

# Dimensionality Reduction

DS 8015

# Outline

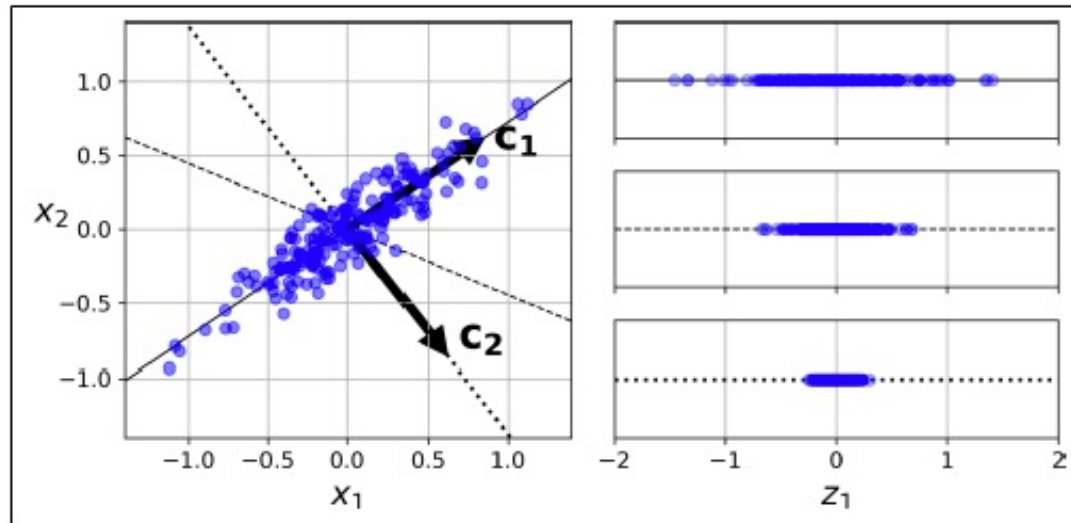
- Principal Component Analysis
- Feature Selection

# Principal Component Analysis

# Dimensionality Reduction

- Purpose:
  - Avoid curse of dimensionality
  - Reduce amount of time and memory required by data mining algorithms
  - Allow data to be more easily visualized
  - May help to eliminate irrelevant features or reduce noise
- Techniques:
  - Principle Component Analysis (PCA)
  - Singular Value Decomposition (SVD)
  - Others: supervised and non-linear techniques

# Dimensionality Reduction



- Assume we have a dataset with two features:  $X_1$  and  $X_2$
- We want to reduce the number of features in order to simplify (and speed up) the computation
- We are trying to find the best axis that can represent with the most variance (or information)
- The axis  $C_1$  has the most variance, then the dashed line, then  $C_2$
- We can use the  $C_1$  axis (one dimension) and convert the data to be represented by this Principal axis

# Dimensionality Reduction

- Linear transform techniques allow you to rotate appropriate datasets that contain correlated variables to a new set of variables where the covariance terms are zero.
- Eigenvalues and eigenvectors are important in the study of covariance matrix structure in statistics.
  - If you calculate the eigenvectors of a covariance matrix the first eigenvector is the axis of greatest variance in the data. The next eigenvector is the axis of next greatest variance, and so on.
- The Principal Component Analysis is a major application to find out the direction of maximum variance.
- Generally it happens that very few principal components can explain most of the variances in maximum part of the multivariate data.

# Linear Algebra: Matrix Determinant

- In the case of 2x2 matrix, the determinant formula is:

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

- Similarly, suppose we have a 3x3 matrix A, and we want the specific formula for its determinant  $|A|$

$$\begin{aligned} |A| &= \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= aei + bfg + cdh - ceg - bdi - afh \end{aligned}$$

- Each determinant of a 2x2 matrix in this equation is called a “**minor**” of the matrix A. More formally, a **minor** of A is the determinant of some smaller square matrix, cut down from A by removing one or more of its rows or columns. The same sort of procedure can be used to find the determinant of a 4x4 matrix, a 5x5 matrix, etc.

# Linear Algebra: Eigenvalue, Eigenvector

- **Definition:** An  $n \times n$  (square) matrix  $A$  is called invertible (a.k.a. nonsingular or nondegenerate) if there exists an  $n \times n$  matrix  $B$  such that

$$AB = BA = I_n$$

- Where  $I_n$  denotes the  $n \times n$  identity matrix and the multiplication used is ordinary matrix multiplication.
- If this is the case, then the matrix  $B$  is uniquely determined by  $A$  and is called the inverse of  $A$ , denoted by  $A^{-1}$ .
- A square matrix that is non invertible is called **singular** or **degenerate**.



# Linear Algebra: Eigenvalue, Eigenvector

- **Definition:** An **eigenvector** of a square matrix  $A$  is a non-zero vector  $v$  that, when the matrix is multiplied by  $v$ , yields a constant multiple of  $v$ , the multiplier being commonly denoted by  $\lambda$ . That is:

$$Av = \lambda v$$

- As this equation uses post-multiplication by  $v$ , it describes a right eigenvector.
- The number  $\lambda$  is called eigenvalue of  $A$  corresponding to  $v$
- If a matrix has  $n$  dimensions (i.e. features) then it contains  $n$  eigenvalues and  $n$  corresponding eigenvectors

# Linear Algebra: Eigenvalue, Eigenvector

- The eigenvalue equation for a matrix  $A$  is

$$Av - \lambda v = 0 \implies (A - \lambda I)v = 0$$

- Where  $I$  is the  $n \times n$  identity matrix.
- It is fundamental result of linear algebra that an equation  $Mv = 0$  has a non-zero solution  $v$  iff the determinant  $\det(M)$  of the matrix  $M$  is zero.
- The eigenvalues of  $A$  are precisely the real numbers  $\lambda$  that satisfy the equation

$$\det(A - \lambda) = 0$$

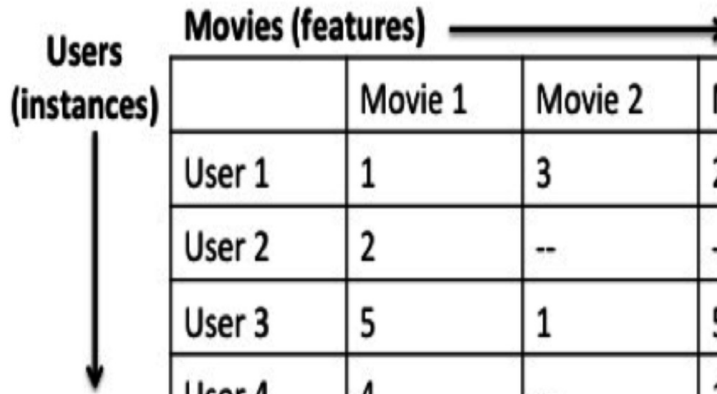
- You can use numpy functions to calculate eigenvalues and eigenvectors

```
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
```

# Principal Component Analysis (PCA)

- Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional datasets into a dataset with fewer variables, where set of resulting variables explains the maximum variance within dataset.
- PCA is used before the unsupervised and supervised machine learning steps to reduce the number of features used in the analysis, thereby reducing the likelihood of error.

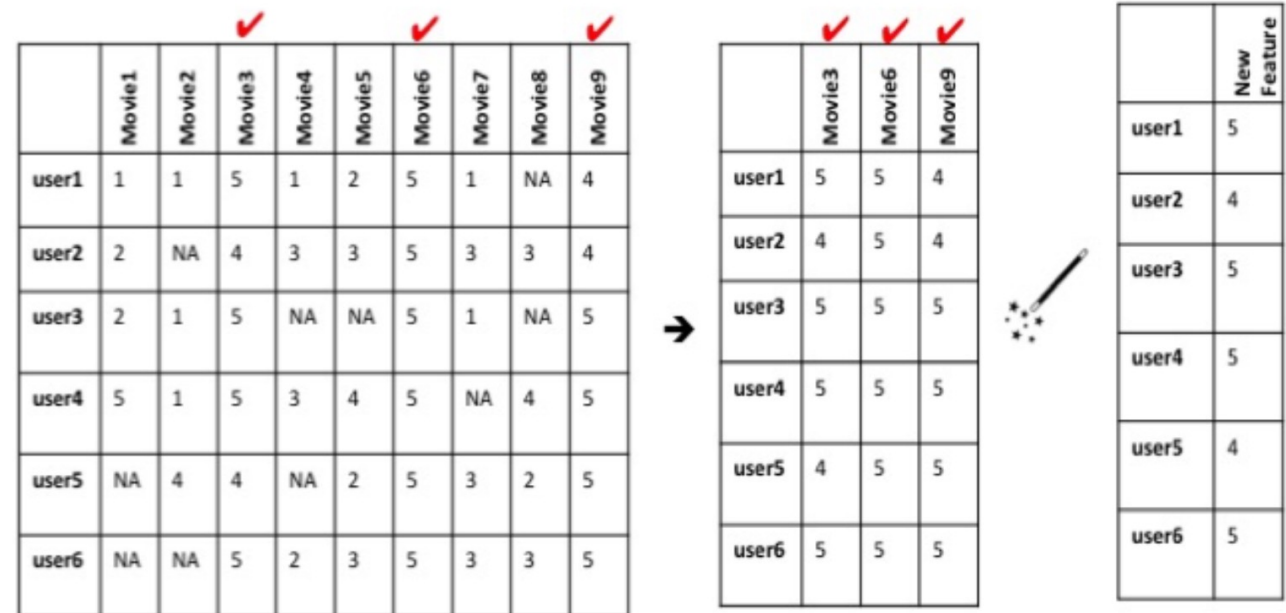
Movie rating system data: All movies ranked by certain users in a similar way might all share similar attributes, and could ultimately be grouped together to form a new feature.



	Movies (features)			
	Movie 1	Movie 2	Movie 3	Movie 4
User 1	1	3	2	1
User 2	2	--	--	5
User 3	5	1	5	3
User 4	4	--	1	4

# Principal Component Analysis (PCA)

- PCA can automatically group features that vary in a similar way within high-dimensional datasets
- Overall goal of PCA is to reduce the number of  $d$  dimensions (features) in a dataset by projecting it into a  $k$  dimensional subspace where  $k < d$



# PCA Step 1 - Standardization

- We need to standardize each feature so that they contribute equally to the analysis.
- If there are large differences between the ranges of initial variables, those variables with large ranges will dominate over the ones with small ranges.

- The standardization can be done as follows

$$Z = \frac{\text{value} - \text{mean}}{\text{standard deviation}}$$

- You can standardize the data using `StandardScaler()`
- Let's name the standardized data as X

# PCA Step 2 – Generating the Covariance Matrix

- We are trying to understand how the variables of the input data set are varying from the mean with respect to each other (or, we want to see if there is any relation between them). Therefore we calculate the covariance matrix

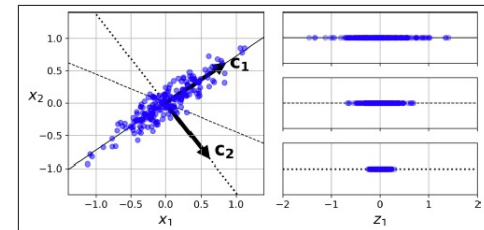
- Covariance matrix is a  $n \times n$  symmetrical matrix ( $n$  is the dimension)

$$\begin{bmatrix} cov(x, x) & cov(x, y) & cov(x, z) \\ cov(y, x) & cov(y, y) & cov(y, z) \\ cov(z, x) & cov(z, y) & cov(z, z) \end{bmatrix}$$

- Notes:
  - $cov(a, a) = var(a)$  This is the case with the main diagonal from left top to bottom right
  - $cov(a, b) = cov(b, a)$  Commutative. Therefore, the above matrix entries are symmetrical with respect to the main diagonal
  - If sign of covariance is positive then the variables are increasing or decreasing together. If sign is negative then one increases when the other decreases.

# PCA Step 3 – Compute Eigenvectors & Eigenvalues

- We will compute the eigenvectors and eigenvalues of the covariance matrix
- Principal Components
  - New variables that are constructed as linear combinations or mixtures of the initial variables.
  - These combinations are done in such a way that the new variables (i.e. principal components) are uncorrelated and most of the information within the initial variables is squeezed/compressed into the first component
- Principal component represent the directions of the data that explains a maximal amount of variance



## PCA Step 3 – Compute Eigenvectors & Eigenvalues

- Eigenvectors of the covariance matrix are the directions of the axes where there is the most variance (most information) and that is called the Principal Component (PC)
- Eigenvalues are simply the coefficients attached to eigenvectors which give the amount of variance carried in each PC
- After finding the eigenvalues ( $\lambda_i$ ), the biggest one corresponds to  $PC_1$ , and the corresponding  $v_i$  is the eigenvector of the same  $PC_1$
- To compute the percentage of variance (information) accounted for by each component, you can divide the eigenvalue of each component by the sum of eigenvalues



# PCA Step 4 – Feature Vector

- In order to reduce the dimensions of the original dataset we pick the principal components that has the highest eigenvalues, and then put their corresponding eigenvectors into a feature vector.

• **Example:**

$$\begin{array}{lll} v_1 = \begin{bmatrix} 0.6778 \\ 0.7351 \end{bmatrix} & \lambda_1 = 1.284028 & \lambda_1 \text{ carries 96\% of the information} \\ v_2 = \begin{bmatrix} -0.7351 \\ 0.6778 \end{bmatrix} & \lambda_2 = 0.04908 & \frac{1.284028}{1.284028 + 0.04908} = 0.96 \end{array}$$

If we pick only  $PC_1$  (because it is 96% of the variation)  
Then the feature vector is:

$$\begin{array}{c} v_1 \\ \begin{bmatrix} 0.6778 \\ 0.7351 \end{bmatrix} \end{array}$$

If we pick both  $PC_1$  and  $PC_2$   
Then the feature vector is:

$$\begin{array}{cc} \begin{array}{c} v_1 \\ \begin{bmatrix} 0.6778 \\ 0.7351 \end{bmatrix} \end{array} & \begin{array}{c} v_2 \\ \begin{bmatrix} -0.7351 \\ 0.6778 \end{bmatrix} \end{array} \end{array}$$

# PCA Step 5 – Recast Data to PC Axes

- In previous step we selected the principle components and formed the feature vector, but the input data set remained in terms of the original axes (i.e. in terms of  $X_i$ )
- We need to transform the original (standardized) data to the new axes
- We will use the feature vector formed using eigenvectors of the covariance matrix to reorient the data from the original axis to the ones represented by the principle components

$$\text{Final Dataset} = \text{FeatureVector}^T * \text{StandardizedOriginalDataset}^T$$

# PCA Example

Iris Dataset

	sepal length	sepal width	petal length	petal width	target
114	5.8	2.8	5.1	2.4	Iris-virginica
62	6.0	2.2	4.0	1.0	Iris-versicolor
33	5.5	4.2	1.4	0.2	Iris-setosa
107	7.3	2.9	6.3	1.8	Iris-virginica
7	5.0	3.4	1.5	0.2	Iris-setosa

Original Dataset

$$\begin{bmatrix} 5.8 & 2.8 & 5.1 & 2.4 \\ 6.0 & 2.2 & 4.0 & 1.0 \\ 5.5 & 4.2 & 4.0 & 1.0 \\ 7.3 & 2.9 & 6.3 & 1.8 \\ 5.0 & 3.4 & 1.5 & 0.2 \end{bmatrix}$$

## Step 1 - Standardization

$$X = \begin{bmatrix} -0.16 & -0.45 & 0.74 & 1.47 \\ 0.1 & -1.34 & 0.17 & -0.14 \\ -0.55 & 1.64 & -1.16 & -1.05 \\ 1.8 & -0.3 & 1.36 & 0.78 \\ -1.2 & 0.45 & -1.11 & -1.05 \end{bmatrix}$$

```
# Standardizing the features  
x = StandardScaler().fit_transform(x)
```

## Step 2 – Covariance Matrix

$$\Sigma = \begin{bmatrix} 1.25 & -0.51 & 1.08 & 0.75 \\ -0.51 & 1.25 & -0.84 & -0.73 \\ 1.08 & -0.84 & 1.25 & 1.13 \\ 0.75 & -0.73 & 1.13 & 1.25 \end{bmatrix}$$

# PCA Example

## Step 3 – Compute Eigenvectors and Eigenvalues

$$\lambda = [3.81 \quad 0 \quad 0.43 \quad 0.76]$$

Sort Eigenvalues in descending order (swap the eigenvectors/columns accordingly)

$$\lambda = [3.81 \quad 0.76 \quad 0.43 \quad 0.0]$$

DESC

## Step 4 – Feature Vector

$$v = \begin{bmatrix} 0.48 & -0.39 & 0.51 & 0.6 \\ -0.43 & 0.15 & -0.41 & 0.79 \\ 0.57 & 0.81 & -0.06 & 0.13 \\ 0.51 & -0.41 & -0.75 & -0.04 \end{bmatrix}$$

$$\hat{v} = \begin{bmatrix} 0.48 & 0.6 & 0.51 & -0.39 \\ -0.43 & 0.79 & -0.41 & 0.15 \\ 0.57 & 0.13 & -0.06 & 0.81 \\ 0.51 & -0.04 & -0.75 & -0.41 \end{bmatrix}$$

# PCA Example

Step 5 – Recast data to the feature axis  
(In this example we used all PCs)

$$FinalDataset = X^T \times \hat{v}^T$$

$$FinalDataset = \begin{bmatrix} -0.16 & -0.45 & 0.74 & 1.47 \\ 0.1 & -1.34 & 0.17 & -0.14 \\ -0.55 & 1.64 & -1.16 & -1.05 \\ 1.8 & -0.3 & 1.36 & 0.78 \\ -1.2 & 0.45 & -1.11 & -1.05 \end{bmatrix}^T \times \begin{bmatrix} 0.48 & 0.6 & 0.51 & -0.39 \\ -0.43 & 0.79 & -0.41 & 0.15 \\ 0.57 & 0.13 & -0.06 & 0.81 \\ 0.51 & -0.04 & -0.75 & -0.41 \end{bmatrix}^T$$

$$FinalDataset = \begin{bmatrix} 1.29 & -0.41 & -1.04 & -0.01 \\ 0.66 & -0.97 & 0.7 & -0.04 \\ -2.17 & 0.86 & -0.09 & -0.05 \\ 2.16 & 0.99 & 0.37 & 0.03 \\ -1.94 & -0.47 & 0.06 & 0.07 \end{bmatrix}$$

# PCA – Python Implementation

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# import iris data set
df = pd.read_csv("iris.csv")
df.head()

features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

# Separating out the features
x = df.loc[:, features].values

# Separating out the target
y = df.loc[:, ['target']].values
```

# PCA – Python Implementation

```
# Standardizing the features
x = StandardScaler().fit_transform(x)

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['pc1', 'pc2'])

finalDf = pd.concat([principalDf, df[['target']], axis = 1)
finalDf.head()
```

	pc1	pc2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa

```
pca.explained_variance_ratio_  
# array([0.72770452, 0.23030523])
```

# PCA – Python Implementation

Check the lecture's implementation with additional diagrams and information



# Choosing the Right Number of Dimensions

- Instead of choosing the number of dimensions arbitrarily, it is simpler to choose the number that add up to a sufficiently large portion of the variance (e.g. 95%)
- If you want to reduce dimensionality for data visualization, then you may want to reduce the number of dimensions down to 2 or 3

- **Method 1**

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

- Then, set the `n_components` to `d` and recalculate

```
pca = PCA(n_components = d)  
X_reduced = pca.fit_transform(X)
```

# Choosing the Right Number of Dimensions

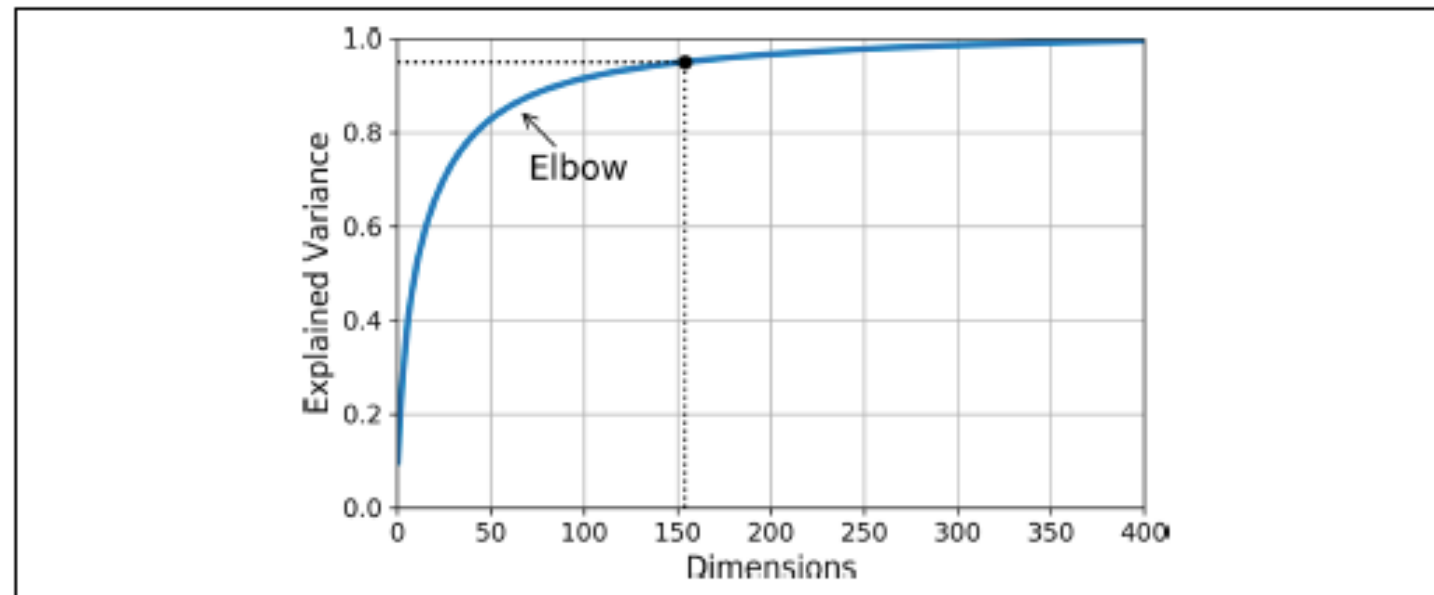
- Method 2 (a better approach)

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

- This automatically selects the number of components that makes sure that the ratio of variance is greater or equal to 95%

# Choosing the Right Number of Dimensions

- Method 3
  - Plot the explained variance as a function of the number of dimensions: `plot cumsum()`
  - Find the elbow of the curve where the explained variance stops growing fast.



# Feature Selection

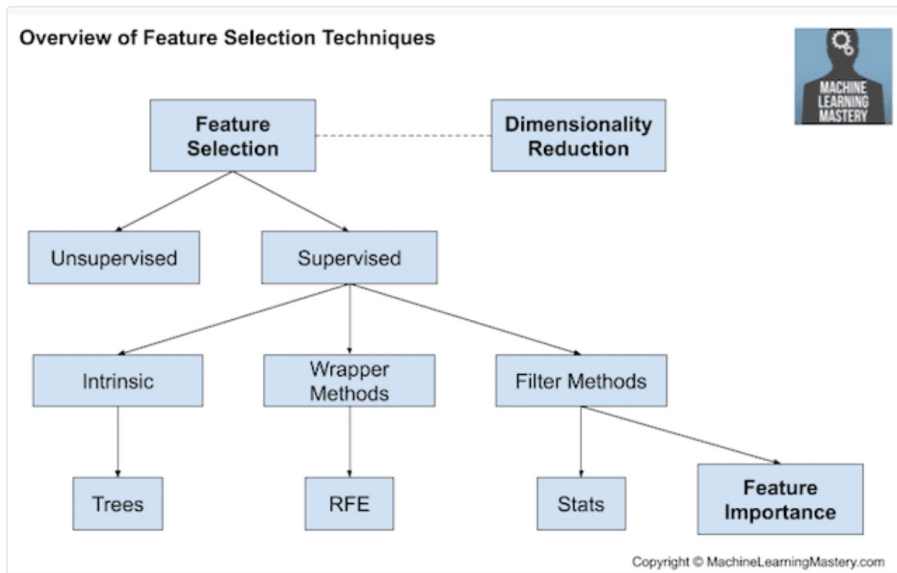
# Feature Selection (FS)

- FS is another way to reduce the dimensionality of data
- What if the number of features is larger than the number of samples or we have a sparsity problem?
- Feature selection helps improving performance of predictions
- FS vs PCA
  - Both tend to reduce the number of attributes in the dataset
  - PCA reduces the number of attributes by creating new combinations of attributes (but less in number)
  - FS methods include and exclude attributes present in the data without changing them

# Feature Selection

- **Redundant features:** Duplicate much or all of the information contained in one or more other attributes
  - Example: purchase price of a product and the amount of sales tax paid (i.e. highly correlated)
- **Irrelevant features:** Contain no information that is useful for the data mining task at hand
  - Example: students' ID is often irrelevant to the task of predicting students' GPA
- Potential benefits of feature selection:
  - It enables the machine learning algorithm to train faster.
  - It reduces the complexity of a model and makes it easier to interpret.
  - It improves the accuracy of a model if the right subset is chosen.
  - It reduces Overfitting.

# Feature Selection Methods



**Feature Selection:** select a subset of input features from the dataset

- **Unsupervised:** do not use the target variable (and remove redundant variables)
  - Use correlation
- **Supervised:** use target variable (to remove irrelevant variables)
  - **Wrapper:** search for well-performing subsets of features
    - Recursive Feature Elimination (RFE)
  - **Filter:** select subsets of features based on their relationship with the target
    - Statistical methods
    - Feature importance methods
  - **Intrinsic:** algorithms that perform automatic feature selection during training
    - Decision trees

# Variable Data Types

Data (input or output) can be of the following types

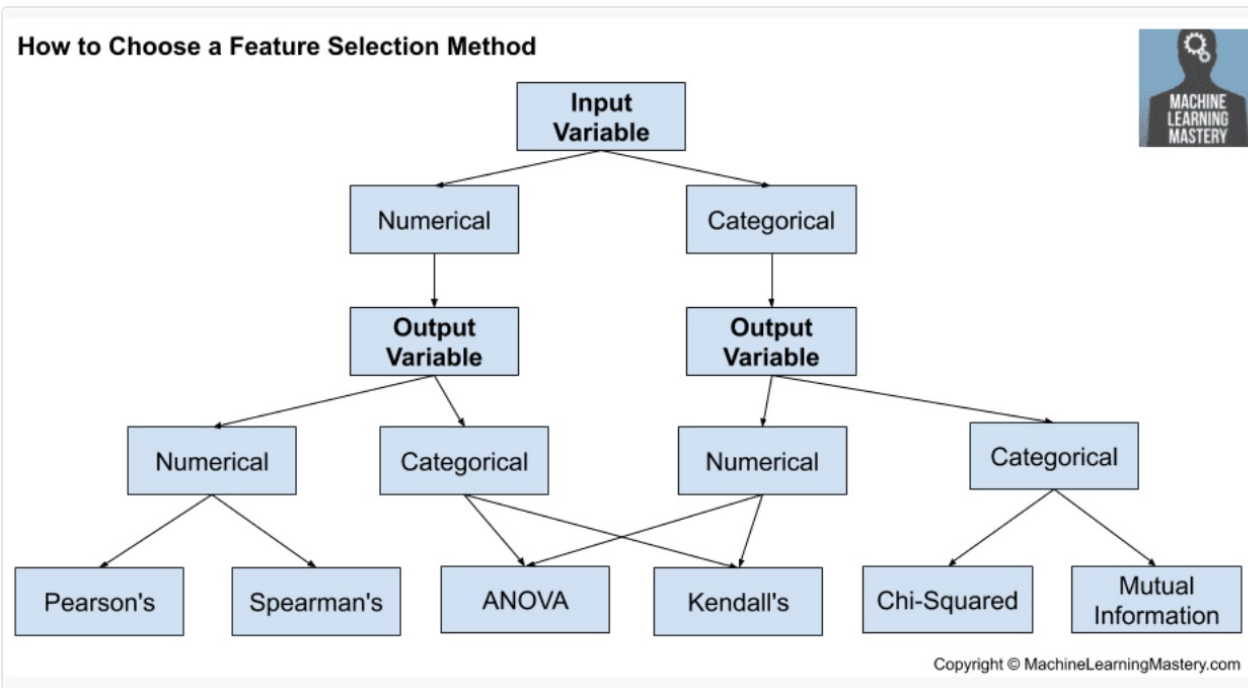
- Numerical information
  - Integer
  - Float
- Categorical information
  - Nominal information. E.g., gender, category name, ocean proximity
  - Ordinal information. Information contains order or rank. E.g. “Very Satisfied, Satisfied, Neutral, Dissatisfied, Very Dissatisfied”, “1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>”
  - Boolean information. True/False



# Statistical Methods for Filter-Based Selection

- Correlation type statistical methods are used between input features and output (target) values for filter based feature selection methods
- The type of output variable indicates the type of predictive modelling problem being performed.
  - **Numerical output** variable -> **regression** predictive modelling problem
  - **Categorical output** variable -> **classification** predictive modelling problem

# Statistical Methods for Filter-Based Selection



## **NUMERICAL INPUT, NUMERICAL OUTPUT -> Regression**

Most common technique: using a correlation coefficient

- Pearson's correlation coefficient for a linear correlation
- Rank-based method (e.g. Spearman's rank coefficient) for nonlinear correlation

## **NUMERICAL INPUT, CATEGORICAL OUTPUT -> Classification**

Most common technique: using a correlation based technique

- ANOVA correlation coefficient for linear correlations
- Kendall's rank coefficient for nonlinear correlations

## **CATEGORICAL INPUT, NUMERICAL OUTPUT -> Regression**

Not a very common problem.

Same techniques as Numerical Input – Categorical Output methods can be used (in reverse order)

## **CATEGORICAL INPUT, CATEGORICAL OUTPUT -> Classification**

Most common technique: Chi-Squared Test

- Chi-Squared test (contingency tables)
- Mutual information (information gain) can be used as well

# Statistical Methods for Filter-Based Selection

- The Scikit-Learn library contains the most of the statistical methods listed on the previous slide.
- Correlation Statistic measures:
  - Pearson's Correlation Coefficient: `f_regression`
  - ANOVA: `f_classif`
  - Chi-Squared: `chi2`
  - Mutual Information: `mutual_info_classif`, `mutual_info_regression`
- Selection methods can be used to pick the best input variables using the correlation statistic measures:
  - Selecting the top k variables: `SelectKBest`
  - Selecting the top percentile variables: `SelectPercentile`

# Feature Selection – Python Implementation

- Regression Feature Selection (Numerical Input, Numerical Output)

```
from sklearn.datasets import make_regression
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

# generate dataset for regression
X, y = make_regression(n_samples=100, n_features=100, n_informative=10)

# define feature selection using Pearson's Correlation Coefficient (-> f_regression)
fs = SelectKBest(score_func=f_regression, k=10)

# apply feature selection
X_selected = fs.fit_transform(X, y)
print(X_selected.shape) # will return (100, 10) because we wanted 10 best features
```

# Feature Selection – Python Implementation

- Classification Feature Selection (Numerical Input, Categorical Output)

```
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# generate dataset for regression
X, y = make_classification(n_samples=100, n_features=20, n_informative=2)

# define feature selection using ANOVA F (-> f_classif)
fs = SelectKBest(score_func=f_classif, k=2)

# apply feature selection
X_selected = fs.fit_transform(X, y)
print(X_selected.shape) # will return (100, 2) because we wanted 2 best features
```

# Feature Selection – Python Implementation

Check the lecture's implementation with additional diagrams and information