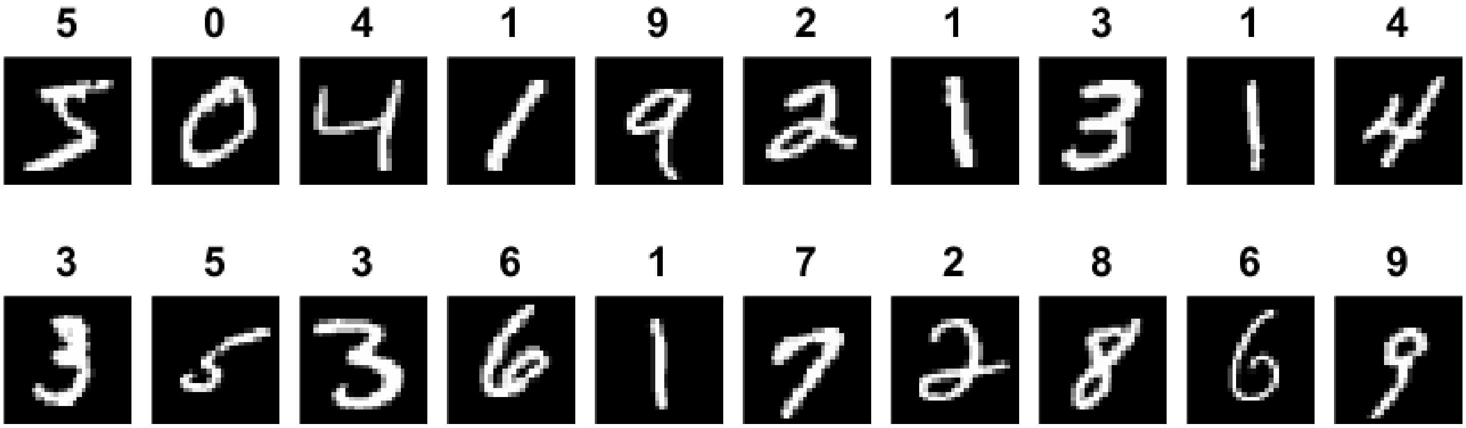
Introduction to Machine Learning
Lecture 8 - MNIST Example

1 Over-fitting and Cross Validation

1.1 MNIST data set

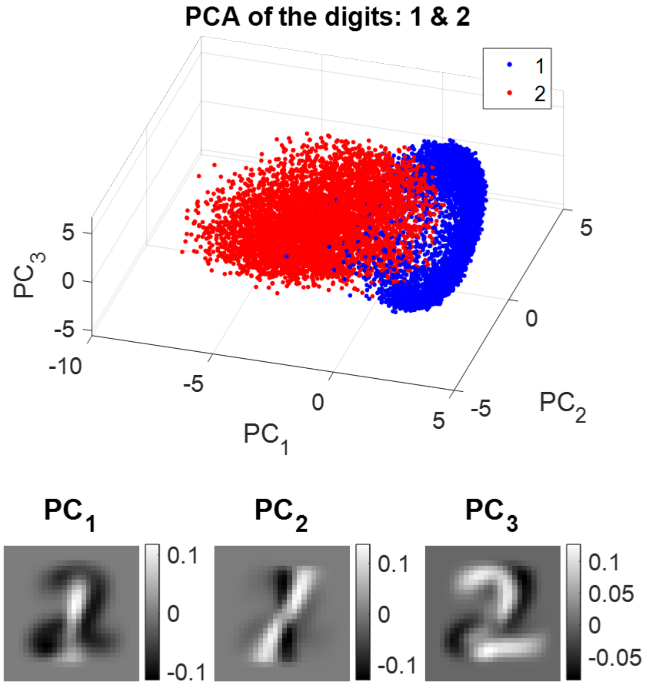
The MNIST data set contains images of digits.

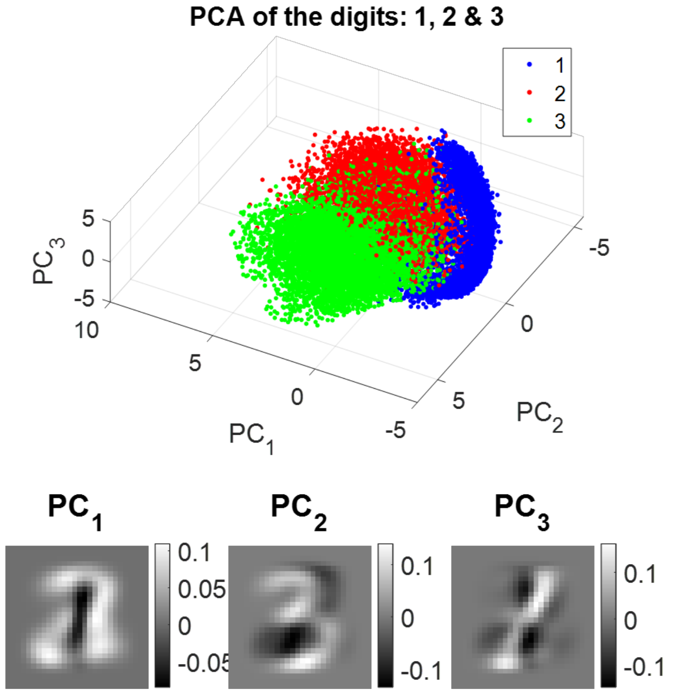


Each image contains 784 pixels. $I \in \mathbb{R}^{28\times 28}$.

1.2 Low dimensionality representation using PCA

We can project each image into $\mathbb{R}^3$ using PCA:





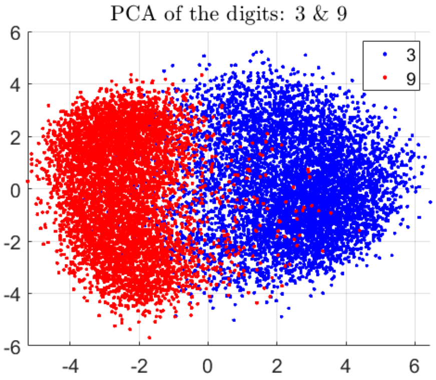
For this data set (MNIST) the linear projection using PCA does provide us with a "good" low-dimensional representation.

1.3 Over-fitting

Consider only the digits 3 and 9.
Our data set contains two subsets:

1. A training set
$$\mathcal{D}_{\text{train}}=\left\{\boldsymbol{x}_i,y_i\right\}_{i=1}^{12{,}080},\qquad y_i\in\{3,9\}$$
2. A test set
$$\mathcal{D}_{\text{test}}=\left\{\boldsymbol{x}_i,y_i\right\}_{i=1}^{2019},\qquad y_i\in\{3,9\}$$

The sets are disjoint.
We plot the $\mathbb{R}^2$ dimensional representation of the digits 3 and 9 using PCA.



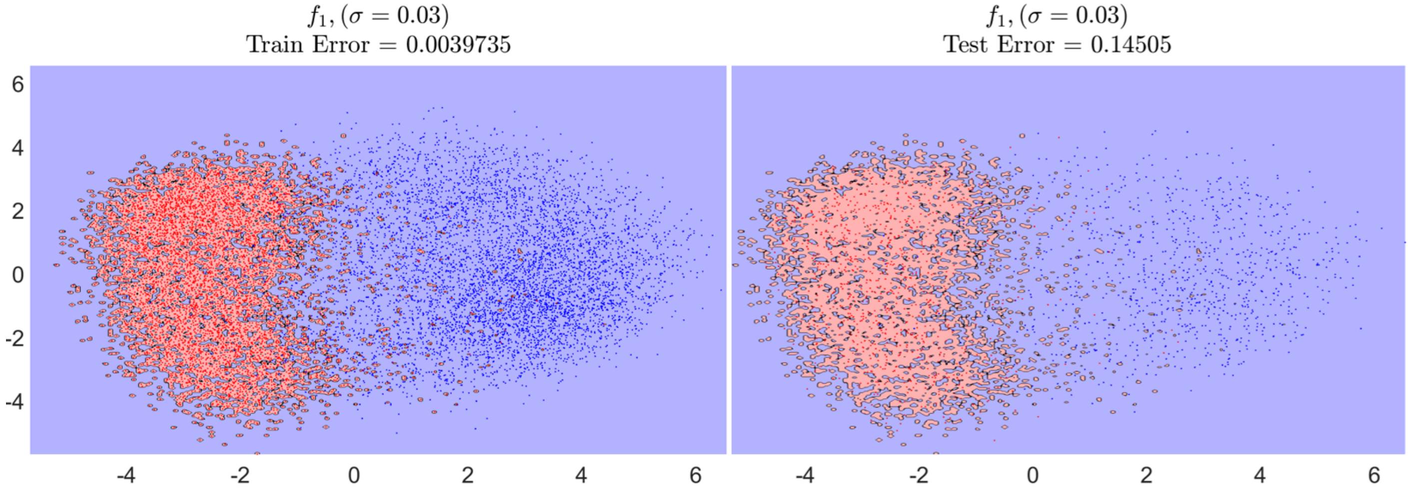
The goal is to obtain a classifier (using only the training set) which minimizes the test error.

$$\text{Train-error}=\frac{1}{\left|\mathcal{D}_{\text{train}}\right|}\sum_{i=1}^{\left|\mathcal{D}_{\text{train}}\right|}\mathbf{1}\left\{\hat{f}\left(\boldsymbol{x}_i\right)\neq y_i\right\},\qquad\left(\boldsymbol{x}_i,y_i\right)\in\mathcal{D}_{\text{train}}$$
$$\text{Test-error}=\frac{1}{\left|\mathcal{D}_{\text{test}}\right|}\sum_{i=1}^{\left|\mathcal{D}_{\text{test}}\right|}\mathbf{1}\left\{\hat{f}\left(\boldsymbol{x}_i\right)\neq y_i\right\},\qquad\left(\boldsymbol{x}_i,y_i\right)\in\mathcal{D}_{\text{test}}$$

1.3.1 Example

We train two Gaussian SVM classifiers:

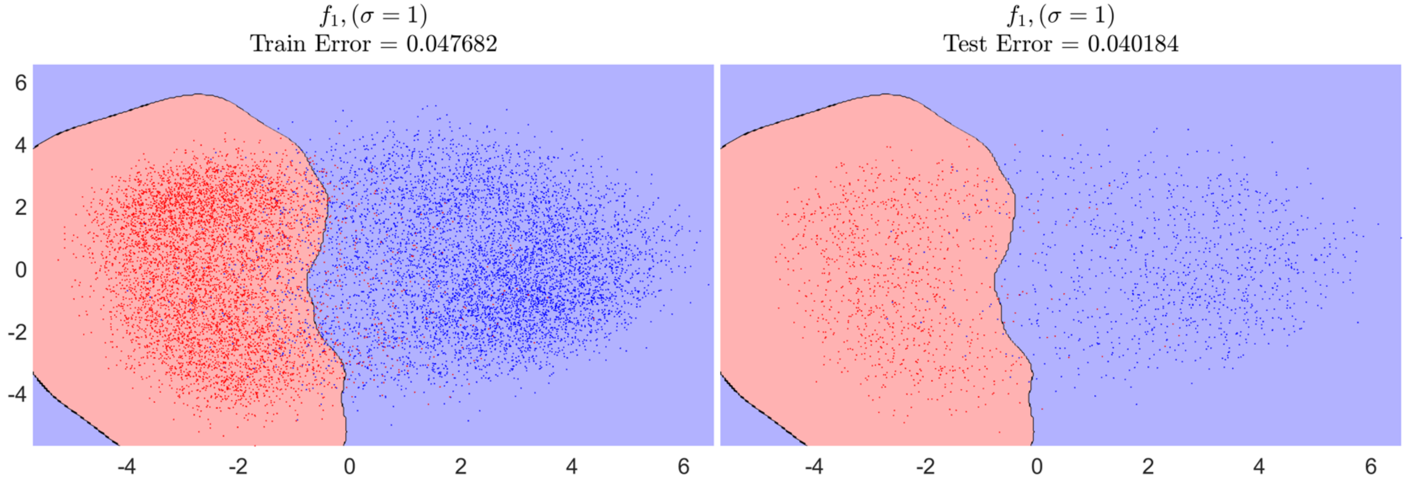
- $\sigma = 0.03$:



$f_1, (\sigma = 0.03)$
Train Error = 0.0039735

$f_1, (\sigma = 0.03)$
Test Error = 0.14505

- $\sigma = 1$:



$f_1, (\sigma = 1)$
Train Error = 0.047682

$f_1, (\sigma = 1)$
Test Error = 0.040184

Notes

- The classifier with $\sigma = 0.03$ has a small train-error, but its test error is much bigger.
  This is known as **over-fitting**.
- The classifier with $\sigma = 1$ has similar train and test errors.
  This classifier does not over-fit the training set.

1.4 Cross validation

To circumvent over-fitting we use cross-validation.

In cross validation we split (randomly) our training set $\mathcal{D}_{\text{train}}$ into K (disjoint) subsets: $\bigsqcup_{k=1}^{K} \mathcal{D}_k = \mathcal{D}_{\text{train}}$.

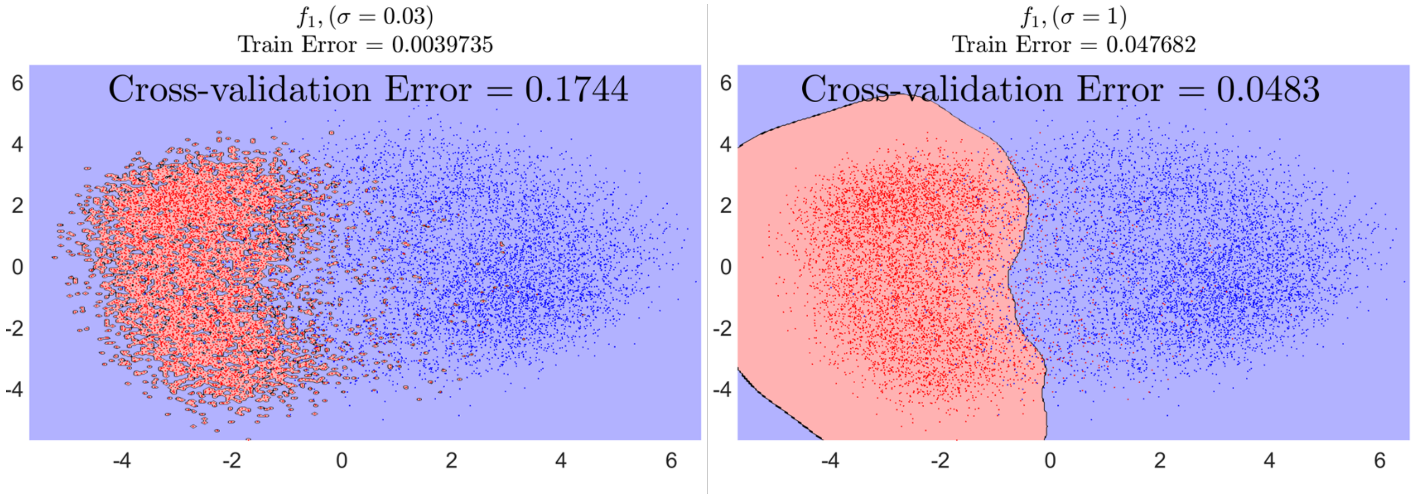
1. **for** $k = 1, \ldots, K$
   (a) Train a classifier using all subsets except the k'th subset.
   (b) Calculate the validation (test) error on the k'th subset:

$$\text{Test-error}\left(k\right) = \sum_{i=1}^{|\mathcal{D}_k|} \mathbf{1}\left\{\hat{f}\left(\boldsymbol{x}_i\right) \neq y_i\right\}, \qquad \boldsymbol{x}_i \in \mathcal{D}_k$$

2. The cross-validation error is the average of all errors:

$$\text{Cross-validation Error} = \frac{1}{N} \sum_{k=1}^{K} \text{Test Error}\left(k\right)$$

Example In the previous problem, we obtained the following cross-validation errors:



From these results we can understand the the left classifier over-fit the training set; whereas the right classifier generalized well.

1.5 Confusion Matrix

1.5.1 Training

From the full MNIST data set we created two subsets:

- A training set:

$$\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N=2,000}, \qquad \boldsymbol{x}_i \in \mathbb{R}^{784},\, y_i \in \{0,1,2,\ldots,9\}$$

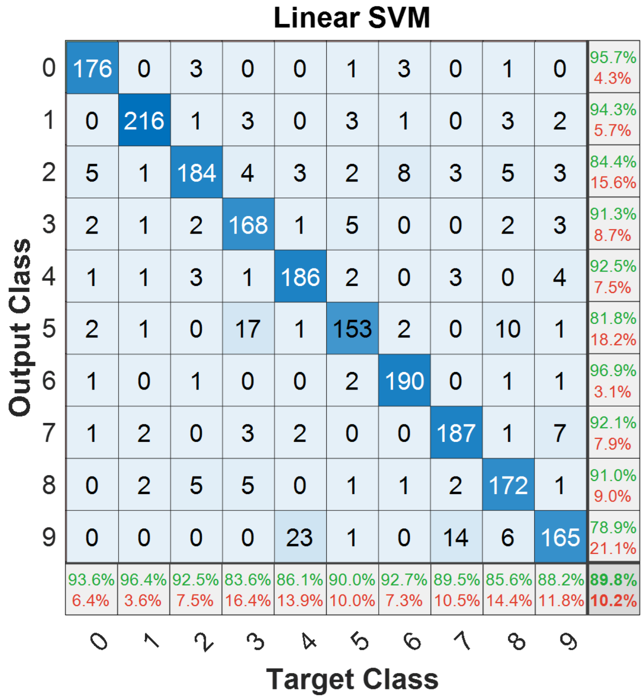
- A test set (which is disjoint with the training set):

$$\mathcal{D}_{\text{test}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N=2,000}, \qquad \boldsymbol{x}_i \in \mathbb{R}^{784},\, y_i \in \{0,1,2,\ldots,9\}$$

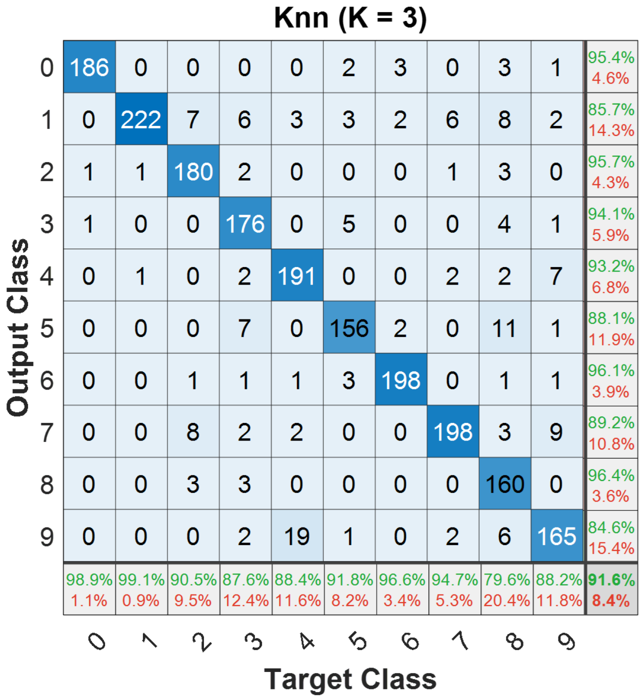
A linear SVM and K-nn ($K = 3$) classifiers were trained on the training set $\mathcal{D}_{\text{train}}$.

1.5.2 Testing

We apply the two classifiers on the testing set.
We plot the performance of each classifier on the test set $\mathcal{D}_{\text{test}}$ using a **confusion matrix**.



	0	1	2	3	4	5	6	7	8	9	
0	176	0	3	0	0	1	3	0	1	0	95.7% 4.3%
1	0	216	1	3	0	3	1	0	3	2	94.3% 5.7%
2	5	1	184	4	3	2	8	3	5	3	84.4% 15.6%
3	2	1	2	168	1	5	0	0	2	3	91.3% 8.7%
4	1	1	3	1	186	2	0	3	0	4	92.5% 7.5%
5	2	1	0	17	1	153	2	0	10	1	81.8% 18.2%
6	1	0	1	0	0	2	190	0	1	1	96.9% 3.1%
7	1	2	0	3	2	0	0	187	1	7	92.1% 7.9%
8	0	2	5	5	0	1	1	2	172	1	91.0% 9.0%
9	0	0	0	0	23	1	0	14	6	165	78.9% 21.1%
	93.6% 6.4%	96.4% 3.6%	92.5% 7.5%	83.6% 16.4%	86.1% 13.9%	90.0% 10.0%	92.7% 7.3%	89.5% 10.5%	85.6% 14.4%	88.2% 11.8%	**89.8%** **10.2%**



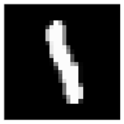
	0	1	2	3	4	5	6	7	8	9	
0	186	0	0	0	0	2	3	0	3	1	95.4% 4.6%
1	0	222	7	6	3	3	2	6	8	2	85.7% 14.3%
2	1	1	180	2	0	0	0	1	3	0	95.7% 4.3%
3	1	0	0	176	0	5	0	0	4	1	94.1% 5.9%
4	0	1	0	2	191	0	0	2	2	7	93.2% 6.8%
5	0	0	0	7	0	156	2	0	11	1	88.1% 11.9%
6	0	0	1	1	1	3	198	0	1	1	96.1% 3.9%
7	0	0	8	2	2	0	0	198	3	9	89.2% 10.8%
8	0	0	3	3	0	0	0	0	160	0	96.4% 3.6%
9	0	0	0	2	19	1	0	2	6	165	84.6% 15.4%
	98.9% 1.1%	99.1% 0.9%	90.5% 9.5%	87.6% 12.4%	88.4% 11.6%	91.8% 8.2%	96.6% 3.4%	94.7% 5.3%	79.6% 20.4%	88.2% 11.8%	**91.6%** **8.4%**

Some errors of the linear SVM classifier:

Label - 7
Predicted - 9


Label - 4
Predicted - 2

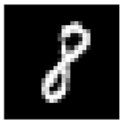

Label - 4
Predicted - 2


Label - 1
Predicted - 5


Label - 6
Predicted - 1


Label - 4
Predicted - 9


Label - 7
Predicted - 2


Label - 8
Predicted - 1


Label - 3
Predicted - 5


Label - 3
Predicted - 5


Label - 3
Predicted - 7


Label - 7
Predicted - 9


Label - 3
Predicted - 1


Label - 4
Predicted - 9


Label - 5
Predicted - 3


Label - 2
Predicted - 0


Label - 8
Predicted - 9

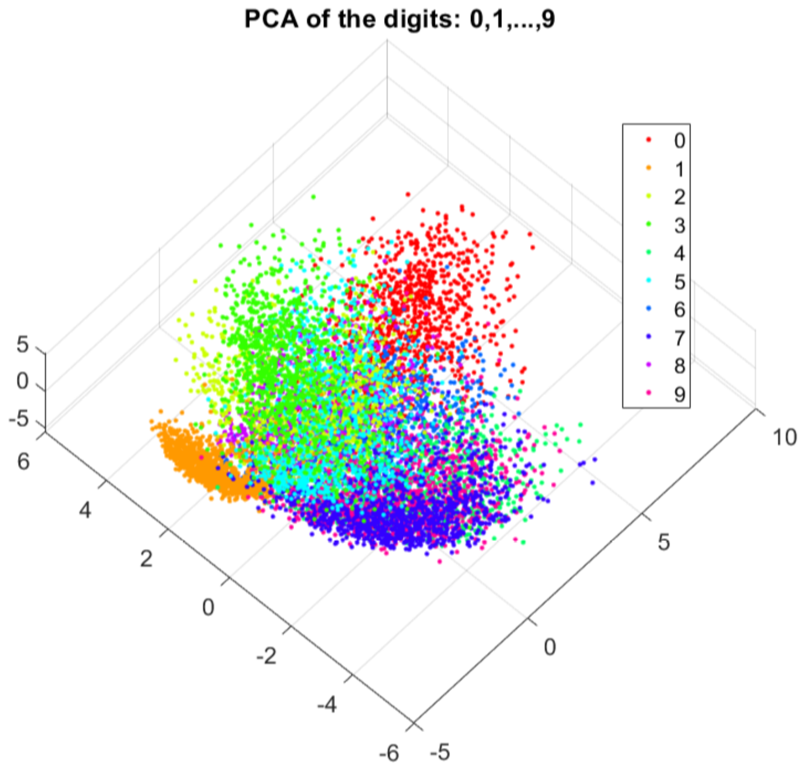

Label - 9
Predicted - 7


Label - 1
Predicted - 8


Label - 4
Predicted - 9


2 t-Distributed Stochastic Neighbor Embedding (tSNE)

Sometimes, linear dimensionality reduction might be insufficient.
For example, using only 3 principle components does not allow us to separate between the different digits in the MNIST data set:



In this case, one can try using a non-linear dimensionality reduction.
There are several algorithm such as:

- Locally-linear embedding (LLE).
- Isomap.
- Laplacian eigenmaps.
- Diffuion Maps.
- etc'

For clustering purposes, the t-SNE algorithm is quite useful (we won't explain the algorithm).
For example, using t-SNE on the MNIST data set provides the following $\mathbb{R}^2$ low dimensionality representation: