EE 516: Mathematical Foundations of Machine Learning **Winter 2023**

Course Project: Subspace Clustering
Due: March 24, 2023, 11:59PM PT

Student Names: Matthew Sullivan and Will Mijangos

1 Paper Descriptions

1.1 Paper 1

Paper Title: Sparse Subspace Clustering: Algorithm, Theory, and Applications
Student Name: Matthew Sullivan

1.1.1 Problem Description & Formulation

Sparse Subspace Clustering: Algorithm, Theory, and Applications contributed a significant framework for clustering high dimensional data. The algorithm, Sparse Subspace Clustering (SSC), relied on the notion of high dimensional data lying in or near some low dimensional structure. In this case, SSC searched for a set of subspaces to cluster data. More importantly, SSC asserted that the correct subspace clustering of data shall be the sparsest [1]. As a result, SSC focused on selecting a sparse representation for the data. This is important as it is robus to common errors in data such as missing and noisy data.

Clustering is typically unsupervised, and this case is no different. SSC relied on the property of self-expressiveness, formally defined as, "each data point in a union of subspaces can be efficiently reconstructed by a combination of other points in the dataset" (Section 2.1) [1]. This can be compared to a linear dependency: a data point that is self-expressive is recoverable through a linear combination of exclusively other data points, Eq. 2.

In the paper, the data is defined to have a set of N noise-free samples, $\{y_1, \cdots, y_N\}$, with each sample of data being D dimensional, $y_i \in \mathbb{R}^D$. The collection of data is slotted into a matrix, $\mathbf{Y}$, as columns altered by an unknown permutation matrix $\mathbf{\Gamma}$, seen in Eq. 1. From Eq. 1, one sees how the samples, y_i, are the columns of the larger $\mathbf{Y}$ matrix. One critical component is that $\mathbf{Y}$ needs to be over-determined rank$(\mathbf{Y}) < N$, to allow linearly dependency between the data. If we do not attain this, some data will lie in a nullspace and won't be self-expressive.

At the same time, there exists a set of n-many linear subspaces, $\{S_\ell\}_{\ell=1}^n$, with each subspace being in d-dimensional, $S_i \in \mathbb{R}^D$. All of the noise-free data points must lie in a union of these linear subspaces. It's important to note that n is a controllable variable. We often do not know how many linear subspaces the data lies in. The paper suggested one can make a few assumptions on the dimension of the subspaces using the weights of the singular values.

$$\mathbf{Y} \triangleq [\mathbf{y_1} \cdots \mathbf{y_N}] = [\mathbf{Y} \cdots \mathbf{Y}]\mathbf{\Gamma} \tag{1}$$

Given that $\mathbf{Y}$ has been made, SSC is a sparse optimization algorithm to find the fewest amount of coefficients to calculate the self-expressive representation. Eq. 3 visualizes self-expressiveness as data that is linearly dependent. The sparse optimization problem locates the coefficients, c, necessary to reconstruct $\mathbf{y_i}$. Ideally, the number of coefficients is equal to the dimension of the subspace, allowing the other data points to be the basis vectors. In practice, this number of often unknown and overestimated as a result.

$$\mathbf{y}_i = \mathbf{Y}\mathbf{c}_i \quad c_{ii} = 0 \tag{2}$$

$$\mathbf{y}_i = c_1\mathbf{y_1} + c_2\mathbf{y_2} + \cdots + 0\mathbf{y_i} + \cdots c_n\mathbf{y_n} \tag{3}$$

To solve SSC, a sparse optimization problem, Eq. 4 must be solved. It can be solved using convex optimization techniques [1].

1.1.2 Algorithm Description

Sparse Subspace Clustering (SSC) is an optimization problem that yields a sparse solution – the coefficients necessary to reconstruct any data point using exclusively other data points. It can be compared to PCA. PCA components are subspaces that capture the most variance in the data in descending order. Instead of capturing variance, SSC focuses on capturing the least amount of coefficients to build subspaces with the most self-expressive points. That is to say: SSC finds the least amount of coefficients to reconstruct $\mathbf{y_i}$ and $\mathbf{y_j}, i \neq j$, given they lie in the same subspace. It does this by locating coefficients for linear combinations in the optimization problem shown in Eq. 4 and Eq. 5. Both of these are convex and can use convex optimization techniques like alternating direction method of multipliers. After the coefficients have been found and stored in a matrix, $\mathbf{C}$, and made sparse – either by a threshold or keeping the top-K largest – a weight matrix is formed through $W = |\mathbf{C}| + |\mathbf{C}^T|$ – which yields a Laplacian. This Laplacian is clustered using spectral clustering with the results returned from the algorithm.

$$\min \|\mathbf{c_i}\|_1 \text{ such that } \mathbf{y_i} = \mathbf{Y}\mathbf{c_i}, \;\; c_{ii} = 0 \tag{4}$$

$$\widehat{\mathbf{c_i}} = \arg\min_{\mathbf{c_i}} \|\mathbf{c_i}\|_1 + \lambda\|\mathbf{y_i} - \mathbf{Y}\mathbf{c_i}\|_1, \;\; c_{ii} = 0 \tag{5}$$

Algorithm 1 Sparse Subspace Clustering

```
**Initialize:** Set of data $\{y_1, \cdots, y_n\}$ that lies in a union of $n$ linear subspaces
Solve Eq. 4 sparsely. We did this by solving 5.
Normalize $\mathbf{c_i} = \frac{\mathbf{c_i}}{\|\mathbf{c_i}\|_1}, \forall i \in N$
Form Weight Matrix $\mathbf{W} = \|C\| + \|C\|^T$
Spectral Cluster $\mathbf{W}$
**Output:** Clusters, Clustered Data
```

The paper also has a section on "Practical Extensions". In clustering, there seems to be a few recurring problems: data with noisy outliers, incomplete data, and data that lies in affine subspaces instead of linear subspaces. Data that lies in an affine subspaces means the data does not appear to be self-expressive – there is no linear combination because there is no origin for the subspace. Loosely speaking, without an origin, there is no starting point for basis vectors, therefore, no basis vectors exist. The paper produces solutions for these predicament that require altering your sparse optimization problem and constraints. It works very well empirically. However, its computational cost is exponential on the dimension of the subspaces.

1.1.3 Theoretical Results

The foundational assumption is that the sparse optimization program recovers coefficients that can recover linearly dependent data. For example, given $N - 1$ even valued data and 1 prime data point, SSC would have to recover odd, even, and negative valued coefficients to reconstruct the prime data point. If it does not, the prime data point will be lost. There is no assumption on the distribution of data in each subspace [1].

There are certain conditions that must be met for sparse subspace recovery. The main condition is that the point to be recovered must lie within some subspace sparse representation to begin with – it cannot lie in the nullspace. It can lie in intersection of subspaces. Furthermore, these conditions must hold for two cases: the smallest coefficients, $\mathbf{c_i}$, to recover a data point and another condition I do not understand, Eq. 6. SSC converges to a local minimum.

$$\mathbf{c_i} = \arg\min_{\mathbf{c_i}} \|c_i\|_1 \text{ such that } \mathbf{y_i} = Y_{-i}c^4 \tag{6}$$

1.1.4 Relation to Course Material

The algorithm leverages convex optimization – a topic loosely mentioned in class. The true sparse optimization problem is a ℓ_0-norm instead of an ℓ_1-norm in Eq. 4. Similar to class, the paper performs a convex relaxation to make it operable. Because it is convex, our initial programming solution used gradient descent. It was slow. I spent a few hours trying to get an ADMM solution, however, I could not produce it. The spectral clustering is the exact same as introduced in the first demo, and the solution code is used in our replication of SSC. The idea of self-expressiveness is just a linear combination of data where basis vectors are the coefficients of other data points – which could be viewed as a new origin to define the subspace.

I do not understand the graph theory very well. I understand more of the sparse theory than I anticipated. I think these should both be possible within the next few months. Looking at the ADMM solution, I understand parts of it, but not as much as I would like.

In my opinion, the theory exceeds the content of the course. While part are recognizable, the interconnect is missing, starting from Section 4.2.

1.2 Paper 2

Paper Title: Subspace Clustering using Ensembles of K-Subspaces
Student Name: Matthew Sullivan

Subspace Clustering using Ensembles of K-Subspaces (EKSS) is accurate to the description of the algorithm: use an ensemble of K-subspaces to vote on the correct clustering. This works on "evidence accumulation clustering": the idea that something about the random clustering must be partially correct [2]. If random clusterings have correct information, the correct clustering is recoverable if one can extract the partially correct information from the random clusters.EKSS extracts the correct information by performing a PCA on the clustered points and counting the number of times a point is assigned to a cluster.

The K-subspaces algorithm wants to minimize the sum of residuals from the original points to their projected subspace, Eq. 7. The estimated clusters, $=\{c_1, \cdots, c_K\}$, lie in subspaces with assigned orthonormal bases, $\mathcal{U} = \{U_1, \cdots, U_K\}$. Eq. 7 is unfortunately very computationally expensive, and is commonly solved using an alternating algorithm to find a global minima. However, if ran, KSS does get some information correct, and this is the foundation for consensus clustering and EKSS.

$$\min \|y - Ax\|_2^2 \longrightarrow \min_{\mathcal{C},\mathcal{U}} \sum_{k=1}^{K} \sum_{i:x_i \in c_k} \|x_i - U_k U_k^T x_i\|_2^2, \;\; A = U_k U_k^T, x_i \in y \tag{7}$$

EKSS uses consensus clustering to extract information from KSS and form a similarity matrix. Consenus clustering is randomly clustering points and assigning them points to each clustering such that, if repeated many times, some pattern of information emerges.

Generally, subspace clustering is an important field as it is a form of unsupervised classification. K-subspaces (KSS) was a previous algorithm discussed for its efficiency and performance. One of its failures, however, was a lack of theoretical guarantees – something this paper provides for specific cases. Furthermore, this paper produced EKSS – a altered form of KSS that has theoretical guarantees and a competive theoretical run time. To simplify things: KSS was a desirable but lacked theorhetical guarntees; EKSS has a specific approach that is competitive to KSS and has theoretical guarantees.

1.2.1 Algorithm Description

EKSS is a geometric algorithm – it performs clustering by utilizing properties of the data. It uses iterations to capture information within the structure of the data to cluster it. Furthermore, it assumes that the potential subspaces are the true possible subspaces for the data and noise free.

Given the noise-free subspaces and the possibly noisy data, clustering repetitively should eventually promote the true clustering for the data. This can be compared to the Central Limit Theorem. Imagine the true subspaces as the true mean. Every time you randomly cluster, you receive a sample mean. Eventually, after enough random clusters, your sample means will converge to the true mean – the correct underlying subspace. A variation of EKSS, EKSS-0, performs well empirically when compared to Thresholded Subspace Clustering.

Algorithm 2 Subspace Clustering using Ensembles of K-Subspaces

```
Initialize: $\mathcal{X}$ data, $\bar{K}$ candidate subspaces, $\bar{d}$ dimensions for each $\bar{K}$ subspace, $K$ number of clusters, threshold $q$, Number of base clusterings $B$, $T$ number of iterations
for b=1, ..., $B$ do
    Generate Random Bases, $\mathcal{U} = \{U_1, \cdots, U_{\bar{K}}\}$, each $\bar{d}$ in dimension for KSS Clustering
    Form cluster, $c_k$, by projecting data onto generated clusters, Eq. 7
    for t=1, ..., $T$ do
        Use PCA to estimate subspace bases
        Refined cluster, $c_k$, using estimated subspace bases
    end for
    form, $C^b$, a matix of clusters. Save up all of these for each $b$.
end for
From all the runs, count up number of times a cluster was used. Store in similarity matrix $A$
Make $A$ sparse by thresholding at $q$, producing $\bar{A}$ State Output: Spectral Clusters, C, by spectral clustering $\bar{A}$ with $K$ clusters
```

1.2.2 Theoretical Results

Given two dot products, $x^T y$ and $x^T z$, and one is significantly larger than the other, $x^T y >> x^T z$, one could say y is more similar to x than z. Thresholding dot products produced a thresholding subspace clustering approach. Dot products can be viewed from a different perspective, however. In Eq. 8, dot products are computed as the magnitude of each vector with the angle, θ between them. Using this approach, one could say similar vectors – vectors that have a larger dot produc – have a small angular separation. This holds true for high dimensional vector spaces. In EKSS, angular separation is used to calculate the angles from one point in a subspace to another point. This is used to quantify the distance between clusters. If the data has a high, positive angular separation, EKSS proved an upper bound to guarantee correct clustering. Specifically, EKSS' probability of clustering points, x_i and x_j, correctly is monotonically increasing with the absolute value of the inner product, 9, [2]. Assuming EKSS preserves angular separation, it has a theoretical "recovery guarantee" that is historically unproved with KSS and thresholding subspace clustering. EKSS, being a repetition of KSS, converges to a local minimum.

$$x^T y = |x|\ |y| \cos\theta \tag{8}$$

$$P(\theta) = \mathbf{P}\{Qx_i, Qx_j \text{ both assigned to } U_1\} \tag{9}$$

1.2.3 Relation to Course Material

There are a few core components that relate to the course material. The idea of residuals and projections is familiar, however, clustering them is new. Instead of clustering by the closest subspace, we stated that PCA told us to cluster using the subspaces that contain the most variance, which we used to unknowingly cluster in Homework 6. Similarly, KSS was generally discussed in the course but not explicitly. The final step of EKSS – spectral clustering – was also covered in the course.

There were a few bits of the paper that I did not catch. First, I did not catch that the subspaces needed to be the true, noise-free subspaces. I ended up reading your dissertation and this talk by Dr. Laura Balzano. I think a student from this course could understand concepts from these resources except angular separation theory – which seems to be a key component in the recovery guarantees. I still do not understand it.

1.3 Paper 3

Paper Title: Adaptive Online k-Subspaces with Cooperative Realization
Student Name: Will Mijangos

1.3.1 Problem description and formulation

This process attempts to solve the rather typical problem of classification. The potential problems lie in large data sets where alternative solutions scale much faster than the proposed algorithm. Not only that, but it may also be able to solve some uncertainties of the true number of subspaces and their dimensions where these are unknown or variable. Applications of this include any where classification is a problem, image processing, signal processing, data mining, data compression and recovery. The typical form fits a collection of k subspaces that are a collection of data points and attempts to separate them with a dependance on similarity. This is standard when you have a known number of subspaces and a finite dataset. As for the algorithm, it is an online algorithm, which means that it can process the data one point at a time, in contrast to other methods, which require the entire data set to be available before they can be run. It is a greedy algorithm, which means that it iteratively refines the set of k subspaces by greedily re-initializing each subspace with the remaining data, meaning it uses the data that it has. This ensures that each subspace is updated to reflect the current state of the data set, and that the final set of subspaces is a good fit for the data. It has been shown to be effective on a variety of data sets, and it has been shown to be a more robust classification method to noise and outliers than alternatives.

1.3.2 Algorithm Description

The algorithm is initialized with k randomly chosen points from the data set. In each iteration, each subspace is re-initialized with the remaining data. There are regularization terms that are functional. The first is a penalty scaled relative to cluster size, another is fixed and this allows convergence for large data sets or unknown number of subspaces. The points are then assigned to the subspace that they are closest to. This process is repeated until the algorithm converges to an acceptable error.

Algorithm 3 CoRe Algorithm

```
Initialize: k-Subspaces, S, with randomly chosen points from the data set, $\epsilon$
error = $\epsilon + 1$
while error > $\epsilon$ do
    Re-initialize each subspace S with the remaining data.
    Assign each point to the subspace S that it is closest to.
    Compute error
end while
```

The CoRe algorithm can be initialized with k randomly chosen points from the data set. However, it has been shown that the algorithm can be improved by initializing the subspaces with the results of a k-means algorithm.

1.3.3 Complexity

The complexity of the CoRe algorithm is $\mathcal{O}(k^2 \log(k))$ where k is the number of bases.

The CoRe algorithm has two tuning parameters:

- k: The number of subspaces.
- ϵ: The convergence tolerance.

The value of k should be chosen such that the data set is well-represented by the subspaces. The value of ϵ should be chosen such that the algorithm converges quickly.

1.3.4 Theoretical results

The main theoretical result of the paper is that CoRe is able to find k subspaces that are close to the true subspaces, even in the presence of noise and outliers. The paper also shows that CoRe is able to achieve better performance than state-of-the-art algorithms, such as k-means and k-svd, on both synthetic and real-world data sets. The experiments show that CoRe is able to find k subspaces that are closer to the true subspaces, even in the presence of noise and outliers using image data and compares results of classification using alternative methods finding improvements. The conclusion includes a discussion on the implications of the results for the field of machine learning. The results suggest that CoRe could be used to improve the performance of machine learning algorithms that require k subspaces, such as image clustering and data compression. The main takeaways are that this scales very well, it may adapt to unknown k (number of subspaces) and d (dimensions) of those subspaces, and it can process data coming in.

1.3.5 Relation to class

This algorithm directly employs Stochastic Gradient Descent (SGD) which was covered in class, albeit with a different regularization term and many other factors but that was immediately discernible. The paper also compares results to k-means and methods using the SVD which were also introduced. Subspace clustering and classification is a recurring problem in this subject field.

2 Comparison of Algorithms

2.1 Interpretability

The Sparse Subspace Clustering was the easiest to understand and implement. It felt like two broad, open to interpretation steps: get some sparse coefficients, spectral cluster them. I felt like this paper alone could birth a wide range of spin-off topics and experiments. It also had the most theory that felt similar and understandable over one pass. EKSS was the easiest to visualize using Dr. Balzano, however, the unfamiliar theory made it difficult to engage with.

I think SSC was easier to understand because it was an optimization problem as well. EKSS, being geometric, is indirectly related to the course while SSC is directly related. Similarly, CoRe is a happy medium that introduces directly related topics such as stochastic gradient descent, least squares, and general programming operations. It also included a few tangential stretch topics like self-expression, Hessian matrices, and batch computations.

2.2 Theoretical Guarantees

Sparse Subspace Clustering offers theoretical guaranteed success if two conditions are met, Eq. 4 and Eq. 6. EKSS has a theoretical guarantee as long as the subspaces are not not close in every direction and the angles are preserved. The probability of success is related to the magnitude of the dot products. While SSC promotes guarantees most related to this course, one could argue that a proper theoretical comparison of SSC and EKSS is related to the trade-offs between SSC Eq. 6 and 4 and EKSS' probability of co-clustering, Eq. 9. This, of course, is comparing an optimization problem to a geometric one. Comparing SCC to CoRe, CoRe seems to be an variation of SSC that uses the Hessian and stochastic gradient descent instead of Eq. 5 and incurs a weaker theoretical guarantee as a result.

2.3 Empirical Guarantees

SSC is guaranteed to recover a sparse solution even after the convex relaxation, Eq. 5 [1]. For CoRe, there are guaranteed intersections as the number of subspace dimensions increases with a fixed number of clusters, and it is guaranteed to cluster all points. EKSS is guaranteed to cluster all points as long as the subspaces are separated. From reading papers, SSC seems to be accepted as a generally good baseline. EKSS is demonstrated to outperform SSC with subspaces angles $\theta \in [0.2, 0.6]$. With other angles, they perform identically. CoRe to be a different version of SSC. Instead of trying to compete for performance, it focuses on computational complexity and scalability.

2.4 Computational Complexity

The computational complexity of Sparse Subspace Clustering grows exponential from the subspace's dimensions – which is also the sparsity level. EKSS boasts a complexity of $\mathcal{O}(\bar{K}D\bar{d}(D+N))$ where $\bar{K}$ is the number of randomly generated subspaces with $\bar{d}$ dimensions, and N samples of data D in dimension. CoRe's most computationally expensive step is $\mathcal{O}(R\hat{k}\hat{d}^2(D+n_{bs})$ and is overall $\mathcal{O}(k^2\log(k))$ with k being the number of subspaces. EKSS could easily become the most computationally efficient – especially with low dimensional data. Given the choice, I would select SSC if there number of subspaces is low.

3 Algorithm Implementation & Testing

We implementing the Sparse Subspace Clustering (SSC) algorithm on a synthetic data set and two digits from the MNIST handwritten digit data set. It seemed the most related to the Spectral Clustering demo and the recently discussed alternative direction method of multipliers. It also had the most citations which means it is the coolest. There are also a few changes we made to the algorithm. The paper states that, unless the data has previously proven subspaces, its best to perform SSC on the whole data set. This, however, is is computationally $\mathcal{O}(e^d)$ depending on the number of dimensions to define a subspace, d. For the MNIST handwritten digit dataset, d could be around 784. In their real world experiments, they briefly discuss how to use a Scree plot to select the number of PCA components by identifying the "knee" from the Scree. The number of components is the sparsity level to span the subspace – which is the number of required coefficients. The alternative method is to recover a complete coefficient matrix and make it sparse using a threshold. For us, this provided worse empirical results.

The link to the Google Colab file can be found **here**. You must be signed into PSU's Google Account to access.

3.1 Results on Synthetic Data

The synthetic data composed of two orthogonal subspaces: one with basis vector $[1, 1, 0]$ and the other with basis vector $[1, 0, 1]$. A total of N many samples were randomly scaled with an even split of samples in both subspaces. Additive noise in $\mathbb{R}^3$ from a normal distribution was added to each sample. The results of the Sparse Subspace Clustering, Figure 1, proved to be unaffected by noisy data.

Increasing the number of samples without changing the number of possible clusters allowed for more unique coefficients – which should improve the performance of clustering. Our results confirmed this intuition, Figure 2.

There are two different parameters worth noting: the weight of the regularizer, λ, and the number of dimensions to consider, P. Their influence is demonstrated in Figure 3 and 4 respectively. If the number of dimensions is unknown, P would change to a tolerance, τ, where coefficients less than τ are set to zero.

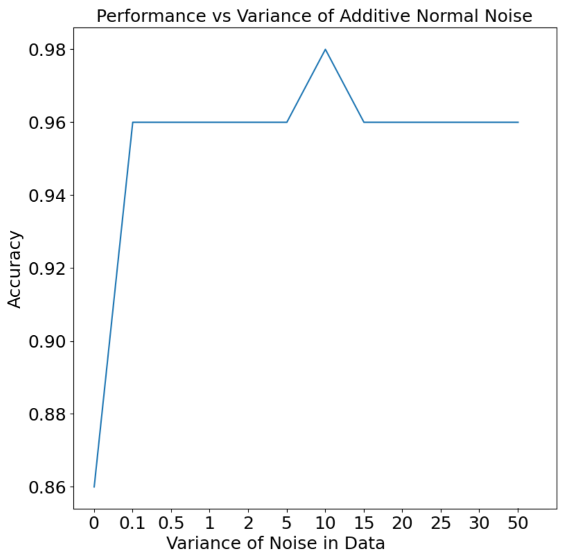


Figure 1: Additive white noise did not affect the performance of SSC.

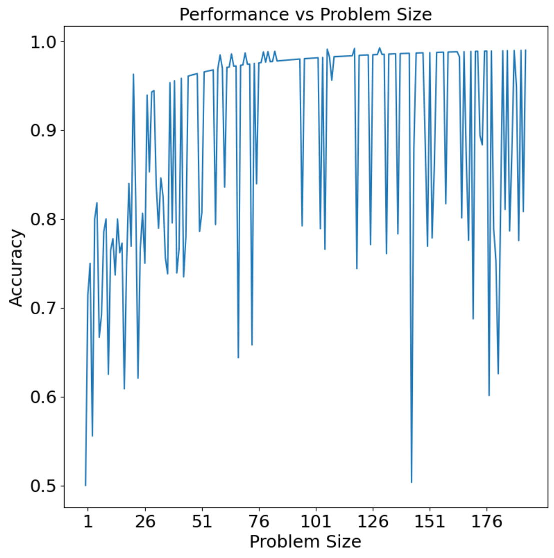


Figure 2: When the number of samples increased and the number of clusters stayed the same, the self-expressiveness of data points improved. There are more possible coefficients to promote tailored linearly dependent combinations to recover difficult data.

The best results came with $\lambda = 1e - 6$ and $P = 5$ coefficients, which produced a 99.5% accuracy, Figure 4. With $N = 400$ and two clusters‘, the run time was 153.08 seconds.

The run time of the algorithm empirically proved to be at least polynomial in terms of the number of samples, N, Figure 7.

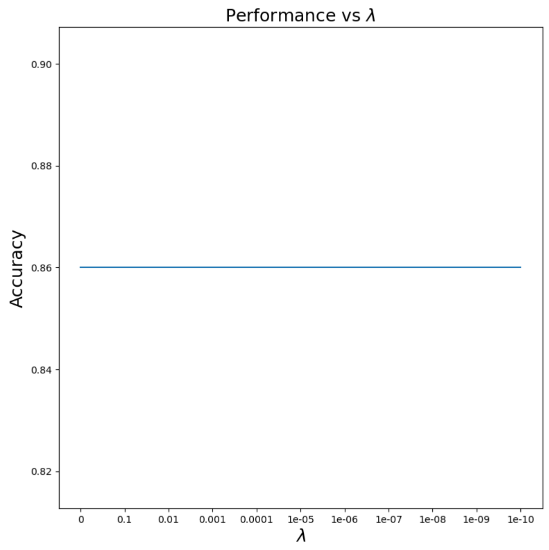


Figure 3: For our synthetic data, the weight of the constraint was uncorrelated with the performance of the SSC. It is assumed that the synthetic data always had some linear combination that perfectly reconstructed all data points. This could indicate that the minimum number of basis vectors is lower than our estimation.

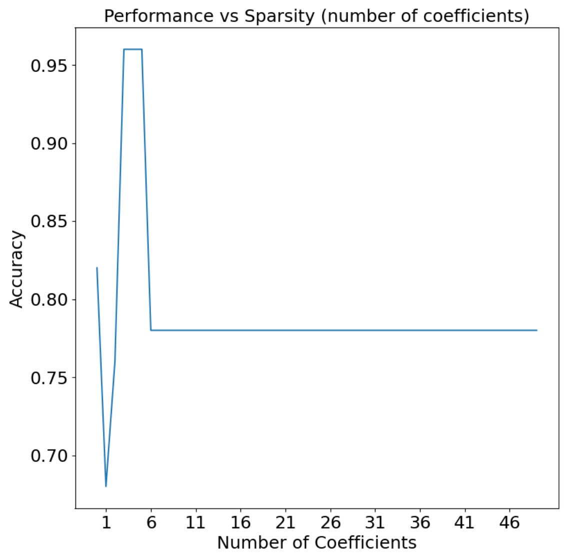


Figure 4: With 50 samples of data, the number of components increased as it approached the true number of basis vectors to span the subspace. As P advanced beyond, the performance dipped as the resulting basis vectors begin to bleed into each other.

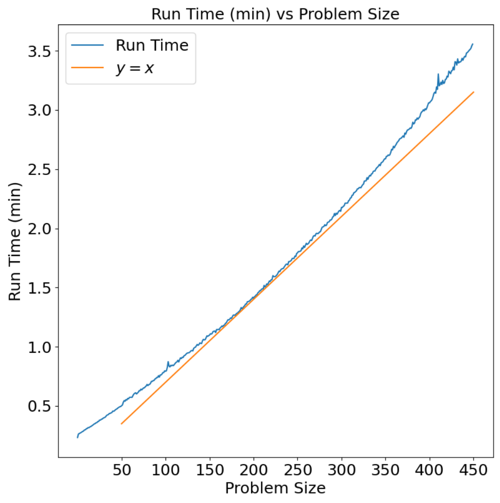


Figure 5: As the number of samples increased, the run time appeared to increase polynomially (blue). A linear line (orange) with a slope determined from the first and final run time is shown for comparison.

3.2 Results on Benchmark Data

SSC was implemented using 200 sample from both digit 1 and digit 5 from MNIST's handwritten digits data set [3]. A sweep across λ and P revealed a local minima for λ while increasing P decreased performance. As P increased, the proportion of number of samples compared to the number of coefficients lowered, producing a harder problem to solve. The performance did not match what was expected. SSC was able to differentiate between many faces while our implementation failed to completely differentiate two numbers. In [1], it was known that the faces has subspaces in $\mathbb{R}^9$. For our implementation, we empirically concluded to subspaces in $\mathbb{R}^5$. Perhaps, we were wrong and missing information that led to incorrect clustering.

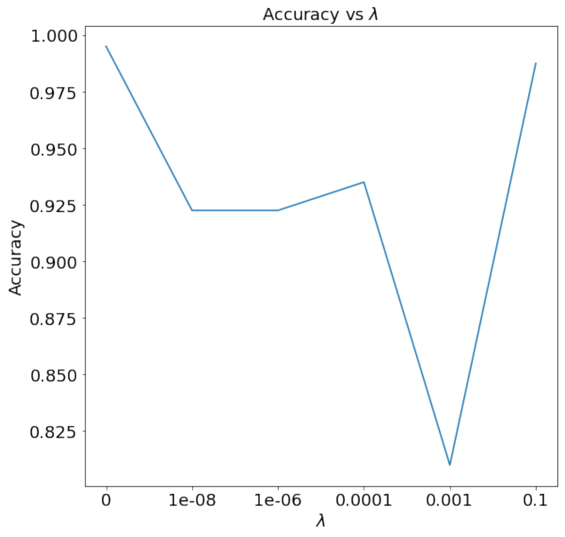


Figure 6: Increasing the weight of the regularizer generally worsened the performance of SSC.

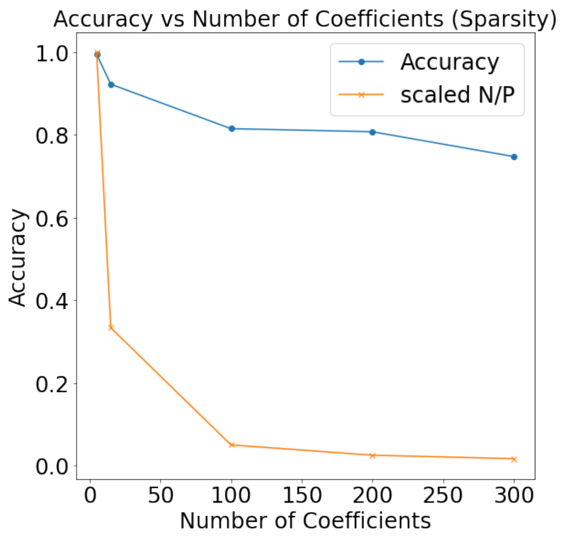


Figure 7: Increasing the number of coefficients per sample shows a decrease in the performance (blue). As the number of coefficients increases, the ratio of coefficients to samples decreases (orange), justifying the worsening accuracy.

References

[1] E. Elhamifar and R. Vidal, "Sparse subspace clustering: Algorithm, theory, and applications," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 2765–2781, Nov. 2013.

[2] J. Lipor, D. Hong, Y. S. Tan, and L. Balzano, "Subspace clustering using ensembles of k-subspaces," *arXiv preprint arXiv:1709.04744*, 2017.

[3] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

A Appendix

The code can be found on GitHub, Google Colab, and below. If hyperlinks don't work, here are the URLs

- `https://github.com/mattsul/EE516Final/tree/main`
- `https://colab.research.google.com/drive/1OMAGztvKcA9csHBR6xI-IQb5W_f-BLia?usp=sharing`

The below plots plot three points: the estimated labels (orange), true labels (black crosses), and the output from spectral clustering (blue). The spectral clustering output numbers clusters in the order it found them. The estimated labels are adjusted such that correctly clustered points should lie underneath a black x. Furthermore, accuracy is computed as a fraction instead of a percentage.

EE516_Final_Project

March 25, 2023

1 Sparse Subspace Clustering

Due March 24, 2023, 11:59PM The general idea is that there are many ways to define points in terms of other points. Another way to say this: there are infinitely many ways to linearly depending points.

This paper introduces Sparse Subspace Clustering (SSC), an algorithm that solves a sparse optimazation using spectral clustering. Beacuse sparse optimization is NP-Hard, the paper asserts a convex relaxation.

Algorithm

1. Input a set of points lying need a union of n-linear subspaces
2. Solve the sparse optimazation program in Eq. 5
3. Normalize the columns, C
4. Form a similarity graph with N nodes representing the data points with the edge weights being $\mathbf{W} = |C| + |C|$.
5. Apply spectral clustering [26] to similarity graph.
6. Output results from Step 5.

1.0.1 mySpectralClustering Function

[7]:
```
import numpy as np
import scipy as sp
import math
from numpy import linalg as lin
from sklearn.cluster import KMeans
from scipy.spatial.distance import squareform, pdist
from scipy.optimize import linear_sum_assignment

def mySpectralClustering(W, K, normalized):
    r"""
    Customized version of Spectral Clustering

    Inputs:
    -------
        W: weighted adjacency matrix of size N x N
        K: number of output clusters
        normalized: 1 for normalized Laplacian, 0 for unnormalized

    Outputs:
    -------
        estLabels: estimated cluster labels
        Y: transformed data matrix of size K x N
    """

    degMat = np.diag(np.sum(W, axis=0))
    L = degMat - W
    n_init = 10
    max_iter = 300

    if normalized == 0:
        D, V = lin.eig(L)
        V_real = V.real
        inds = np.argsort(D)
        Y = V_real[:, inds[0:K]].T

        k_means = KMeans(n_clusters=K, n_init=n_init, max_iter=max_iter).fit(Y.T)
        estLabels = k_means.labels_
    else:
        # Invert degree matrix
        degInv = np.diag(1.0 / np.diag(degMat))
        Ln = degInv @ L

        # Eigen decomposition
        D, V = lin.eig(Ln)
        V_real = V.real
        inds = np.argsort(D)
        Y = V_real[:, inds[0:K]].T

        k_means = KMeans(n_clusters=K, n_init=n_init, max_iter=max_iter).fit(Y.T)
        estLabels = k_means.labels_

    return estLabels, Y
```

[47]:
```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp

def SparseSubspaceClustering(Xtrain, K, lam=1e-4, P=15, normalize=0):
    r"""
    Sparse Subspace Clutering with P Coefficients

    Inputs:
    -------
```

```
        Xtrain: (D x N) Data matrix where the columns are samples of␣
↪D-dimensional data
        K: number of output clusters for spectral clustering
        lam: weight of regularization term
        P: number of coefficients used to reconstruct a data point
        normalized: 1 for normalized Laplacian for spectral clustering, 0 for␣
↪unnormalized
    Outputs:
    --------
        estLabels: estimated cluster labels
        Y: transformed data matrix of size K x N
    """

    N, D = Xtrain.shape
    C = np.zeros((D, D))
    for ii, y in enumerate(Xtrain.T):
        ### Documentation here: https://www.cvxpy.org/
        Xtrain_no_y = np.delete(Xtrain, ii, axis=1)
        c = cp.Variable(D-1)
        prob = cp.Problem(cp.Minimize(cp.norm(c) + lam*cp.norm(y - Xtrain_no_y @␣
↪c)), [cp.sum(c) == 1])
        result = prob.solve()
        c_values = np.insert(c.value, ii, 0)
        ## Sparsify manually. Really a sad state of affairs.
        idx = np.argpartition(c_values, -P)[-P:]
        new_c = np.zeros(D)
        new_c[idx] = c_values[idx]

        C[ii,:] = new_c
        C[:,ii] = new_c

    C = np.nan_to_num(C)
    for ii in range(D):
        C[:,ii] = C[:,ii] / np.linalg.norm(C[:,ii], ord=np.inf)

    W = abs(C) + abs(C.T)
    W = np.nan_to_num(W)

    estLabels, Y = mySpectralClustering(W, K, normalize)
    return estLabels, Y
```

1.0.2 Synthetic Experiment

Generate synthetic data from two orthogonal bases [1,1,0] and [1,0,1].

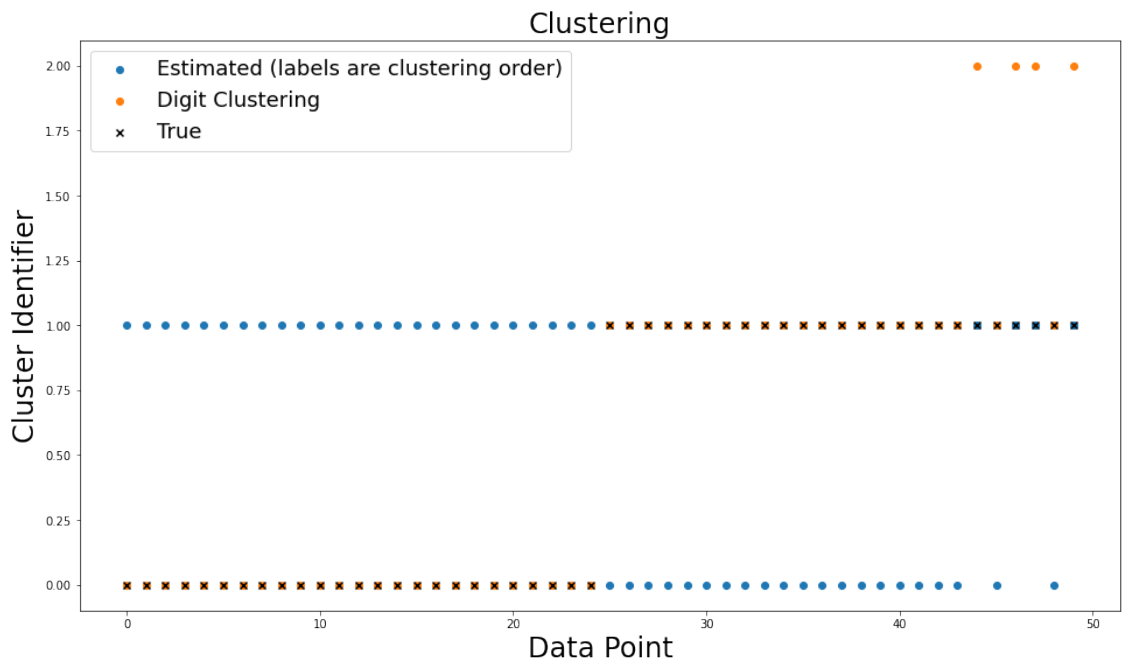
[66]:
```
import numpy as np
```

```
N = 50
D = 3
half = int(N/2)
x_train = np.zeros((D, N))
y_train = np.zeros(N)
y_train[half:] = 1

for ii in range(N):
    if ii < int(N/2):
        x_train[:,ii] = np.array([2*np.random.randint(0, 10), 2*np.random.
 ↪randint(0, 10), 0])
    else:
        x_train[:,ii] = np.array([2*np.random.randint(0, 10), 0, 2*np.random.
 ↪randint(0, 10)])

## Comment out when running tests. Helps makes things run faster when␣
 ↪demonstrating.
x_train[:,0] = [1,1,0]
x_train[:,-1] = [1,0,1]
```

[92]:
```
K = 2
P = 50
lam = 1e-2

estLabels, Y = SparseSubspaceClustering(x_train, K, lam, P, normalize=0)

## This adjusts so the clusters are assigned to the digits. The cluster numbers␣
 ↪are assigned based off of which clustered first.
counts = np.bincount(estLabels[:half])
clust0 = np.argmax(counts)

counts = np.bincount(estLabels[half:])
clust1 = np.argmax(counts)

newEstLabels = np.concatenate([(estLabels[:half] - clust0), (estLabels[half:] -␣
 ↪clust1 + 1)])

acc0 = sum((estLabels[:half] - clust0) == y_train[:half])
acc1 = sum((estLabels[half:] - clust1 + 1) == y_train[half:])
tot_acc = (acc0 + acc1) / N
print('Total Accuracy:', tot_acc)

plt.figure(figsize=(16, 9))
plt.title("Clustering", fontsize=f)
plt.scatter(np.arange(0, N, 1), estLabels, label='Estimated (labels are␣
 ↪clustering order)')
plt.scatter(np.arange(0, N, 1), newEstLabels, label='Digit Clustering')
plt.scatter(np.arange(0, N, 1), y_train, marker='x', c='k', label='True')
plt.xlabel("Data Point", fontsize=f)
plt.ylabel("Cluster Identifier", fontsize=f)
plt.legend(fontsize=18)
```

```
Total Accuracy: 0.92
```

[92]:
```
<matplotlib.legend.Legend at 0x7f80c209b2e0>
```



1.0.3 Real World Experiment: MNIST Dataset

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
import cvxpy as cp
import time

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

The below line of code shows theres 60,000 images with 28x28 dimensions. I am only going to get N samples from the digits 0-3.

```
[ ]: print(x_train.shape)
     print(y_train.shape)

     N = 200
     idx_0s = np.where(y_train == 0)[0][:N]
     idx_1s = np.where(y_train == 1)[0][:N]
     idx_2s = np.where(y_train == 2)[0][:N]
     idx_3s = np.where(y_train == 3)[0][:N]
     idx_5s = np.where(y_train == 5)[0][:N]
     subsampled_indices = np.concatenate([idx_0s, idx_1s, idx_2s, idx_3s])

     # ## Using only 0s, 1s, and 2s
     # It does not like the digit 1.
     subsampled_indices = np.concatenate([idx_0s, idx_2s, idx_5s])
     subsampled_indices = np.concatenate([idx_0s, idx_5s])


     Xtrain = x_train[subsampled_indices, :, :].reshape(len(subsampled_indices),␣
      ↪28*28).T
     Ytrain = y_train[subsampled_indices]

     X0 = x_train[idx_0s, :, :].reshape(N, 28*28)
     X1 = x_train[idx_1s, :, :].reshape(N, 28*28)
     X2 = x_train[idx_2s, :, :].reshape(N, 28*28)
     X3 = x_train[idx_3s, :, :].reshape(N, 28*28)

     print(Xtrain.shape)
     print(Ytrain.shape)
```

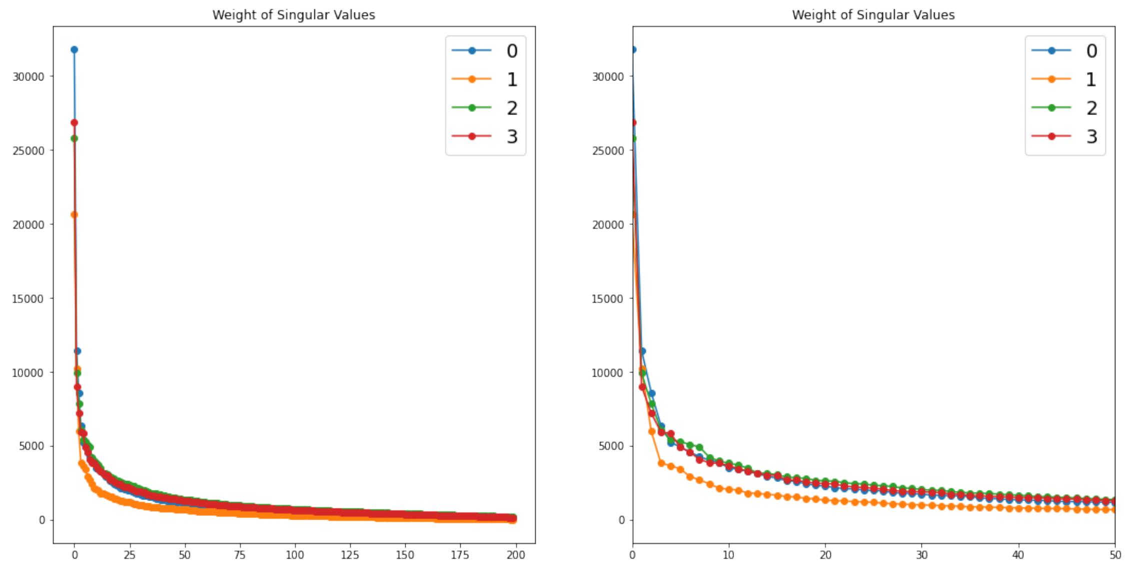
```
(60000, 28, 28)
(60000,)
(784, 400)
(400,)
```

Section 7.2 seems to suggest that the “knee” occurs when the singular values start tapering off. It appears to me that the “knee” occurs at 10-15 singular values. Maybe even 20.

The images can be modeled as corrupted data points lying close to a union of 10-20-dimensional subspaces. We will see about that. It might just be like 5 with a bunch of unused subspaces.

```
[ ]: _, s0, _ = np.linalg.svd(X0)
     _, s1, _ = np.linalg.svd(X1)
     _, s2, _ = np.linalg.svd(X2)
     _, s3, _ = np.linalg.svd(X3)

     plt.figure(figsize=(18,9))
     plt.subplot(1, 2, 1)
     plt.title('Weight of Singular Values')
     plt.plot(s0, marker='o', label='0')
     plt.plot(s1, marker='o', label='1')
     plt.plot(s2, marker='o', label='2')
     plt.plot(s3, marker='o', label='3')
     plt.legend(fontsize=18)
     plt.subplot(1, 2, 2)
     plt.title('Weight of Singular Values')
     plt.plot(s0, marker='o', label='0')
     plt.plot(s1, marker='o', label='1')
     plt.plot(s2, marker='o', label='2')
     plt.plot(s3, marker='o', label='3')
     plt.xlim(0, 50)
     plt.legend(fontsize=18)
```

```
[ ]: <matplotlib.legend.Legend at 0x7f8d4cd1a7c0>
```



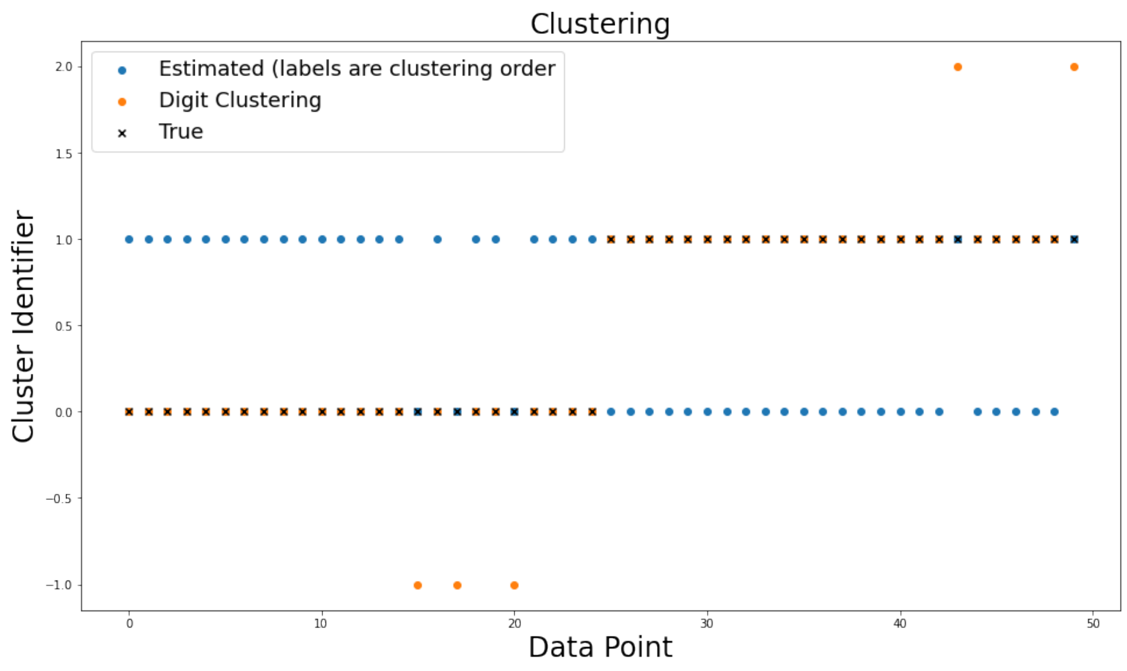
Sparse Subspace Clustering MNIST Keep about 15 coefficients.

```
[91]: K = 2
P = 15
lam = 1e-6

estLabels, Y = SparseSubspaceClustering(Xtrain, K, lam, P, normalize=0)

## This adjusts so the clusters are assigned to the digits. The cluster numbers␣
 ↪are assigned based off of which clustered first.
counts = np.bincount(estLabels[:half])
clust0 = np.argmax(counts)
```

```
counts = np.bincount(estLabels[half:])
clust1 = int(not clust0)

newEstLabels = np.concatenate([(estLabels[:half] - clust0), (estLabels[half:] -␣
 ↪clust1 + 1)])

acc0 = sum((estLabels[:half] - clust0) == y_train[:half])
acc1 = sum((estLabels[half:] - clust1 + 1) == y_train[half:])
tot_acc = (acc0 + acc1) / N
print('Total Accuracy:', tot_acc)

plt.figure(figsize=(16, 9))
plt.title("Clustering", fontsize=f)
plt.scatter(np.arange(0, N, 1), estLabels, label='Estimated (labels are␣
 ↪clustering order)')
plt.scatter(np.arange(0, N, 1), newEstLabels, label='Digit Clustering')
plt.scatter(np.arange(0, N, 1), y_train, marker='x', c='k', label='True')
plt.xlabel("Data Point", fontsize=f)
plt.ylabel("Cluster Identifier", fontsize=f)
plt.legend(fontsize=18)
```

```
Total Accuracy: 0.9
```

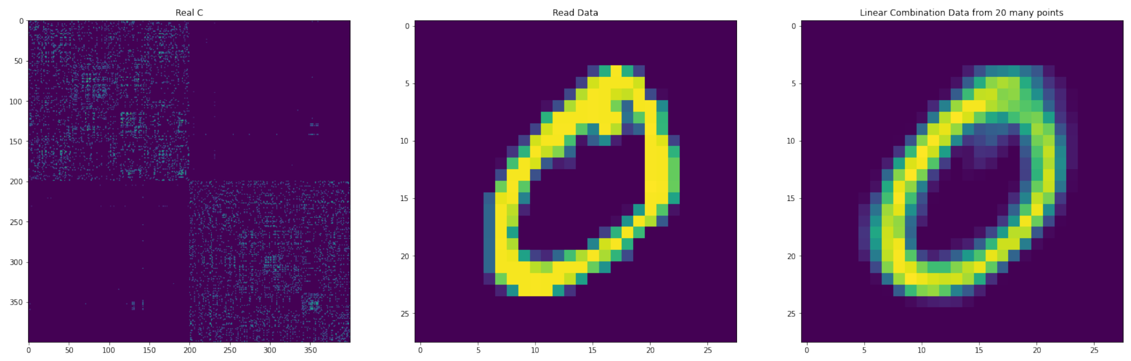
```
[91]: <matplotlib.legend.Legend at 0x7f80c2127970>
```



1.0.4 Development Cells.

```
[ ]: Xhat = Xtrain @ C
     ## Visualize for successs
     plt.figure(figsize=(27, 9))
     plt.subplot(1, 3, 1)
     plt.title("Real C")
     plt.imshow(C)
     plt.subplot(1, 3, 2)
     plt.title("Read Data")
     plt.imshow(Xtrain[:,0].reshape(28,28))
     plt.subplot(1, 3, 3)
     plt.title("Linear Combination Data from " + str(P) + ' many points')
     plt.imshow(Xhat[:,0].reshape(28,28))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f8d4d316400>
```



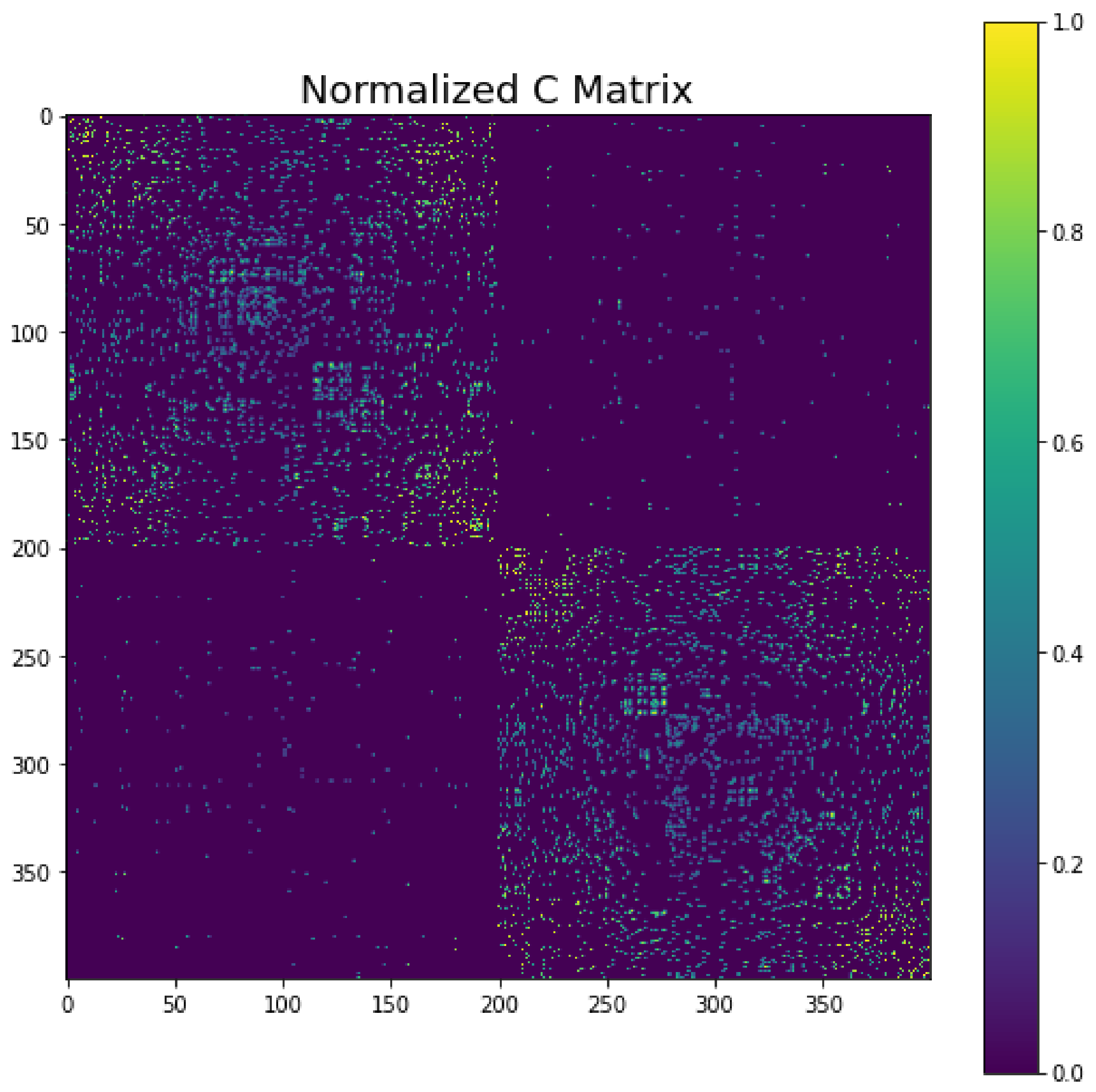
Tolerance Sparsifying

```
[ ]: ## This is less than ideal. I think the idea is that there's only like X many␣
      ↪dimensions worth considering
     ## The face example says 9 vectors -- which implies only 9 c values are non-zero␣
      ↪for each data point?
     ## I think we can do the same with the largest X many c values -- kind of like␣
      ↪the top X PCA stuff -- which I think is 10-20 from the knee of the spree plot
     tol = 1e-3
     # idx = np.where(C < tol)[0] ## doesn;t work for 2D matrices
     C_tol = C.copy()
     C_tol[ C_tol < tol] = 0
     Xhat = Xtrain @ C_tol

     plt.figure(figsize=(16,16))
     plt.subplot(2, 2, 1)
     plt.title("Fake C")
     plt.imshow(C)
plt.subplot(2, 2, 2)
plt.imshow(Xhat[:,0].reshape(28,28))


plt.subplot(2, 2, 3)
plt.title("Real C")
plt.imshow(C)
plt.subplot(2, 2, 4)
plt.imshow(Xhat[:,0].reshape(28,28))
```

####Step 2: Do $\frac{c_i}{\|c_i\|_\infty}$ßC

[ ]:
```
N, D = C.shape
for ii in range(D):
    C[:,ii] = C[:,ii] / np.linalg.norm(C[:,ii], ord=np.inf)

f= 18
plt.figure(figsize=(9,9))
plt.title("Normalized C Matrix", fontsize=f)
plt.imshow(C)
plt.colorbar()
```

[ ]:
```
<matplotlib.colorbar.Colorbar at 0x7fc6a1fc2c10>
```



Step 3: Similarity Graph This is where things might get a little tricky. I do not see the purpose of actually forming a graph. I am going to call back to Demo 1. We did spectral clustering there. We will do it again here. Instead of an actual graph, we will use the Laplacian.

[ ]:
```
W = abs(C) + abs(C.T)
W = np.nan_to_num(W)
# print(np.argwhere(np.isnan(W)))
# np.count_nonzero(~np.isnan(W[:,208]))
```

Step 4: Spectral Cluster I am going to copy-paste John's Demo Solution code.

```
[ ]: import numpy as np
     import scipy as sp
     import math
     from numpy import linalg as lin
     from sklearn.cluster import KMeans
     from scipy.spatial.distance import squareform, pdist
     from scipy.optimize import linear_sum_assignment

     def mySpectralClustering(W, K, normalized):
         r"""
         Customized version of Spectral Clustering

         Inputs:
         -------
             W: weighted adjacency matrix of size N x N
             K: number of output clusters
             normalized: 1 for normalized Laplacian, 0 for unnormalized

         Outputs:
         -------
             estLabels: estimated cluster labels
             Y: transformed data matrix of size K x N
         """

         degMat = np.diag(np.sum(W, axis=0))
         L = degMat - W
         n_init = 10
         max_iter = 300

         if normalized == 0:
             D, V = lin.eig(L)
             V_real = V.real
             inds = np.argsort(D)
             Y = V_real[:, inds[0:K]].T

             k_means = KMeans(n_clusters=K, n_init=n_init, max_iter=max_iter).fit(Y.T)
             estLabels = k_means.labels_
         else:
             # Invert degree matrix
             degInv = np.diag(1.0 / np.diag(degMat))
             Ln = degInv @ L

             # Eigen decomposition
             D, V = lin.eig(Ln)
             V_real = V.real
             inds = np.argsort(D)
             Y = V_real[:, inds[0:K]].T
```

```
        k_means = KMeans(n_clusters=K, n_init=n_init, max_iter=max_iter).fit(Y.T)
        estLabels = k_means.labels_

    return estLabels, Y
```

[ ]:
```
def train(Xtrain, lam):
    # lam = 1e-4
    P = 15

    N, D = Xtrain.shape
    C = np.zeros((D, D))
    for ii, y in enumerate(Xtrain.T):
        ### Documentation here: https://www.cvxpy.org/
        Xtrain_no_y = np.delete(Xtrain, ii, axis=1)
        c = cp.Variable(D-1)
        prob = cp.Problem(cp.Minimize(cp.norm(c) + lam*cp.norm(y - Xtrain_no_y @␣
 ↪c)), [cp.sum(c) == 1])
        result = prob.solve()
        c_values = np.insert(c.value, ii, 0)

        ## Sparsify manually. Really a sad state of affairs.
        idx = np.argpartition(c_values, -P)[-P:]
        new_c = np.zeros(D)
        new_c[idx] = c_values[idx]

        C[ii,:] = new_c
        C[:,ii] = new_c
    N, D = C.shape
    for ii in range(D):
        C[:,ii] = C[:,ii] / np.linalg.norm(C[:,ii], ord=np.inf)
    W = abs(C) + abs(C.T)
    W = np.nan_to_num(W)
    return W
```

[ ]:
```
K = 2

estLabels, Y = mySpectralClustering(W, K, 0)
# print(f"Classification Accuracy (unnormalized): {(sum(estLabels == Ytrain) /␣
 ↪len(Ytrain)) * 100}%")

# estLabels, Y = mySpectralClustering(W, K, 1)
# print(f"Classification Error (normalized): {sum(estLabels == Ytrain) /␣
 ↪len(Ytrain) * 100}%")

## now pair up the clusters so the error function makes sense
```

```
N = 200
predicted_0 = estLabels[:N]
# predicted_1 = estLabels[N:2*N]
# predicted_2 = estLabels[2*N:3*N]
predicted_5 = estLabels[N+1:]

counts = np.bincount(predicted_0)
clust0 = np.argmax(counts)
accuracy0 = sum((predicted_0 - clust0) == Ytrain[:N])

counts = np.bincount(predicted_5)
clust5 = np.argmax(counts)
accuracy5 = sum((predicted_5 - clust5 + 5) == Ytrain[N+1::])

total_accuracy = (accuracy0 + accuracy5) / (2*N)
print('Accuracy:', total_accuracy)
```

```
Classification Accuracy (unnormalized): 0.75%
Classification Error (normalized): 0.25%
```

[ ]:
```
a = 1
print(int(not a))
```

```
0
```

[ ]:
```
l_vals = [0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7]
accuracy = np.zeros(len(l_vals))
for ii, l in enumerate(l_vals):
    W = train(Xtrain, l)
    estLabels, Y = mySpectralClustering(W, K, 0)

    N = 200
    predicted_0 = estLabels[:N]
    predicted_5 = estLabels[N+1:]

    counts = np.bincount(predicted_0)
    clust0 = np.argmax(counts)
    accuracy0 = sum((predicted_0 - clust0) == Ytrain[:N])

    clust5 = int(not clust0)
    accuracy5 = sum((predicted_5 - clust5 + 5) == Ytrain[N+1::])

    total_accuracy = (accuracy0 + accuracy5) / (2*N)
    accuracy[ii] = total_accuracy
    print('Accuracy:', total_accuracy)
```

```
Accuracy: 0.4975
```

```
<ipython-input-24-712b95ebfabd>:24: RuntimeWarning: invalid value encountered in
true_divide
  C[:,ii] = C[:,ii] / np.linalg.norm(C[:,ii], ord=np.inf)
```

```
Accuracy: 0.4975
```

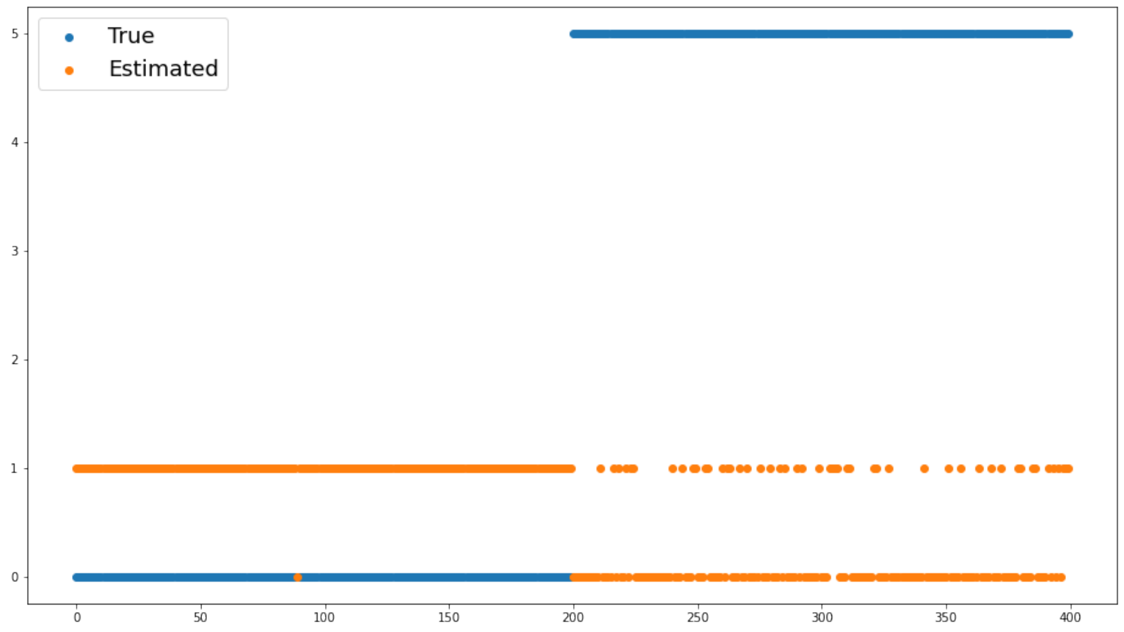
```
        ---------------------------------------------------------------------------

        SolverError                               Traceback (most recent call last)

        <ipython-input-25-f2db86a40dfa> in <module>
          2 accuracy = np.zeros(len(l_vals))
          3 for ii, l in enumerate(l_vals):
    ----> 4     W = train(Xtrain, l)
          5     estLabels, Y = mySpectralClustering(W, K, 0)
          6


        <ipython-input-24-712b95ebfabd> in train(Xtrain, lam)
         10         c = cp.Variable(D-1)
         11         prob = cp.Problem(cp.Minimize(cp.norm(c) + lam*cp.norm(y -␣
 ↪Xtrain_no_y @ c)), [cp.sum(c) == 1])
    ---> 12         result = prob.solve()
         13         c_values = np.insert(c.value, ii, 0)
         14


        /usr/local/lib/python3.9/dist-packages/cvxpy/problems/problem.py in␣
 ↪solve(self, *args, **kwargs)
        491         else:
        492             solve_func = Problem._solve
    --> 493         return solve_func(self, *args, **kwargs)
        494
        495     @classmethod


        /usr/local/lib/python3.9/dist-packages/cvxpy/problems/problem.py in␣
 ↪_solve(self, solver, warm_start, verbose, gp, qcp, requires_grad, enforce_dpp,␣
 ↪ignore_dpp, canon_backend, **kwargs)
       1066         end = time.time()
       1067         self._solve_time = end - start
    -> 1068         self.unpack_results(solution, solving_chain, inverse_data)
       1069         if verbose:
       1070             print(_FOOTER)
        /usr/local/lib/python3.9/dist-packages/cvxpy/problems/problem.py in␣
 ↪unpack_results(self, solution, chain, inverse_data)
       1391             warnings.warn(INF_OR_UNB_MESSAGE)
       1392         if solution.status in s.ERROR:
    -> 1393             raise error.SolverError(
       1394                     "Solver '%s' failed. " % chain.solver.name() +
       1395                     "Try another solver, or solve with verbose=True␣
 ↪for more "


        SolverError: Solver 'ECOS' failed. Try another solver, or solve with␣
 ↪verbose=True for more information.
```

[ ]:
```python
print(clust0)
print(clust5)
print(estLabels)
```

```
0
1
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[ ]:
```python
## Manually Compute Accuracy
t = np.arange(0, len(Ytrain), 1)
plt.figure(figsize=(16, 9))
plt.scatter(t, Ytrain, label='True')
plt.scatter(t, estLabels, label='Estimated')
plt.legend(fontsize=f)
```

[ ]:
```
<matplotlib.legend.Legend at 0x7fc69dd732e0>
```



```
[ ]: ## now pair up the clusters so the error function makes sense
     N = 200
     predicted_0 = estLabels[:N]
     # predicted_1 = estLabels[N:2*N]
     # predicted_2 = estLabels[2*N:3*N]
     predicted_5 = estLabels[N+1:]

     counts = np.bincount(predicted_0)
     clust0 = np.argmax(counts)
     accuracy0 = sum((predicted_0 - clust0) == Ytrain[:N])

     counts = np.bincount(predicted_5)
     clust5 = np.argmax(counts)
     accuracy5 = sum((predicted_5 - clust5 + 5) == Ytrain[N+1::])

     total_accuracy = (accuracy0 + accuracy5) / (2*N)
     print('Accuracy:', total_accuracy)
```

```
Accuracy: 0.8725
```

```
[ ]: acc = [0.8675, 0.9625,  0.9625,  0.9625,  0.9625,  0.9805, 0.9625,  ]
```