Curse of Dimensionality

- The **curse of dimensionality** refers to various phenomena that arise when analyzing data in high-dimensional spaces, typically with many features or variables.
- As the number of dimensions grows, several issues emerge, making it harder to effectively analyze, visualize, and draw conclusions from the data.
- The difficulties related to training machine learning models due to high dimensional data is referred to as ‘Curse of Dimensionality’.

Some problem sets may have

- Large number of features
- Making the model extremely slow
- Even making it difficult to find a solution
- This is referred to as '**Curse of Dimensionality**'
- Fortunately, it is often possible to reduce the number of features considerably, turning an intractable problem into a tractable one.

The Curse of Dimensionality

Example: MNIST Dataset

- (28*28) number of features for each image.
- Border features had no importance and could be ignored.
- Neighbouring pixels are highly correlated
  - They can be merged into one (taking mean intensity) without losing much of information, further reducing the dimensions or features.

Approaches to Mitigating the Curse of Dimensionality

1. **Dimensionality Reduction Techniques**:
    - **Principal Component Analysis (PCA)**: Transforms data to a lower-dimensional space by finding the directions (principal components) of maximum variance. Often used to reduce noise and redundancy.
    - **t-Distributed Stochastic Neighbor Embedding (t-SNE)**: A non-linear technique useful for visualizing high-dimensional data in 2 or 3 dimensions, often for exploratory data analysis.
    - **Linear Discriminant Analysis (LDA)**: Similar to PCA but supervised, aiming to find feature space that maximizes class separability.
    - **Autoencoders**: Neural networks that learn an efficient, compressed representation of data, useful for non-linear dimensionality reduction.
2. **Feature Selection**:
    - **Filter Methods**: Select features based on statistical measures (e.g., correlation, chi-square test) without involving a predictive model.

- **Embedded Methods**: Integrate feature selection directly into model training (e.g., Lasso regression).

3. **Regularization**:
   - Techniques like **L1 (Lasso) and L2 (Ridge) regularization** in machine learning penalize large coefficients, effectively reducing the impact of less important features and mitigating overfitting in high-dimensional settings.
4. **Random Projections**:
   - Projects high-dimensional data onto a lower-dimensional subspace using random matrices while approximately preserving distances. This approach is computationally efficient and often surprisingly effective.
5. **Data Sampling or Aggregation**:
   - Reducing the dimensionality by aggregating or summarizing data can sometimes simplify high-dimensional data without significant information loss.
   - Clustering or binning can also help reduce the dimensionality, especially in cases of categorical features with many levels.

1. Conceptual Steps of PCA

1. Standardize the Dataset
2. Compute the Covariance Matrix
3. Compute Eigenvalues and Eigenvectors
4. Sort Eigenvalues and Select Principal Components
5. Transform the Dataset

Step 1: Standardize the Dataset

Standardization ensures that all features contribute equally to the analysis by scaling them to have zero mean and unit variance.

For a dataset $\mathbf{X}$ with n samples and m features:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

Where:

- x_{ij}: The i-th observation of the j-th feature.
- $\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$: Mean of the j-th feature.
- $\sigma_j = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \mu_j)^2}$: Standard deviation of the j-th feature.

Result: Standardized data matrix $\mathbf{Z}$. ↓

Step 2: Compute the Covariance Matrix

The covariance between two features X and Y is:

$$\text{Cov}(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_X)(y_i - \mu_Y)$$

For m-dimensional standardized data, the covariance matrix $\mathbf{C}$ is:

$$\mathbf{C} = \frac{1}{n-1} \mathbf{Z}^\top \mathbf{Z}$$

Where:

- $\mathbf{Z}$: Standardized data matrix ($n \times m$).
- $\mathbf{C}$: Covariance matrix ($m \times m$).

Step 3: Compute Eigenvalues and Eigenvectors

The eigenvalue problem is:

$$(\mathbf{C} - \lambda \mathbf{I})\mathbf{v} = 0$$

Where:

- λ: Eigenvalue.
- $\mathbf{v}$: Eigenvector.
- $\mathbf{I}$: Identity matrix.

Solve the characteristic equation:

$$\det(\mathbf{C} - \lambda \mathbf{I}) = 0$$

1. Solve for λ (eigenvalues).
2. Substitute each λ to find $\mathbf{v}$ (eigenvectors).

Step 4: Sort Eigenvalues and Select Principal Components

1. Arrange eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ in descending order.
2. Select the top k eigenvalues to determine the number of principal components.
3. Form the projection matrix $\mathbf{V}_k$ using the corresponding eigenvectors.

Step 5: Transform the Dataset

The principal components are obtained by projecting the standardized data onto the eigenvectors:

$$\mathbf{Z}' = \mathbf{Z}\mathbf{V}_k$$

Where:

- $\mathbf{Z}'$: Transformed data matrix ($n \times k$).
- $\mathbf{V}_k$: Matrix of top k eigenvectors ($m \times k$).

3. Numerical Example

Dataset $$\mathbf{X} = \begin{bmatrix} 2 & 0 \\ 4 & 2 \\ 6 & 4 \\ 8 & 6 \end{bmatrix}$$

Step 1: Standardize the Dataset

1. Calculate $\mu_1 = 5, \mu_2 = 3$.
2. Calculate $\sigma_1 = 2.236, \sigma_2 = 2.236$.
3. Standardize: $$\mathbf{Z} = \begin{bmatrix} -1.34 & -1.34 \\ -0.45 & -0.45 \\ 0.45 & 0.45 \\ 1.34 & 1.34 \end{bmatrix}$$

Step 2: Covariance Matrix

$$\mathbf{C} = \frac{1}{n-1}\mathbf{Z}^\top \mathbf{Z} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Step 3: Eigenvalues and Eigenvectors

1. Solve $\det(\mathbf{C} - \lambda \mathbf{I}) = 0$:

$$\begin{vmatrix} 1-\lambda & 1 \\ 1 & 1-\lambda \end{vmatrix} = 0$$

$$(1-\lambda)^2 - 1 = 0 \implies \lambda^2 - 2\lambda = 0 \implies \lambda = 2, 0$$

2. Eigenvectors:
   - For $\lambda = 2$: $\mathbf{v}_1 = [1, 1]^T$.
   - For $\lambda = 0$: $\mathbf{v}_2 = [-1, 1]^T$.

Step 4: Transform the Data

1. Projection matrix $\mathbf{V}_k = [\mathbf{v}_1]$:

$$\mathbf{V}_k = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

2. Transform data:

$$\mathbf{Z}' = \mathbf{Z}\mathbf{V}_k = \begin{bmatrix} -1.34 \\ -0.45 \\ 0.45 \\ 1.34 \end{bmatrix}$$

4. Interpretation

1. The first principal component explains the maximum variance (2).
2. The second principal component has negligible variance (0).
3. The data can be effectively reduced to 1 dimension without significant loss of information.

Principal Component Analysis

So, PCA involves two steps

- SVD and
- Projection of the training dataset onto the orthogonal principal components

PCA using SVD in Sklearn	PCA using PCA in Sklearn

# Centering the data and doing SVD 	
X_centered = X - X.mean(axis=0)
U,s,V = np.linalg.svd(X_centered)
 	
# Extracting the components and projecting the original dataset 	
W2 = V.T[:, :2]
X2D = X_centered.dot(W2)
	from sklearn.decomposition import PCA
	# Directly using PCA and transforming the original dataset
	# Takes care of centering
	
	pca = PCA(n_components = 2)
	X2D = pca.fit_transform(X)

PCA - Explained Variance Ratio

Variances explained by each of the components is important

- Available via the explained_variance_ratio_ variable
- The ratio indicates the proportion of the dataset's variance that lies along each principal component.

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

PCA - Number of PCs

How to select the number of principal components

- A much better option is to set n_components to be a float between 0.0 and 1.0, indicating the ratio of variance you wish to preserve

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

PCA - Number of PCs

How to select the number of principal components

- Yet another option is to plot the explained variance as function of the number of dimensions.
- There will usually be an elbow in the curve, where the explained variance stops growing fast.

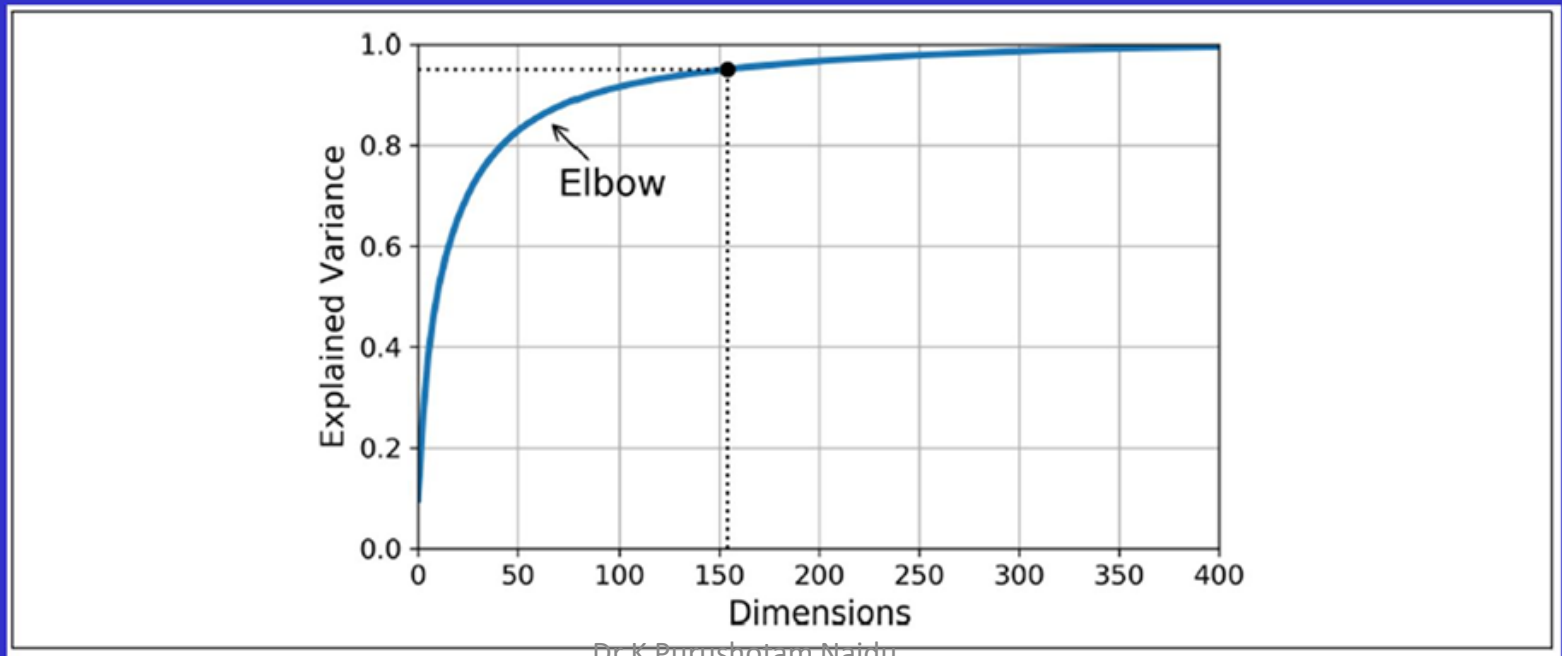


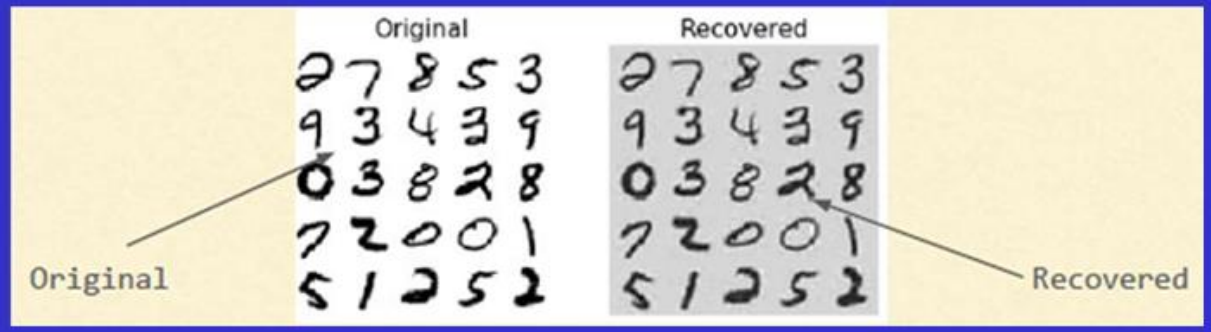
Figure 8-8. Explained variance as a function of the number of dimensions

PCA for Compression

- After dimensionality reduction, the training set takes up much less space.
- As an example, if we try applying PCA to the MNIST dataset while preserving 95% of its variance, each instance will have just over 150 features, instead of the original 784 features.
  - 80% reduction!
  - Training time improves tremendously.

PCA for Compression

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



Equation 8-3. PCA inverse transformation, back to the original number of dimensions

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}}\mathbf{W}_d^{\mathsf{T}}$$