UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3253 Analytic Techniques and Machine Learning

Module 8: Dimensionality Reduction

Course Plan

Module Titles
Module 1 – Introduction to Machine Learning
Module 2 – End to End Machine Learning Project
Module 3 – Classification
Module 4 – Clustering and Unsupervised Learning
Module 5 – Training Models and Feature Selection
Module 6 – Support Vector Machines
Module 7 – Decision Trees and Ensemble Learning
Current Focus: Module 8 – Dimensionality Reduction
Module 9 – Introduction to TensorFlow
Module 10 – Introduction to Deep Learning and Deep Neural Networks
Module 11 – Distributing TensorFlow, CNNs and RNNs
Module 12 – Final Assignment and Presentations (no content)

Learning Outcomes for this Module

- Be aware of "The Curse of Dimensionality" and strategies for minimizing it
- Apply dimensionality reduction techniques such as
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Local Linear Embedding

Topics for this Module

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

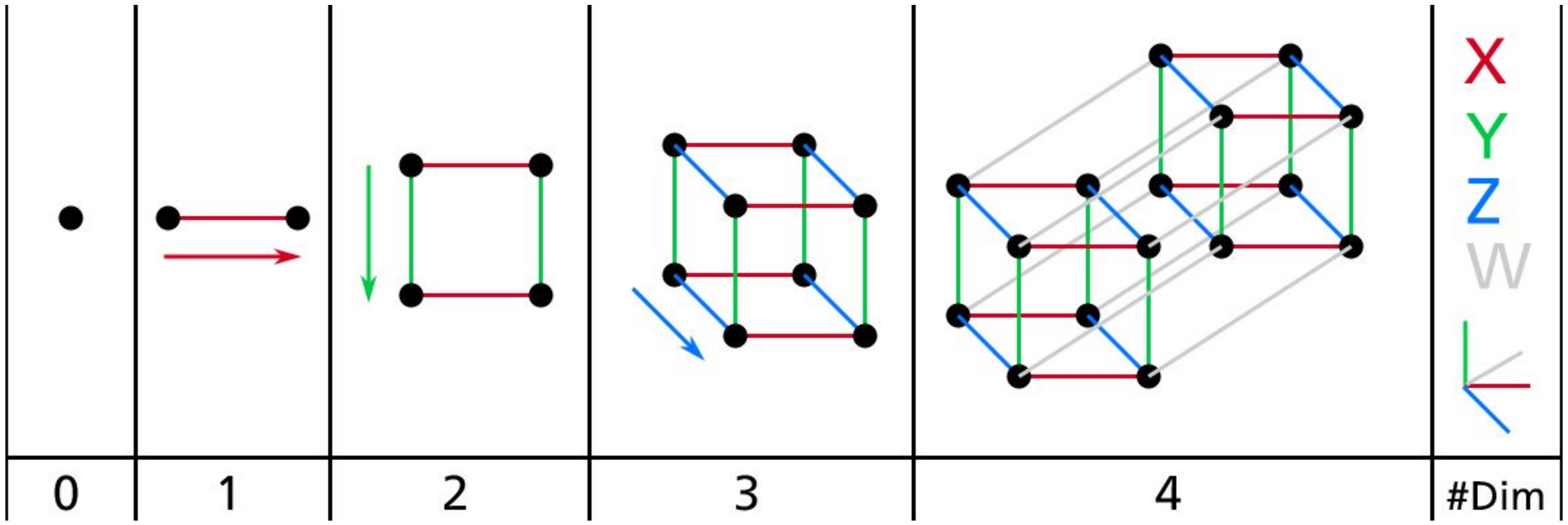
Module 8 – Section 1

The “Curse” of Dimensionality

Introduction

- Machine Learning problems involve thousands or even millions of features for each training instance:
  - Too many features makes training extremely slow, it can also make it much harder to find a good solution
  - This problem is often referred to as the "curse" of dimensionality
- It is often possible to reduce the number of features considerably, turning an intractable problem into a tractable one.
- Dimensionality reduction is also very useful for data visualization

N-Dimensions

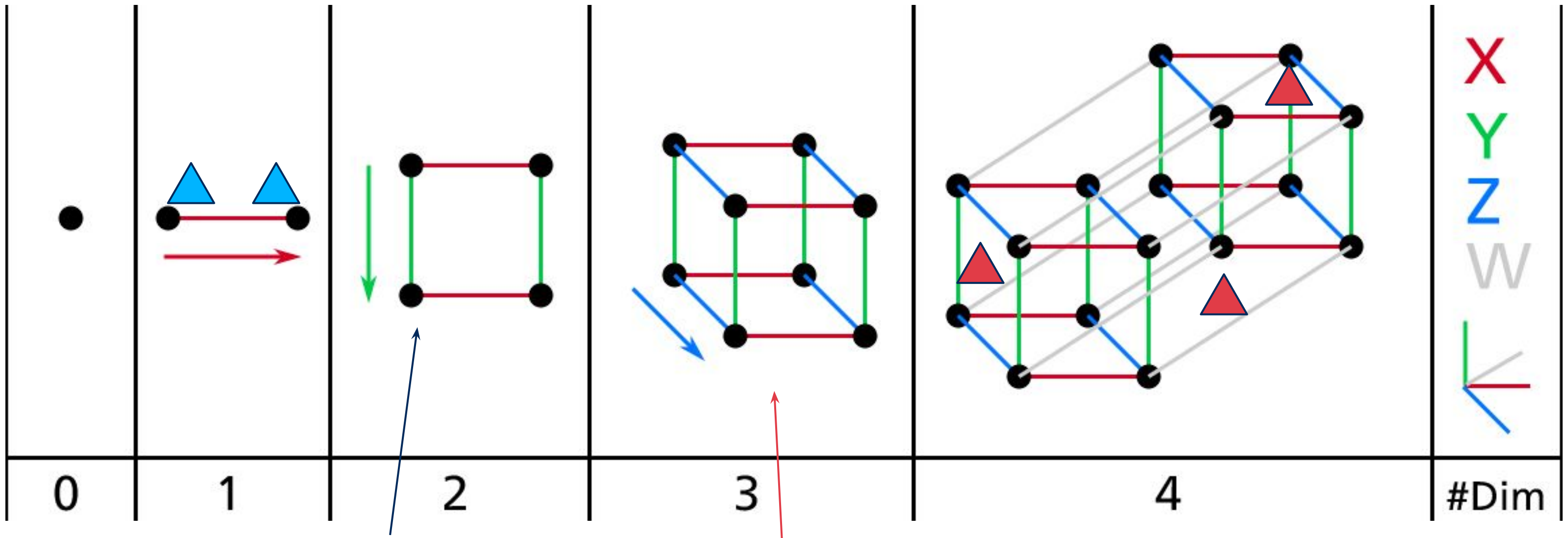


Even a basic 4D hypercube is incredibly hard to picture in our mind

N-Dimensions

Blue triangles part of a 1 feature space are comparatively very close

Blue triangles part of a 4 features space can be very far



within a square, distance between two random points will be, on average 0.52

within a cube, distance between two random points will be, on average 0.66

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

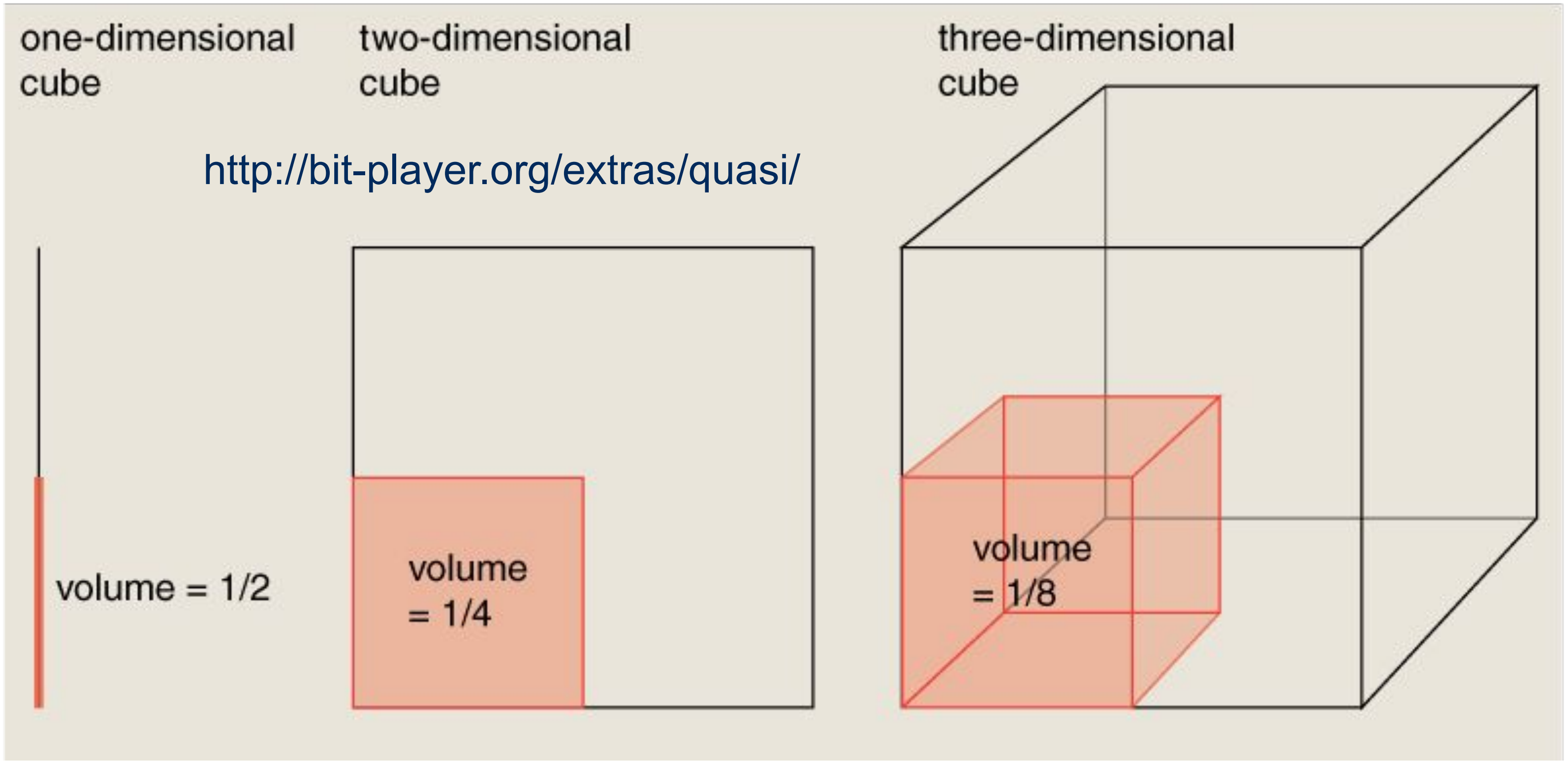
Curse of Dimensionality: Sparcity

- Things behave very differently in high-dimensional space.
  - Probability of a random point being close to the border:
    - In a 1 × 1 square, there is about a 0.4% chance of the point being located less than 0.001 from a border.
    - For a 10,000-dimensional unit hypercube, this probability is greater than 99.999999%.
    - Most points in a high-dimensional hypercube are very close to the border.
  - Two random chosen points inside a cube:
    - In a unit square, the distance between these two points will be, on average, roughly 0.52.
    - In a unit 3D cube, the average distance will be roughly 0.66.
    - In a 1,000,000-dimensional hypercube will be about 408.25
    - High-dimensional datasets are at risk of being very sparse: most training instances are likely to be far away from each other.

More instances to help sparsity

- A new instance will likely be far away from any training instance, making predictions much less reliable than in lower dimensions, since they will be based on much larger extrapolations.
  - The more dimensions the training set has, the greater the risk of overfitting it.
- One solution to the curse of dimensionality could be to increase the size of the training set to reach a sufficient density of training instances.
  - In practice, the number of training instances required to reach a given density grows exponentially with the number of dimensions.

Curse of Dimensionality (cont'd)



- As dimensions grow, fewer observations per region
- The volume of the unit sphere tends to zero as the dimensionality of the sphere increases

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

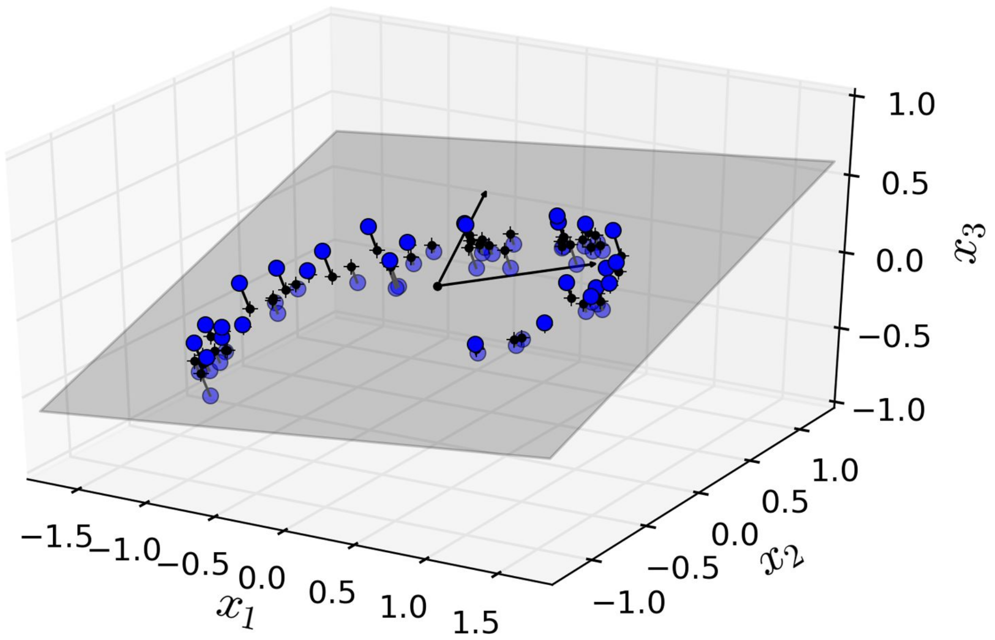
Module 9 – Section 2

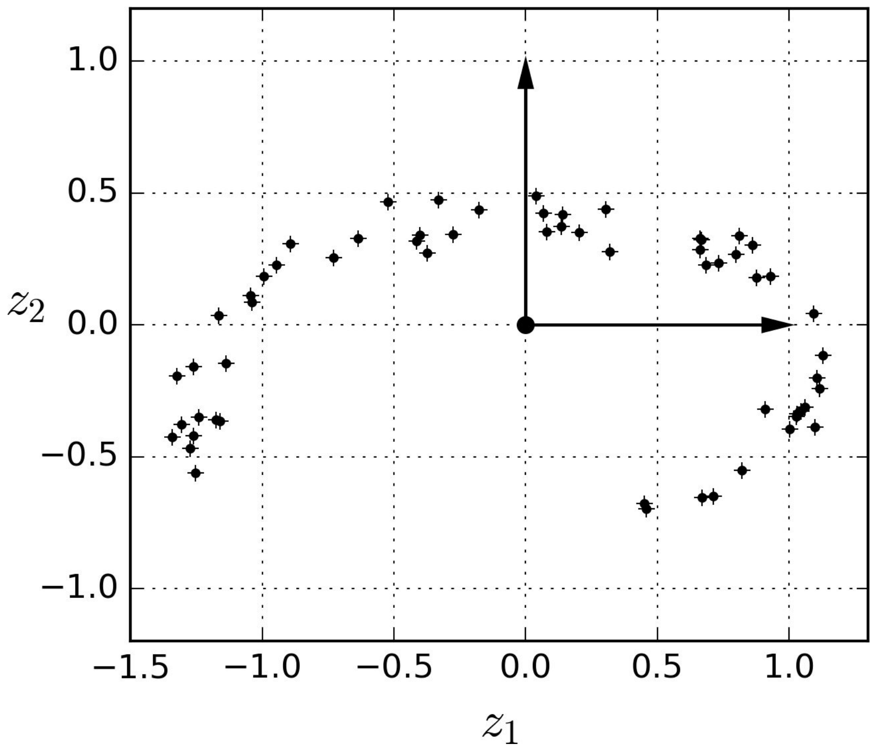
Dimensionality Reduction using Projection

Dimensionality Reduction – Projection

- A technique to reduce dimensionality
- Data is not spread out uniformly across all dimensions.
  - Many features are almost constant, while others are highly correlated
  - All training instances actually lie within a much lower-dimensional subspace of the high-dimensional space.
- If we project every training instance perpendicularly onto this subspace, we get the new 2D dataset
  - We have just reduced the dataset’s dimensionality from 3D to 2D.

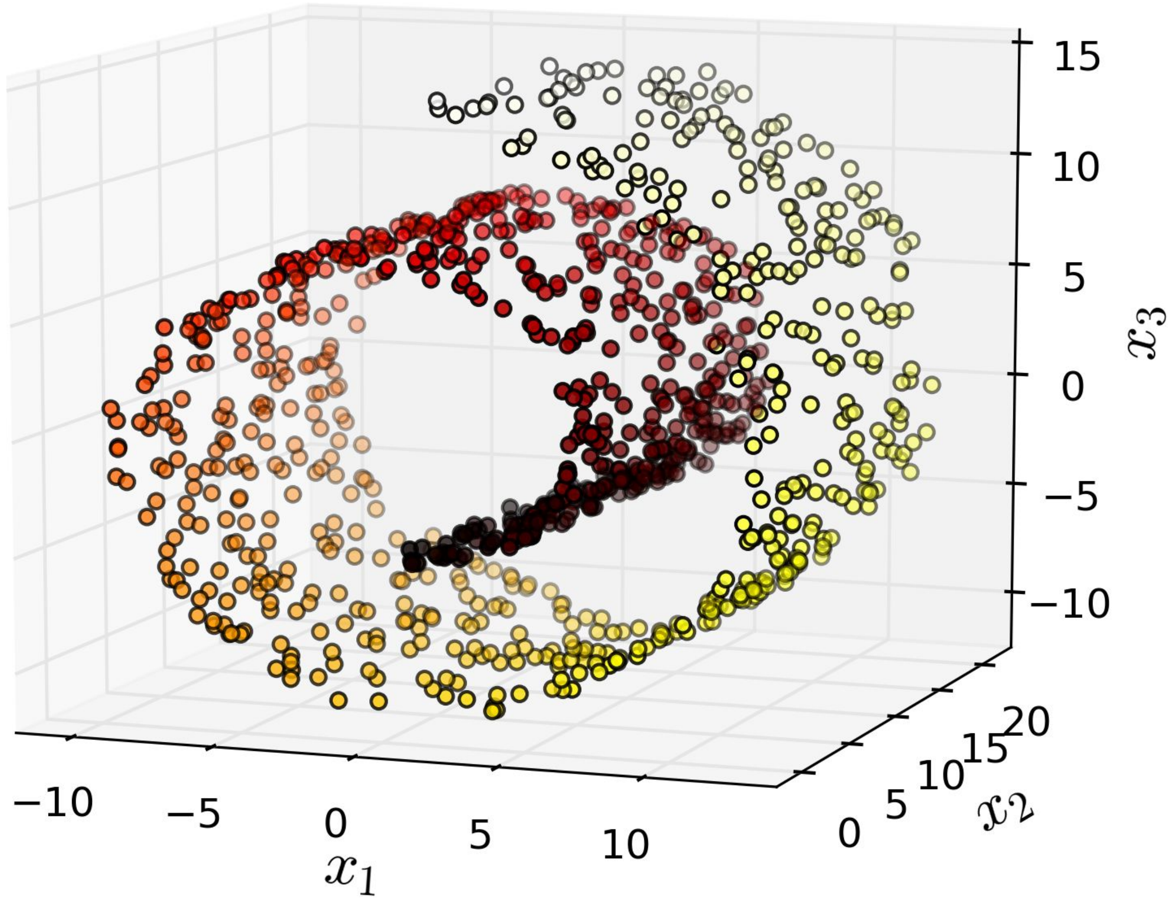
Dimensionality Reduction – Projection (cont'd)





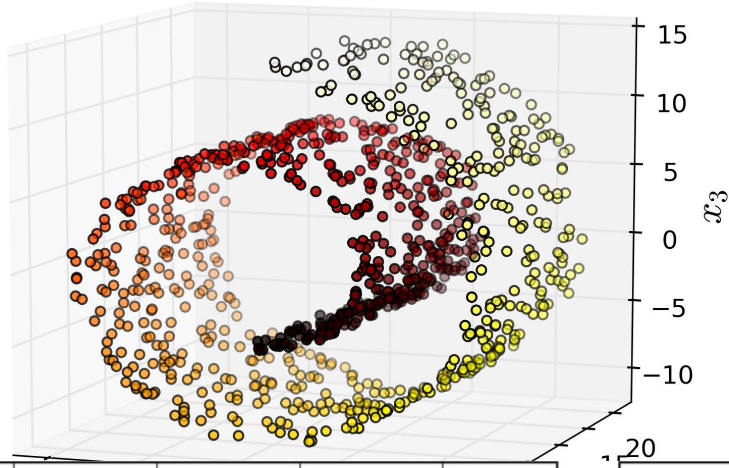
Reduction of feature space from 3 to 2

Dimensionality Reduction – Projection (cont’d)

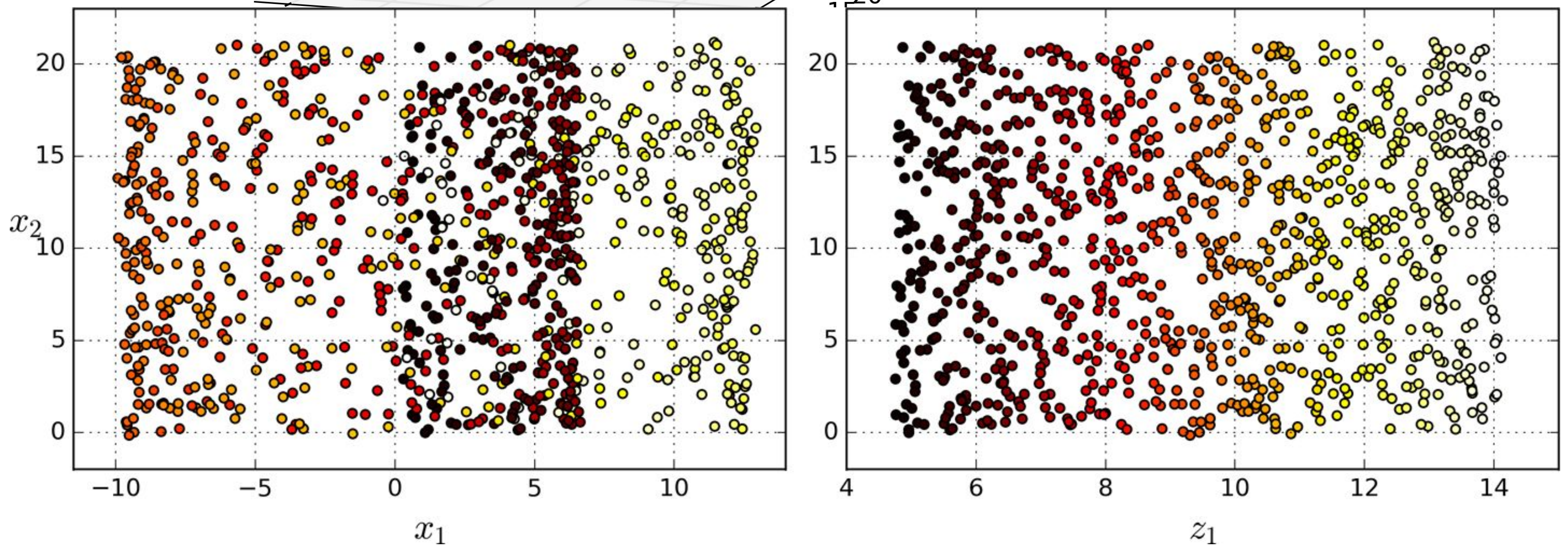


But… does it always work?

Dimensionality Reduction – Projection (cont'd)



projecting onto a plane would squash different layers of the Swiss roll together



UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

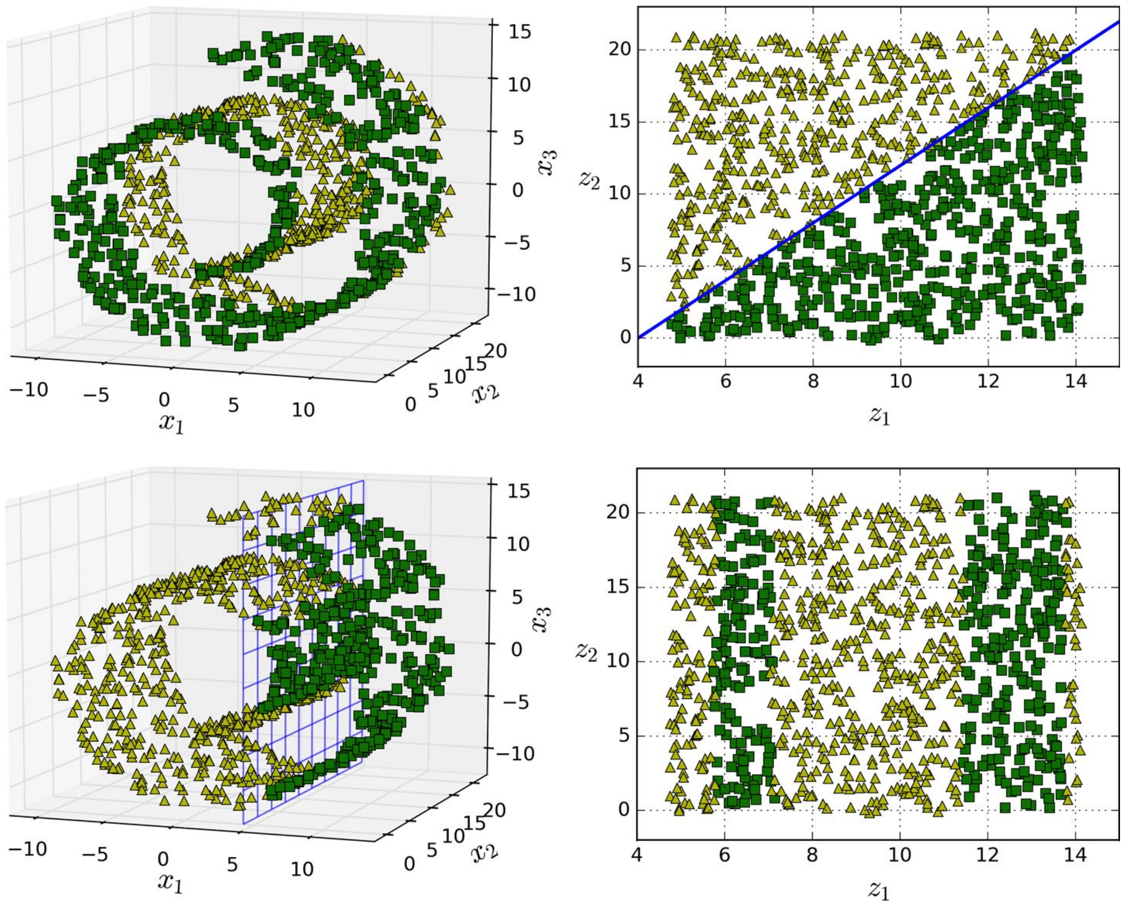
Module 8 – Section 3

Dimensionality Reduction using Manifolds

Manifold Learning

- A d-dimensional manifold is a part of an n-dimensional space (where $d < n$) that locally resembles a d-dimensional hyperplane.
  - In the case of the Swiss roll, $d = 2$ and $n = 3$: it locally resembles a 2D plane, but it is rolled in the third dimension.
- Most real-world high-dimensional datasets lie close to a much lower-dimensional manifold.
- The task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space of the manifold.

Manifold Learning (cont'd)



If you reduce the dimensionality of your training set before training a model, it will definitely speed up training,
However, it may not always lead to a better or simpler solution; it all depends on the dataset.

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

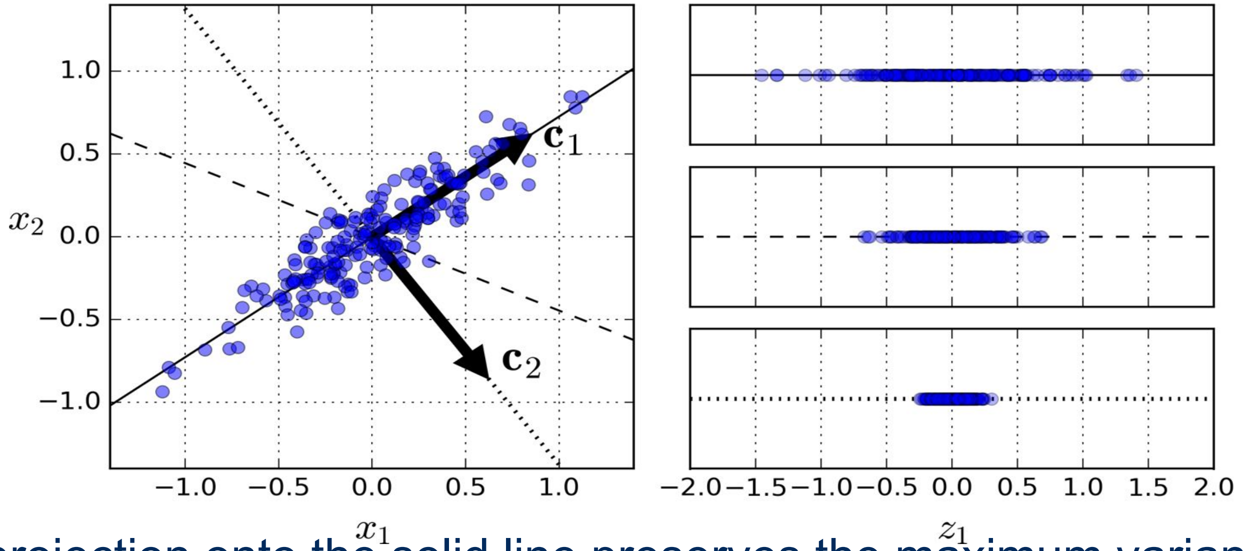
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 8 – Section 4

Principal Component Analysis (PCA)

PCA

- Principal Component Analysis (PCA)
- Is a **linear transformation** and assumes it can create a linear model of data (from N dimension to d)
- Vanilla PCA **fails** to model non-linear relationship
- First it identifies the hyperplane that lies closest to the data, and then it projects the data onto it.

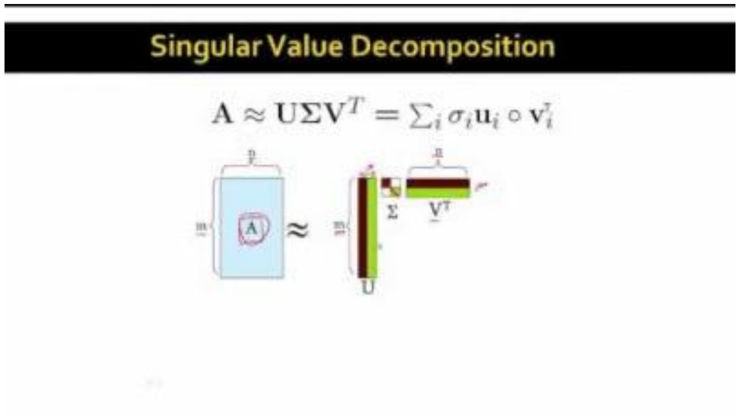


The projection onto the solid line preserves the maximum variance, while the projection onto the dotted line preserves very little variance.

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

PCA (cont'd)

- PCA Minimizes MSE: The main axis of PCA minimizes the mean squared distance between the original dataset and its projection onto that axis
- The unit vector that defines the i^th axis is called the i^th principal component (PC)
- How to find the PCs?
  - Use the matrix factorization technique called Singular Value Decomposition (SVD). SVD decomposes a matrix X into the product of three matrices $U \cdot \Sigma \cdot V^T$, where ***V*** **contains all the principal components** that we are looking for


Singular Value Decomposition
$\mathbf{A} \approx \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i$
n
m
A
≈
m
Σ
V^T
U

Projecting down n to d using PCA

- The dimensionality of the dataset can be reduced by projecting n dimensions onto the hyperplane defined by the first d principal components

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X} \cdot \mathbf{W}_d$$

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

W2 has the first 2 components from PCA

$\longleftarrow$ X2D is the data in 2 (lower) dimensions

PCA sklearn

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

components_: After fitting the PCA transformer to the dataset, you can access the principal components using the components_ variable

explained_variance_ratio_ variable: Indicates the proportion of the dataset’s variance that lies along the axis of each principal component.

```
>>> pca.explained_variance_ratio_
array([ 0.84248607,  0.14631839])
```

Choosing n-dimensions

Choose the number of dimensions that add up to a sufficiently large portion of the variance

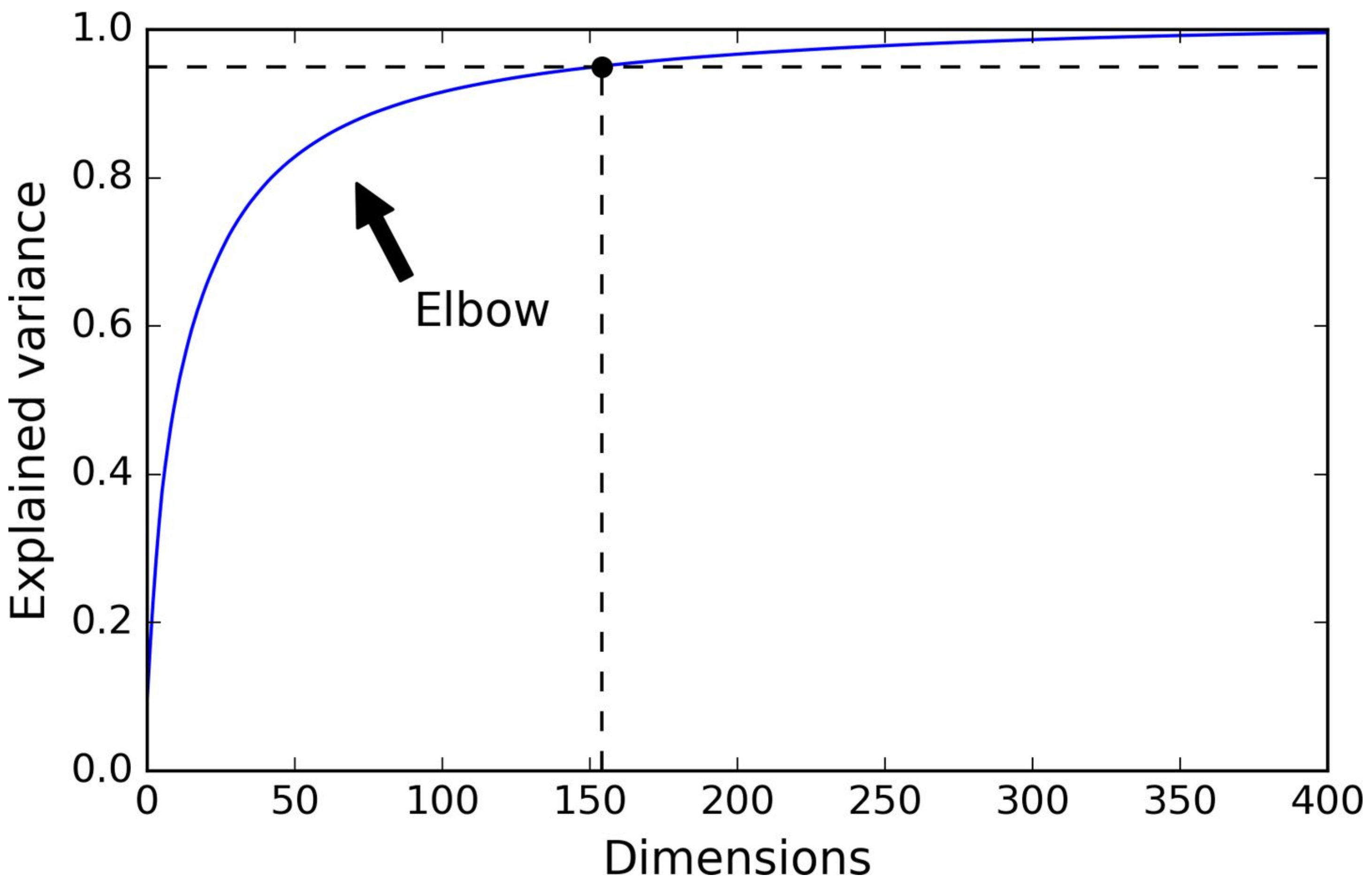
Example: Preserve 95% Variance

```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

Alternatively:

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

Choosing n-dimensions (cont'd)



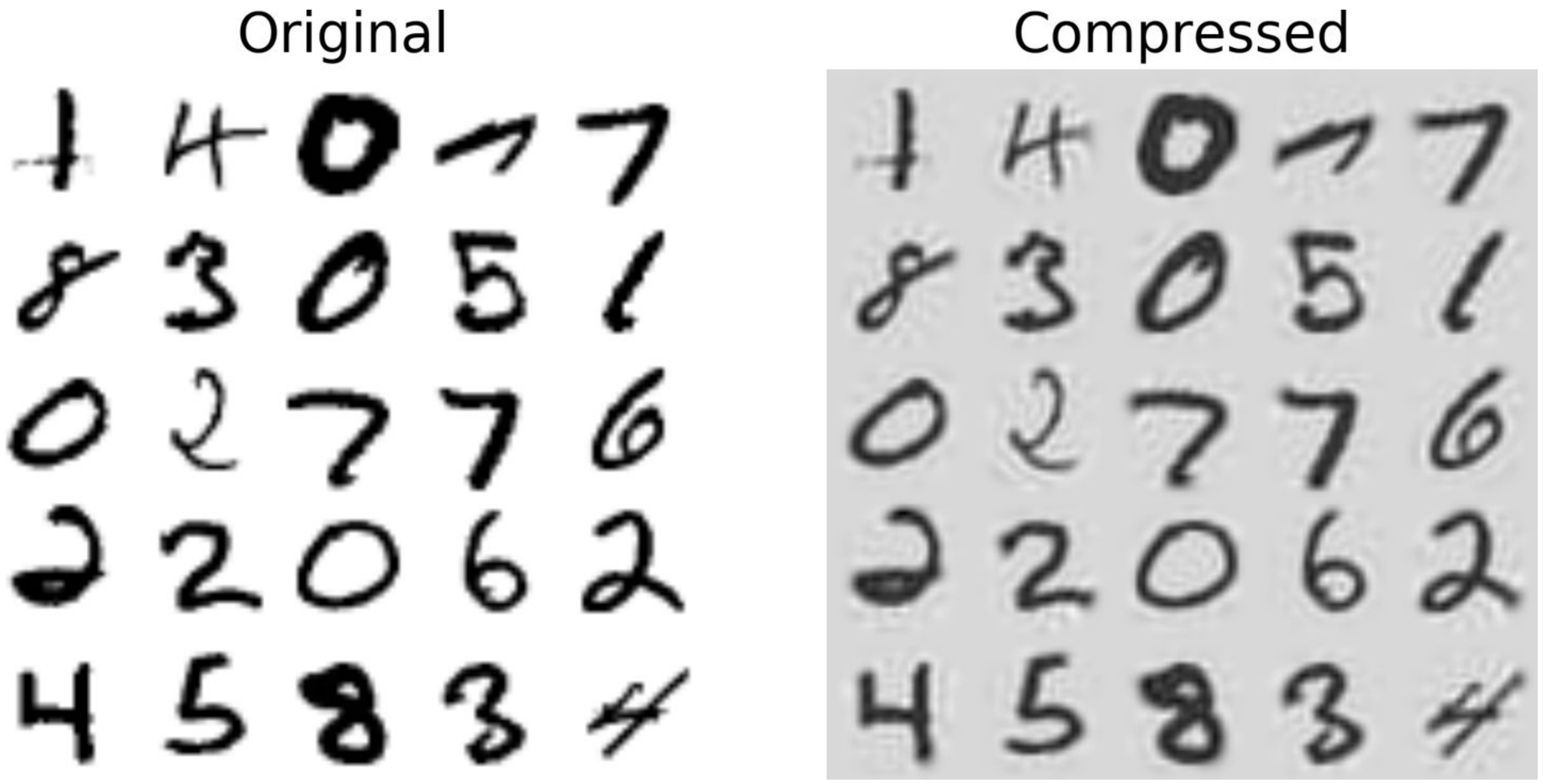
Hands-On: Lets see it in action here

PCA for Compression

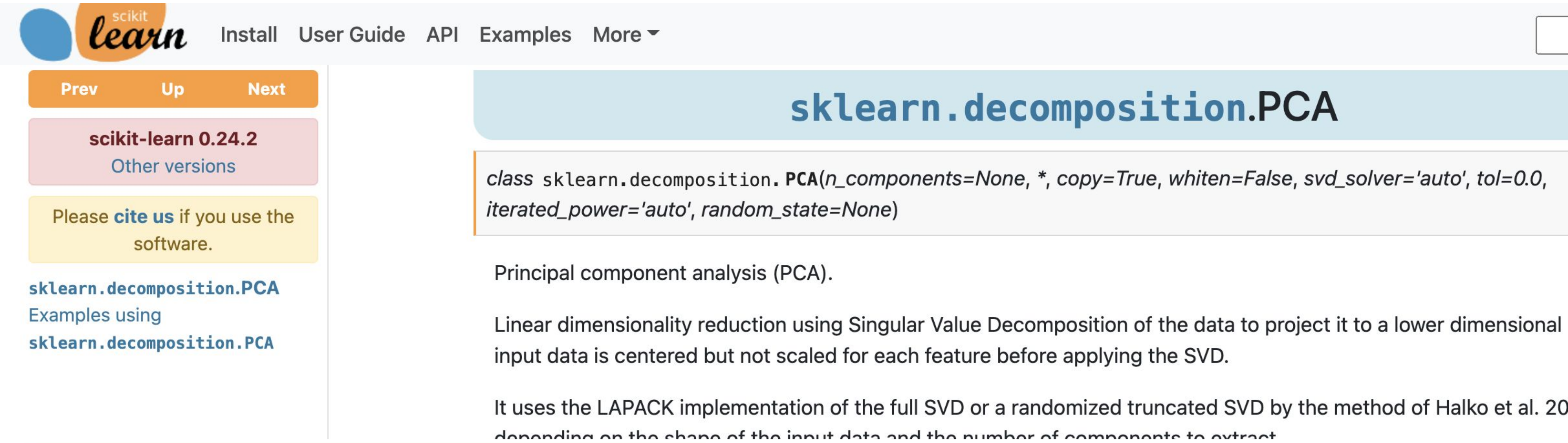
- Example: PCA on MNIST (95% variance):
  - 150 features, instead of the original 784 features
- The dataset is now less than 20% of its original size! This can speed up a classification algorithm (such as an SVM classifier) tremendously.

PCA for Compression (cont’d)

```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```



Flavors of PCA



scikit-learn 0.24.2
Other versions

Please cite us if you use the software.

sklearn.decomposition.PCA
Examples using sklearn.decomposition.PCA

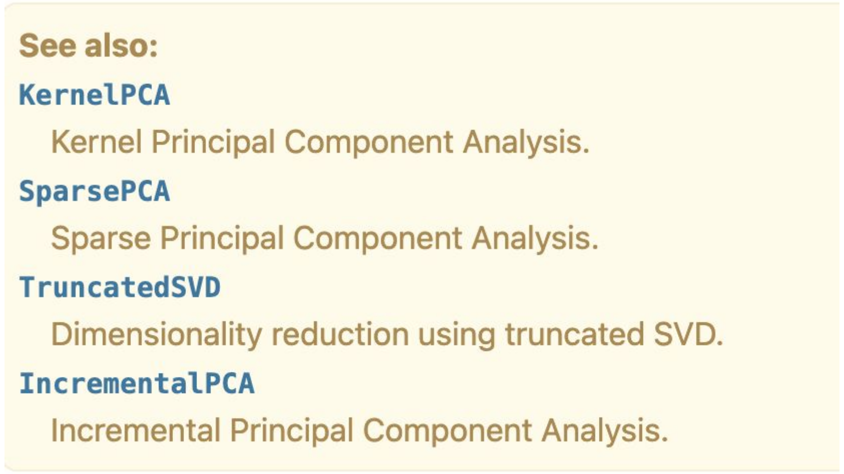
sklearn.decomposition.PCA

class `sklearn.decomposition.`**`PCA`**(*n_components=None*, *, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*)

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional
input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 20



See also:

`KernelPCA`
Kernel Principal Component Analysis.

`SparsePCA`
Sparse Principal Component Analysis.

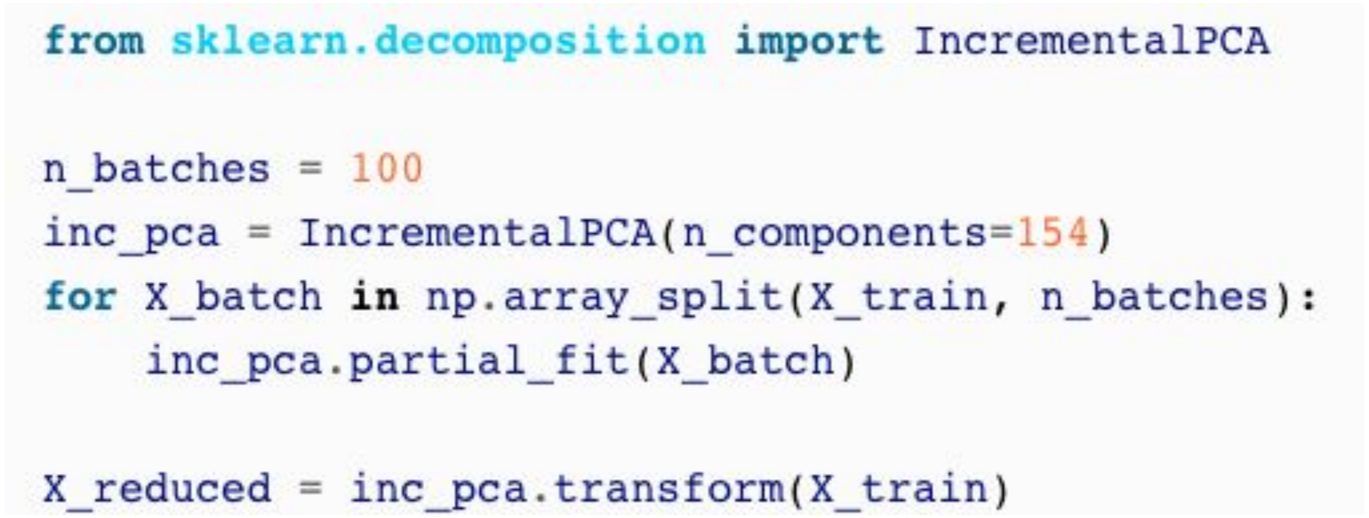
`TruncatedSVD`
Dimensionality reduction using truncated SVD.

`IncrementalPCA`
Incremental Principal Component Analysis.

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Incremental PCA

- Split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time
  - Useful for large datasets or online PCA



```python
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

Randomized PCA

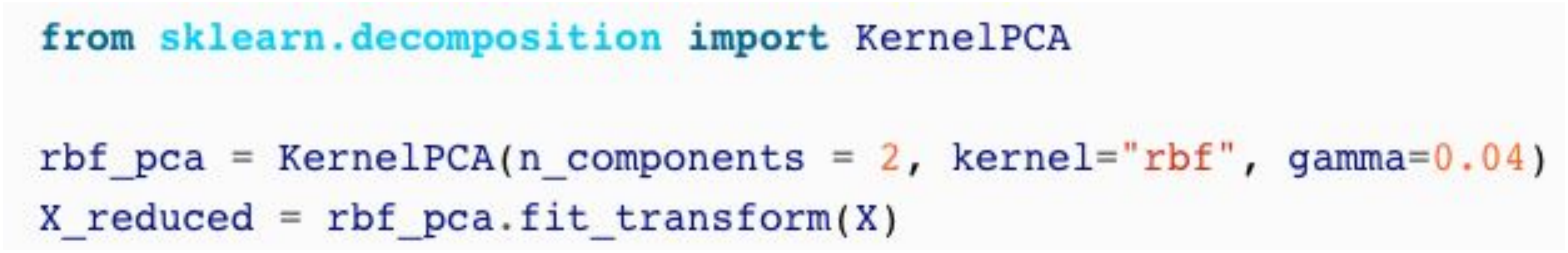
- Stochastic algorithm that quickly finds an approximation of the first d principal components (i.e., finds estimate for W)
  - Its computational complexity is O(m × d2) + O(d3), instead of O(m × n2) + O(n3), so it is dramatically faster than the previous algorithms when d is much smaller than n.

```
rnd_pca = PCA(n_components=154, svd_solver="randomized")
X_reduced = rnd_pca.fit_transform(X_train)
```

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X} \cdot \mathbf{W}_d$$

Kernel PCA

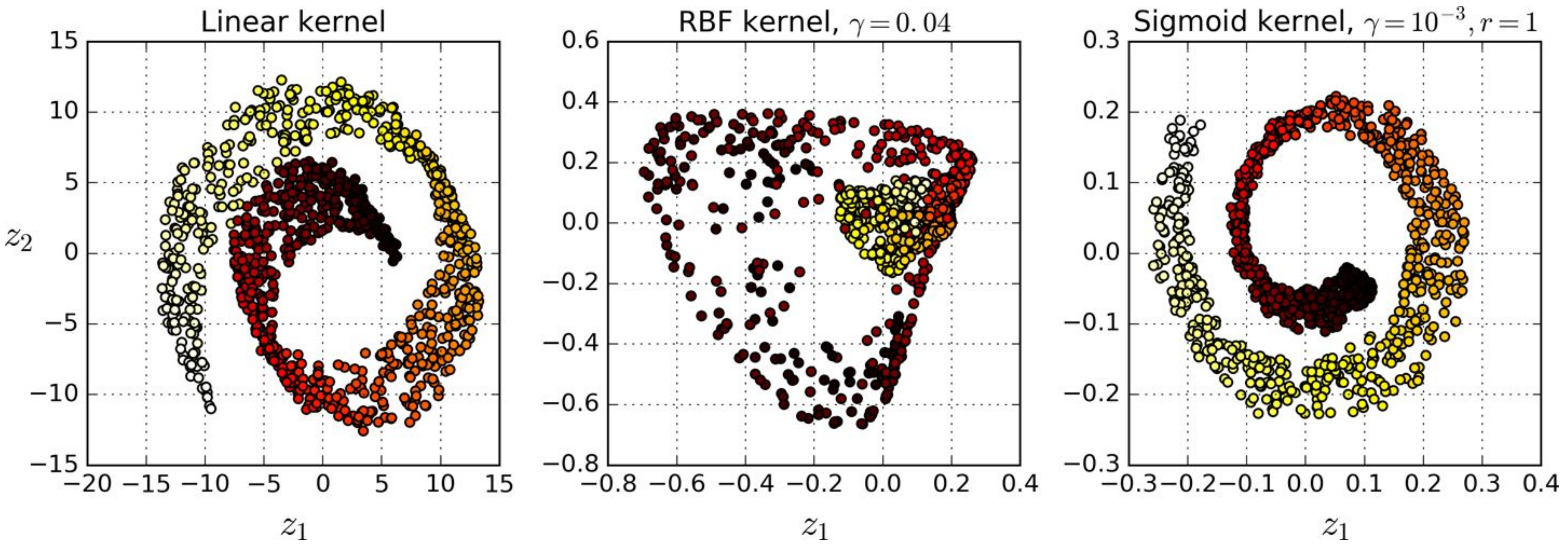
- Recall that a linear decision boundary in the high-dimensional feature space corresponds to a complex nonlinear decision boundary in the *original space*.
- It is often good at preserving clusters of instances after projection, or sometimes even unrolling datasets that lie close to a twisted manifold.



```python
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

Kernel PCA (cont'd)



Swiss roll, reduced to two dimensions using :

- linear kernel (equivalent to simply using the PCA class)
- RBF kernel
- sigmoid kernel (Logistic).

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Kernel PCA (cont'd)

```python
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

Selecting Kernel and Hyperparameters

- Dimensionality reduction is often a preparation step for a supervised learning task:
- Use a grid search to select the kernel and hyperparameters that lead to the best performance on that task. A simple workflow would be as below (this is an example):
  - Reduce dimensionality to two dimensions using kernel PCA
  - Apply Logistic Regression for classification.
  - Uses GridSearchCV to find the best kernel and gamma value for kPCA in order to get the best classification accuracy at the end of the pipeline

Selecting Kernel and Hyperparameters (cont’d)

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression())
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

```
>>> print(grid_search.best_params_)
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```

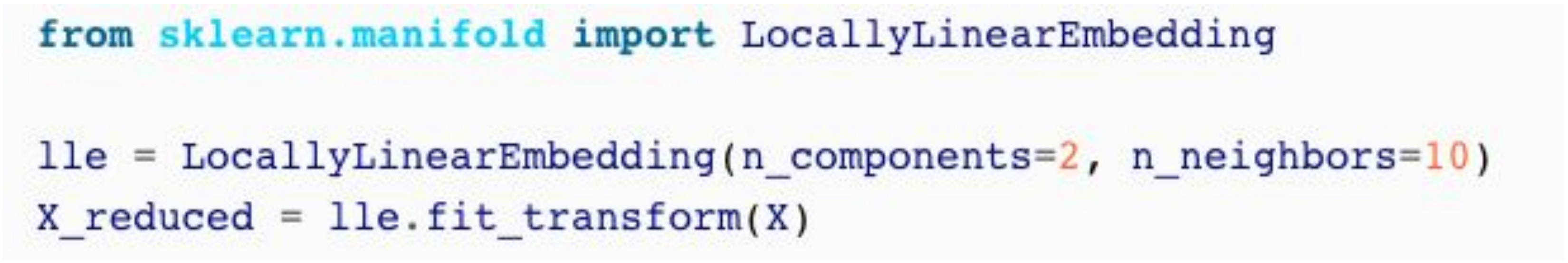
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 5

Local Linear Embedding (LLE)

Local Linear Embedding (LLE)

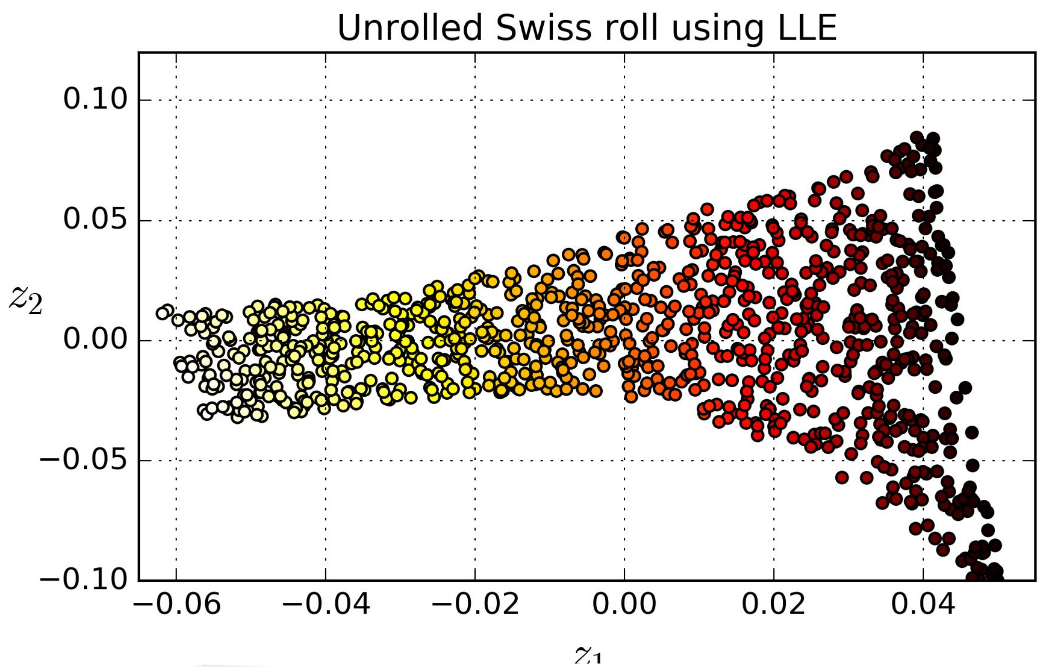
- LLE works by first measuring how each training instance linearly relates to its closest neighbors (c.n.), and then looking for a low-dimensional representation of the training set where these local relationships are best preserved
  - Particularly good at unrolling twisted manifolds, especially when there is not too much noise.

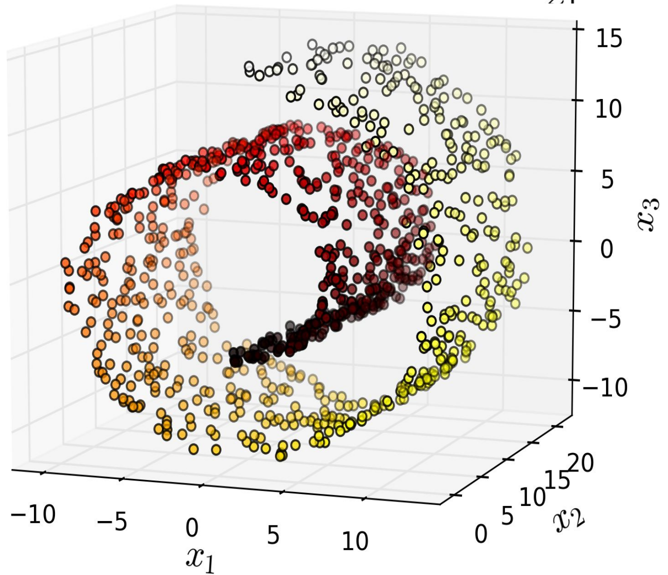


```python
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
X_reduced = lle.fit_transform(X)
```

Local Linear Embedding (cont'd)





The Swiss roll is completely unrolled and the distances between instances are locally well preserved.

However, compared to original roll (lower plot), distances are not preserved on a larger scale: the left part of the unrolled Swiss roll is squeezed, while the right part is stretched.

Local Linear Embedding (cont'd)

- For each training instance $\mathbf{x}^{(i)}$, the algorithm identifies its ***k* closest neighbors**, then tries to **reconstruct $\mathbf{x}^{(i)}$** as a **linear** function of these neighbors.
- It finds the weights $w_{i,j}$ such that the squared distance between $\boldsymbol{x}^{(i)}$ and

$$\sum_{j=1}^{m} w_{i,j}\mathbf{x}^{(j)}$$

is as small as possible, assuming $w_{i,j} = 0$ if $\boldsymbol{x}^{(j)}$ is not one of the k closest neighbors of $\boldsymbol{x}^{(\mathrm{i})}$

How LLE works – STEP 1

- In the first step, we calculate the weights that best represents the relationships between instances.

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{m} \| \mathbf{x}^{(i)} - \sum_{j=1}^{m} w_{i,j}\mathbf{x}^{(j)} \|^2$$

$$\text{subject to } \begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^{m} w_{i,j} = 1 & \text{for } i = 1, 2, \cdots, m \end{cases}$$

- $\hat{W}$ (W hat) will be the weights for such a representation

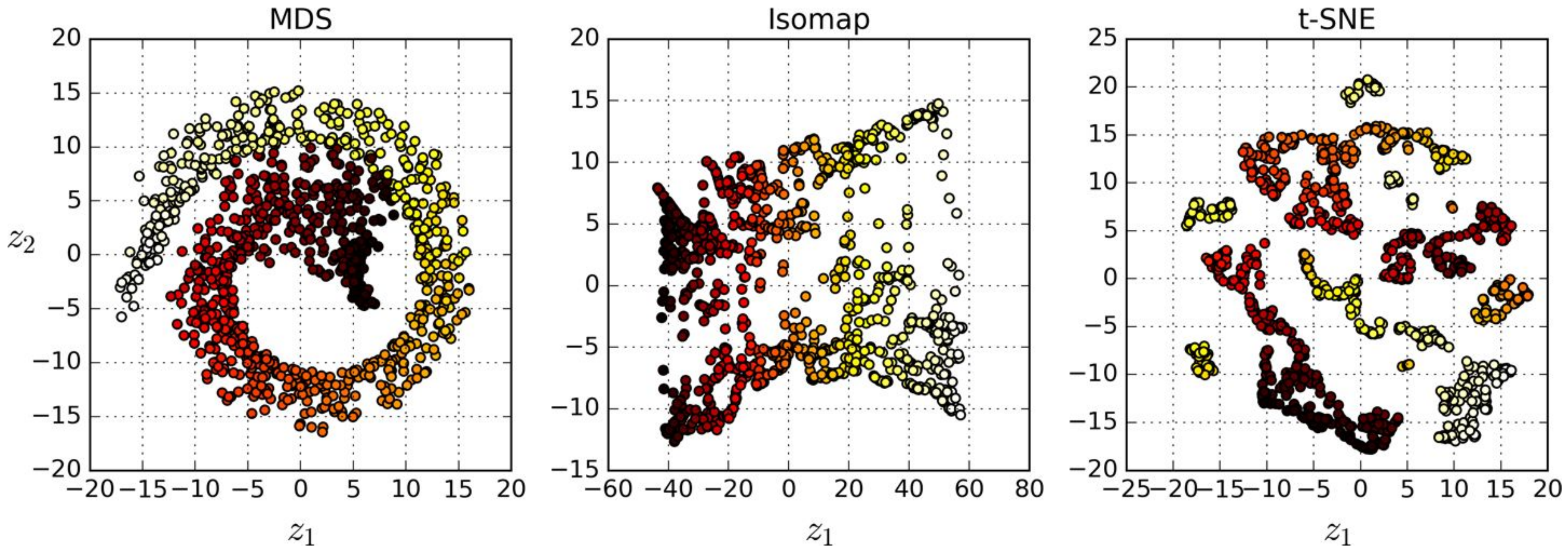
UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

How LLE works – STEP 2

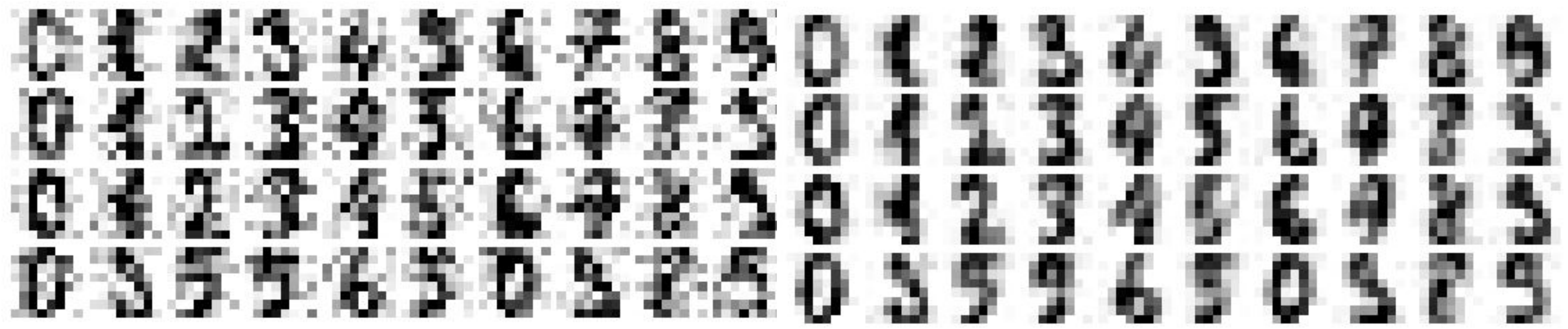
- The second step is to map the training instances into a d-dimensional space (where d < n) while preserving these local relationships as much as possible.
- If ***z***$^{(i)}$ is the image of ***x***$^{(i)}$ in this d-dimensional space, then we want the squared distance between $\sum_{j=1}^{m} \hat{w}_{i,j}\mathbf{z}^{(j)}$ to be as small as possible.

$$\hat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{i=1}^{m} \| \mathbf{z}^{(i)} - \sum_{j=1}^{m} \hat{w}_{i,j}\mathbf{z}^{(j)} \|^2$$

Other Methods



PCA for Noise Reduction



https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 9 – Section 6

Resources and Wrap-up

Resources

- Hands-On Machine Learning with Scikit-Learn and Tensorflow:
  - Chapter 8

Assessment

- See Jupyter Notebook

Next Class

- Introduction to Tensorflow

Follow us on social

Join the conversation with us online:

facebook.com/uoftscs

@uoftscs

linkedin.com/company/university-of-toronto-school-of-continuing-studies

@uoftscs

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies