CS 5786 - MACHINE LEARNING FOR DATA SCIENCE

Competition 1
Classifying Handwritten Digits

By
Anant Agarwal (aa2387)
Prachi Dutta (pd385)
Ria Mirchandani (rpm245)
Taru Saraswat (ts752)

Kaggle Team Name - Digit Recognition

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. APPROACHES

1.1. K-MEANS CLUSTERING

1.1.1. WITH RANDOM CENTROIDS

1.1.1.1. Motivation

K-means clustering is one of the simplest clustering methods - both to implement and run. Since the only parameter required for implementing K-means is K and since we are given the number of clusters beforehand, we decided to first obtain preliminary clusters using K-means.

1.1.1.2. Implementation Details

We implemented K-means in Python using the KMeans function from the sklearn package.

1.1.1.3. Model

- Use of Data:
  The data used here was the given "features.csv" file to perform K-means clustering.
- Parameters:
  The only parameter used here was the number of clusters K initialized to 10.

1.1.1.4. Results and Analysis

Performing K-means clustering on the features using random centroids gave a public score of 0.09533 on Kaggle. Hence, it was clear that the method was inefficient in forming correct clusters.

1.1.2. WITH CENTROIDS INITIALIZED USING SEED DATA

1.1.2.1. Motivation

Since K-means is highly sensitive to the initial centroid assignment it was decided that seed data should be used to initialize centroids.

1.1.2.2. Implementation Details

We implemented K-means in Python using the KMeans function from the sklearn package. However, instead of using random centroids we used the average of the features of the digit ID's given in the seed data as initial centroids.

1.1.2.3. Model

- Use of Data:
  The data used here was the given "features.csv" file to perform K-means clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10 and the computed initial centroids.

1.1.2.4. Results and Analysis

As expected, performing K-means clustering on the features using centroids computed from the seeds gave a higher public score of 0.37033 on Kaggle than performing K-means using random centroids. Therefore, we were able to mitigate the sensitivity of K-means to initial centroids to some extent by this approach. This also implied that the seed data was a fair representative of the actual clusters. However, since K-means is also sensitive to noise we decided to perform dimensionality reduction before performing K-means in order to improve the score.

1.1.3. WITH PRINCIPAL COMPONENT ANALYSIS

1.1.3.1. Implementation Details

We carried out dimensionality reduction in Python using the PCA function from the sklearn package.

1.1.3.2. Model

- Use of Data:
  The data used here was the given "features.csv" file to perform K-means clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10, the computed initial centroids and the number of dimensions of the features to be reduced to.

1.1.3.3. Experiments

To determine the number of dimensions of the features to be reduced to we carried out a number of experiments. We plotted the amount of variance explained by each of the selected components as shown in Figure 1. From the plot it can be seen that the maximum variance is explained by the first 3-4 components.

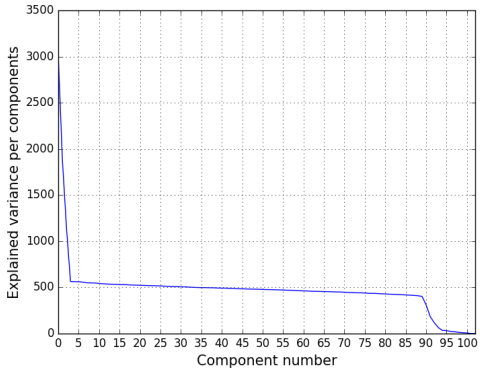

Figure 1.1: Explained variance for different components

However, the first three components only contain 12.37% of the total variance information. Almost 97.5% of the total variance is explained by the first 89 components. Therefore, we experimented with both value 3 and 89 for PCA and obtained public scores of 0.47067 and 0.36767 on Kaggle respectively.

1.1.3.4. Results and Analysis

As expected, performing K-means after reducing the dimensions of the features to 3 using PCA gave an increased public score of 0.47067 on Kaggle. Therefore, it was concluded that removing noise from the data is important to improve clustering.

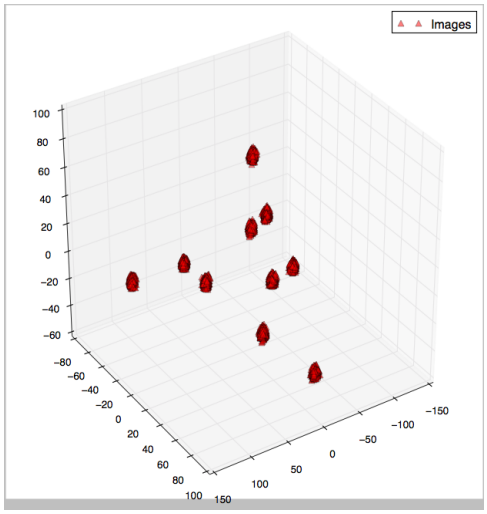

Figure 1.2: Clusters formed after performing PCA

1.1.4. WITH RANDOM PROJECTION

1.1.4.1. Implementation Details

Another way of reducing dimensions is using Random Projections. We implemented random projection in Python.

1.1.4.2. Model

- Use of Data:
  The data used here was the given "features.csv" file to perform K-means clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10, the computed initial centroids and the number of dimensions of the features to be reduced to determined to be 3 in the previous experiments.

1.1.4.3. Results and Analysis

Performing K-means clustering after reducing the dimensions of the features using RP gave a public score of 0.37367 on Kaggle. Since this was not an improvement over the results obtained using PCA we decided to use PCA for dimensionality reduction in the following experiments.

1.1.5. WITH CONCATENATED VIEWS

1.1.5.1. Implementation Details

Here, we concatenated the the normalized features data with the adjacency matrix to obtain more information and performed K-means on the concatenated matrix.

1.1.5.2. Model

- Use of Data:
  The data used here was concatenation of "features.csv" file with "adjacency.csv" to perform K-means clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10 and the computed initial centroids.

1.1.5.3. Results and Analysis

Performing K-means clustering on the concatenated views gave a public score of 0.10783 on Kaggle. Hence, simply concatenating the views to obtain more information is not sufficient. Therefore, we experimented with different way of combining information from both the views as mentioned in the following sections.

1.2. GMM CLUSTERING

1.2.1. WITHOUT PRINCIPAL COMPONENT ANALYSIS

1.2.1.1. Motivation

K-means works well for spherical clusters, but the results we got after applying K-Means clustering had the accuracy below 0.5. This pointed to the possibility of data points not

belonging to spherical clusters. We know that GMM algorithm works well for clusters having different shapes and sizes. Thus, we decided to experiment with GMM clustering algorithm and see its impact on the model performance.

1.2.1.2. Implementation Details

We used the Sklearn's sklearn.mixture module for implementing the GMM algorithm. For the initial means we used the average of the seeds provided.

1.2.1.3. Model

- Use of Data:
  The data used here was the "features.csv" to perform GMM clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10 and the computed initial centroids.

1.2.1.4. Results and Analysis

Performing GMM clustering on the features gave a public score of 0.52517 on Kaggle. As our model performance increased with GMM as compared to K-Means clustering, which indicated that the clusters weren't spherical in shape.

1.2.2. WITH PRINCIPAL COMPONENT ANALYSIS

1.2.2.1. Motivation

GMM performed better than K Means but still the results did not have very high accuracy. So we decided to pre-process the feature set by performing dimensionality reduction, to reduce the noise.

1.2.2.2. Implementation Details

We carried out dimensionality reduction in Python using the PCA function from the sklearn package.

1.2.2.3. Model

- Use of Data:
  The data used here was the "features.csv" to perform GMM clustering and the given "seed.csv" file was used to compute the initial centroids.

- Parameters:
  The parameters used here were the number of clusters K initialized to 10, the computed initial centroids and the number of dimensions of the features to be reduced to set to 3.

1.2.2.4. Results and Analysis

Performing K-means clustering after reducing the dimensions of the features using PCA gave a public score of 0.34333 on Kaggle. This was surprising initially, but taking a deeper look at the data made it clear that PCA may not actually reduce the noise, because the variation among the noise is high, and PCA will try to capture maximum variation and hence assign more weights to noise dimensions.

1.3. SPECTRAL CLUSTERING

1.3.1. WITH GIVEN ADJACENCY MATRIX

1.3.1.1. Motivation

As the adjacency graph matrix was provided, we looked at the problem as a graph clustering problem and applied Spectral Clustering. The nodes of the graph can be viewed as each image and the edges are as provided in adjacency. Unnormalized Spectral Clustering is known to be prone to Outliers as the images with least number of edge connection are highly likely to be disconnected from the graph into a separate cluster because of the mincut approach. Additionally, it was given that the clusters were of roughly equal size, which also motivated us to pick up the normalized clustering approach.

1.3.1.2. Implementation Details

We implemented the Jordan-Weiss algorithm in Python, using [1]. We also used Sklearns' Spectral Clustering module for verifying the implementation[2]. We also tweaked the k-means step of spectral clustering to use the mean of spectral embeddings corresponding to the seeds provided as the initial centroids.

1.3.1.3. Model

- Use of Data:
  The data used here was the "adjacency.csv" to perform spectral clustering.
- Parameters:
  Dimensions of Spectral Embedding - Because we were building 10 clusters so we hoped to get 10 eigen values close to zeros and took the eigen vectors corresponding to those.

1.3.1.4. Results and Analysis

Performing spectral clustering on the adjacency matrix gave a public score of 0.13533 on Kaggle. As we have seen significantly better results with K means with centroids initialized with seeds - we conclude that the adjacency graph alone isn't powerful enough to cluster the images. And a combination of both the feature set and the adjacency graph should be used.

1.3.2. WITH EUCLIDEAN DISTANCE

(Bonus Technique)

1.3.2.1. Motivation

As concluded in the 1.3.1.3, we try to use both the feature and adjacency matrix together. We thought of making the adjacency matrix as the weighted matrix, where the weight was calculated as 1/(Euclidean Distance + 1e-10) for the nodes that were connected.

1.3.2.2. Implementation Details

Calculating Euclidean distance between the images for 103 dimensions was turning out to be very expensive, so we applied PCA to speedup the distance computation. The rest of the spectral clustering was same as discussed in 1.3.1.

1.3.2.3. Model

- Use of Data:
  The data used here was the "features.csv" to compute the new adjacency matrix with euclidean distances.
- Parameters:
  Dimensionality of PCA was chosen as 3, following the same reasoning as in K-Means. The dimensions of spectral embeddings was kept as 10, following the same reasoning as in Spectral Clustering using the given adjacency matrix.

1.3.2.4. Results and Analysis

Performing spectral clustering on the new adjacency matrix with euclidean distances gave a public score of 0.63767 on Kaggle. The weighted euclidean adjacency graph did help in improving the performance, resulting in confirmation that the graph along with the feature set should be used to do the clustering.

1.3.3. WITH CANONICAL CORRELATION ANALYSIS ON FEATURES AND SPECTRAL EMBEDDING

1.3.3.1. Motivation

Looking at the performance gain in method 1.3.2.1, we formulated other techniques to combine graph and feature set together. So we thought that the spectral embeddings is a good representation of adjacency graph and extracting commonalities between the embeddings and feature set could be a good approach to go forward.

1.3.3.2. Implementation Details

We get the spectral embeddings for the unweighted adjacency matrix using the sklearn package. Perform CCA with feature set, CCA is also used from the sklearn package. We experiment with the dimensions of spectral embeddings and dimensions of CCA to arrive on the final number of dimensions. Then K-means clustering was performed on the projected features view to obtain the clusters.

1.3.3.3. Model

- Use of Data:
  The data used here was the "features.csv" and the "adjacency.csv" as the two views to perform CCA.
- Parameters:
  We observe the performance of different parameters on classifying the seeds correctly and choose the best one. The charts below show the performance on varying the embeddings and CCA dimensions(K value).

1.3.3.4. Experiments

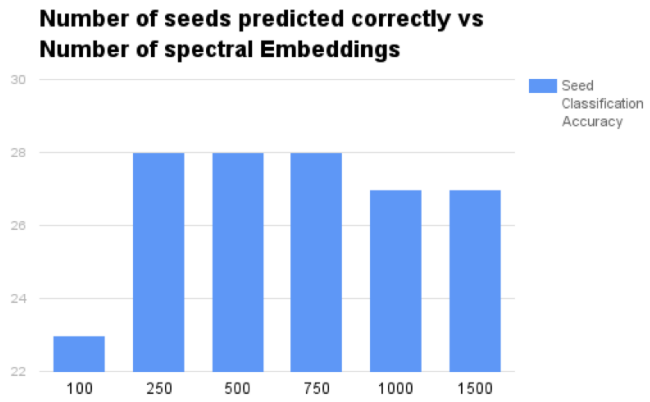

Figure 1.3: Accuracy of seed prediction using different spectral embeddings

The above chart showed that choosing the spectral embeddings of 250, 500 and 750 had the maximum number of correctly predicted seeds, that is 28 out of 30 seeds in total. So we decided to choose 500 spectral embedding for our model.

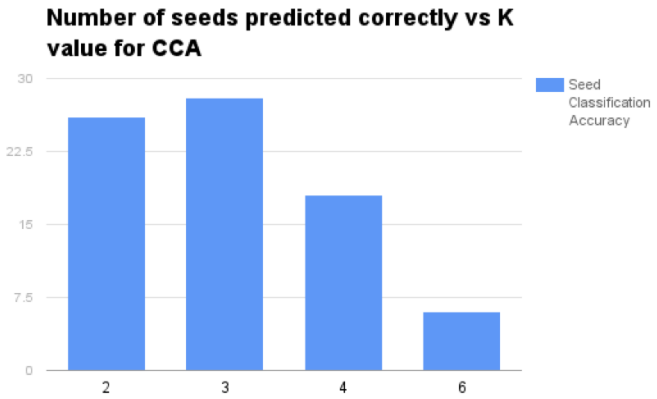

Figure 1.4: Accuracy of seed prediction using different K values for CCA

The above chart showed that our model performed best when CCA was done for K=3.

1.3.3.5. Results and Analysis

This approach gave a public score of 0.938 on Kaggle. Hence, the performance improved significantly by efficiently combining the feature set and the adjacency matrix together.

1.3.4. WITH CANONICAL CORRELATION ANALYSIS ON SUBSET OF FEATURES AND SPECTRAL EMBEDDING

(Bonus Technique)

1.3.4.1. Motivation

We did further analysis on data to see if we could combine the graph and data more efficiently. In particular, we looked at seeds provided, and calculated the standard deviation of dimensions of seeds. The reasoning here is that dimensions representing the real features should have low standard deviation and the dimensions representing the noise should have high standard deviation (by noise we mean the background of the image, and by real features we mean the actual digit in the image).

1.3.4.2. Implementation Details

We computed standard deviation for each dimension on each set of seeds. (i.e. we did this process for all the digits separately) Then we picked the most frequently occurring columns for each seed with least standard deviation values. Then K-means clustering was performed on the projected features view to obtain the clusters.

1.3.4.3. Model

- Use of Data:
  The data used here was the "features.csv" and the "adjacency.csv" as the two views to perform CCA.
- Parameters:
  The dimension of spectral embedding and the K value of CCA were set as determined in the previous experiments. The number of dimensions to be retained in the features set was determined in the following experiment.

1.3.4.4. Experiments

We observed that the most frequently occurring columns were:
0,1,2,3,4,5,6,7,8,9,10,11,12, 100, 101, 102.
Then among these we skipped the least frequently occurring columns and measured the performance on seed classification accuracy and the Kaggle score. We saw that the best result were obtained by choosing the following set of columns:
0,1,2,3,4,5,6,7,8,9,10,11,12.

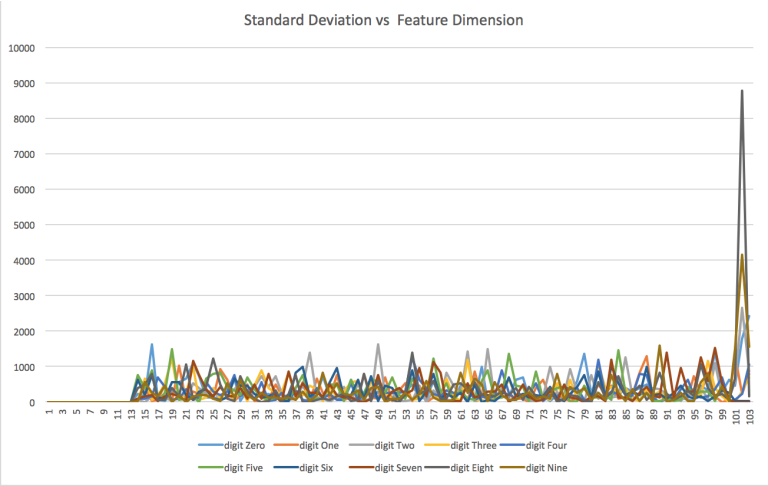

Figure 1.5: Standard deviation across different dimensions of features

1.3.4.5. Results and Analysis

This approach gave a public score of 0.999 on Kaggle. The performance improved significantly by using the above technique to filter out the correct set of dimensions from the feature set. The accuracy of 0.999 for predicting 6000 images meant that we were classified only 6 images incorrectly.

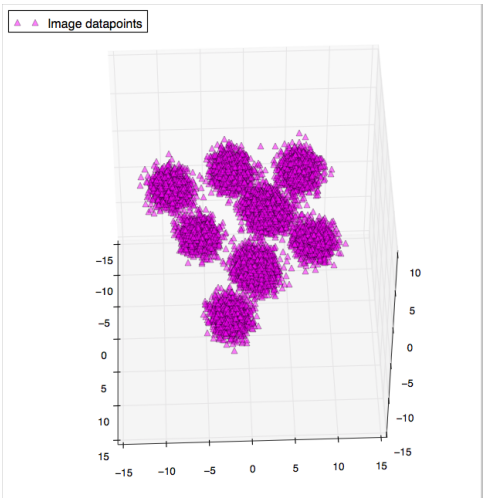

Figure 1.6: Clusters obtained using projected CCA view

1.3.5. WITH CANONICAL CORRELATION ANALYSIS ON SUBSET OF FEATURES AND SPECTRAL EMBEDDING USING 30 CLUSTERS

(Bonus Technique)

1.3.5.1. Motivation

To get the last set of points classified correctly, we thought of applying K means with 30 centroids, and then merging the 30 clusters into 10 clusters. We followed this methodology in the hope of covering any outliers that might be getting classified incorrectly.

1.3.5.2. Implementation Details

Applied K-Means with 30 centroids, the same way as the seed points. For the 3 seed points given for every digit, we merged their corresponding clusters into one, by reassigning the label to be equal to label/3.

1.3.5.3. Model

- Use of Data:
  The data used here was the "features.csv" and the "adjacency.csv" as the two views to perform CCA.
- Parameters:
  The dimension of spectral embedding and the K value of CCA were set as determined in the previous experiments. The number of dimensions to be retained in the features set was set to the best value found in the previous experiment.

1.3.5.4. Results and Analysis

This approach gave a public score of 0.9985 on Kaggle. The performance went down slightly, which could be accounted by the change of the shape of the clusters.

1.3.6. ENSEMBLE

(Bonus Technique)

1.3.6.1. Motivation

We made another attempt to classify the 6 incorrectly identified images. The idea here was to take the majority vote of multiple (best performing) models for deciding the final labels of the images.

1.3.6.2. Implementation Details

We took the majority vote of the following models:

- se(500)_cca(3)_x(13)
- se(500)_cca(3)_x(13)_clusters30
- se(500)_cca(3)_x(12)
- se(500)_cca(3)_x(16)

se(500) - indicates that spectral embeddings were taken of 500 dimensions.
x(13) / x(12) / x(16) represents the number of dimensions chosen from the feature set based on the most frequently occurring dimensions with least standard deviation (as described in method 1.3.4)
cca(3) - indicates that 3 dimensions were chosen after applying CCA
clusters30 - indicates that 30 clusters were formed and then merged into 10 (as described in method 1.3.5)

1.3.6.3. Model

- Use of Data:
  The data used here was the "features.csv" and the "adjacency.csv" as the two views to perform CCA.
- Parameters:
  The dimension of spectral embedding and the K value of CCA were different for different inputs as mentioned above. The number of dimensions to be retained in the features set was set to the best value found in the previous experiment.

1.3.6.4. Results and Analysis

This approach gave a public score of 0.999 on Kaggle. The majority vote also inclined towards the best performing model, as we did not observe any gain in model performance. We felt that taking a few other low performance models could have helped.

2. COMPARISON AND RESULTS

SECTION	APPROACH	PUBLIC SCORE	PRIVATE SCORE
1.1.1	K-Means with random centroids	0.09533	0.09983
1.1.2	K-Means with initialized centroids	0.37033	0.36983
1.1.3	PCA + K-Means	0.47067	0.46367
1.1.4	RP + K-Means	0.37367	0.36567
1.1.5	K-Means on concatenated views	0.10783	0.10950
1.2.1	GMM clustering	0.52517	0.52650
1.2.2	PCA + GMM clustering	0.34333	0.34150
1.3.1	Unweighted spectral clustering	0.13533	0.12883
1.3.2	Spectral clustering using euclidean distance	0.63767	0.64383
1.3.3	Spectral embedding on adjacency + CCA with features + K-Means	0.93800	0.93333
1.3.4	Spectral embedding on adjacency + CCA with subset of features + K-Means	0.99900	0.99817
1.3.5	Spectral embedding on adjacency + CCA with subset of features + K-Means into 30 clusters + Merge into 10 clusters using seeds	0.99850	0.99700
1.3.6	Ensemble of multiple predictions	0.99900	0.99800

Table 2.1: Kaggle scores for approaches

3. WORK DISTRIBUTION

1.1.1 - K Means with Random Centroids - Ria
1.1.2 - K Means with Centroids using Seed Data - Ria
1.1.3 - K Means with PCA - Ria, Taru
1.1.4 - K Means with RP - Ria, Taru
1.1.5 - K Means with Concatenated Views - Ria
1.2.1 - GMM without PCA - Taru
1.2.2 - GMM with PCA - Taru
1.3.1 - Spectral Clustering with given Adjacency matrix - Prachi, Anant
1.3.2 - Spectral Clustering with Euclidean Distance - Anant
1.3.3 - Spectral Clustering with CCA on Features - Anant, Ria, Taru
1.3.4 - Spectral Clustering with CCA on Subset of Features - Anant, Ria, Taru
1.3.5 - Spectral Clustering with CCA on Features with 30 KMeans cluster - Anant
1.3.6 - Ensemble - Anant

BIBLIOGRAPHY

[1] https://www.mathworks.com/matlabcentral/fileexchange/34412-fast-and-efficient-spectral-clustering/content/files/SpectralClustering.m

[2] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#sklearn.cluster.SpectralClustering