imagine would influence whether the ℓ_1 distance would work better or worse than the ℓ_2 distance?

   (**Gustavo's Answer:**)

   Please refer to **Figure 1** for simple case that can get different results using $l1$ and $l2$ classifiers.

   The property of the data set that influence the distance measure to use is the spread of the data. In other words, how many outliers does that data have.

   L1 norm is less sensitive to outliers but more sensitive to small scale behavior. L2 norm is more sensitive to outliers but less sensitive to small scale behavior.

2. (15%) One of the biggest problems with kNN classifiers is that they are very expensive to apply at test time, even if you use clever data structures and clever applications of the triangle inequality. One way to speed up a kNN classifier would just be to have fewer training points. Suppose you're given a training set of N labeled pairs $(x_n, y_n)_{n=1}^N$. Suppose that we want to *throw out* some subset of these points. What criteria would you use for deciding which points to throw out? Sketch an algorithm for doing so.

   (**Gustavo's Answer:**)

   To speed up the K-NN classification the K-means algorithm can be implemented in order to obtain a set of k-centroids per class label that will be used in the training process. These k-centroids will represent the mean value for that specific label's cluster therefore giving good "representative candidates" for that label. This will reduces the amount of points used at train time therefore increasing the speed of the K-NN classification.

   The algorithm consists of first obtaining the training data by finding k centroids per label. Then, You proceed to calculate the euclidean distance of each point in the test data set to all the K-centroids by label. Following this, you select the 10 centroids that are closest to the

point to classify and perform majority vote on all the those values. The value that appears the most in these minimum distances will classify the new point.

Please refer to the KNN_MNIST.m file attached with this document for the algorithm implementation.

3. (10%) An important notion for a classifier is that of *consistency*. Roughly, a classification algorithm is *consistent* if, whenever it has access to *infinite* amounts of training data, its error rate approaches the optimal error rate. Consider the noise-free setting, where the optimal error rate is zero. Is the one-nearest-neighbor algorithm consistent in this setting?

   (**Gustavo's Answer:**)

   The one nearest-neighbor algorithm will be consistent to always classify the data with the correct label given an infinite amount of data to train with. The value that will be measure closest to the data point will have the correct label to that data point.

2 Programming Task [CSE326 (60%)]

Students shall submit your own codes and results in one folder. All experimental results including plots and discussions are required to be reported in a single PDF file with the above written exercises. Grades for all three parts of experiments are equally distributed.

(1) We experiment with unsupervised learning: K-means. Your implementation objectives are: (1) `clusterInit.m`, which should initialize random clusters according to the "furthest first" heuristic (these "real" centers should be as far as possible from any of the previous centers, e.g., if you've already selected four centers, the fifth center should be the point whose distance to any of the 4 centers is *maximum*.), and (2) `test_kmeans.m` function for testing the implementation on a simple 2D data (`provided as "2D_data.txt"`) If you don't see reasonable clusters coming out, something is probably broken.

Your results should include:

- A plot of the data, unclustered
- The data clustered with $K = 2$ and random initialization
- The data clustered with $K = 3$ and random initialization (run 20 times... you should see a small amount of variability in the outputs). The plots also include distance scores.
- The data clustered with $K = 3$ and "furthest" initialization (run 20 times... you should see a small amount of variability in the outputs). The plots also include distance scores. Note that this won't work until you do the furthest-first initialization below.
- A plot of $K \in \{2, 3, 4, 5, 6, 8, 10, 15, 20\}$ versus distance score.

Note: Please use different colors for different clusters.

(**Gustavo's Answer:**)

Please refer to **Figures 2-6**

(2) In this experiment, test your K-means clustering with the furthest-first initialization on the MNIST handwritten digits database (`provided as "mnist.mat"`) with different values of K.

- Report the means found for $K = 5, 10, 15$ (you have to reshape each row of the "trainX" or "testX" as a 28^2 image).

  (**Gustavo's Answer:**)

  Please refer to **Figures 7-9** for the outputs

- For each of these, do you see clusters that look like the actual digits? How big does K have to be before you essentially see a mean for each "true" digit? Why do you suppose some digits are "over-represented?"

  (**Gustavo's Answer:**)

  Most of the clusters can be distinguished as actual digits. I believe you need a minimum of k = 10 in order to have 10 clusters that would ideally represent each label. Some digits are over-represented due to the fact that some numbers are written in different ways which would generate separate clusters for the same number.

(3) Implement KNN classification by using clustering as a method of speeding up. What you do is: for each class y, generate K clusters. We think of these K clusters as "prototypical" points for class y. Thus, for MNIST, since there are 10 classes, we end up with $10 \times K$ points. These are now the "training data" that is fed into the KNN classifier. When $K = 1$, each class just gets a single representative; when $K = 5$, each class gets five representatives.

- Display test accuracy and test time as a function of the number of clusters per (true) class. The test time should monotonically increase as a function of the number of clusters (per class). Does test time increase linearly? Why or why not? Does the test accuracy monotonically increase: that is, does adding more clusters always help? Why or why not? What number of clusters would you choose?

(**Gustavo's Answer:**)

Does test time increase linearly? Why or why not?

According to the graphs from **Figure 10 -11** the time seems to increase linearly as we increment the number of representatives per class. This is because every time you increase K you increase the number of values you are calculating the distance to by a factor of 10 which is a constant term.

This means that the slope of the linear trend should be 10 with an intercept at the time it takes to test k = 1.

Does the test accuracy monotonically increase: that is, does adding more clusters always help? Why or why not? What number of clusters would you choose?

According to the outputs of accuracy vs. k-value from **Figures 10 -11**, there is a point at which is K is increased the accuracy of the program starts to decrease. This value seems to be in the realm of k = 10-14. The accuracy of the test increases to this point until you reach a point where you are over representing each label of the data and giving more representation to outliers. This will cause more erroneous classification therefore lowering the accuracy.

Figures for PS2 (next page)

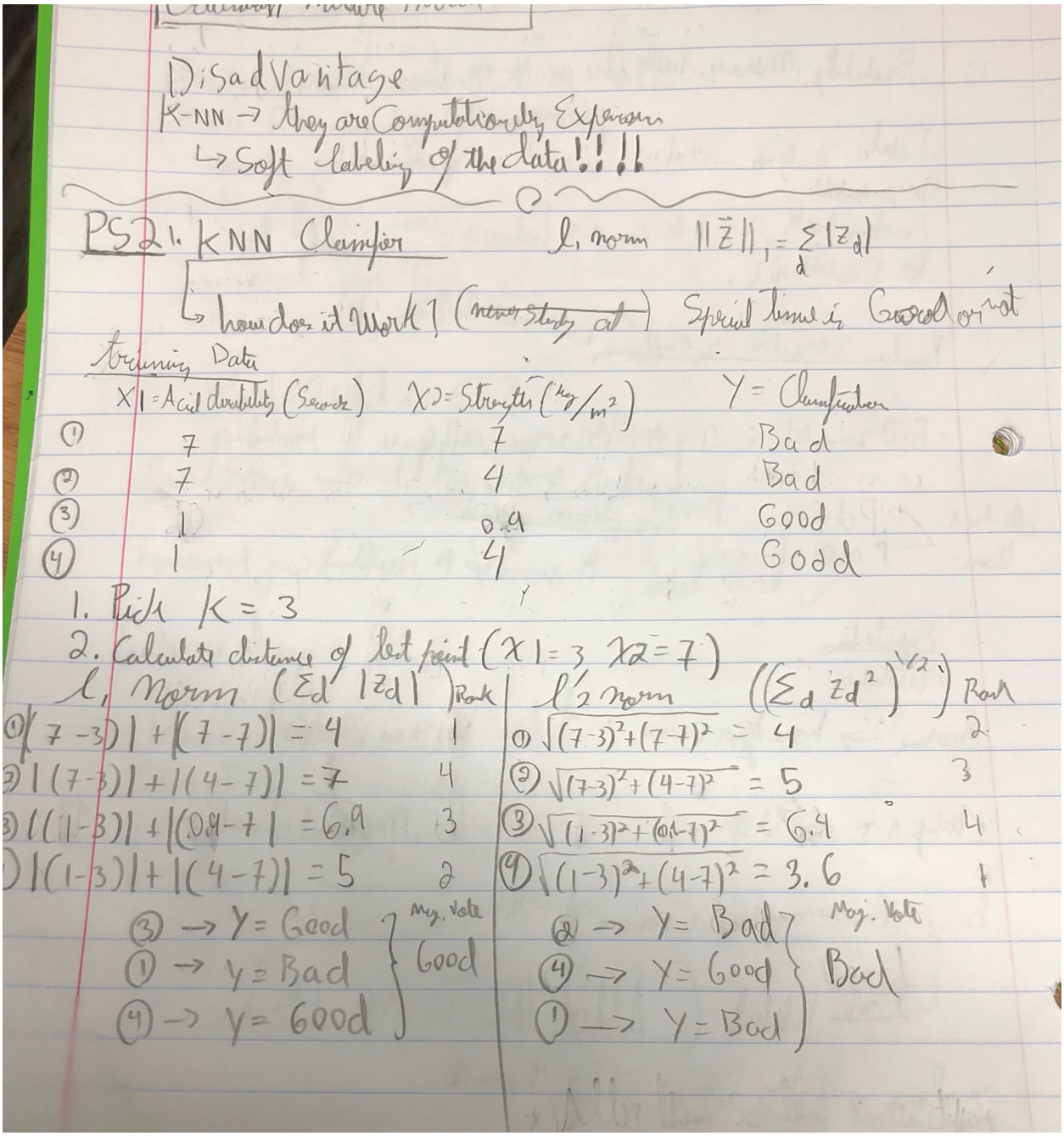

Figure 1: PS2 Question 1: example with $l2$ and $l1$

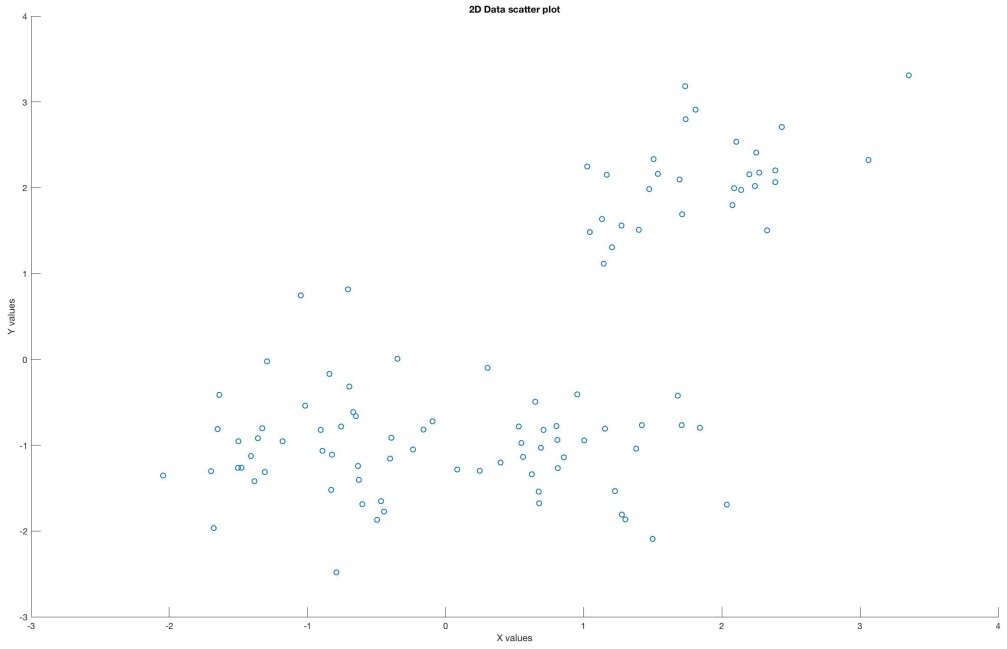

Figure 2: Unclustered Data

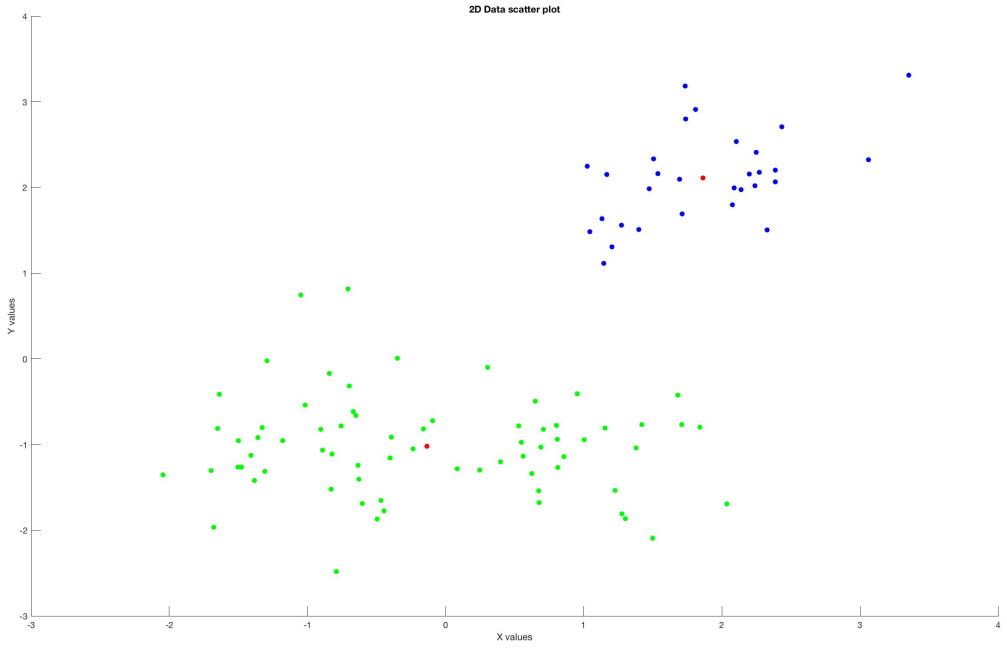

Figure 3: Clustered Data with $K = 2$ and random initialization. The red points are the centroids of the cluster

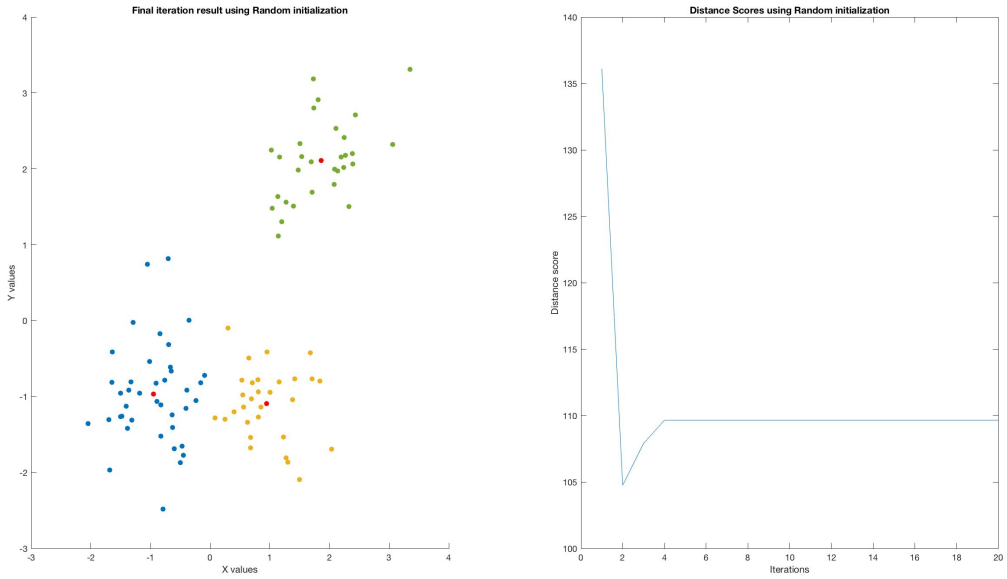

Figure 4: Clustered Data with $K = 3$ and random initialization and distance score plotted. The red points are the centroids of the cluster.

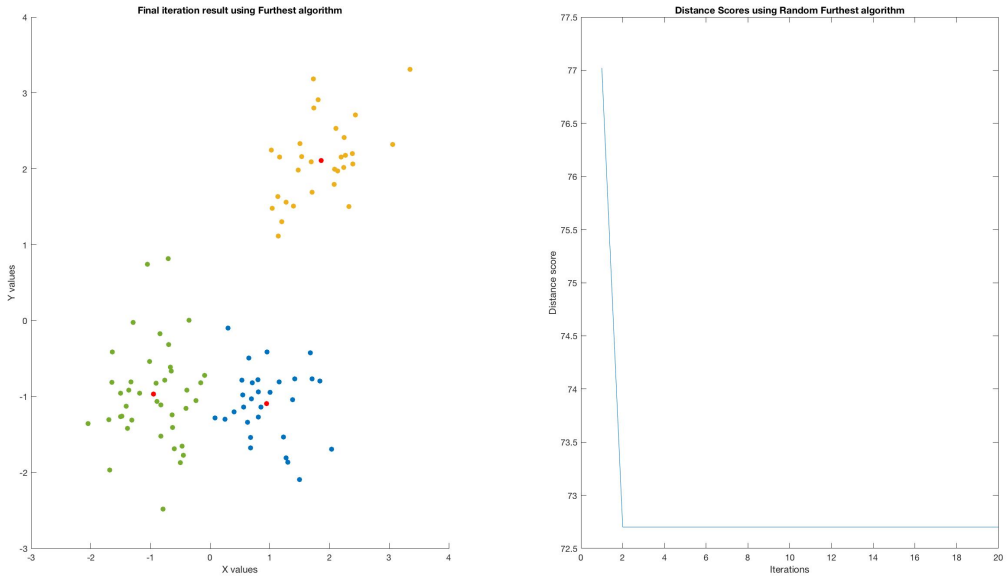

Figure 5: Clustered Data with $K = 3$ and furthest heuristic initialization and distance score plotted. The red points are the centroids of the cluster.

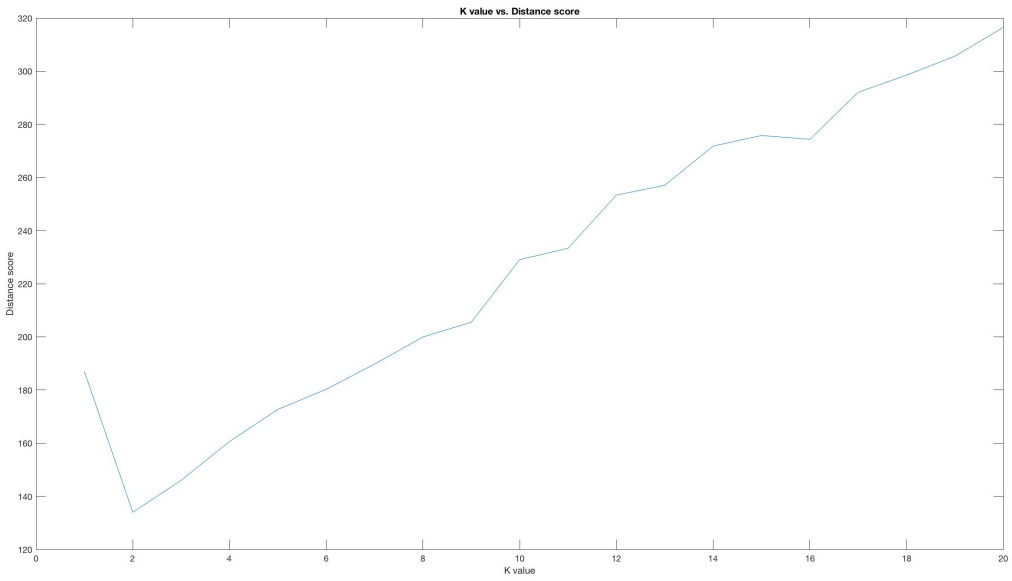

Figure 6: K value vs. Distance score using furthest heuristic algorithm. Using random initialization my code would crash because there would be clusters with no assignments

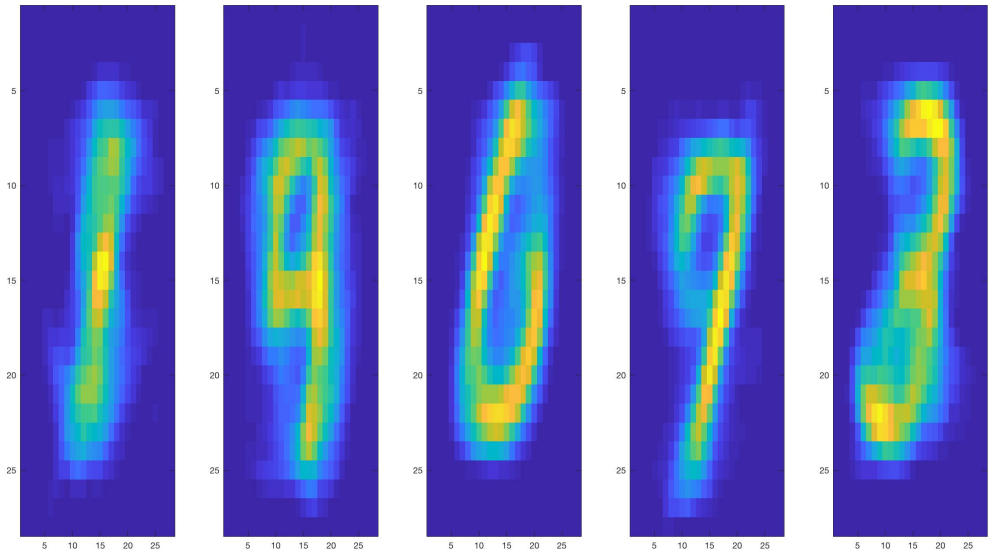

Figure 7: Mean found from the MNIST data using k-means of $k = 5$

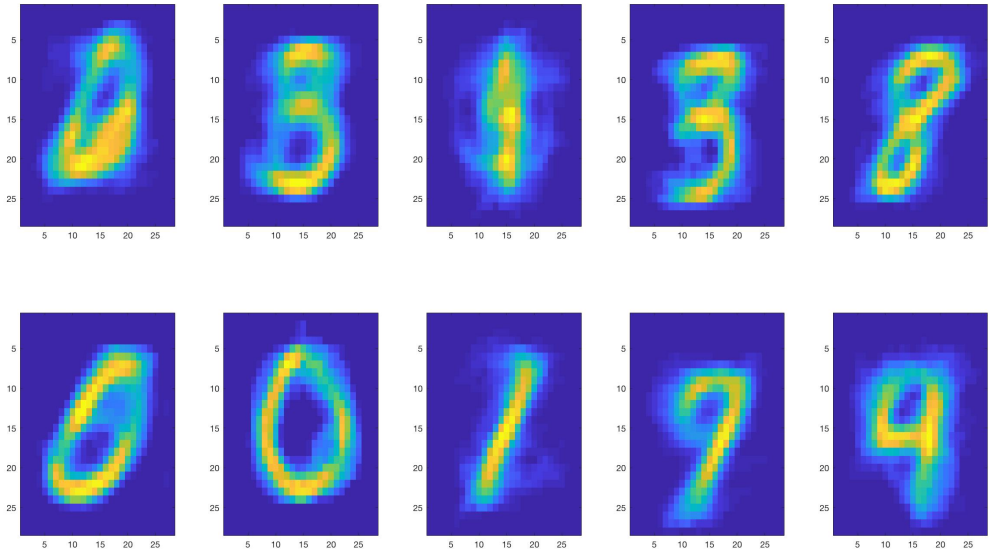

Figure 8: Mean found from the MNIST data using k-means of $k = 10$

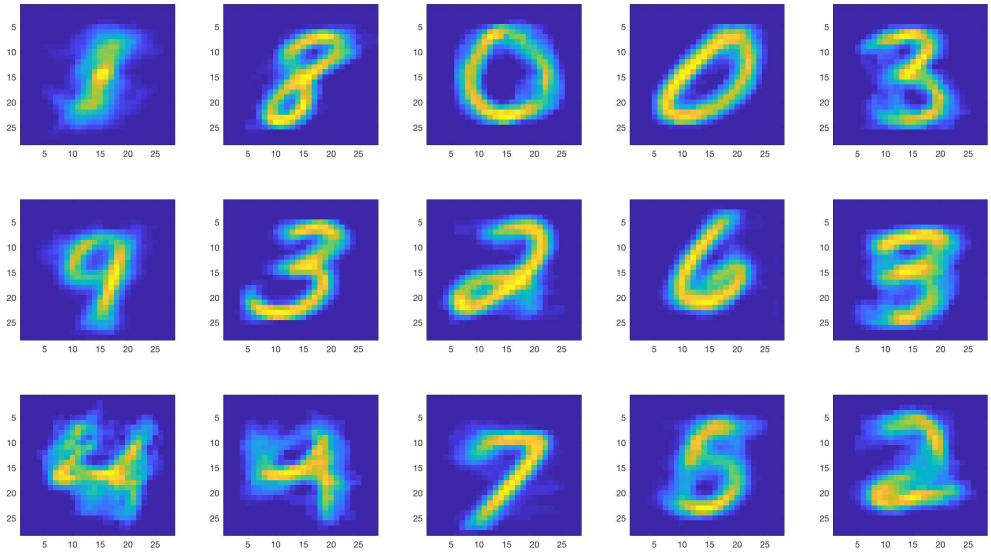

Figure 9: Mean found from the MNIST data using k-means of $k = 15$

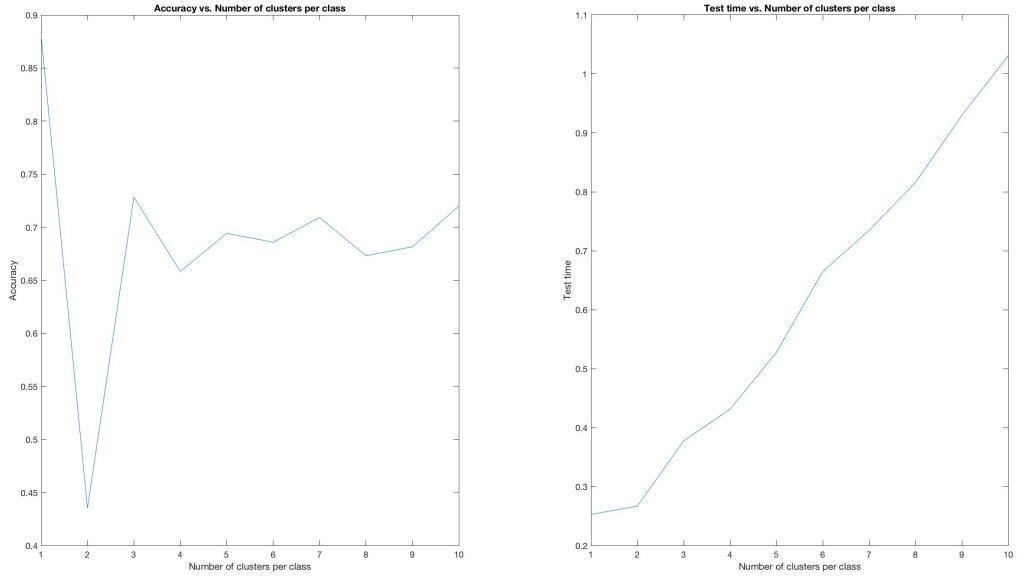

Figure 10: Test accuracy and test time as a function of the number of clusters per (true) class k-nn of $k = 10$

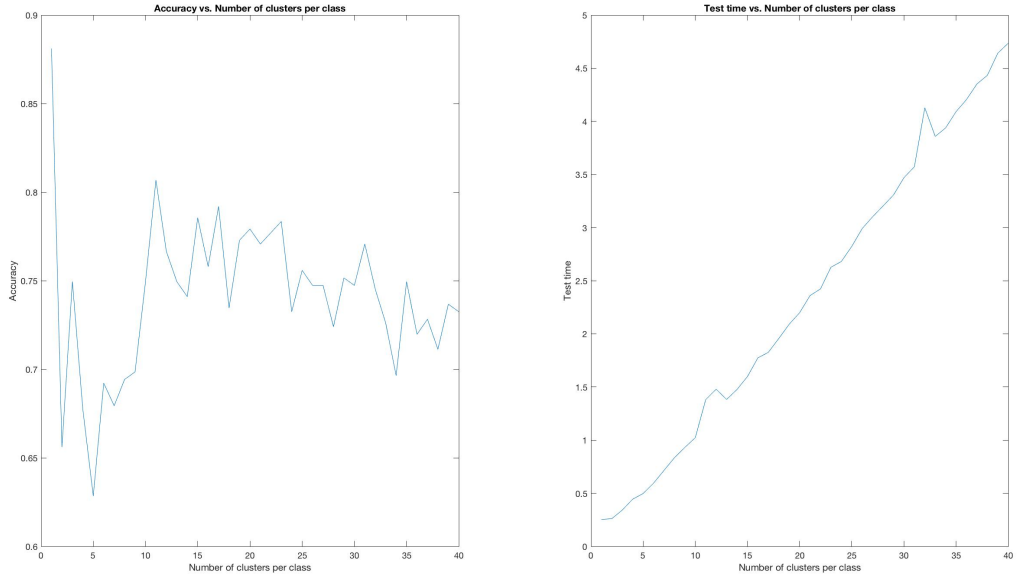

Figure 11: Test accuracy and test time as a function of the number of clusters per (true) class $k = 40$