CS 4786/5786 - Kaggle Competition 1 Report

Ishaan Jain(irj4)
Nithish Raja Chidambaram(nr293)
Tommy Shum(ts539)
Kaggle Team: FissionChips

Notes:

We used Scikit-Learn, Numpy, matplotlib, t-sne, tableau

Part 1 - K Nearest Neighbors

Our first approach was to use K - Nearest Neighbors on the features file to assign labels to the unlabeled points. K - Nearest Neighbors works in the following manner: for a given unlabeled point P, we see which k labeled points are closest to P, and assign a label to P based on these k points. We used Scikit Learn's K - Nearest Neighbor algorithm. And we decided to give point P the same label as the closest neighbor. This is equivalent to K = 1. We also experimented with K = 3, 5, and 7. We settled on K = 1, because it had the best Kaggle results. We used the seed points as the candidates for the K Nearest Neighbors.

We decided to use K - Nearest Neighbors for a few reasons. First, K - Nearest Neighbors is effective with limited labeled data. Second, we wanted to try a simple naive baseline system to have a comparison for our follow-up systems.

Our K-Nearest Neighbors model achieved 8 percent accuracy on Kaggle. K-Nearest Neighbors did not perform well for a few reasons. First, the data was high dimensional, and KNN suffers from curse of dimensionality. Also, KNN is effective for classification of images, and our dataset is not of images.

Part 2 - K - Means Clustering

The second naive approach that we tried was K-Means Clustering, and we had a few different ideas for it.

i. Tried K-Means with random cluster centers with k=10 to produce 10 different clusters. We then manually mapped the 10 cluster labels that we got to the true labels (0-9 of the digits). We did this by counting the labeled points (seeds) within each cluster and assigning the label of the mode of the labeled points within each cluster.

ii. Tried K-Means with initial cluster centers to be one of the seed points explicitly. The idea behind this is, because we have few labeled points, we know which clusters they should belong to; and this was a good reason as to why they can be thought of as the cluster centers, even though the cluster centers could change after each iteration.

We could see that both of the above approach wasn't working too well, before even making submissions. This is because, while we were trying to map the labels from clustering to the true labels of the digits, we were getting a lot of overlaps- in the sense that there were a lot of clusters getting the same labels(even though they are potentially different clusters) and some of the clusters needed tie breaking to be assigned a label. This was because there were 2 sets of labeled points of equal counts within a single cluster. All of these were indicators for bad clusters!
We tried experimenting with increased K values, to produce more clusters (>10) and were thinking of using the 'elbow approach' to choose a good 'K'. But this was leading to the formation of clusters, that sometimes did not have any labeled(seed) points at all in them. This discouraged us from exploring more in this idea. In the end K-means Clustering was only able to achieve 22 percent accuracy on Kaggle.

Part 3 - Label Propagation

Since we have only a few labelled points, and mostly unlabelled points, we tried to use a semi-supervised approach - without the use of the provided similarity graph. Speaking of similarity graphs, we tried a technique that creates its own similarity graph.

Label propagation creates a similarity graph by using one of two kernels

- KNN simply uses an extension of the KNN algorithm used in part 1 - similarity is assigned based on nearest neighbours over the unlabelled points. This graph is then used for labelling

Since we already tried a variant of KNN in part 1, we tried using the other kernel:

- The Radial Basis Function (rbf) kernel uses the namesake function to create the graph. This is more robust to noise and is more suited to high-dimensional spaces.

The downside to this is that the graph created is a dense matrix representing a fully connected graph. This leads to much slower performance, especially in higher dimensions.

Another addition to Label Propagation was the use of the Label Spreading Algorithm.
The difference is that Label Spreading minimizes a loss function and is more robust to noise. Also, Label Spreading normalizes the edge weights in the graph to create a normalized graph Laplacian Matrix.

Modifiable Parameters:

- Tol: tolerance for convergence. Did not affect the model much
- Alpha: Clamping factor that decides how much of the initial information is retained. 0.2 performed the best, although it did not much affect the accuracy
- Gamma: The gamma hyperparameter for the rbf kernel. Higher gamma gave us better results, but after a particular value (20), the normalizer would not be able to handle the large values and return 0 for all labels. A lower gamma would be too loose and vulnerable to noise.

We tried both Label Propagation and Label Spreading, however both did not perform well. Our Label Propagation method achieved a score of 36 percent on the Kaggle private leaderboard and the Label Spreading algorithm achieved a score of 45 percent. Label Spreading was slightly better than Label Propagation, possibly due to some error that we could not catch, or due to the limited labels.

Part 4 - Neural Networks

As with most machine learning competitions, no competition is without a neural network. Since we had some labelled data, we tried a deep model using the seed points and a hot vector representing the label.

The data that we have consists of one row with 1084 columns per sample. Also, since we know that the data is not a vectorized image, a convolutional network was not an option. This limited us to using low level sequential models.

We used a simple 9 layer model with some dense, activation and dropout layers. We only trained on the 60 seed points for about 20 epochs.
We saw some decent results despite the lack of labelled data.

Modifiable Parameters:

- Layers: perhaps the most important parameter in any network, the number and type of layers affects the model performance. Since we have limited labelled data, increasing the number of layers would only overfit on the data and slow down model convergence. The only thing we could do was set a fixed number of 9 layers and change layer types and parameters passed to them.

GANs - despite the advanced technique and its application, in this case, the GAN would

Simply serve to overfit on the data by generating points from the original distribution. Even if we used an Auxiliary Classifier GAN, that generates points from individual labels rather than the entire distribution, we would just be performing a k-means on a higher number of points, which would not improve classification.

Our Neural Network model was only able to achieve a score of 33 percent on Kaggle. This is because Neural Networks is more suited for supervised tasks and we did not have a lot of labeled data.

Part 5 - Spectral Clustering

We tried spectral clustering on the 10,000 points using an affinity matrix, as it gives a way to use the similarity information. We created the affinity matrix in the following manner: if two points I and J are similar, then their corresponding entry in the affinity matrix will have value 1, otherwise it will have value of 0. This is similar to the construction of an adjacency matrix. We tried to perform spectral clustering using this affinity matrix, but we got a warning that the affinity matrix was not fully connected. This makes sense since the similarity graph only had similarity information for the first 6,000 points. Therefore, we decided to consider only the first 6,000 points and created the affinity matrix with just the first 6,000 points. We performed spectral clustering on just these 6000 points and created 10 clusters. We used similar mapping techniques using the seed points to get the labels of the clusters. Then we used KNN (K=1) to label the last 4,000 points.

The figure below shows labels generated by spectral clustering(F Labels) vs actual labels(T Labels). The line is a simple projection based on the labels. If the labels were correctly matched, the line would be of the form $y = x$. However, the trend line is fairly constant at $y \approx 4.6$

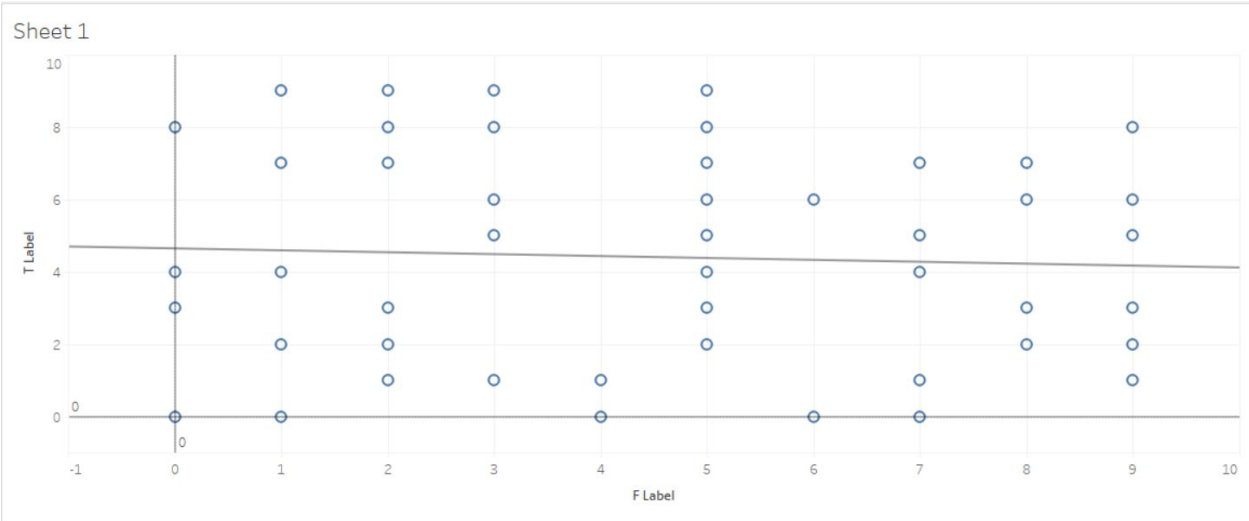


Fig 5.1 - True Labels vs Fake Labels

Another look at the T labels vs F labels.
The x-axis shows the distinct label ids (total 60) and the points for that label

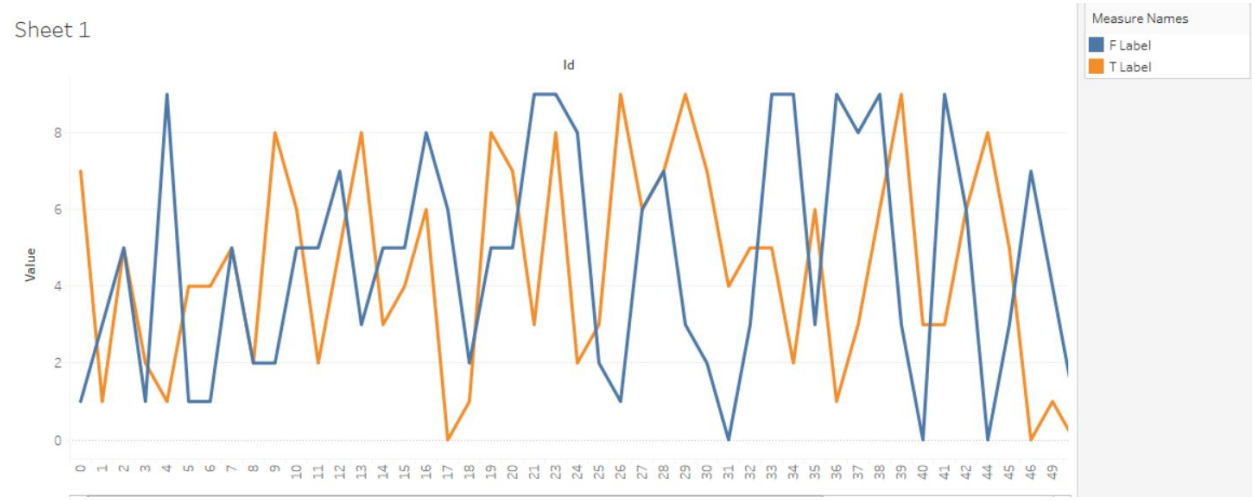


Fig 5.1a - Labels vs Ids

Remaining ids:

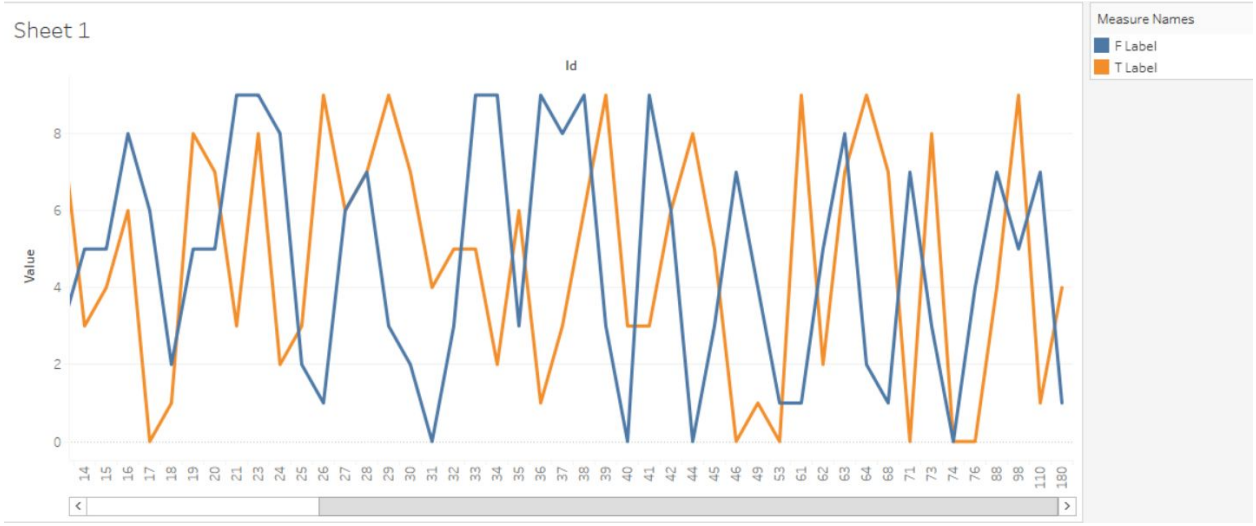


Fig 5.1b - Labels vs Ids continued

The bubble chart below shows number of overlaps between labels as the size.
Ie. If there are 5 points that have the same T and F label, they will appear bigger than one that has 4 points with the same label.
It is fairly well distributed, but some label correlations stand out - 1:4, 9:1 etc.

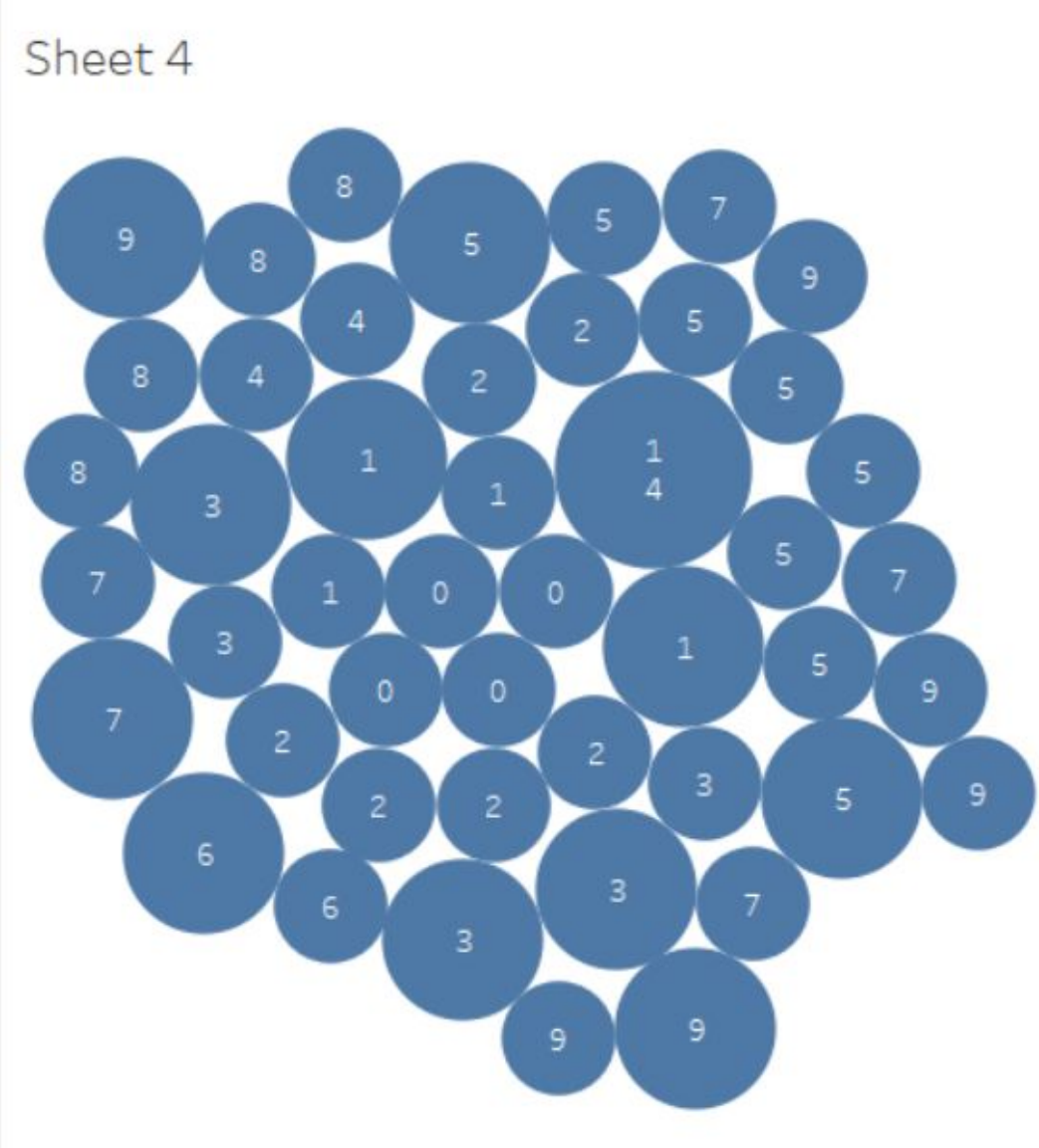


Fig 5.2 - Label overlaps as counts

Spectral Clustering did not perform too well. Our best Kaggle score with Spectral Clustering was only 45 percent. Spectral Clustering did not perform well because it only used the information from the similarity graph and not the features.

Part 6 - Spectral Embedding and Related Techniques

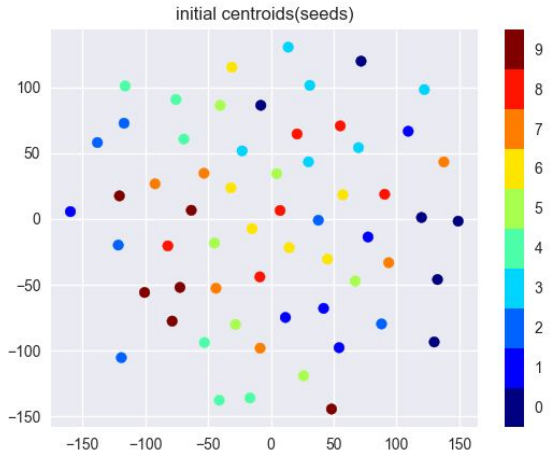
While looking at the Scikit Learn library we noticed that there is an algorithm called Spectral Embedding, which gives a projection alone without clustering. We did not want to perform clustering; we wanted to treat the embedding as view 2 and perform CCA using this and the original features.

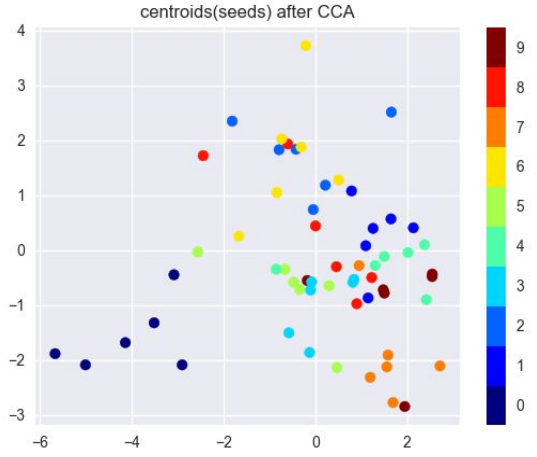
The competition rubric encourages us to use information from both the similarity graph and features to develop our model. Therefore, we decided to perform spectral embedding of similarity graph and combine this information with the original features through CCA. CCA finds the directions along which there is maximum correlation between 2 views, which is what we want - to find the most common information from the similarity graph and the original features. We were also suggested to look into this method by a Teaching Assistant.

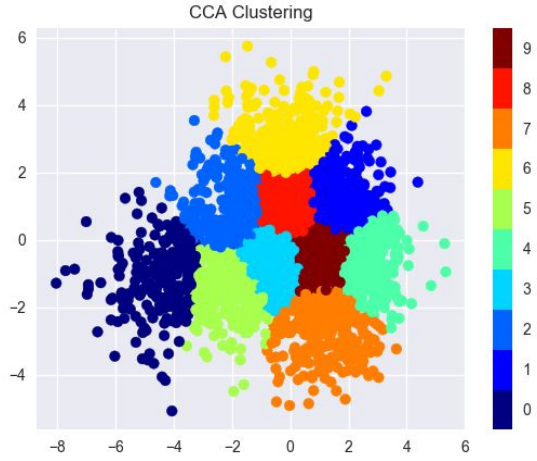
We performed spectral embeddings with default n_components, which led to projection into 2D. So view 2 had 6000 points each of 2 dimensions. For view 1, we experimented with having original set of 1084 features as well as using PCA to get rid of dimensions that did not have a lot of variance(noisy). Not reducing dimensions using PCA was more helpful than using it.

We then applied CCA and found 2 directions that had the most correlation between view 1 and view 2 and projected the points onto these directions. We then performed K-Means clustering and found 10 distinct clusters. This time, the clusters were more well formed. In the sense that, there wasn't much tie breaking that was needed and no clusters had redundant labels while performing the mapping. This is because, now we are using the most relevant information, (found using CCA) based on which the clustering is performed.

The following 3 figures shows the visualization of different data:
Figure - Initial Centroids(seeds): This figure displays the initial seeds, plotted after using t-sne to reduce to 2D. We can see that there are no clear patterns that we can discern from here
Figure - centroids(seeds) after CCA: This figure shows the same seeds after we have used spectral clustering and applied CCA on the first 6000 points. From here we can see that there are faint indications of where each cluster's center would lie
Figure - CCA Clustering: This figure shows the last 4000 points, whose labels are to be predicted, after performing CCA. From here, it's just a matter of mapping the labels of these clusters to the labels of the corresponding seed points.


initial centroids(seeds)
100
50
0
−50
−100
−150
−150
−100
−50
0
50
100
150
9
8
7
6
5
4
3
2
1
0


centroids(seeds) after CCA
4
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
9
8
7
6
5
4
3
2
1
0


CCA Clustering
6
4
2
0
−2
−4
−8
−6
−4
−2
0
2
4
6
9
8
7
6
5
4
3
2
1
0

Our method that combined Spectral Embedding, PCA, CCA and Clustering resulted in a Kaggle score of 55 percent on the private leaderboard, our highest score. This method was particularly effective compared to our previous method because it used relevant information from the similarity graph, the features and the seeds!