High-Dimensional Data: Dimensionality Reduction

Techniques for Categorizing High-Dimensional Data — Part 1 of 2

Prof. Michael Mascagni

Department of Computer Science
Department of Mathematics
Department of Scientific Computing
Graduate Program in Molecular Biophysics
Florida State University, Tallahassee, FL 32306 **USA**
E-mail: mascagni@fsu.edu
URL: http://www.cs.fsu.edu/∼mascagni

Outline

What Is High-Dimensional Data?

Definition

Data is **high-dimensional** when the number of features (dimensions) d is large relative to the number of samples n. In many modern datasets, $d \gg n$.

Examples encountered in practice:

- **Images:** A 256×256 grayscale image has $d = 65{,}536$ dimensions
- **Genomics:** Gene expression arrays: $d \approx 20{,}000$ genes, $n \approx 100$ patients
- **NLP:** Bag-of-words vocabulary vectors: $d \approx 100{,}000$
- **Sensor networks:** Hundreds of correlated time-series channels
- **Finance:** Thousands of correlated asset returns

Core Challenge

Most learning algorithms assume data fills the feature space reasonably densely. In high dimensions, data becomes **sparse** and **distance metrics break down**.

The Curse of Dimensionality

Hughes Phenomenon (1968): Classification accuracy can *decrease* as dimensionality increases if the sample size is fixed.

Volume grows exponentially:

$$\text{Volume of unit hypercube in } \mathbb{R}^d = 1^d = 1 \text{ but Volume of inscribed ball} = \frac{\pi^{d/2}}{\Gamma(d/2+1)} \cdot \frac{1}{2^d} \to 0$$

The ball occupies a vanishing fraction of the cube as $d \to \infty$.

Distance concentration:

$$\frac{d_{\max} - d_{\min}}{d_{\min}} \to 0 \quad \text{as } d \to \infty$$

All pairwise distances become nearly equal — nearest-neighbor search loses meaning.

Practical consequences:

- Exponentially more samples needed to fill space
- Overfitting becomes severe
- Visualization is impossible without projection
- Redundant features add noise, not signal

Strategies for High-Dimensional Data

Dimensionality Reduction (This Lecture)

- **Linear:** PCA, SVD, LDA, ICA
- **Nonlinear:** t-SNE, UMAP, Isomap
- **Learned:** Autoencoders, VAE

Goal: Map $\mathbf{x} \in \mathbb{R}^d \to \mathbf{z} \in \mathbb{R}^k$ with $k \ll d$, preserving structure.

Feature Selection (complementary)

- Filter methods: variance, correlation
- Wrapper methods: recursive elimination
- Embedded: LASSO regularization

Goal: Retain a subset of original features, not a combination.

Guiding Principle

Real-world high-dimensional data often lies near a **low-dimensional manifold** embedded in $\mathbb{R}^d$. Dimensionality reduction finds and exploits this manifold.

PCA: Intuition and Goal

Goal

Find an orthogonal linear projection that maximizes the **variance** of the projected data, thereby capturing the most information in the fewest dimensions.

Setup:

- Data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, centered: $\bar{\mathbf{x}} = \mathbf{0}$
- Seek unit vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ (principal components)
- First PC: $\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|=1} \mathbf{w}^\top \mathbf{S} \mathbf{w}$
- Sample covariance: $\mathbf{S} = \frac{1}{n-1} \mathbf{X}^\top \mathbf{X}$

Solution via eigen-decomposition:

$$\mathbf{S}\mathbf{w}_i = \lambda_i \mathbf{w}_i, \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$$

The k eigenvectors with largest eigenvalues form the projection matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$.

PCA: Mathematics

Step-by-step algorithm:

1. Center the data: $\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^\top$
2. Compute covariance: $\mathbf{S} = \frac{1}{n-1}\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$
3. Eigen-decompose: $\mathbf{S} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top$
4. Select top-k eigenvectors: $\mathbf{W}_k \in \mathbb{R}^{d\times k}$
5. Project: $\mathbf{Z} = \tilde{\mathbf{X}}\mathbf{W}_k \in \mathbb{R}^{n\times k}$

Explained variance ratio:

$$\text{EVR}(k) = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$$

Choose k such that EVR ≥ 0.90 (or by scree plot elbow).

Reconstruction Error

$$\|\mathbf{X} - \mathbf{Z}\mathbf{W}_k^\top\|_F^2 = \sum_{}^{d}\lambda_i$$

PCA: Python Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler

# Load 64-dimensional handwritten digit dataset
digits = load_digits()
X, y = digits.data, digits.target    # X: (1797, 64)

# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit PCA
pca = PCA()
pca.fit(X_scaled)
```

PCA: Python Example (Cont.)

```python
# Plot cumulative explained variance
cum_var = np.cumsum(pca.explained_variance_ratio_)
plt.plot(range(1, len(cum_var)+1), cum_var)
plt.axhline(0.90, color='r', linestyle='--', label='90% threshold')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Scree Plot -- Digits Dataset')
plt.legend(); plt.tight_layout(); plt.show()

# Project to 2D for visualization
pca2 = PCA(n_components=2)
Z = pca2.fit_transform(X_scaled)
print(f"EVR (2 components): {sum(pca2.explained_variance_ratio_):.3f}")
```

```
EVR (2 components): 0.286
# ~29% of variance in 2 PCs (64 -> 2 is aggressive)
# 90% threshold reached at ~21 components
```

PCA: Strengths, Weaknesses, and Summary

Strengths

- Closed-form, globally optimal linear solution
- Computationally efficient: $O(d^2 n + d^3)$
- Interpretable: each PC is a linear combination
- Removes correlation between features
- Well-understood theory

Weaknesses

- Linear only — misses curved manifolds
- Sensitive to outliers
- Assumes Gaussian-like, variance-capturing structure
- PCs not always semantically meaningful
- Covariance matrix expensive for $d \gg n$

When to Use PCA

Preprocessing step before classification; noise removal; visualization of linear structure; whitening for downstream algorithms. Default first choice for linear dimensionality reduction.

SVD: Definition and Connection to PCA

Theorem (SVD)

Every matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ factors as:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}$ (left singular vectors), $\mathbf{\Sigma} \in \mathbb{R}^{n \times d}$ (diagonal, $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$), $\mathbf{V} \in \mathbb{R}^{d \times d}$ (right singular vectors).

Connection to PCA:

- Right singular vectors $\mathbf{V}$ = eigenvectors of $\mathbf{X}^\top\mathbf{X}$ (the PCA directions)
- $\lambda_i = \sigma_i^2/(n-1)$ (eigenvalues of $\mathbf{S}$)
- PCA scores: $\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}$
- SVD is numerically more stable than forming $\mathbf{X}^\top\mathbf{X}$ explicitly

Truncated SVD (rank-k approximation):

$$\mathbf{X} \approx \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top \quad \text{(best rank-}k\text{ approximation by Eckart–Young theorem)}$$

SVD: Applications in High-Dimensional Data

Latent Semantic Analysis (LSA):

- Term-document matrix $\mathbf{X} \in \mathbb{R}^{n_{\text{docs}} \times n_{\text{terms}}}$
- Truncated SVD reveals latent topics
- Reduces synonymy and polysemy noise

Image Compression:

$$\text{Storage: } n \cdot d \quad \rightarrow \quad k(n + d + 1) \text{ values}$$

Rank-50 approximation of a 512×512 image retains most visual structure.

Randomized SVD (Halko et al., 2011):

- For very large matrices, exact SVD is $O(nd \min(n, d))$
- Randomized algorithm achieves rank-k in $O(nd \log k)$
- Used in scikit-learn's `TruncatedSVD` and `PCA(svd_solver='randomized')`

Key Insight

SVD reveals the **intrinsic rank** of data. If $r \ll \min(n, d)$, the data lives on a low-dimensional linear subspace.

SVD: Python Example — Image Compression

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_sample_image

# Load sample image and convert to grayscale float
china = load_sample_image('china.jpg')
X = china[:, :, 0].astype(float)  # shape (427, 640)

# Full SVD
U, s, Vt = np.linalg.svd(X, full_matrices=False)

# Reconstruct at various ranks
fig, axes = plt.subplots(1, 4, figsize=(14, 4))
for ax, k in zip(axes, [5, 20, 50, 200]):
    X_k = U[:, :k] @ np.diag(s[:k]) @ Vt[:k, :]
    ax.imshow(X_k, cmap='gray')
    compression = k*(427+640+1) / (427*640)
    ax.set_title(f'rank={k}\n({compression:.1%} data)')
    ax.axis('off')
plt.suptitle('Rank-k SVD Image Compression')
plt.tight_layout(); plt.show()
```

SVD: Python Example — Image Compression (Cont.)

```python
# Explained variance
var_explained = np.cumsum(s**2) / np.sum(s**2)
k90 = np.searchsorted(var_explained, 0.90) + 1
print(f"90% variance explained at rank k = {k90}")
```

```
90% variance explained at rank k = 37
# Image has 427 x 640 = 273,280 pixels but only rank ~37 needed for 90% fidelity
```

LDA: Supervised Dimensionality Reduction

Motivation

PCA and SVD are **unsupervised**: they ignore class labels. LDA is **supervised**: it finds directions that *maximize class separability*.

Fisher's LDA Criterion:

Define:

- $\mathbf{S}_W$ = **within-class scatter**: $\mathbf{S}_W = \sum_c \sum_{\mathbf{x} \in C_c} (\mathbf{x} - \boldsymbol{\mu}_c)(\mathbf{x} - \boldsymbol{\mu}_c)^\top$
- $\mathbf{S}_B$ = **between-class scatter**: $\mathbf{S}_B = \sum_c n_c (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^\top$

Maximize the Rayleigh quotient:

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}}$$

Solution: Generalized eigenvalue problem $\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$. At most $C - 1$ nonzero eigenvalues for C classes.

LDA vs PCA: Key Differences

Property	**PCA**	**LDA**
Type	Unsupervised	Supervised
Criterion	Max variance	Max class separation
Uses labels	No	Yes
Max components	$\min(n, d) - 1$	$C - 1$
Assumption	None	Gaussian classes, equal covariance
Optimal for	Reconstruction	Classification
Sensitive to	Outliers	Class imbalance

Important Constraint

LDA reduces to at most $C - 1$ dimensions (C = number of classes). For binary classification: only **1 discriminant axis** is available. LDA is a classification algorithm as well as a reduction technique.

LDA: Python Example

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score

# Iris dataset: 4 features, 3 classes
X, y = load_iris(return_X_y=True)

# LDA: reduce to C-1 = 2 components
lda = LinearDiscriminantAnalysis(n_components=2)
Z = lda.fit_transform(X, y)
```

LDA: Python Example (Cont.)

```python
# Visualize
colors = ['steelblue', 'tomato', 'forestgreen']
labels = load_iris().target_names
fig, ax = plt.subplots(figsize=(7, 5))
for c, label in enumerate(labels):
    mask = (y == c)
    ax.scatter(Z[mask, 0], Z[mask, 1],
               color=colors[c], label=label, alpha=0.8)
ax.set_xlabel('LD1'); ax.set_ylabel('LD2')
ax.set_title('LDA on Iris Dataset')
ax.legend(); plt.tight_layout(); plt.show()

# Classification accuracy via cross-validation
lda_clf = LinearDiscriminantAnalysis()
scores = cross_val_score(lda_clf, X, y, cv=5)
print(f"LDA 5-fold CV accuracy: {scores.mean():.3f} +/- {scores.std():.3f}")
```

```
LDA 5-fold CV accuracy: 0.980 +/- 0.013
# Classes are nearly linearly separable in LDA space
```

Nonlinear Dimensionality Reduction: Why?

Linear methods fail on curved manifolds:

- Swiss roll: a 2D sheet embedded in 3D — PCA cannot unroll it
- Handwritten digits: digit classes form curved clusters in pixel space
- Word embeddings: semantic relationships are nonlinear

Manifold Hypothesis:

High-dimensional real-world data concentrates near a smooth, low-dimensional manifold $\mathcal{M} \subset \mathbb{R}^d$.

Nonlinear methods (overview):

Method	Preserves	Best For
Isomap	Geodesic distances	Smooth manifolds
LLE	Local geometry	Convex neighborhoods
t-SNE	Local neighborhoods	Cluster visualization
UMAP	Local + global topology	General purpose

t-SNE: t-Distributed Stochastic Neighbor Embedding

Core Idea (van der Maaten & Hinton, 2008): Represent pairwise *similarities* in high-d and low-d space, then minimize their KL divergence.

Step 1 — High-dimensional similarities:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)}, \qquad p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

Step 2 — Low-dimensional similarities (Student-t with 1 d.f.):

$$q_{ij} = \frac{(1 + \|\mathbf{z}_i - \mathbf{z}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\mathbf{z}_k - \mathbf{z}_l\|^2)^{-1}}$$

Step 3 — Minimize KL divergence via gradient descent:

$$\mathcal{L} = \mathrm{KL}(P\|Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Why Student-t in low-d?

Heavy tails allow dissimilar points to be placed far apart, preventing the **crowding problem**

UMAP: Uniform Manifold Approximation and Projection

Core Idea (McInnes et al., 2018): Model data as a weighted k-nearest-neighbor graph and find a low-dimensional graph that is as topologically similar as possible.

High-level algorithm:

1. Build a fuzzy simplicial complex from k-NN graph in $\mathbb{R}^d$
2. Initialize low-dimensional embedding (e.g., via spectral methods)
3. Optimize cross-entropy between high-d and low-d fuzzy sets

Property	**t-SNE**	**UMAP**
Speed	Slow $O(n^2)$ / $O(n \log n)$	Fast $O(n^{1.14})$
Global structure	**Poor**	**Good**
Cluster separation	**Excellent**	**Good**
Scalability	$\leq$ 10,000 (practical)	Millions of points
Reproducibility	Stochastic	Stochastic (seedable)
Theory	Information-theoretic	Topological/Riemannian

t-SNE and UMAP: Python Comparison

```python
from sklearn.manifold import TSNE
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
import umap              # pip install umap-learn
import matplotlib.pyplot as plt

X, y = load_digits(return_data=True)
X_s = StandardScaler().fit_transform(X.data)

# t-SNE
tsne = TSNE(n_components=2, perplexity=30,
            random_state=42, n_iter=1000)
Z_tsne = tsne.fit_transform(X_s)
```

t-SNE and UMAP: Python Comparison (Cont.)

```python
# UMAP
reducer = umap.UMAP(n_components=2, n_neighbors=15,
                    min_dist=0.1, random_state=42)
Z_umap = reducer.fit_transform(X_s)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for ax, Z, title in zip(axes,
    [Z_tsne, Z_umap], ['t-SNE', 'UMAP']):
    sc = ax.scatter(Z[:,0], Z[:,1], c=y.target,
                    cmap='tab10', s=5, alpha=0.8)
    ax.set_title(title); ax.axis('off')
plt.colorbar(sc, ax=axes[-1], label='Digit class')
plt.tight_layout(); plt.show()
```

Autoencoders: Learned Nonlinear Reduction

Definition

An **autoencoder** is a neural network trained to reconstruct its input through a low-dimensional *bottleneck* layer (latent code).

Architecture:

$$\underbrace{\mathbf{x} \in \mathbb{R}^d}_{\text{input}} \xrightarrow{\text{encoder } f_\theta} \underbrace{\mathbf{z} \in \mathbb{R}^k}_{\text{latent code } k \ll d} \xrightarrow{\text{decoder } g_\phi} \underbrace{\hat{\mathbf{x}} \in \mathbb{R}^d}_{\text{reconstruction}}$$

Training objective:

$$\mathcal{L}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - g_\phi(f_\theta(\mathbf{x}_i))\|^2$$

Nonlinear PCA: With linear activations and k hidden units, an autoencoder recovers exactly the same subspace as PCA. Nonlinear activations (ReLU, sigmoid) learn *curved* manifolds.

Autoencoder: Python Example (PyTorch)

```python
import torch, torch.nn as nn, torch.optim as optim
from sklearn.datasets import load_digits
from sklearn.preprocessing import MinMaxScaler
import numpy as np

X = MinMaxScaler().fit_transform(load_digits().data)  # (1797, 64)
X_t = torch.tensor(X, dtype=torch.float32)

class Autoencoder(nn.Module):
    def __init__(self, d=64, k=8):
        super().__init__()
        self.enc = nn.Sequential(
            nn.Linear(d, 32), nn.ReLU(),
            nn.Linear(32, k))
        self.dec = nn.Sequential(
            nn.Linear(k, 32), nn.ReLU(),
            nn.Linear(32, d), nn.Sigmoid())
    def forward(self, x):
        return self.dec(self.enc(x))
```

Autoencoder: Python Example (PyTorch) (Cont.)

```python
model = Autoencoder(d=64, k=8)
opt   = optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.MSELoss()

for epoch in range(300):
    loss = loss_fn(model(X_t), X_t)
    opt.zero_grad(); loss.backward(); opt.step()

with torch.no_grad():
    Z = model.enc(X_t).numpy()       # (1797, 8) latent codes
print(f"Final reconstruction loss: {loss.item():.4f}")
```

```
Final reconstruction loss: 0.0083
# 64 -> 8 dimensions with <1% reconstruction error
```

Method Comparison Table

Method	Type	Supervised?	Scales to	Preserves	Best Use
PCA	Linear	No	Millions	Global variance	Preprocessing, noise removal
SVD	Linear	No	Very large	Global structure	LSA, image compression
LDA	Linear	Yes	Thousands	Class separation	Classification preprocessing
t-SNE	Nonlinear	No	$\leq$50k	Local clusters	2D/3D visualization
UMAP	Nonlinear	No	Millions	Local + global	General visualization
AE	Deep/Nonlinear	No	Millions	Task-specific	Generative, anomaly detection

Computational complexity:

Method	Fit Time	Transform Time
PCA (exact)	$O(nd^2 + d^3)$	$O(ndk)$
PCA (randomized)	$O(ndk)$	$O(ndk)$
LDA	$O(nd^2 + d^3)$	$O(ndk)$
t-SNE	$O(n^2)$ / $O(n \log n)$	No out-of-sample
UMAP	$O(n^{1.14})$	$O(n_{\text{new}})$

Decision Guide: Which Method to Use?

Start with these questions:

1. Do you have class labels? $\rightarrow$ **LDA** for $C \leq 20$ classes
2. Is the goal visualization (2D/3D)? $\rightarrow$ **UMAP** (fast, preserves global) or **t-SNE** (sharp clusters)
3. Is interpretability required? $\rightarrow$ **PCA** (each axis is a linear combination)
4. Is the data a term-document or image matrix? $\rightarrow$ **SVD / TruncatedSVD**
5. Is the manifold highly nonlinear and data large? $\rightarrow$ **UMAP** or **Autoencoder**
6. Do you need out-of-sample transforms? $\rightarrow$ Avoid t-SNE; prefer PCA, UMAP, or AE

Practical Recommendation

Always run PCA first as a baseline and to whiten/denoise. Then apply UMAP for visualization or a supervised method for downstream classification. Use autoencoders when labelled data is scarce and representation learning is needed.

Summary of Part 1

Key Takeaways:

- The **curse of dimensionality** makes high-d data sparse and distance-based methods unreliable
- **PCA**: maximize variance via covariance eigenvectors; fast, interpretable, linear
- **SVD**: equivalent to PCA; numerically superior; underpins LSA and compression
- **LDA**: maximize class separation with $\mathbf{S}_W^{-1}\mathbf{S}_B$; supervised; at most $C-1$ components
- **t-SNE**: probabilistic; local neighborhood preservation; excellent 2D cluster visualization
- **UMAP**: topological; faster and more scalable than t-SNE; better global structure
- **Autoencoders**: neural, nonlinear; flexible bottleneck; no linear assumption

Part 2 Preview

Classification and Clustering in High-Dimensional Spaces: k-NN, SVM, Random Forests, k-Means, DBSCAN, Gaussian Mixture Models, Spectral Clustering, and evaluation metrics.

References and Further Reading

- Jolliffe, I.T., *Principal Component Analysis*, 2nd ed., Springer, 2002.
- Halko, N., Martinsson, P.G., Tropp, J.A., "Finding structure with randomness," *SIAM Review*, 53(2), 2011.
- van der Maaten, L., Hinton, G., "Visualizing data using t-SNE," *JMLR*, 9, 2008.
- McInnes, L., Healy, J., Melville, J., "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," arXiv:1802.03426, 2018.
- Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning*, MIT Press, 2016. Ch. 14 (Autoencoders).
- Bellman, R., *Adaptive Control Processes*, Princeton, 1961. (Origin of "curse of dimensionality")
- scikit-learn documentation: `scikit-learn.org`

Questions?

Abbreviations and Acronyms — Part 1

Dimensionality Reduction Methods

Acronym	Expansion
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
LDA	Linear Discriminant Analysis
ICA	Independent Component Analysis
t-SNE	t-distributed Stochastic Neighbor Embedding
SNE	Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection
AE	Autoencoder
VAE	Variational Autoencoder
MDS	Multidimensional Scaling
LLE	Locally Linear Embedding
NMF	Non-negative Matrix Factorization

Linear Algebra & Statistics

Acronym	Expansion
PC	Principal Component
EVR	Explained Variance Ratio
EVD	Eigenvalue Decomposition
MSE	Mean Squared Error
KL	Kullback–Leibler (divergence)
KDE	Kernel Density Estimation
CLT	Central Limit Theorem
IID	Independent and Identically Distributed
LSA	Latent Semantic Analysis
BoW	Bag of Words
PDF	Probability Density Function

Abbreviations and Acronyms — Part 2

Machine Learning & Neural Networks

Acronym	Expansion
ML	Machine Learning
k-NN	k-Nearest Neighbors
SVM	Support Vector Machine
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
MLP	Multi-Layer Perceptron
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
LASSO	Least Absolute Shrinkage and Selection Operator

Datasets, Tools & General

Acronym	Expansion
MNIST	Modified NIST digit dataset
NLP	Natural Language Processing
API	Application Programming Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
CS	Computer Science
FSU	Florida State University
2D	Two-Dimensional
3D	Three-Dimensional
d	Number of original features
k	Number of reduced dimensions
n	Number of data samples

Abbreviations and Acronyms — Part 3

Note on LDA

LDA in this presentation refers to **Linear Discriminant Analysis** (supervised dimensionality reduction), *not* Latent Dirichlet Allocation (an NLP topic model).