Machine Learning
Assignment 2

Darshan Agarwal
201225189

Link to video:
http://researchweb.iiit.ac.in/~darshan.agarwal/b-1.ogv

Problem Setting

Handwritten digit recognition is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The applications of the handwritten digit recognition include automatically address reading and mail routing, postal mail sorting, bank check processing, etc. In this assignment, we will be tweaking the feature representation as well as the classifier to see their impact on performance in solving handwritten digit recognition problems.

Problem Space

Given an unseen image of a handwritten digit, assign relevant label to the image. The input consists of white or black pixels, the digits are usually well-separated from the background, and there are only ten output categories. The problem deals with objects in a real two-dimensional space and the mapping from image space to category space has both considerable regularity and considerable complexity. This problem is of great interest in the pattern recognition research community because of its applicability to many fields towards more convenient input devices and more efficient data organization and processing. The problem in the recognition of handwritten digits is that of within class variance since a same digit can be written in multiple ways. There has been much research in this field where different approaches of feature extraction have been implemented in an attempt to improve the discrimination ability. It is seen from the experiments that extraction of features in terms of direction, local structure and curvature improves the accuracy of classification.

Dataset

We have used MNIST and USPS datasets for the assignment. MNIST has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from

NIST. The digits have been size-normalized and centered in a fixed-size image. USPS has a training set of 4649 images and a test set of 4649 examples.

- **STAGE 1**

In this stage, the raw data samples in original dimensions were taken from both the datasets and classification performance was obtained for the following classifiers :

I. **K Nearest neighbour :** In k-Nearest Neighbour classification, the output is a class membership. The K is used to signify how many votes are used for decision making. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors. *k* is a positive integer. The optimal value of K to choose is odd so that it eliminates a tie between two sets.

II. **SVM :** A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. The optimal separating hyperplane *maximizes* the margin of the training data. SVM commonly used with linear, polynomial, RBF and sigmoid kernels. We have used a multiclass SVM classification (libsvm) in our model with different kernels of 1) linear 2) RBF 3)Chi Square 4) Polynomial

   A. **RBF Kernel :** The Radial Basis Function kernel on two samples x and x', represented as feature vectors in some *input space*, is defined as

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

   B. **Polynomial Kernel :** For degree-*d* polynomials, the polynomial kernel is defined as where *x* and *y* are vectors in the *input space*, i.e. vectors of features computed from training or test samples and $c \geq 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial.

$$K(x, y) = (x^{\mathsf{T}} y + c)^d$$

   C. **Linear Kernel :** For all objects $u, v \in X$, where X is a set of training examples.
   $K(u,v) = f(u).f(v) = f_1(u)f_1(v) + f_2(u)f_2(v)$

Confusion Matrices for MNIST and USPS datasets

1. **Confusion Matrices for Linear Kernel**

USPS dataset Accuracy : 95.6

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	769	0	4	2	3	4	0	1	3	0
1	0	644	0	0	0	0	2	0	1	0
2	4	1	430	4	6	3	1	1	3	1
3	5	0	11	385	2	8	0	1	6	0
4	4	3	6	0	421	3	1	2	0	3
5	2	0	4	10	3	328	5	0	1	2
6	4	0	2	0	3	3	402	0	0	0
7	0	0	0	0	3	1	0	389	2	7
8	2	1	5	7	3	2	1	2	308	0
9	0	0	3	1	5	3	0	8	0	379

MNIST dataset Accuracy : 91.73

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	964	0	1	2	1	2	5	3	1	1
1	0	1115	3	2	0	1	4	1	8	1
2	8	4	929	16	7	6	13	11	35	3
3	5	0	21	919	2	21	3	11	20	8
4	1	5	3	1	913	0	11	5	5	38
5	8	3	0	34	13	780	19	5	24	6
6	8	3	4	2	7	15	915	0	4	0
7	2	9	22	4	9	1	1	950	4	26
8	9	9	6	17	13	27	10	12	860	11
9	8	8	1	15	37	12	1	19	10	898

2. Confusion Matrices for Polynomial Kernel

USPS dataset Accuracy : 97.16

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	623	163	0	0	0	0	0	0	0	0
1	0	647	0	0	0	0	0	0	0	0
2	18	305	130	0	0	0	0	0	1	0
3	32	308	8	70	0	0	0	0	0	0
4	7	429	3	0	0	0	0	0	0	4
5	104	241	1	4	0	4	0	0	0	1
6	73	303	1	0	0	0	37	0	0	0
7	0	395	4	0	0	0	0	0	0	3
8	37	259	12	2	0	0	0	0	19	2
9	2	367	0	0	0	0	0	0	0	30

MNIST dataset Accuracy : 97.6300

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	972	0	1	1	0	3	1	0	2	0
1	0	1126	2	1	1	0	3	0	2	0
2	8	0	1006	0	2	0	5	8	3	0
3	0	2	1	987	0	6	0	5	6	3
4	2	0	2	0	965	0	3	1	0	9
5	2	0	0	10	1	867	3	1	5	3
6	4	5	1	0	3	6	937	0	2	0
7	0	10	9	2	1	0	0	1000	0	6
8	5	0	1	3	4	4	1	4	950	2
9	3	6	1	5	9	3	1	1	3	977

3. Confusion Matrices for χ^2 kernel

USPS dataset Accuracy : 95.9131

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	786	0	0	0	0	0	0	0	0	0
1	47	600	0	0	0	0	0	0	0	0
2	453	0	1	0	0	0	0	0	0	0
3	418	0	0	0	0	0	0	0	0	0
4	442	1	0	0	0	0	0	0	0	0
5	355	0	0	0	0	0	0	0	0	0
6	358	0	0	0	0	0	56	0	0	0
7	326	0	0	0	0	0	0	76	0	0
8	331	0	0	0	0	0	0	0	0	0
9	274	0	0	0	0	0	0	0	0	125

MNIST Dataset Accuracy : 93.51

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	967	0	1	1	1	6	3	1	0	0
1	0	1125	1	3	0	1	2	1	2	0
2	7	1	978	5	4	6	7	8	14	2
3	2	0	15	946	3	20	0	7	12	5
4	0	1	7	1	945	2	4	0	2	20
5	5	2	1	24	4	825	13	1	14	3
6	8	2	7	1	5	12	920	0	3	0
7	0	6	20	5	7	0	0	966	0	24
8	5	7	7	20	7	16	3	5	897	7
9	5	5	1	7	19	3	0	12	3	954

4. Confusion Matrices for RBF Kernel

USPS dataset Accuracy : 94.49

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	766	0	3	2	5	3	5	0	2	0
1	0	642	0	0	1	0	3	0	1	0
2	4	1	422	7	9	1	4	2	4	0
3	2	1	5	385	2	16	0	0	5	2
4	1	4	6	0	415	0	3	1	1	12
5	3	0	4	10	7	317	7	0	4	3
6	6	0	8	0	4	2	393	0	1	0
7	0	0	0	0	6	1	0	379	1	15
8	3	6	3	9	3	4	1	3	294	5
9	0	1	0	1	5	1	0	9	2	380

MNIST dataset Accuracy : 11.35

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	980	0	0	0	0	0	0	0	0
1	0	1135	0	0	0	0	0	0	0	0
2	0	1032	0	0	0	0	0	0	0	0
3	0	1010	0	0	0	0	0	0	0	0
4	0	982	0	0	0	0	0	0	0	0
5	0	892	0	0	0	0	0	0	0	0
6	0	958	0	0	0	0	0	0	0	0
7	0	1028	0	0	0	0	0	0	0	0
8	0	974	0	0	0	0	0	0	0	0
9	0	1009	0	0	0	0	0	0	0	0

5. Confusion Matrices for LibLinear SVM

USPS dataset Accuracy : 94.49

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	763	0	5	5	1	4	1	0	5	2
1	0	641	0	0	2	0	2	0	2	0
2	3	1	419	10	9	3	1	3	4	1
3	2	0	9	379	1	12	0	7	6	2
4	4	2	9	0	411	2	2	1	4	8
5	1	0	3	6	5	324	8	0	5	3
6	3	0	3	1	3	2	400	0	2	0
7	1	0	0	1	4	1	0	385	0	10
8	6	2	9	5	5	4	2	3	295	0
9	0	0	2	1	4	2	0	13	1	376

MNIST dataset Accuracy : 86.24

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	906	0	15	1	0	3	23	4	25	3
1	0	1123	4	0	0	1	4	1	1	1
2	11	33	866	16	3	2	36	13	45	7
3	5	8	49	806	0	42	7	14	68	11
4	4	7	9	2	836	2	35	9	15	63
5	8	9	9	50	6	637	48	12	106	7
6	4	3	7	0	2	5	935	0	2	0
7	2	19	27	1	2	2	1	930	9	35
8	7	40	15	12	7	27	18	10	823	15
9	9	18	4	10	19	11	0	110	66	762

6. Confusion Matrices for KNN

USPS dataset Accuracy : 96.60

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	782	0	3	1	0	0	0	0	0	0
1	0	646	0	0	0	0	1	0	0	0
2	7	2	434	4	1	1	0	5	0	0
3	1	0	2	408	0	3	0	0	3	1
4	2	3	4	0	428	0	0	1	0	5
5	7	0	2	25	1	318	1	0	0	1
6	6	3	1	0	2	2	400	0	0	0
7	0	2	0	1	4	0	0	390	1	4
8	2	2	2	10	2	5	1	1	304	2
9	0	0	0	1	7	0	0	9	1	381

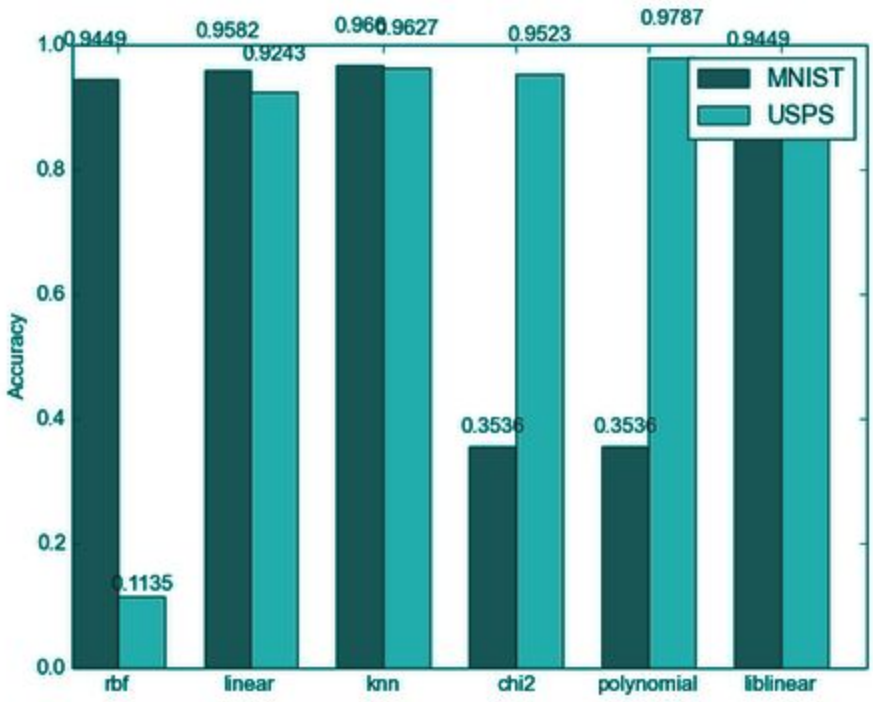
MNIST dataset Accuracy : 96.27

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	976	1	1	0	0	1	0	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	11	10	995	1	2	0	0	12	1	0
3	1	1	8	981	1	9	0	6	2	1
4	3	7	0	0	959	0	2	3	0	8
5	6	2	0	25	2	850	2	1	1	3
6	7	3	0	0	5	4	939	0	0	0
7	0	29	8	2	3	0	0	981	0	5
8	10	2	8	28	9	29	4	5	876	3
9	6	6	3	9	19	4	1	22	2	937

In the case of MNIST data, it is observed that a polynomial kernel gave the best results. It is followed by KNN classifier. SVM (libsvm) using a rbf kernel gave extremely bad results and a

linear SVM using liblinear gave slightly lower accuracy. The relatively high accuracy of linear SVMs compared to rbf can say that the classes are more or less linearly separable except for a few outliers in each class, which were taken care of properly by a polynomial kernel.

In the case of USPS data, it is observed that a knn classifier gave the best results. It is followed by SVM (libsvm) using a linear kernel. SVM (libsvm) using rbf kernel and linear SVM using liblinear gave almost the same accuracy and similar number of misclassified samples. Same as USPS it is seen that, linear SVMs has high accuracy, so we can come to a conclusion that the classes are more or less linearly separable except for a few outliers in each class.



Runtimes For Linear SVM

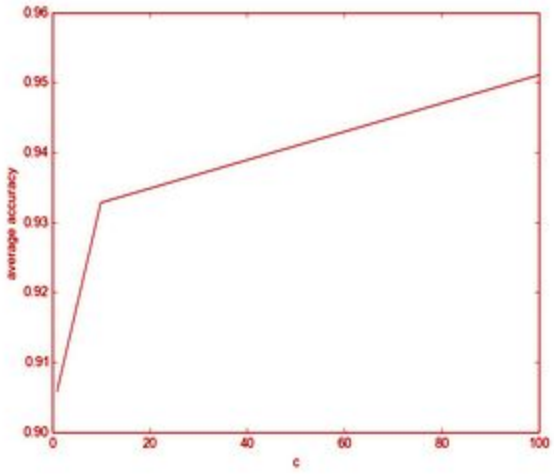
	USPS Training time	MNIST Training time	USPS Testing time	MNIST Testing time
LibLinear	**2.25seconds**	**1min50sec**	**0.5seconds**	**2.1seconds**
LibSvm	**28.0seconds**	**140min0sec**	**24seconds**	**49seconds**

Liblinear was observed to be much faster than libsvm. The main reason behind the different run times for LibLinear and Libsvm is that Libsvm uses a 1 vs 1 strategy while liblinear uses a 1 vs rest strategy.

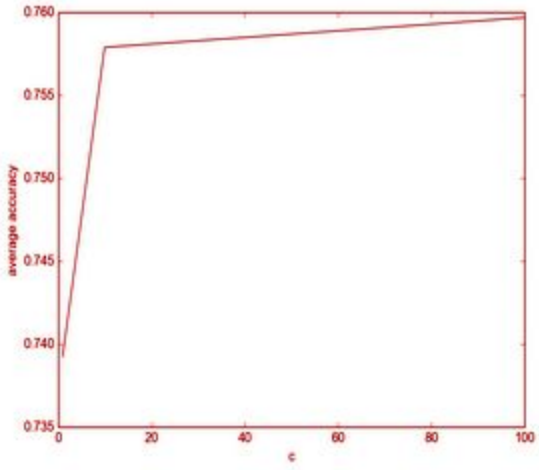
Hyperparameter tweaking

- **Parameter c**

Polynomial Kernel

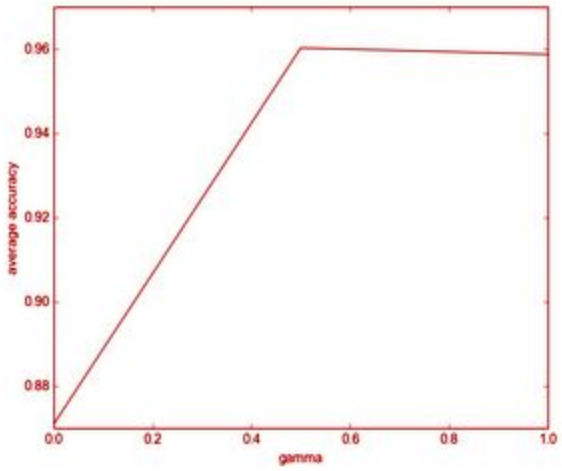


RBF Kernel

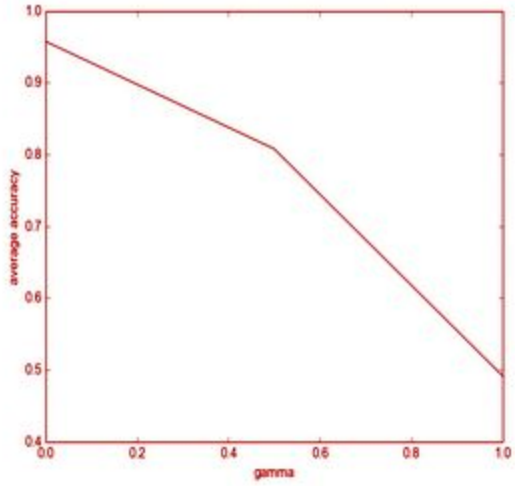


- **Parameter gamma**

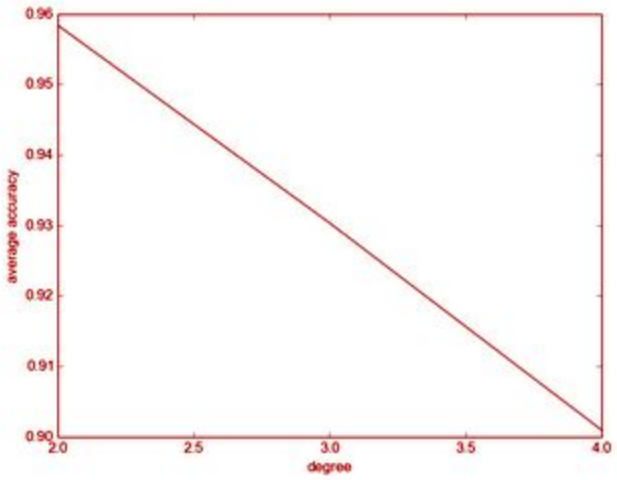
Polynomial Kernel



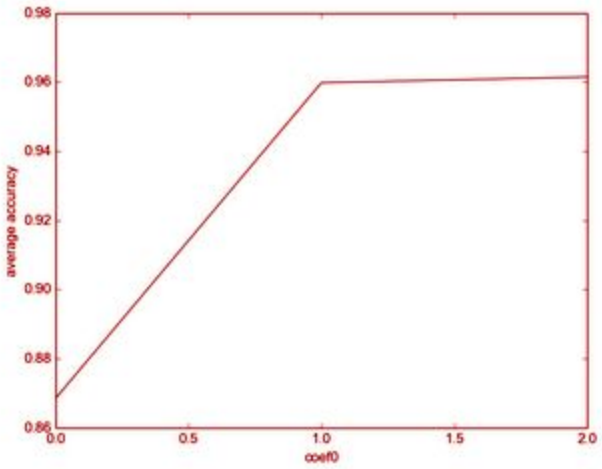
RBF Kernel



- **Parameter <u>degree</u>**



- **Parameter <u>coeff</u>**



In both RBF Kernel and Polynomial kernel, the performance was observed to be increasing in case of the c parameter. The performance decreased with gamma in the case of rbf, while there was no such decrease observed in case of polynomial kernel, it had an optimum as gamma=0.5. One thing to note is that, in case of parameter 'degree' the performance of polynomial kernel degraded linearly indicating the linear separability of data.

- **STAGE 2**

Three different representations chosen are : PCA, sphog and FLD.

1. **Raw Data. Accuracy : 92.55**

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	964	0	8	5	1	8	8	2	9	8
1	0	1115	4	0	5	3	3	9	9	8
2	1	3	929	21	3	0	4	22	6	1
3	2	2	16	919	1	34	2	4	17	15
4	1	0	7	2	913	13	7	9	13	37
5	2	1	6	21	0	780	15	1	27	12
6	5	4	13	3	11	19	915	1	10	1
7	3	1	11	11	5	5	0	950	12	19
8	1	8	35	20	5	24	4	4	860	10
9	1	1	3	8	38	6	0	26	11	898

2. **sphog Accuracy : 98.9**

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	977	0	4	0	1	1	6	0	4	3
1	1	1129	4	0	0	0	1	2	0	3
2	0	2	1016	2	1	1	0	7	2	1
3	0	1	1	1002	0	5	0	2	2	1
4	0	1	1	0	975	0	3	0	2	4
5	1	0	0	3	0	881	1	0	2	6
6	1	0	0	0	2	1	944	0	1	0
7	0	1	5	1	0	0	0	1014	3	5
8	0	1	1	2	1	3	3	1	954	3
9	0	0	0	0	2	0	0	2	4	983

3. **PCA. Accuracy : 9.5**

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	6	3	148	77	71	64	15	52	465
1	0	12	2	176	75	91	78	14	52	507
2	0	9	0	147	76	81	75	17	41	489
3	0	6	0	160	62	99	76	9	25	464
4	0	8	0	168	61	83	78	18	41	431
5	0	6	0	158	54	73	59	15	37	413
6	0	9	0	132	67	100	57	12	56	424
7	0	10	3	164	65	82	72	17	49	479
8	0	13	2	145	61	76	67	15	44	447
9	0	8	3	143	64	109	60	13	50	456

4. **FLD Accuracy : 90.01**

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	948	0	4	2	0	7	12	2	5	0
1	0	1095	4	6	1	0	3	2	24	0
2	13	9	914	23	12	5	18	8	28	2
3	3	2	28	880	1	43	2	16	28	7
4	0	1	6	1	911	1	19	4	7	32
5	13	2	9	45	11	749	15	9	34	5
6	18	3	10	0	10	20	894	0	3	0
7	2	15	20	13	11	0	0	909	3	55
8	12	23	9	32	17	44	15	10	799	13
9	8	3	3	15	60	6	1	37	10	866

The performance of PCA was observed to be very bad with accuracy of only 9.5. The reason for this could be due to the fact that the data was just a collection of pixels and wasn't consisting of

any features as such that were extracted from the images. SPHOG gave extremely good results probably because of the fact that it is scale invariant.

- **STAGE 3**

Isomap is low-dimensional embedding methods, where geodesic distances on a weighted graph are incorporated with the classical scaling. Isomap is used for computing a quasi-isometric, low-dimensional embedding of a set of high-dimensional data points. The algorithm provides a simple method for estimating the intrinsic geometry of a data manifold based on a rough estimate of each data point's neighbors on the manifold. Isomap is one representative of isometric mapping methods, and extends metric multidimensional scaling(MDS) by incorporating the geodesic distances imposed by a weighted graph. Isomap is distinguished by its use of the geodesic distance induced by a neighborhood graph embedded in the classical scaling.This is done to incorporate manifold structure in the resulting embedding.Isomap defines the geodesic distance to be the sum of edge weights along the shortest path between two nodes (computed using Dijkstra's algorithm, for example). The top n eigenvectors of the geodesic distance matrix, represent the coordinates in the new n-dimensional Euclidean space.

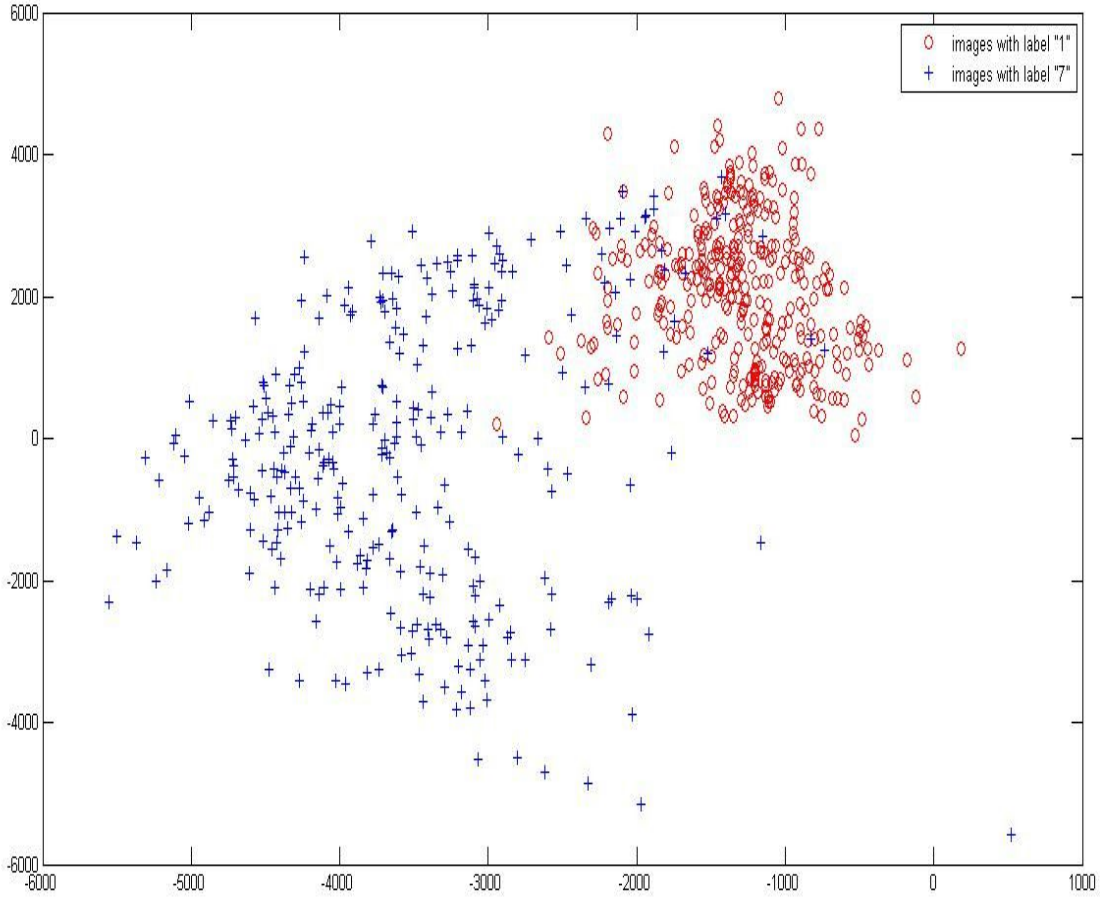
Unlike classical techniques such as principal component analysis (PCA) and multidimensional scaling (MDS), this approach is capable of discovering the non linear degrees of freedom that underlie complex natural observations, such as human handwriting or images of a face under different viewing conditions. In contrast to previous algorithms for nonlinear dimensionality reduction, this algorithm efficiently computes a globally optimal solution, and, for an important class of data manifolds, is guaranteed to converge asymptotically to the true structure.

STEPS

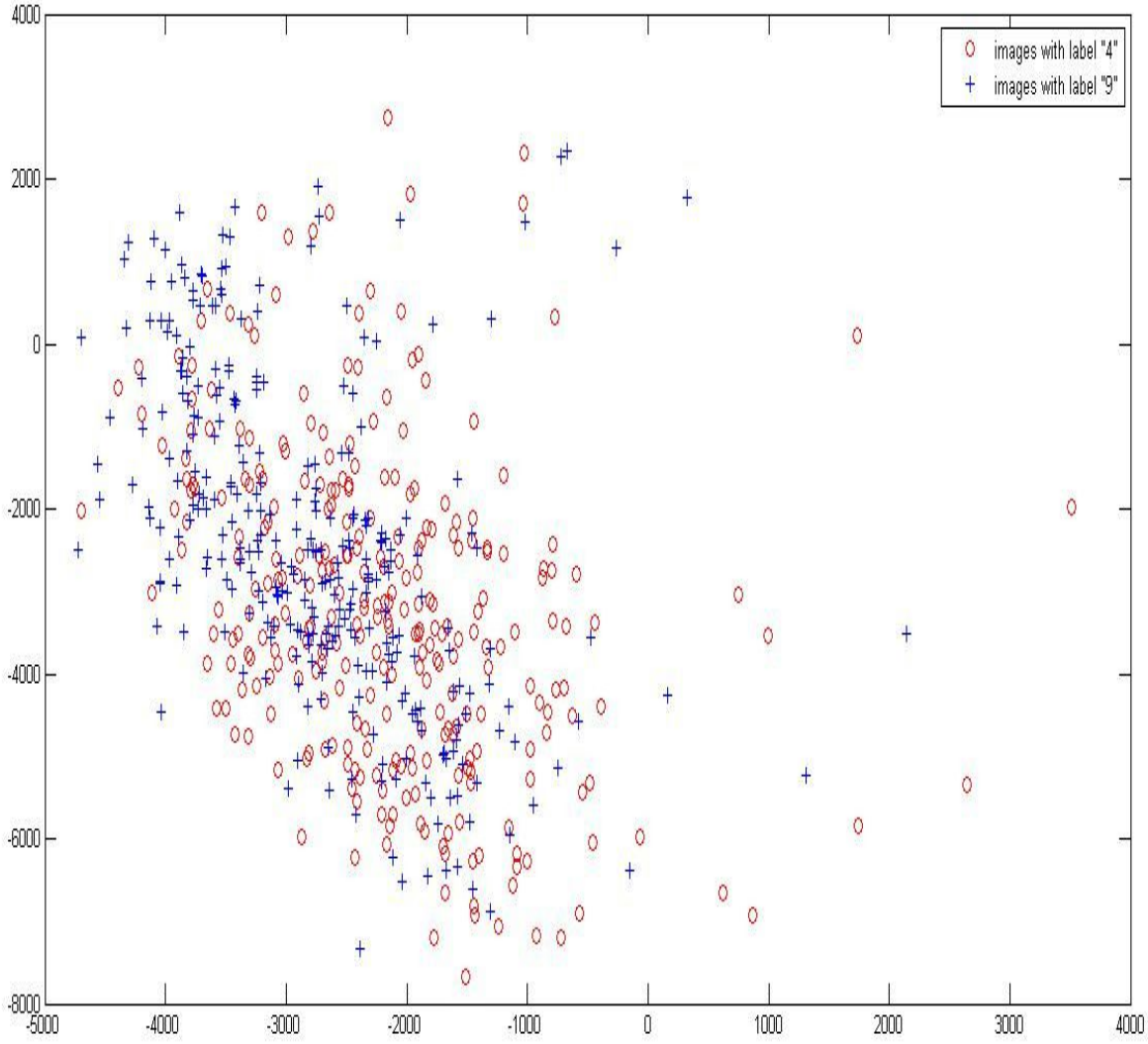
1. **Construct neighborhood graph**
   Define the graph G over all data points by connecting points i and j if [as measured by dX(i, j)] they are closer than e (e-Isomap), or if i is one of the K nearest neighbors of j (K-Isomap). Set edge lengths equal to dX(i, j).
2. **Compute Shortest Paths**
   Initialize dG(i, j) 5 dX(i, j) if i, j are linked by an edge; dG(i,j) 5 ` otherwise. Then for each value of k 5 1, 2,..., N in turn, replace all entries dG(i, j) by min{dG(i, j), dG(i,k) 1 dG(k, j)}. The matrix of Þnal values DG 5 {dG(i, j)} will contain the shortest path distances between all pairs of points in G (16, 19).
   Let lp be the p-th eigenvalue (in decreasing order) of the matrix t(DG) (17), and vp i be the i-th component of the p -th
   eigenvector. Then set the p-th component of the d-dimensional coordinate vector yi equal to=lpvp

2-D Isomap with Euclidean distance

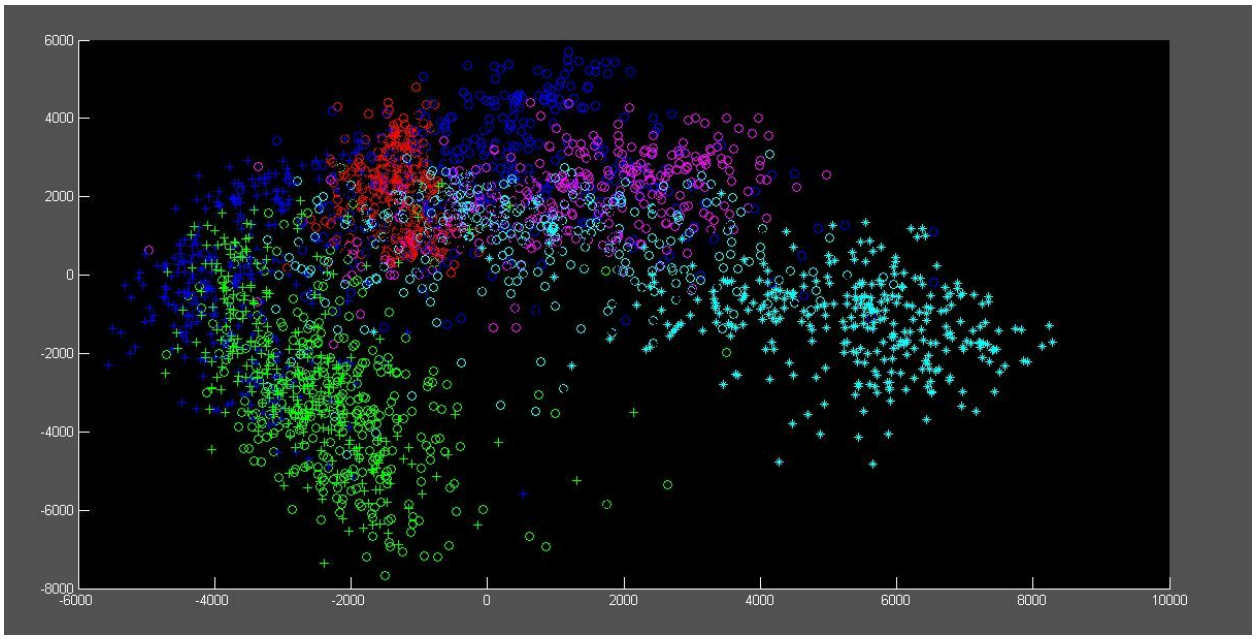
- Projection of 1's and 7's on a 2-D isomap

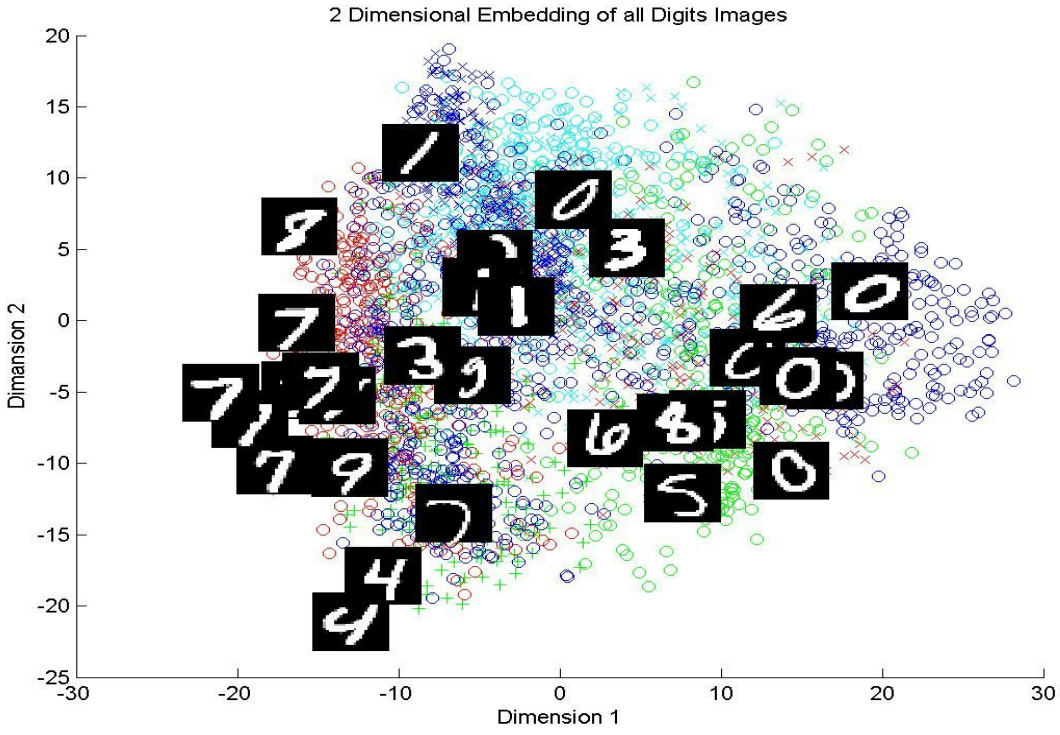


- Projection of 4's and 9's on a 2-D isomap.



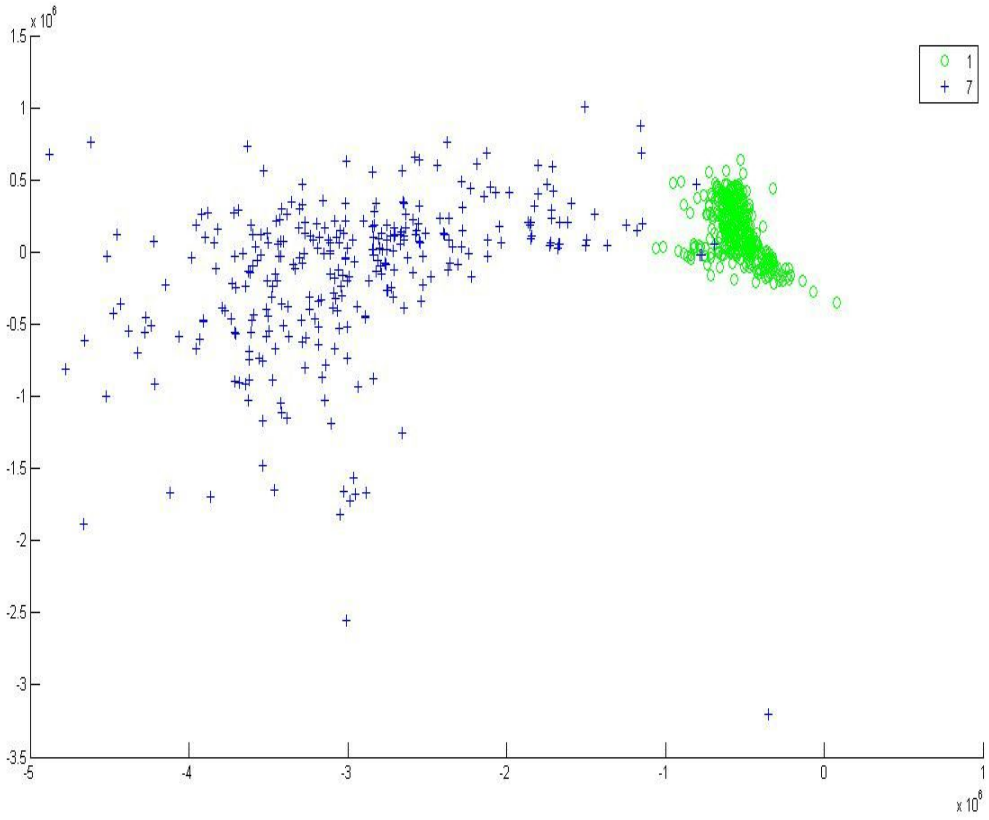
- Projection of all the ten digits on a 2-D isomap. Each of the digits is represented by a different type of color/symbol. (1-Red(circle), 2-blue(circle), 3-pink(circle), 4-green(circle), 5-light blue(circle), 6-yellow(circle), 7-blue(plus), 8-yellow(plus), 9-green(plus), 0-light blue(plus).



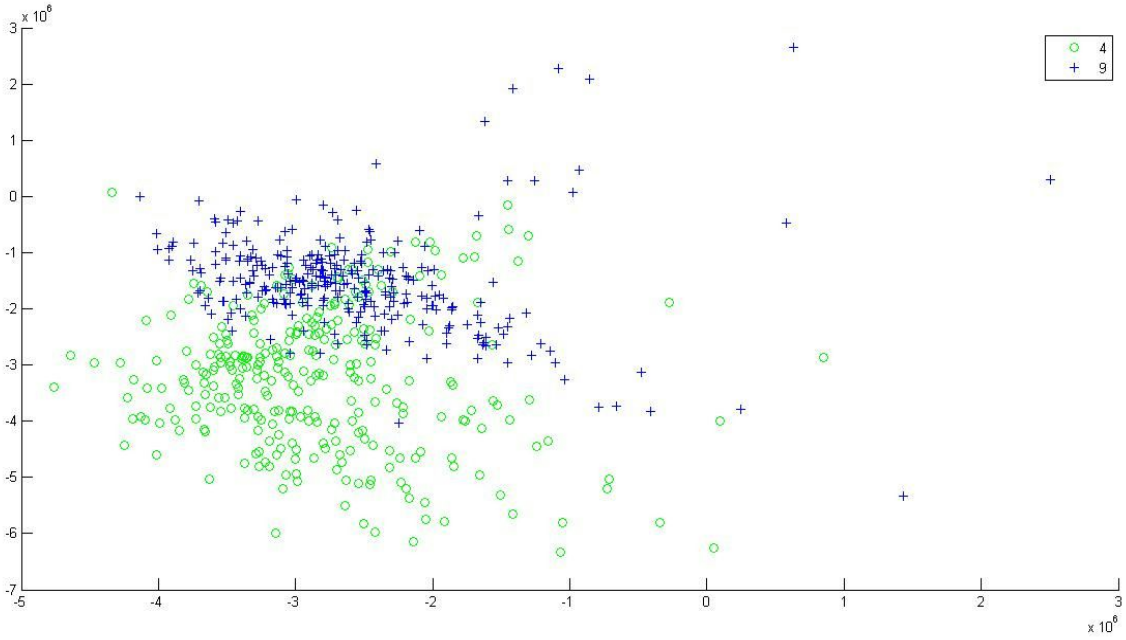


2-D Isomap with Tangent distance

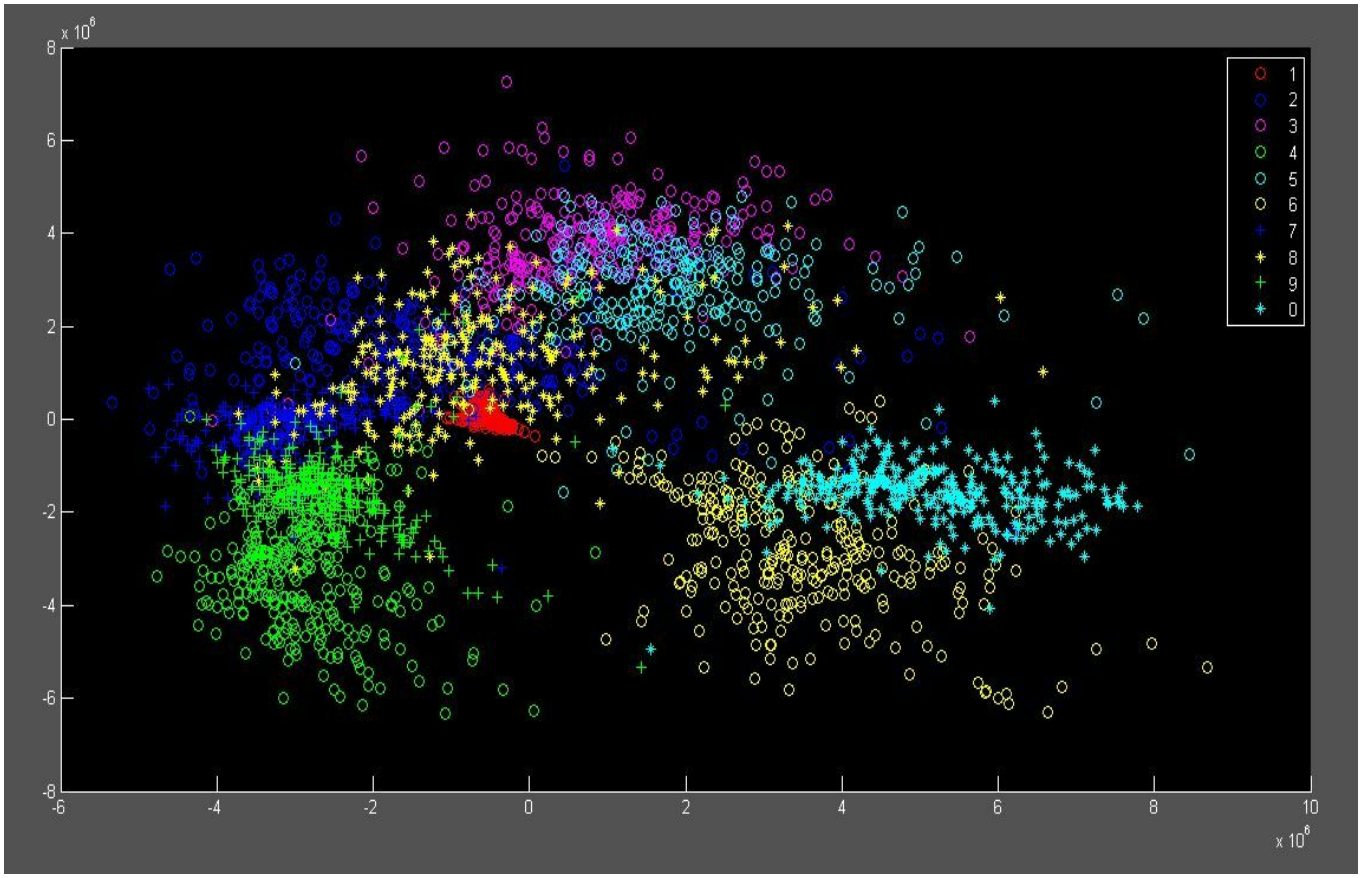
- **Projection of 1 and 7 on a 2-D Isomap**



- **Projection of 4 and 9 on a 2-D Isomap**



- **Projection of all the ten digits on a 2-D isomap**



OBSERVATIONS :

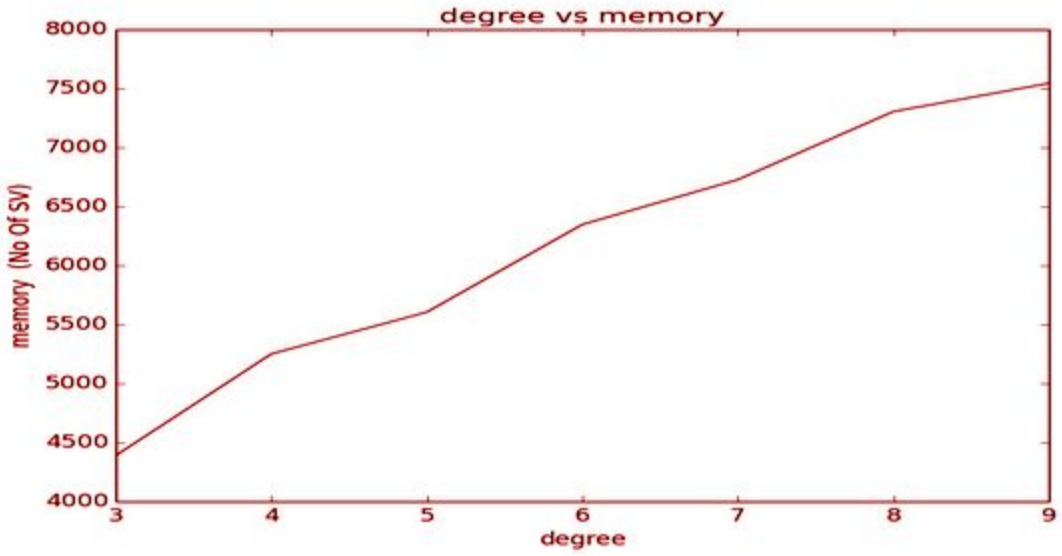
When tangent distances are used, they lead to more distinguishable clustering. This could be easily seen in the case of 4's and 9's which were hardly distinguishable with Euclidian distance but formed distinct clusters when tangent distances were used. In case of "4" and "9", the sample images with higher x coordinate is clearly different from the image with different label, whereas those having lower or negative x coordinate are more similar with the ones with different labels. Tangent distance maps the digits on a larger scale [-150 150] and the variance among instances of same digits is lesser. Although mapping done using euclidean distance is faster than that using tangential distance. Isomap is not able to create much clearer regions for all the digits it seems for more clear boundaries our decision region should be of more dimensions.

Stage 4

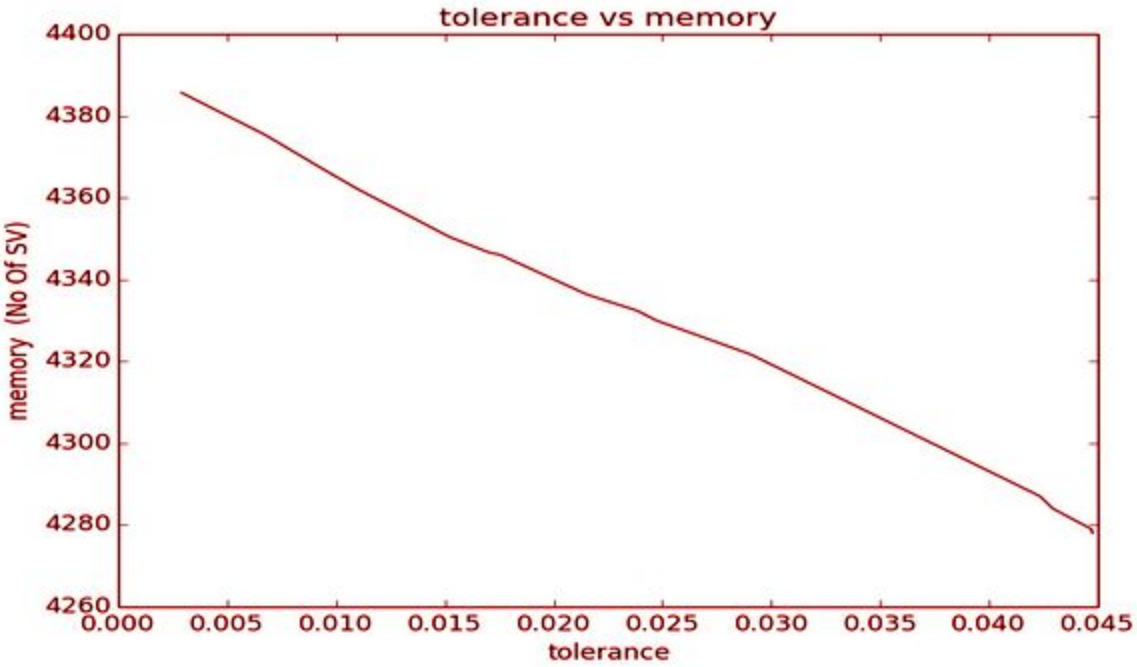
The parameters that affect the computational requirements are:

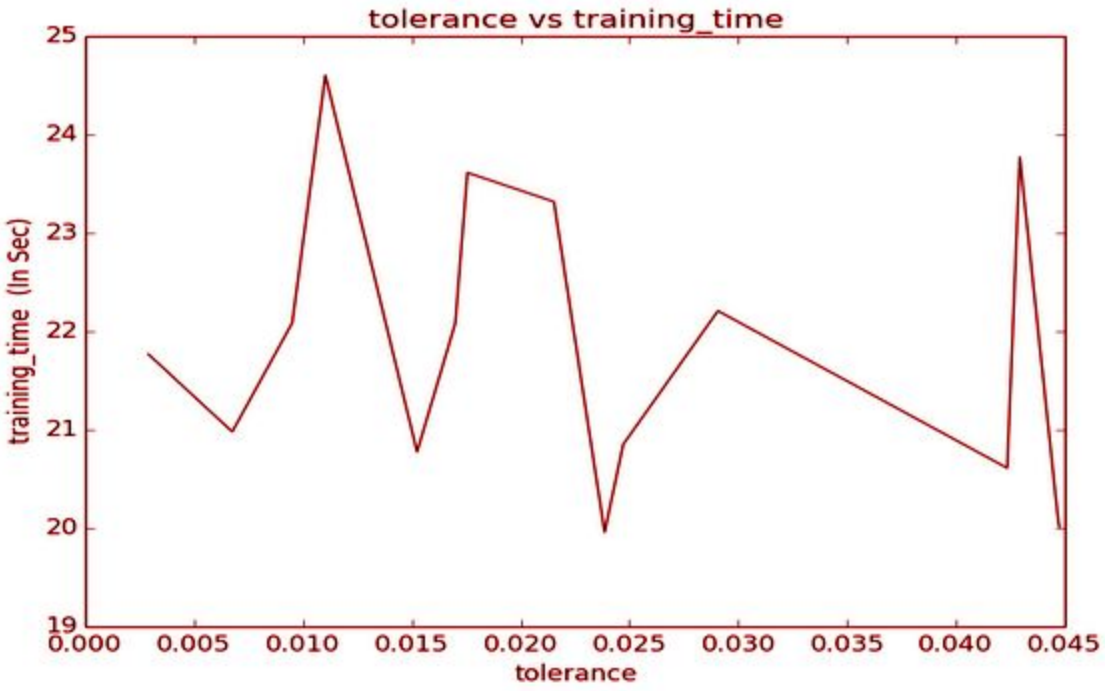
1. Number of samples, their dimensionality and their distribution
2. Number of classes
3. Tolerance (mainly affects memory)
4. Kernel used and its hyperparameters in non linear svm's like the degree of a polynomial kernel
5. Multiclass strategy (1-v-1 or 1-v-rest)

Effect of degree of polynomial on computational requirements



Effect of tolerance on computational requirements





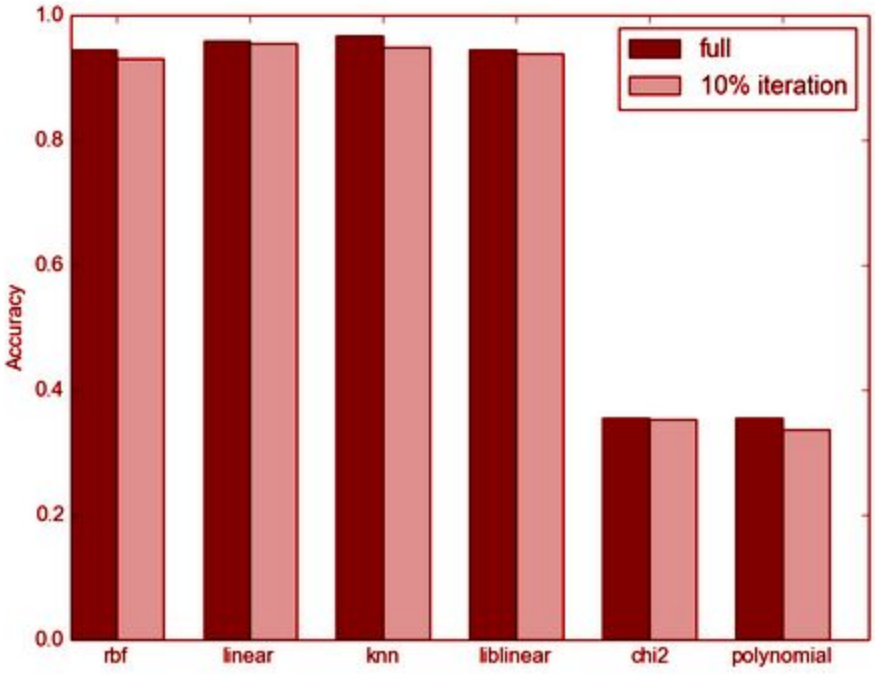
When the following training scheme is used:

(i) Train with only 10% of the training data and test on the rest of the training data.

(ii) If there are any misclassification, add them and retrain.

This training scheme can give a performance close to that training with the full dataset. The support vectors would mostly remain same even if we the samples that aren't support vectors are ignored while training. Hence, this training scheme would end up learning the support vectors in a quicker way as they are very few in number and independent of the number of non support vector samples.

The main **advantage** is that it **saves a lot of time** because we are training only on a small portion of the dataset. The disadvantage of this scheme is that there may not be enough outlier points to minimize the error in decision boundary margin between classes. Also, the distribution of the final data that is considered may be an accurate representation of real world data.



Accuracy of above mentioned training scheme on USPS dataset