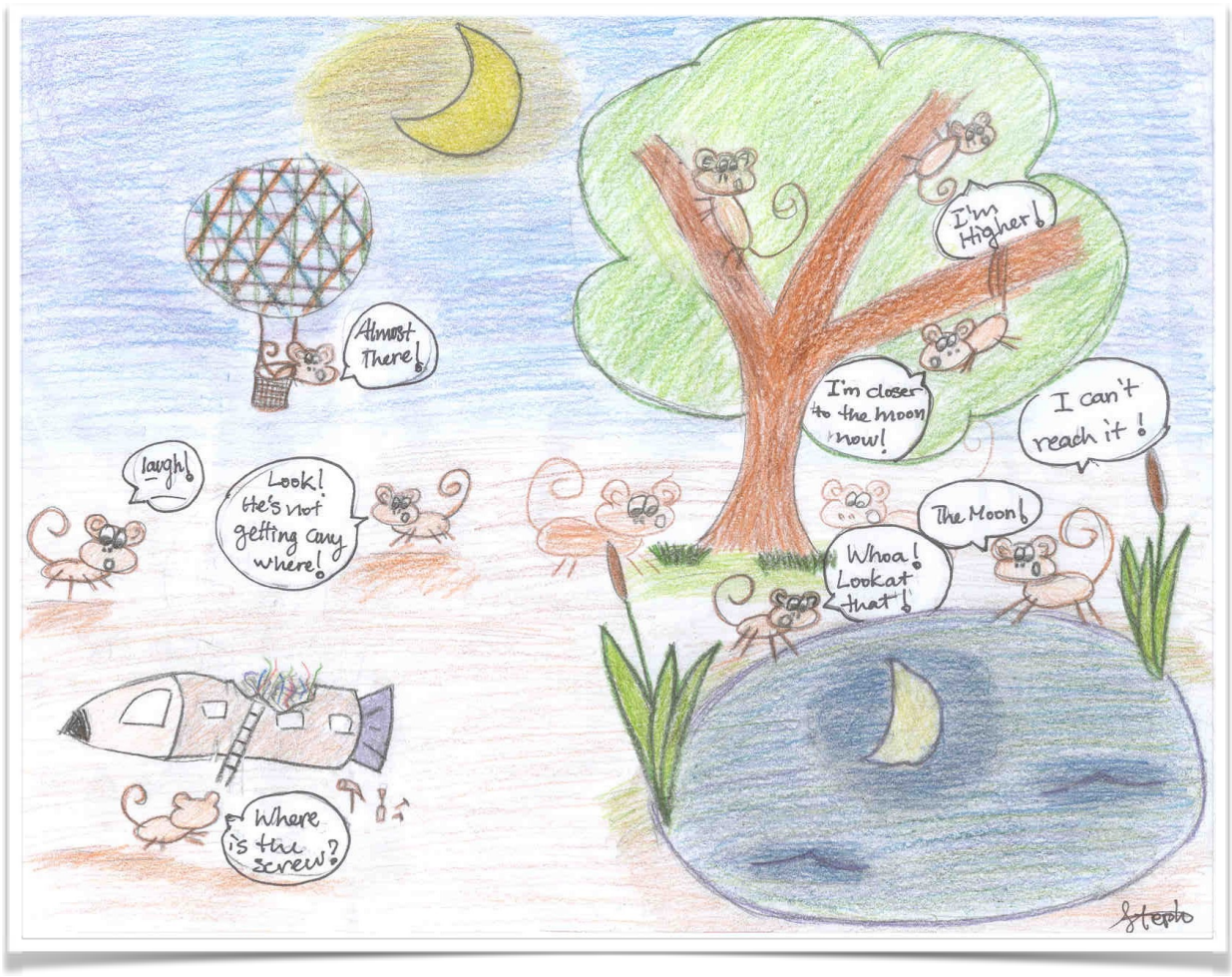
Machine Learning Report

Analysis of Classification Techniques



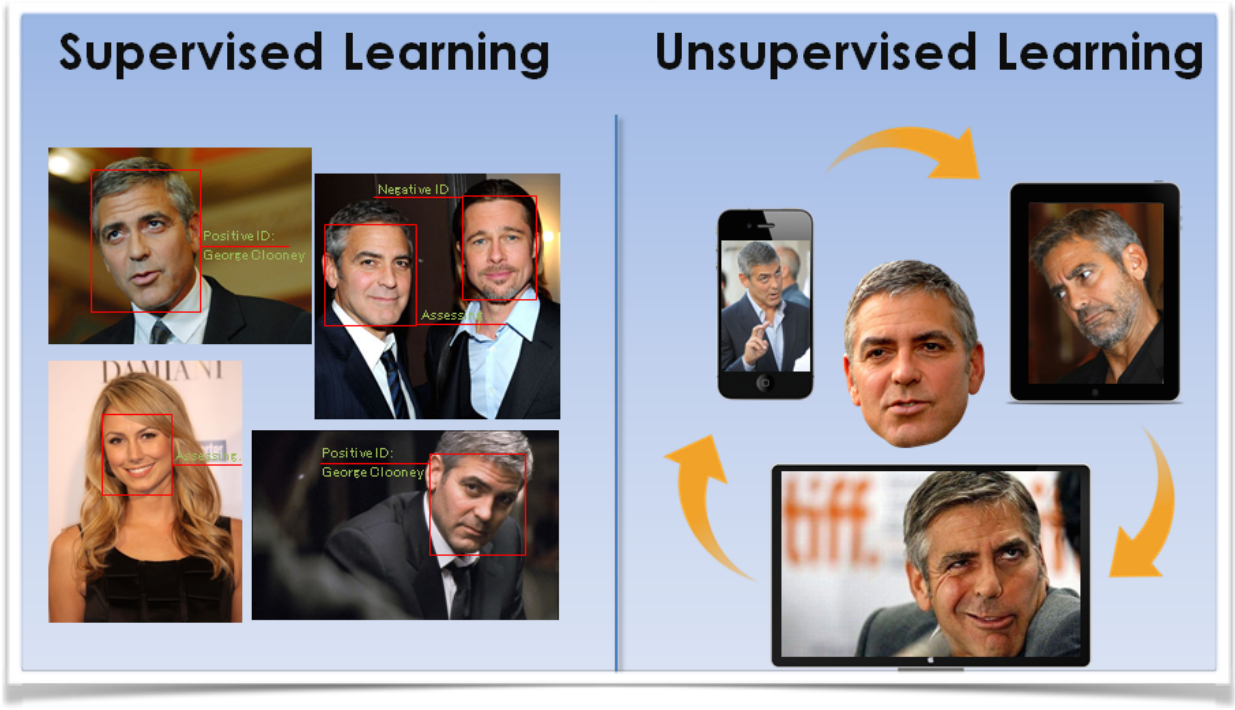
Mohit Jain - 201202164

Fall 2015

Classification Problem

All you need to know…

In machine learning, classification is the problem of identifying to which of a set of categories, a new observation belongs on the basis of a training set of data containing observations (or instances) whose category membership is known. An examples would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence of absence of certain symptoms, etc.)



In the terminology of machine learning, classification is considered an instance of *supervised learning*, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is knows as *clustering*, and involves grouping data into categories based on some measures of inherent similarity or distance.

Often, the individual observations are analysed into a set of quantifiable properties, known variously as explanatory variables or *features*. Other classifiers work by comparing observations to previous observations by means of a *similarity* or *distance function*.

An algorithm that implements classification, especially in a concrete implementation, is known as a *classifier.* The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes.

The most frequent terminology used in this report has been defined above. Any special terms used later shall be explained along with. The report is divided into 4 parts, each dealing with a different genre of the classification problem and each finally reducing down to their performance on the famous handwritten digits dataset MNIST by Yann Lecun and the USPS dataset.

Stage - 1

Nearest Neighbour

Linear - RBF - Chi Square - Polynomial SVM

Performance Comparison

I used the *scikit-learn* library in *Python* to tackle this section for comparison of various model performances.

Some of the terminologies used in the experiment reports are as follows :

- The **precision** is the ratio *tp / (tp + fp)* where *tp* is the number of true positives and *fp* the number of false positives.
- The **recall** is the ratio *tp / (tp + fn)* where *tp* is the number of true positives and fn the number of false negatives.
- The **F-beta** score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.
- The **support** is the number of occurrences of each class in y_true.

The following are my results for the experiments conducted…

~K-Nearest Neighbours :

- K=3 : *KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',* metric_params=None, n_neighbors=3, p=2, weights='distance')

K=3	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	980
1	0.97	1.00	0.98	1135
2	0.98	0.97	0.98	1032
3	0.97	0.97	0.97	1010
4	0.98	0.97	0.97	982
5	0.96	0.96	0.96	892
6	0.98	0.99	0.98	958
7	0.96	0.97	0.96	1028
8	0.99	0.95	0.97	974
9	0.96	0.96	0.96	1009
Avg/Total	**0.97**	**0.97**	**0.97**	**10000**

Confusion - k-3										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	9	7	997	2	0	0	1	14	2	0
3	0	1	4	975	1	13	1	7	4	4
4	0	5	0	0	948	0	5	4	1	19
5	4	1	0	12	2	860	5	1	3	4
6	4	3	0	0	4	3	994	0	0	0
7	0	18	4	0	2	0	0	994	0	10
8	7	0	3	13	5	11	3	4	923	5
9	3	4	2	7	9	4	1	8	2	969

- *K=5 : KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=5, p=2, weights='distance'):*

K=5	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	980
1	0.96	1.00	0.98	1135
2	0.98	0.96	0.97	1032
3	0.97	0.96	0.97	1010
4	0.98	0.96	0.97	982
5	0.97	0.97	0.97	892
6	0.98	0.99	0.98	958
7	0.96	0.96	0.96	1028
8	0.98	0.94	0.96	974
9	0.95	0.97	0.95	1009
Avg/Total	**0.97**	**0.97**	**0.97**	**10000**

Confusion Matrix - k-5

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	11	7	989	2	0	0	2	17	4	0
3	0	2	3	973	1	13	1	7	4	6
4	2	7	0	0	943	0	4	3	0	23
5	4	0	0	9	2	861	6	1	4	5
6	5	3	0	0	3	2	945	0	0	0
7	0	20	4	0	3	0	0	990	0	11
8	7	3	5	12	5	11	5	5	916	5
9	3	5	3	7	7	3	1	11	2	967

We see no change in overall performance from K=3 case. However, digit-specific performance has changed a bit for the digits in the latter half.

- *K=7 : KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=7, p=2, weights='distance'):*

K=7	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	980
1	0.96	1.00	0.98	1135
2	0.99	0.96	0.97	1032
3	0.97	0.96	0.97	1010
4	0.98	0.96	0.97	982
5	0.97	0.97	0.97	892
6	0.98	0.99	0.98	958
7	0.96	0.96	0.96	1028
8	0.99	0.95	0.96	974
9	0.95	0.96	0.96	1009
Avg/Total	**0.97**	**0.97**	**0.97**	**10000**

Confusion Matrix - k-7

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	11	7	989	2	0	0	2	17	4	0
3	0	2	3	973	1	13	1	7	4	6
4	2	7	0	0	943	0	4	3	0	23
5	4	0	0	9	2	861	6	1	4	5
6	5	3	0	0	3	2	945	0	0	0
7	0	20	4	0	3	0	0	990	0	11
8	7	3	5	12	5	11	5	5	916	5
9	3	5	3	7	7	3	1	11	2	967

We see no change in overall performance from K=3,5 case. However, digit-specific performance has changed a bit in compareson to K=3, while lesser so from the K=5 case.

~SVM :

The values of *gamma & C* were fixed after multiple iterations of different values.

- *Linear : SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0073, kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False):*

Linear	Precision	Recall	F1-Score	Support
0	0.92	0.92	0.92	980
1	0.94	0.95	0.94	1135
2	0.95	0.95	0.95	1032
3	0.95	0.95	0.95	1010
4	0.97	0.96	0.97	982
5	0.93	0.95	0.94	892
6	0.95	0.95	0.95	958
7	0.98	0.97	0.97	1028
8	0.98	0.98	0.98	974
9	0.96	0.95	0.95	1009
Avg/Total	**0.95**	**0.95**	**0.95**	**10000**

Confusion Matrix - svm-linear

	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	29	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	3	5	3	7	7	3	1	11	2	967

- *RBF : SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0073, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False):*

RBF	Precision	Recall	F1-Score	Support
0	0.93	0.96	0.94	980
1	0.92	0.95	0.96	1135
2	0.94	0.94	0.92	1032
3	0.91	0.96	0.93	1010
4	0.93	0.96	0.97	982
5	0.92	0.97	0.94	892
6	0.94	0.96	0.97	958
7	0.94	0.97	0.98	1028
8	0.95	0.95	0.98	974
9	0.93	0.96	0.93	1009
Avg/Total	**0.94**	**0.96**	**0.95**	**10000**

Confusion Matrix - svm-rbf										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	767	0	2	2	5	2	5	3	1	1
1	0	1101	2	3	0	1	4	15	8	1
2	4	8	920	15	17	6	13	11	35	3
3	2	0	24	930	1	32	3	11	29	8
4	1	5	3	1	913	0	11	5	5	38
5	13	3	0	34	13	775	19	5	24	6
6	8	3	4	2	7	15	910	5	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	3	5	3	7	7	3	1	11	2	967

Performance seems to have degraded a bit as compared to SVM with linear kernel.

- *Chi-Square : SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0073, kernel='chi', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False):*

Chi-Square	Precision	Recall	F1-Score	Support
0	0.94	0.96	0.95	980
1	0.94	0.95	0.94	1135
2	0.94	0.94	0.94	1032
3	0.95	0.96	0.96	1010
4	0.96	0.96	0.96	982
5	0.95	0.97	0.95	892
6	0.96	0.96	0.96	958
7	0.98	0.97	0.98	1028
8	0.95	0.95	0.95	974
9	0.95	0.6	0.96	1009
Avg/Total	**0.96**	**0.96**	**0.96**	**10000**

Confusion Matrix - svm-chisq

	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	2	9	22	4	9	1	1	950	4	26
8	9	9	6	17	13	27	10	12	860	11
9	3	5	3	7	7	3	1	16	2	962

Chi-square kernel outperforms all kernel modifications applied till now! But the polynomial kernel has bigger plans for this race…

- *Polynomial : SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0073, kernel='poly', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False):*

Polynomial	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	980
1	0.98	0.99	0.99	1135
2	0.97	0.98	0.97	1032
3	0.97	0.98	0.97	1010
4	0.98	0.98	0.98	982
5	0.97	0.97	0.97	892
6	0.99	0.97	0.97	958
7	0.98	0.97	0.97	1028
8	0.97	0.97	0.97	974
9	0.97	0.96	0.96	1009
Avg/Total	**0.98**	**0.98**	**0.98**	**10000**

Confusion Matrix - svm-poly

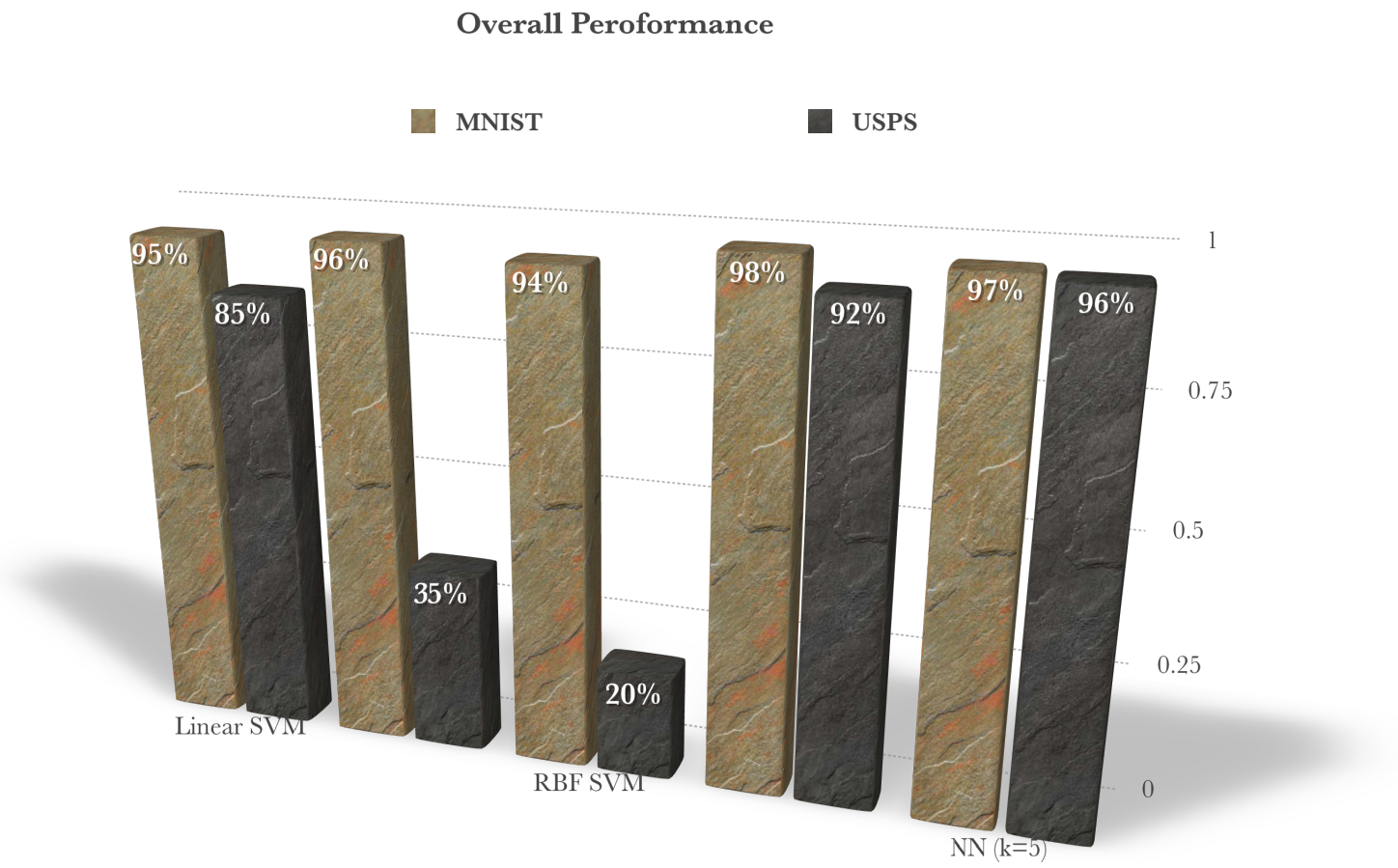
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	971	0	2	1	0	2	2	0	2	0
1	1	1128	3	0	0	1	1	1	0	0
2	5	2	1007	1	1	0	2	5	8	1
3	0	0	3	987	0	7	0	3	6	4
4	1	0	6	0	962	0	1	1	0	11
5	4	1	0	13	1	865	3	0	3	2
6	6	2	3	0	3	7	934	0	3	0
7	1	8	10	5	3	0	0	993	0	8
8	5	1	6	4	2	3	2	5	941	5
9	2	4	2	9	10	4	0	6	3	969

Polynomial kernel with SVM beats all other tested methods!

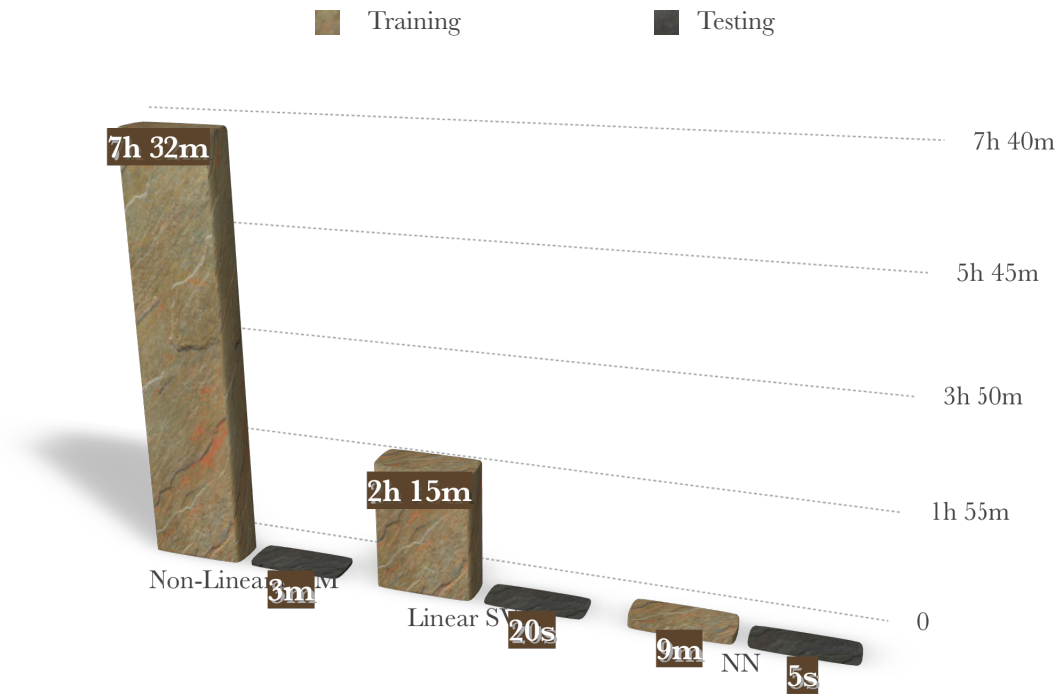
Major Takeaways…

The experiments showed that the SVM with a polynomial kernel performed the best while the one with a RBF kernel performs relatively poor. NN neighbour techniques perform approximately as good as the best SVM techniques.

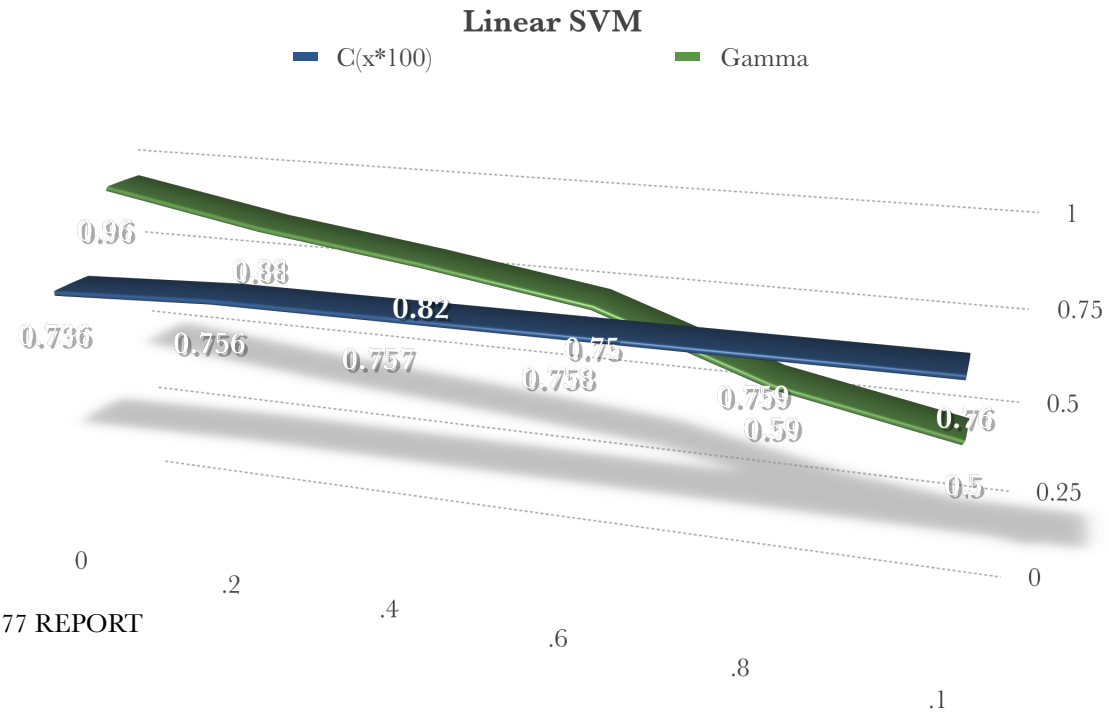
This shows that the data is not inherently different in a non-linear fashion, rather we can draw our decision boundaries in the linear plane of digits itself and hence the good performance of most linear kernels using SVM. Some of the outliers were captured by the polynomial and NN techniques and hence the outperform the other classifiers.



The time taken for the linear SVMs to finish training/testing tasks was also considerably lesser than that of their non-linear counterparts. However, the NN techniques took the least amount of time.



Another important feature of these experiments was the choice of the learning-rate constants, *gamma & C*. The following is a graphical representation of the procedure of choosing these values.



Part - 2

Improved Features & Additive Kernel SVMs

Performance Comparison

Using improved features and additive-kernels to achieve state of the art performance for these classification tasks.

Confusion Matrix - PCA - Precision - 0.1										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	971	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	82	72	17	49	479
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

Confusion Matrix - SPHOG - Precision - 0.99										
	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

Confusion Matrix - FLD - Precision - 0.90										
	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

Major Takeaways…

We are able to receive state-of-the-art performance with SPHOG. The success of this technique in achieving such high performance can be attributed to the scale invariance of these features.

PCA does an extremely bad job at this task as there were no underlying features in the images that could have been done away with. We gave as input the raw images and hence any removal in this data was a major change in the image structure itself.

FLD does a decent job, much better than PCA which had no feature extraction involved whatsoever, but not as good as the SPHOG.

Part - 3

Model construction for Handwritten Digits

Model Setting

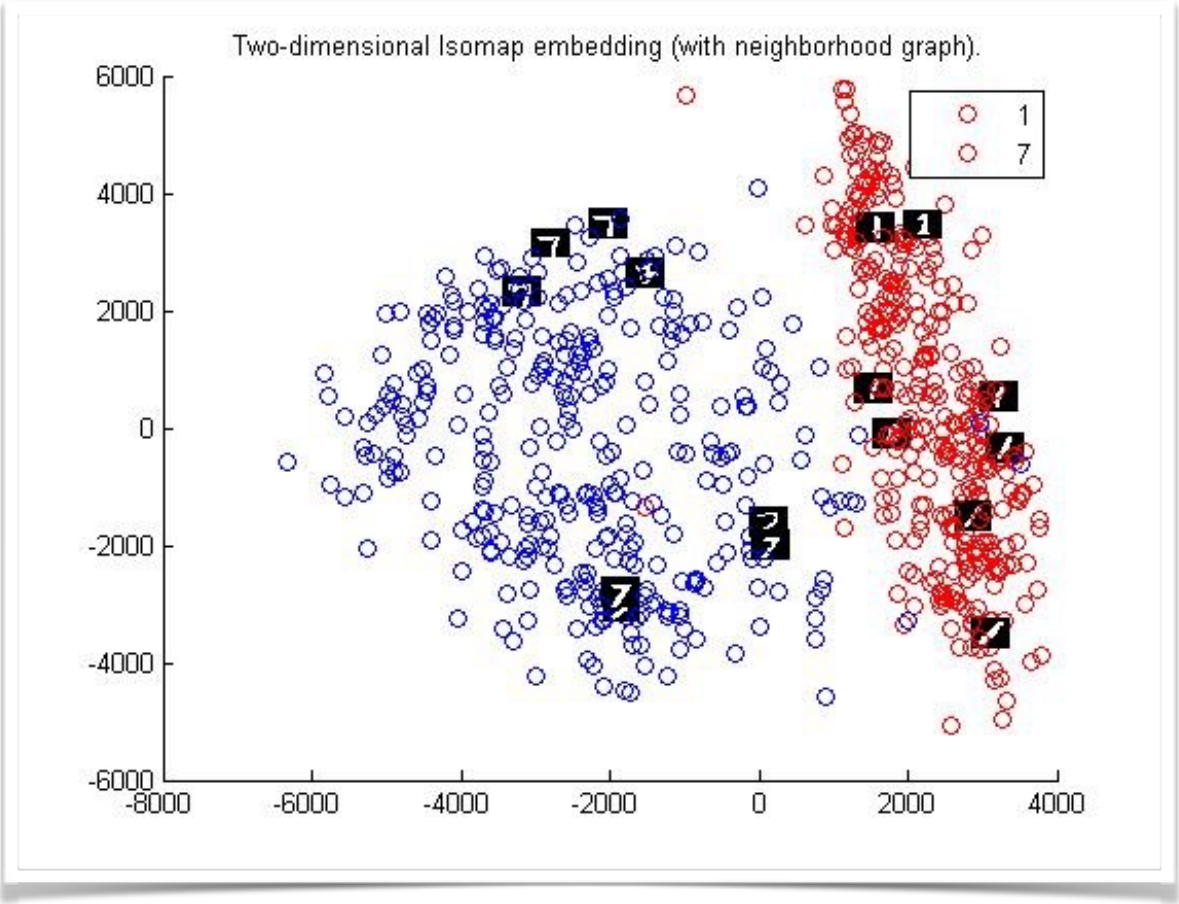
Isomap is an low dimensional embedding method where we handle high dimensional data set and embed the data points on a low dimensional space and calculate the residual variance, which gives us an idea as to how bad we failed to embed.

The experiment tries to calculate the geodesic distance on a low dimensional manifold by guessing the neighbourhood using the k-NN approach and then using Dijkstra's algorithm to calculate the geodesic distance between 2 given points and then using these formula to calculate the points in these subspace.

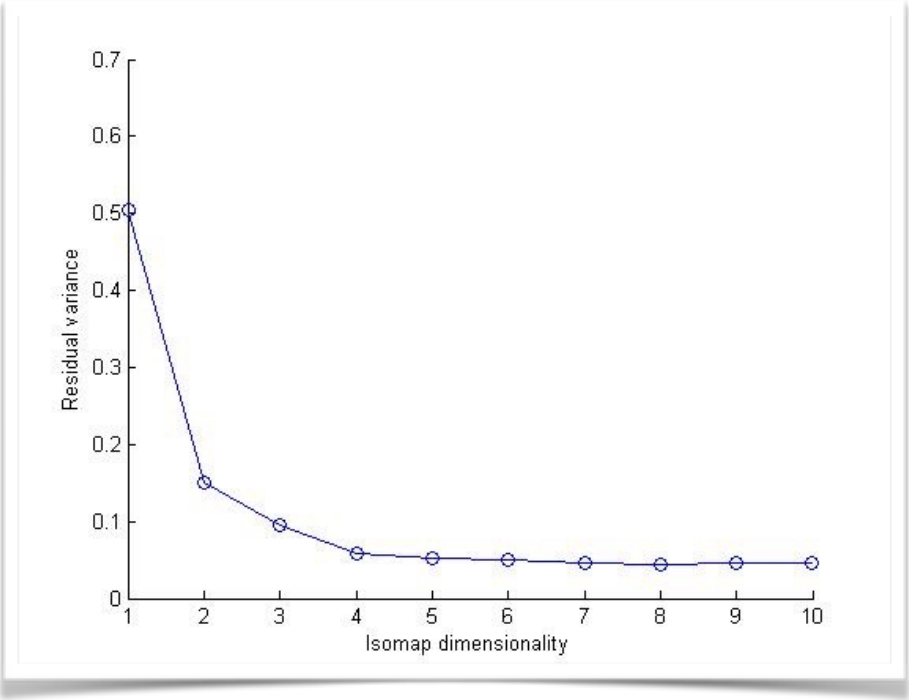
Here we are trying to reduce the dimensionality of data space ,and that will make our further calculations less complex. And we are simultaneously trying to coverage of data set in our calculations so that dimensionality reduction will be less affected.

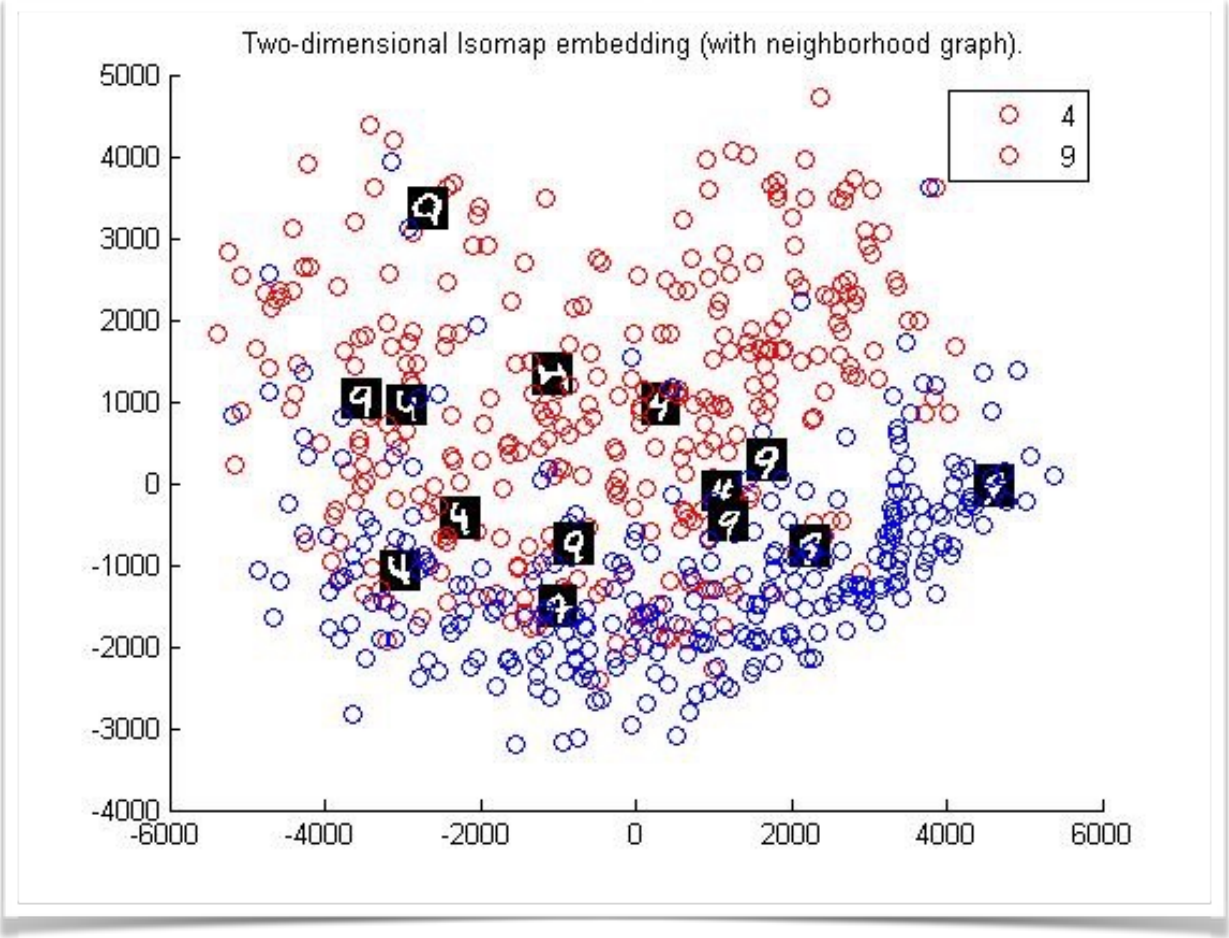
We would be considering 2 types of distance measures for the construction of the model, namely Euclidean distance and the Tangent distance.

Euclidean Model

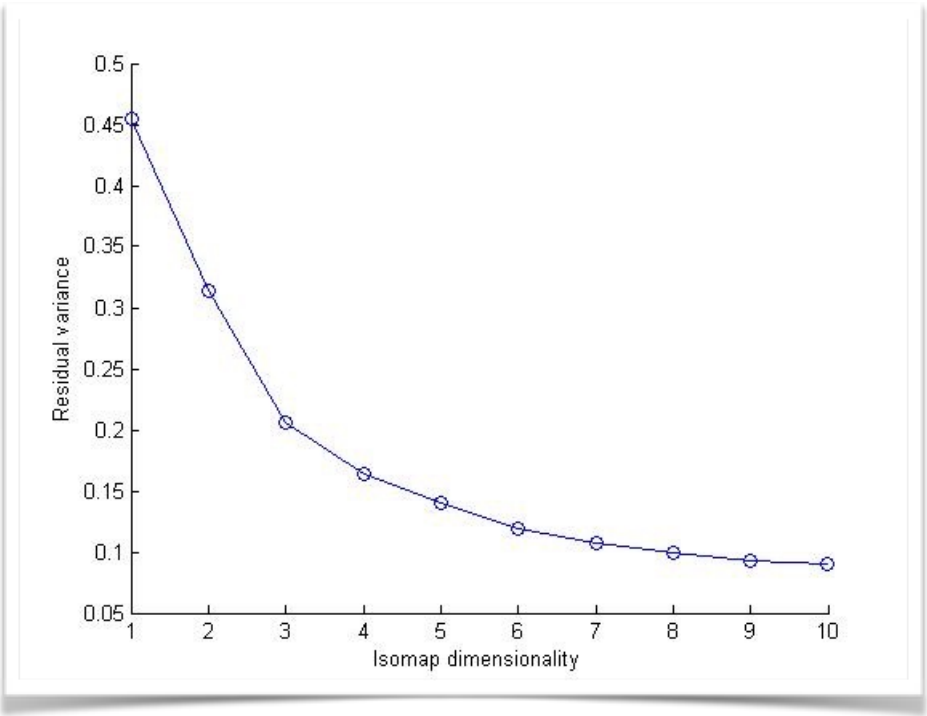


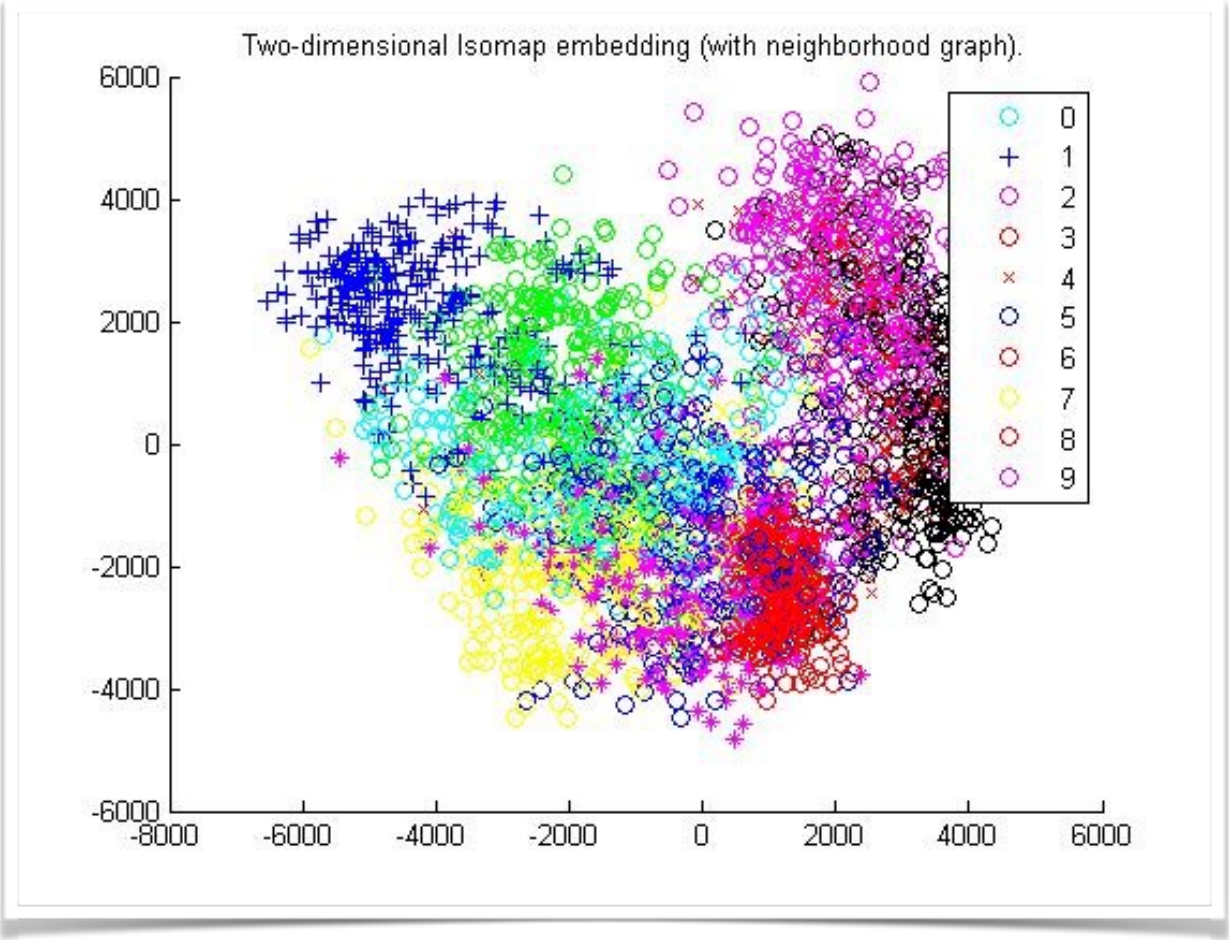
- Residual variance curve for digits 1 & 7 :



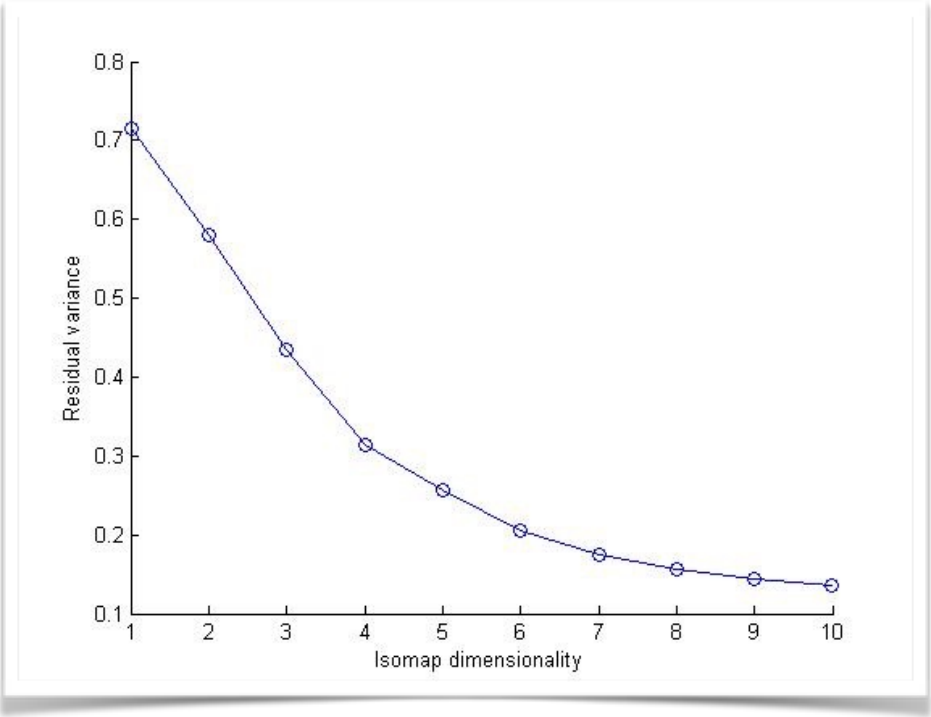


- Residual variance for the digits 4 & 9 :

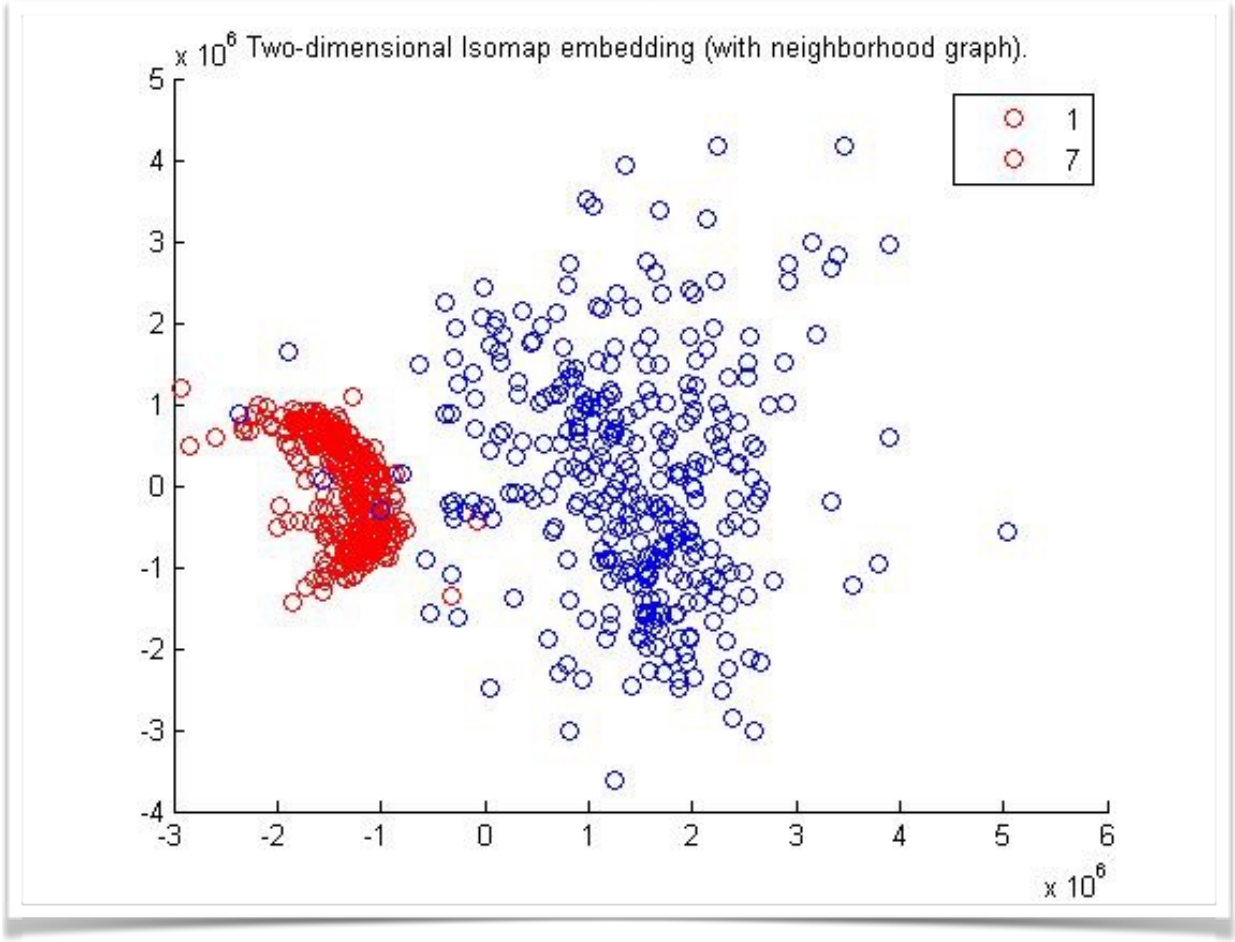


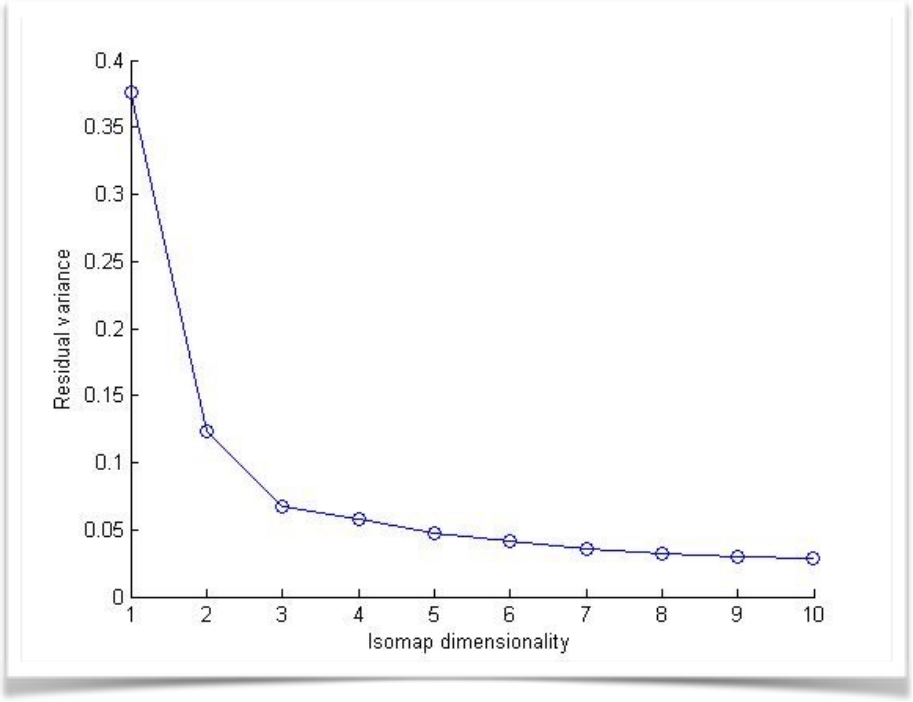


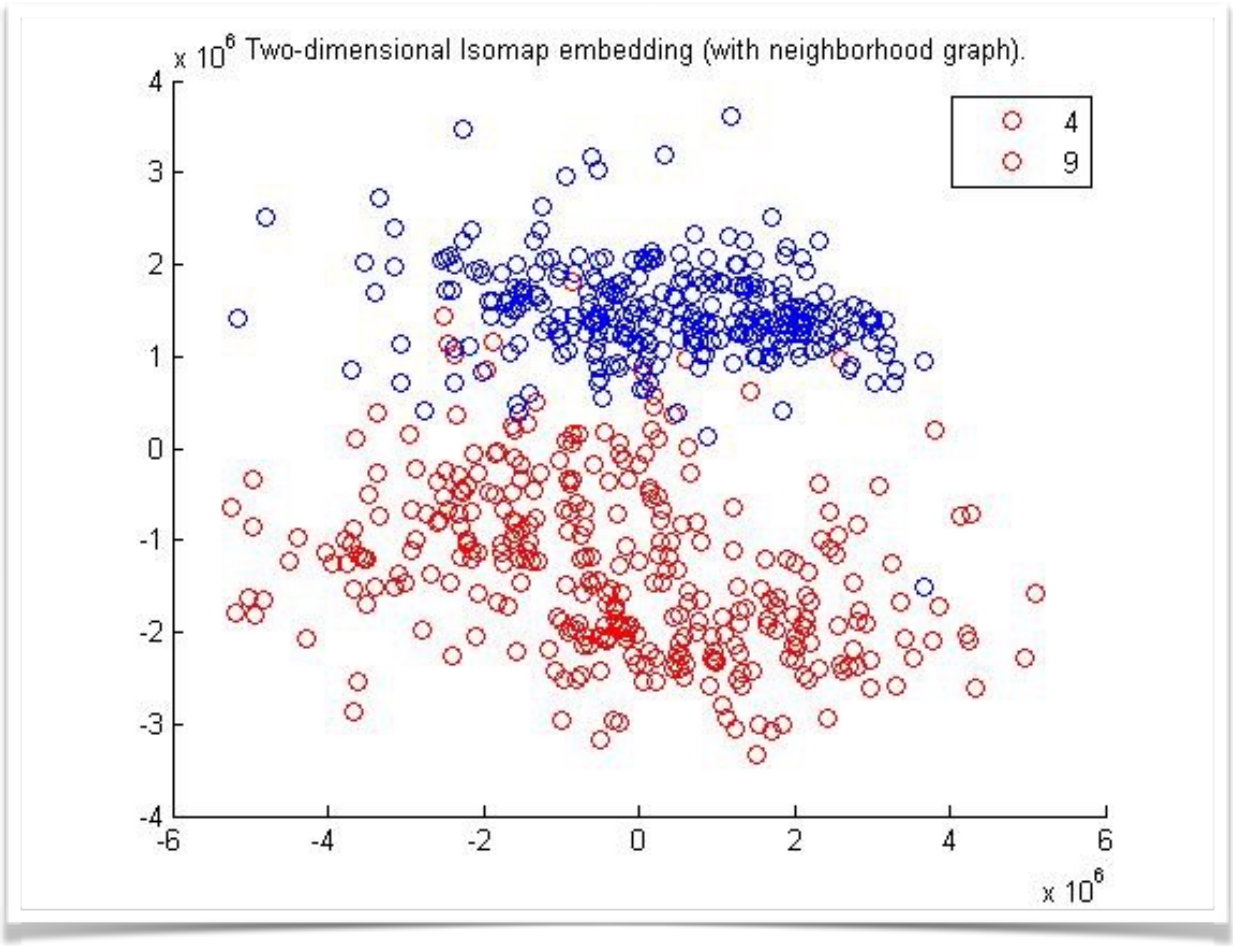
- Residual variance for all the digits :

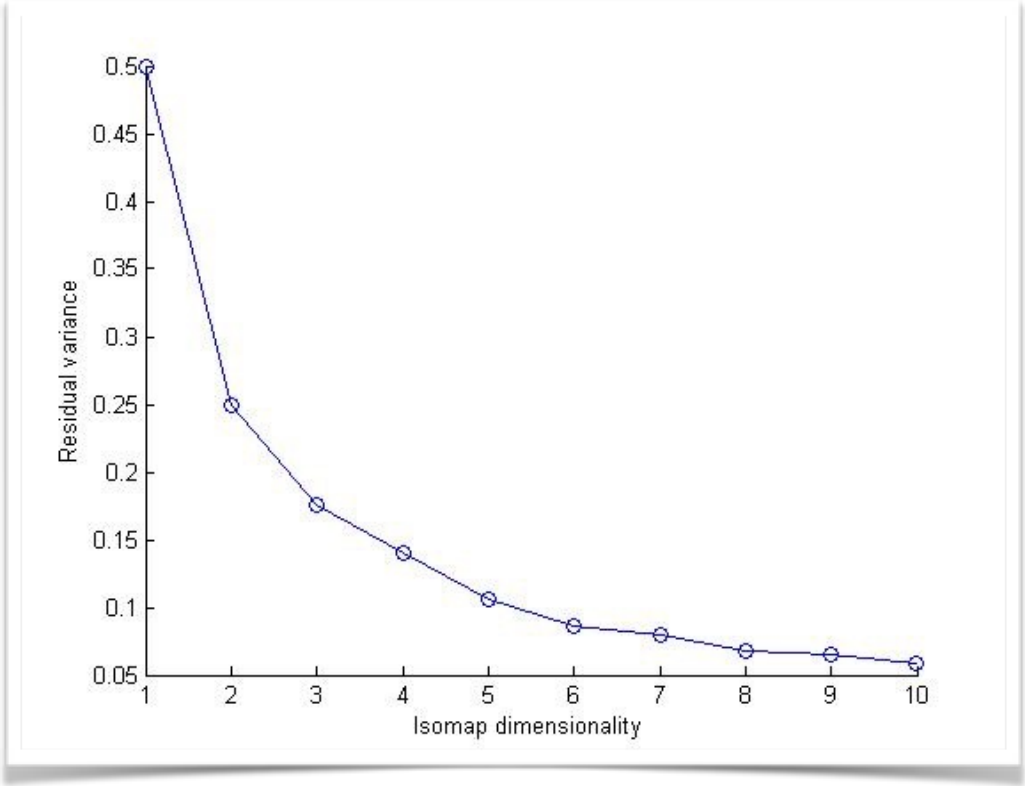


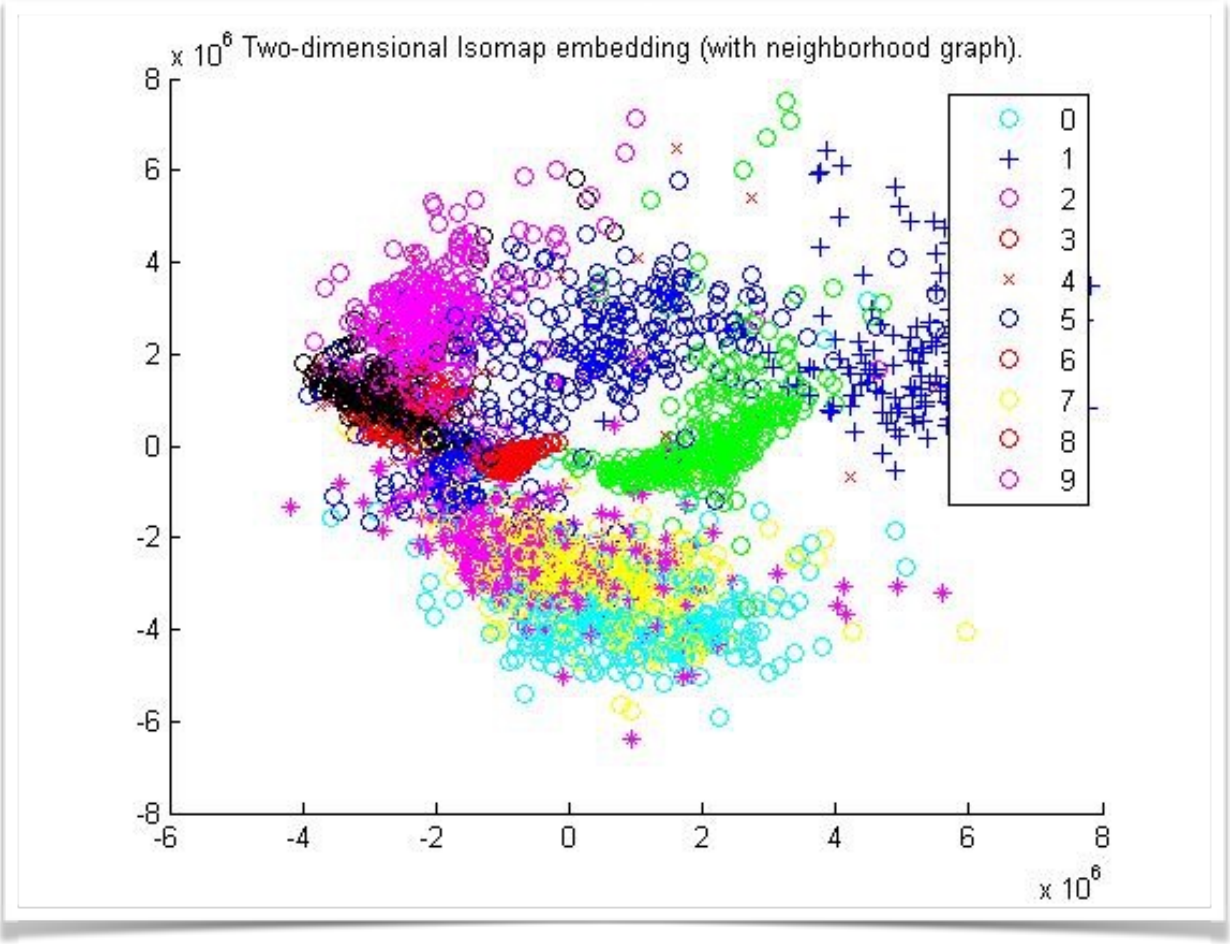
Tangent Distance

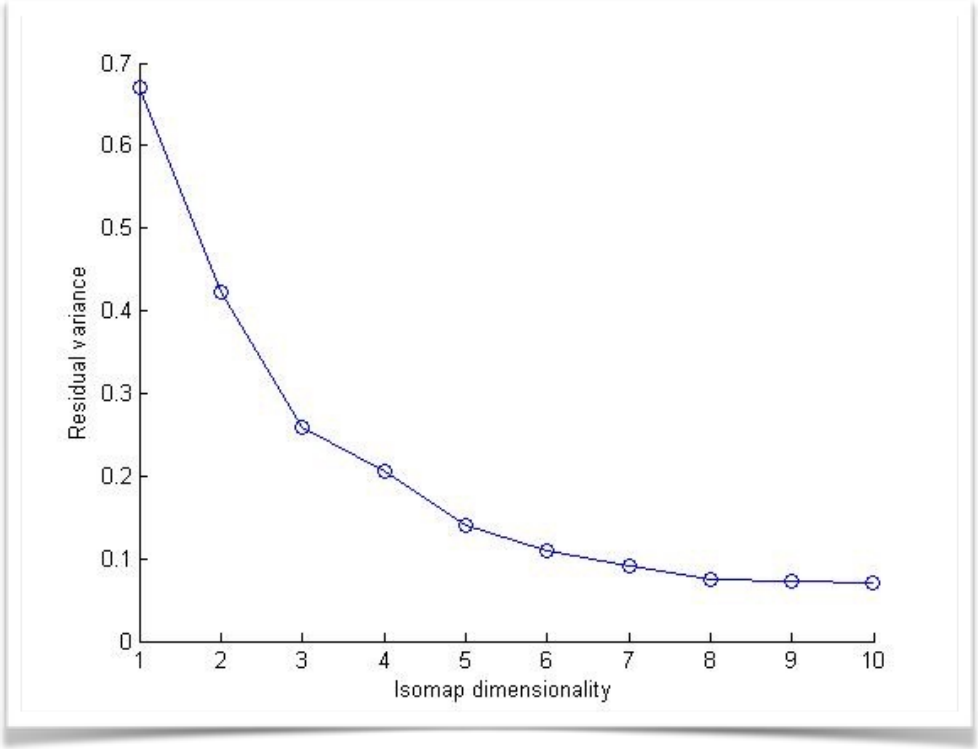





x 10^6 Two-dimensional Isomap embedding (with neighborhood graph).
4
9
x 10^6


Residual variance
Isomap dimensionality


x 10^6 Two-dimensional Isomap embedding (with neighborhood graph).
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6
8
x 10^6
0
1
2
3
4
5
6
7
8
9


Residual variance
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
10
Isomap dimensionality

Major Takeaways…

From the isomap models of tangent and euclidean distance measures created, it can be seen that 1 and 7 are well separated in both the graphs whereas 4 and 9 are not. There are inherent features in digits 4 and 9 which make them similar even in the plane of the isomap.

The variance graphs tell us that tangent distance performs better at the classification task of handwritten digits as compared to the euclidean distance.

1 and 7 are well separated in both graphs but 4 and 9 are not . And also the observation in variance tells that separation is better in tangent distance than in euclidean distance.

Part - 4

Computational Analysis & Iterative Learning

Experiment Description

(Part - A)

We will now be evaluating the efficiency parameters involved in the computation of the various experiments discussed above. The major parameters that control the performance are :

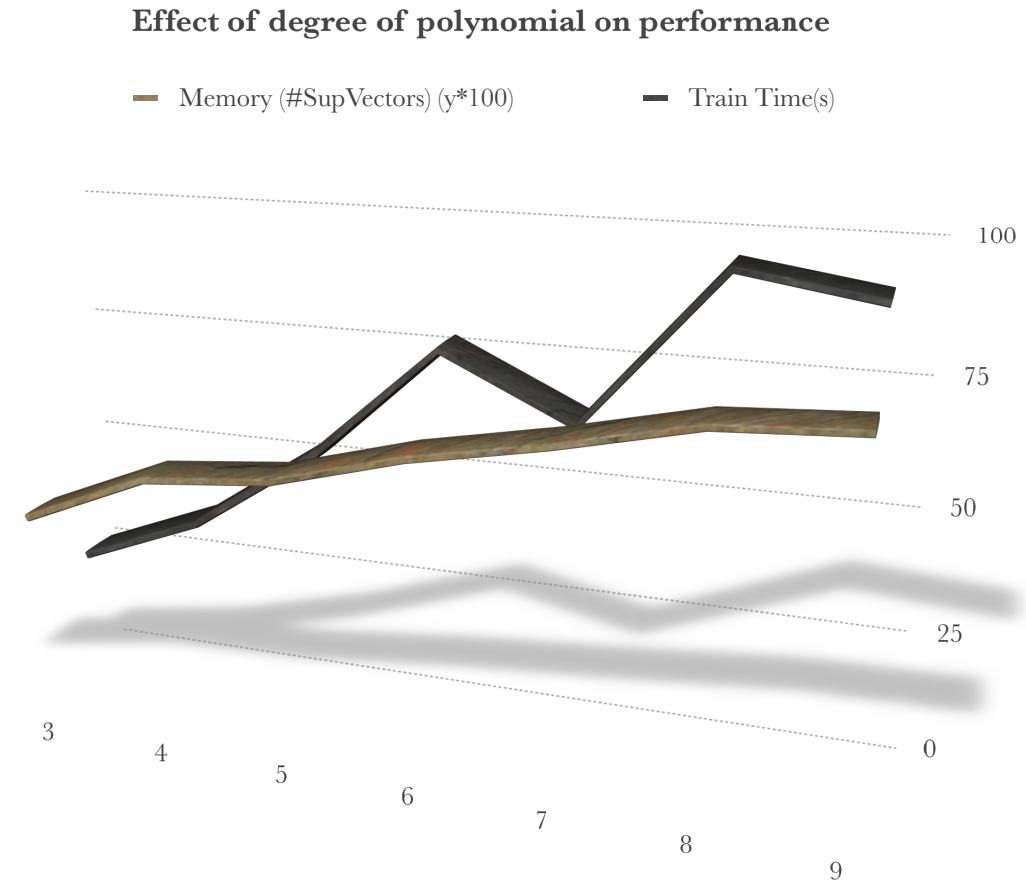
- Number of classes
- Number of data-samples, their dimensionality and distribution
- Kernel type and its meta-data hyper-parameters
- Multiclass stratergy (1v1 or 1vRest)
- Tolerance

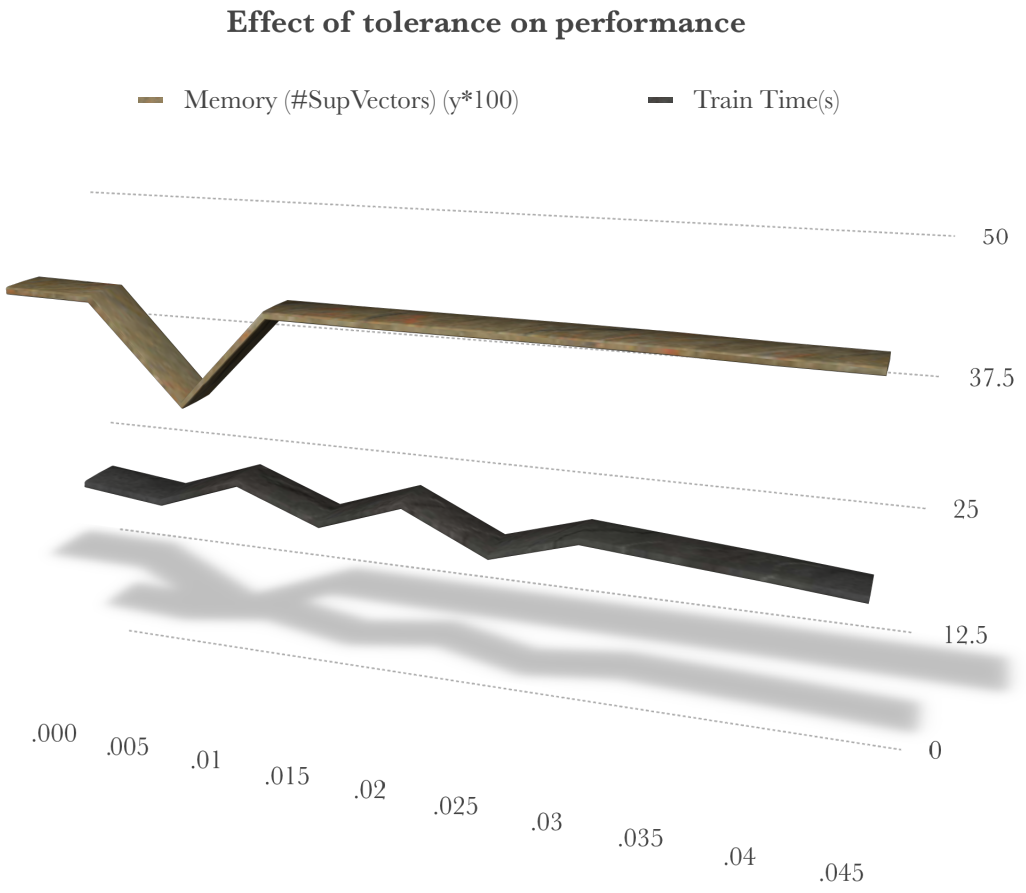
(Part - B)

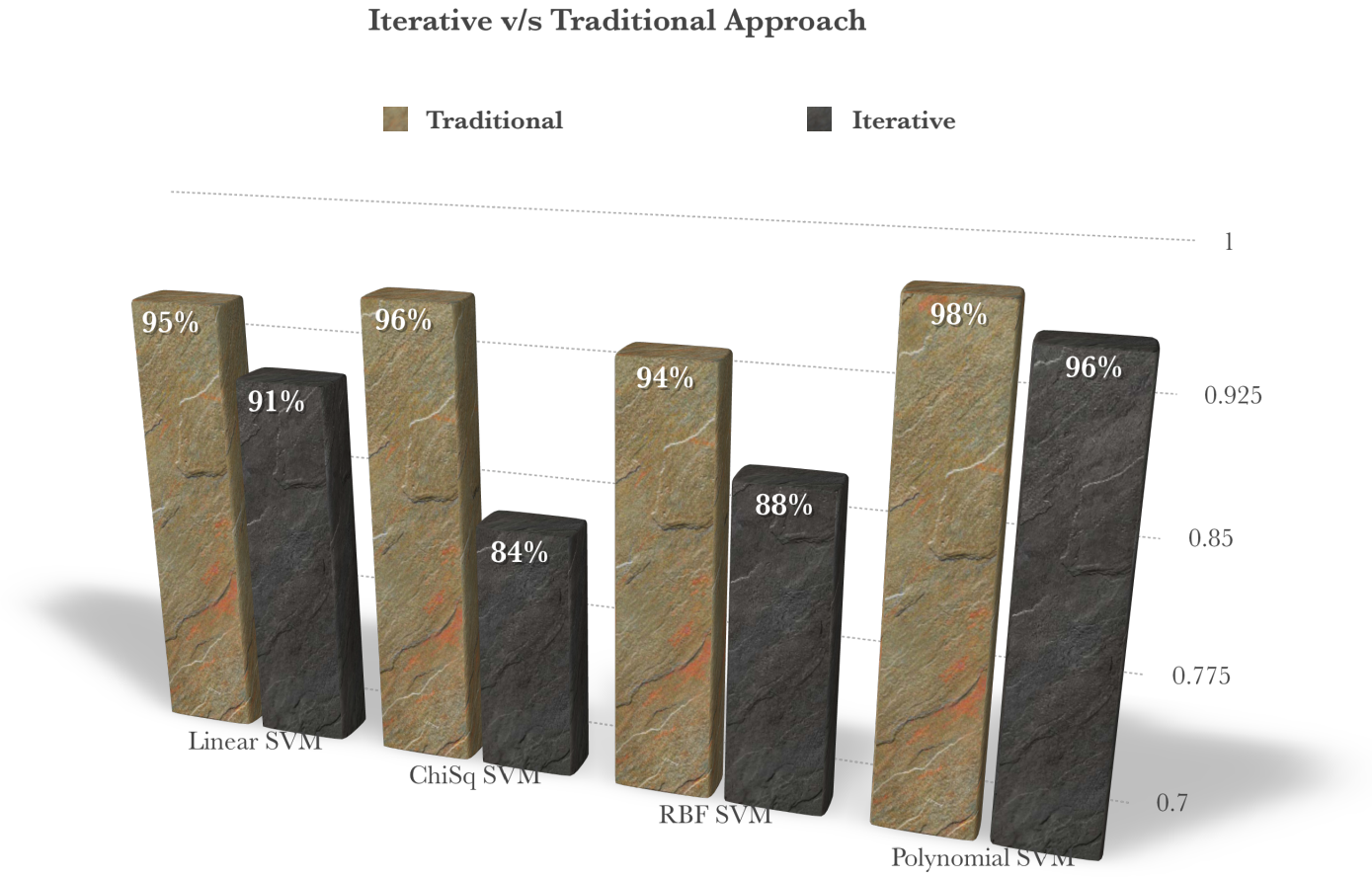
We would now be incorporating iterative learning into our learning procedure. Starting only with 10% of the MNIST dataset, we train the on this smaller training data. For every sample that is misclassified by this system, we add it to the training set and retrain our model with the updated training data.

This process is continued until the amount of change in accuracy between consecutive iterations is greater than eta = 0.1%

We would finally try to infer the advantages and disadvantages of such an approach by comparing the performance using the parameters discussed in part-A of this subpart.


Effect of degree of polynomial on performance
Memory (#SupVectors) (y*100)
Train Time(s)
100
75
50
25
0
3
4
5
6
7
8
9


Effect of tolerance on performance
Memory (#SupVectors) (y*100)
Train Time(s)
50
37.5
25
12.5
0
.000
.005
.01
.015
.02
.025
.03
.035
.04
.045



The major advantage of an iterative approach is the high reduction in time complexity of the task. By only training on a small portion of the dataset and then iteratively building up on the training data, we reduce a lot of redundant iterations of learning over the data. Hence, we see that our performance is nearly equal to the state-of-the-art methods without much parameter tuning as the support vectors in the iterative and traditional methods remain almost the same.

The marginal drop of performance however is because of the fact that we do not have all the outliers considered in our training set now and hence the margin for the SVM is not the most efficient bound as compared to the traditional method. We might apply this greedy approach to reduce time complexity, but the possibility of having tampered our original dataset exists.

Thank You.

Note :

- All codes used to produce these results can be found at my github page.
  Link : https://github.com/NightFury13/ClassificationMNIST/

- For a detailed walk through of this assignment, you can check out the report voice-over.
  Link : https://goo.gl/cjQEoj