2.2 - Data Pre-Processing

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Introduction to Machine Learning for Finance

Bologna - February-March, 2025

Data Pre-Processing

Data Pre-Processing

- Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm.
- On the other hand, the success of a machine learning algorithm highly depends on the quality of the data fed into the model.
- Real-world data is often dirty containing outliers, missing values, wrong data types, irrelevant features, or non-standardized data.
- The presence of any of these will prevent the machine learning model to properly learn.
- For this reason, transforming raw data into a useful format is an essential stage in the machine learning process.

Data Pre-Processing

The topics that we will cover in this lesson are as follows:

- Removing and imputing missing values from the dataset
- Getting categorical data into shape for machine learning algorithms
- Feature Normalization
- Selecting relevant features for the model construction

Data Cleaning

- Dealing with inconsistent recording
- Removing unwanted observations
- Removing duplicates
- Investigating outliers
- Dealing with missing items

Subsection 1

Missing Data

Dealing with Missing Data

- The real-world data often has a lot of missing values.
- The cause of missing values can be data corruption or failure to record data.
- The handling of missing data is very important during the preprocessing of the dataset as many machine learning algorithms do not support missing values.

Dealing with Missing Data

Delete Rows/Columns with Missing Values

- One of the easiest ways to deal with missing data is simply to remove the corresponding features (columns) or training examples (rows) from the dataset entirely;
- Missing values can be handled by deleting the rows or columns having null values. If columns have more than half of the rows as null then the entire column can be dropped. The rows which are having one or more columns values as null can also be dropped.
- Remember that, in pandas, rows with missing values can easily be dropped via the **dropna** method.

Dealing with Missing Data

Delete Rows/Columns with Missing Values

Pros:

- A model trained with the removal of all missing values creates a robust model.

Cons:

- Loss of a lot of information.
- Works poorly if the percentage of missing values is excessive in comparison to the complete dataset.

Dealing with Missing Data

Imputing missing values with Mean/Median:

- Columns in the dataset which are having numeric continuous values can be replaced with the mean, median, or mode of remaining values in the column.
- This method can prevent the loss of data compared to the earlier method.

Dealing with Missing Data

Imputing missing values with Mean/Median

Pros:

- Prevent data loss which results in deletion of rows or columns
- Works well with a small dataset and is easy to implement.

Cons:

- Works only with numerical continuous variables.
- Can cause data leakage
- Do not factor the covariance between features.

Subsection 2

Categorical Data

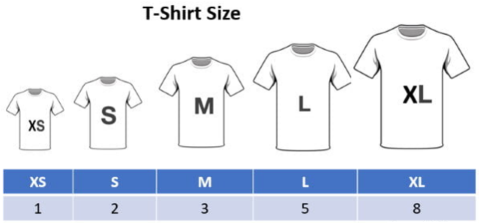
What is Categorical Data?

- Categorical data is a form of data that takes on values within a finite set of discrete classes.
- It is difficult to count or measure categorical data using numbers and therefore they are divided into categories.
- There are two types of categorical variables: **I. Ordinal Variables** and **II. Nominal Variables**

Categorical Data: Ordinal Variables

Ordinal Variables

- These variables maintain a natural order in their class of values.
- If we consider the level of education then we can easily sort them according to their education tag in the order of *High School* ¡ *Under-Graduate* ¡ *post-Graduate* ¡ *PhD*.
- The review rating system can also be considered as an ordinal data type where 5 stars is definitely better than 1 star.



XS	S	M	L	XL
1	2	3	5	8

Categorical Data: Nominal Variables

Nominal Variables

- These variables do not maintain any natural/logical order.
- The color of a car can be considered as Nominal Variable as we cannot compare the color with each other.
- It is impossible to state that **Red** is better than **Blu** (subjective!).
- Similarly, Gender is a type of Nominal Variable as again we cannot differentiate between Male, Female, and Others.

Categorical Data: Encoding

- Most of the Machine learning algorithms are designed to work with numeric data.
- Hence, we need to convert Categorical (text) data into numerical data for model building.
- There are multiple encoding techniques to convert categorical data into numerical data. Let us look at some of them.

Categorical Data: Ordinal Encoding

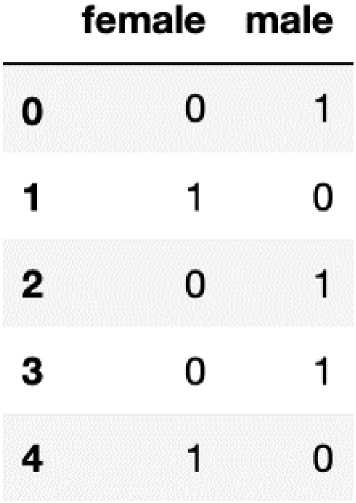
- **Ordinal Encoding** primarily encodes ordinal categories into ordered numerical values.
- Ordinal encoding maps each unique category value to a specific numerical value based on its order or rank.
- Consider for example an education degree feature in a given data frame. Here, we can define the ordering of the categories when creating an ordinal encoder using sklearn. so, for example, we can arrange the order inside the categories as a list in ascending order. First, we have the High School followed by Associate,Master, and then Ph.D. at the end.
- *OrdinalEncoder(categories=[['HS', 'AS', 'M.S', 'Ph.D.']])*

Categorical Data: One Hot Encoding

- if there is no ordinal relationship between the categorical variables then ordinal encoding might mislead the model.
- This is because the ordinal encoder will try to force an ordinal relationship on the variables to assume a natural ordering, thus resulting in poor performance.
- In this case, One Hot encoder should be used to treat our categorical variables.
- It will create dummy variables by converting N categories into N features/columns.

Categorical Data: One Hot Encoding

- Considering the gender example. If we have a male in the first row, then its value is 1. Also if we have a female in the second row then its value is 0. Whenever the category exists its value is 1 and 0 where it does not.
- We can one-hot encode categorical variables in two ways: by using **get_dummies** in pandas and by using **OneHotEncoder** from sklearn.



	female	male
0	0	1
1	1	0
2	0	1
3	0	1
4	1	0

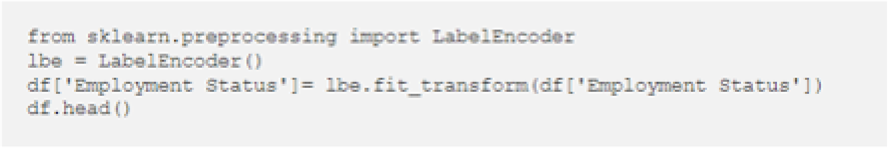
Categorical Data: Label Encoding

- The label encoder will convert each category into a unique numerical value.
- If implemented with Sklearn, then this encoder should be used to encode output values, i.e. y, and not the input X.
- It is similar to the ordinal encoder except, here the numeric values are assigned automatically without following any sort of natural order.
- Generally, the alphabetical order of the categorical values is used to determine which numerical value comes first.

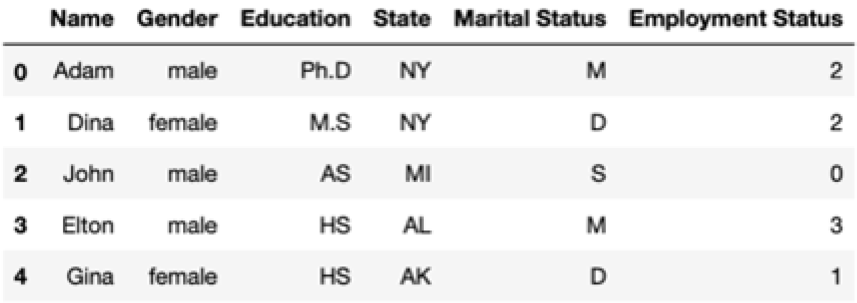
Categorical Data: Label Encoding

- Consider for example a target variable 'Job Status' that has four different categories.
- After applying label encoding to this column the four different categories are mapped into integers 0: Full Time, 1: Intern, 2: Part-Time, and 3:Unemployed.
- With this, it can be interpreted that Unemployed have a higher priority than Part-Time, Full Time, and Intern while training the model, whereas, **there is no such priority or relation between these statuses**.
- We cannot define the order of labels with the label encoding technique.

Categorical Data: Label Encoding



```
from sklearn.preprocessing import LabelEncoder
lbe = LabelEncoder()
df['Employment Status']= lbe.fit_transform(df['Employment Status'])
df.head()
```



	Name	Gender	Education	State	Marital Status	Employment Status
0	Adam	male	Ph.D	NY	M	2
1	Dina	female	M.S	NY	D	2
2	John	male	AS	MI	S	0
3	Elton	male	HS	AL	M	3
4	Gina	female	HS	AK	D	1

Subsection 3

Feature Scaling (Normalization)

Feature Scaling

- Before using many ML algorithms (including those for unsupervised learning), it is important to scale feature values so that they are comparable.
- Z-score scaling involves calculating the mean (μ) and SD (σ) from the values of each feature from the training set. Scaled feature values for all data sets are then created by subtracting the mean and dividing by the SD.
- The scaled feature values have a mean of zero and SD of one.

$$\text{scaled feature value} = \frac{V - \mu}{\sigma} \tag{1}$$

Feature Scaling

- Min-max scaling involves calculating the maximum and minimum value of each feature from the training set.
- Scaled feature values for all data sets are then created by subtracting the minimum and dividing by the difference between the maximum and minimum.
- The scaled feature values lie between zero and one:

$$\text{scaled feature value} = \frac{V - V_{min}}{V_{max} - V_{min}} \tag{2}$$

Feature Selection

Introduction to Feature Selection

- Feature selection is the process of identifying and retaining only the most relevant variables for a machine learning model.
- The goal is to improve model performance by reducing noise and redundancy in the dataset.
- High-dimensional datasets often contain irrelevant or correlated features that do not contribute meaningful information for prediction.
- Unlike feature extraction, which transforms existing features into new ones (e.g., PCA, LDA), feature selection retains a subset of the original features.
- Feature selection enhances interpretability, reduces training time, and prevents overfitting, leading to a more efficient and effective model.

Why Feature Selection is Important?

- **Reduces Overfitting**: By removing irrelevant or redundant features, the model focuses only on meaningful variables, reducing the risk of learning noise instead of true patterns.
- **Improves Accuracy**: By eliminating uninformative features, the model generalizes better to unseen data, leading to improved prediction performance.
- **Speeds Up Training**: Fewer features mean lower computational complexity, reducing the time required to train and optimize machine learning models.

Why Feature Selection is Important?

- **Enhances Interpretability**: Models with fewer, well-selected features are easier to understand and explain, which is especially important in regulated industries such as finance and healthcare.
- **Mitigates Curse of Dimensionality**: High-dimensional data can negatively impact model performance due to sparsity; selecting relevant features helps mitigate this issue.
- **Facilitates Model Deployment**: A model with fewer variables is easier to implement in production environments, requiring less storage and computational resources.

How do we select features?

- A feature selection procedure combines a search technique with an evaluation method. The search technique proposes new feature subsets, and the evaluation measure determines the how good the subset is.
- In a perfect world, a feature selection method would evaluate all possible subsets of feature combinations and determine which one results in the best performing machine learning model.
- However, computational cost inhibits such a practice in reality. In addition, the optimal subset of features varies between machine learning models. A feature subset that optimizes one model‘s performance won‘t necessarily optimize another‘s.

Methods of Feature Selection

1. **Filter Methods**: These methods apply statistical techniques to evaluate the importance of each feature independently of the model. Common techniques include correlation analysis, Chi-square test, and mutual information.
2. **Wrapper Methods**: These methods iteratively evaluate subsets of features using machine learning models, selecting the best-performing combination. Examples include Recursive Feature Elimination (RFE), Forward Selection, and Backward Elimination.
3. **Embedded Methods**: These methods integrate feature selection within the model training process. Techniques such as Lasso (L1) regression and Decision Tree-based feature importance are commonly used.

Filter Methods

- A typical filter algorithm consists of two steps: it ranks features based on certain criteria and then chooses the highest-ranking features to train the machine learning models.
- Filter methods are generally univariate, so they rank each feature independently of the rest. Because of this, the filter methods tend to ignore any interactions that occur between features. Thus, redundant variables will not necessarily be eliminated by filter methods.
- However, some multivariate filter selection methods exist as well. These consider features in relations to others in the data set, making them naturally capable of handling redundant features. Their selection criteria scans for duplicated features and correlated features and provide simple but powerful methods to quickly remove redundant information.

Filter Methods

Constant, quasi-constant, and duplicated features

- The most basic and intuitive methods for feature selection consist of removing constant, quasi-constant, or duplicated features.
- Constant features only show one value for all the observations in the data set. That is, they show absolutely no variability.
- Quasi-constant features are similar; if most observations share the same value, then we'd label that feature quasi-constant. In practice, quasi-constant features typically refer to those variables where more than 95 to 99 percent of the observations show the same value.
- Duplicated features, as the name indicates, are those that are in essence, identical. That is, for every observation, they show the same value.

Filter Methods

Constant, quasi-constant, and duplicated features

- Although it sounds obvious and overly simple, many datasets contain a lot of constant, quasi-constant, and duplicated features.
- In fact, duplicated features often arise when generating new features by one-hot encoding of categorical variables.
- Removing these features is an easy but effective way to reduce the dimension of the feature space, without losing any significant information.

Filter Methods

Correlation

- Correlation measures the linear association between two or more variables.
- The higher the correlation, the more linearly associated the variables are.
- The central hypothesis is that good feature sets contain features that are highly correlated with the target, yet uncorrelated with each other.
- If two variables are correlated, we can predict one from the other.
- Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information.

Filter Methods

Mutual Information

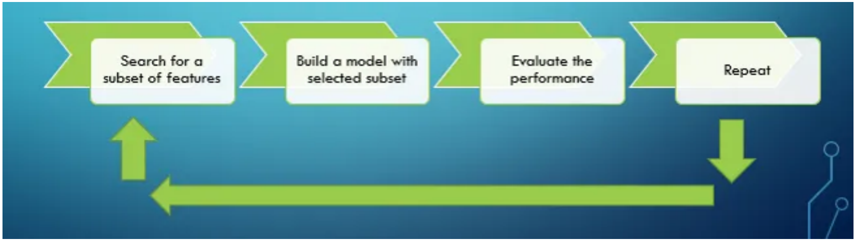
- Mutual information measures the mutual dependence between two variables, in this case, the feature and the target. Mutual information is similar to correlation, but more general; it doesn‘t strictly represent linear association. It measures how much knowing one of these variables reduces uncertainty in the other.

Fisher Score

- Fisher score uses the Chi-Square distribution to measure the dependency between two variables and works with categorical or discrete variables.
- Fisher score essentially compares the actual frequencies of the values of a variable with the expected frequencies if it had no relation to the target.

Wrapper Methods

- In comparison to filter methods, wrapper methods tend to be more computationally expensive, but select a better set of features.
- Wrapper methods use a specific machine learning algorithm in the selection process and choose the best subset of features tailored for that algorithm.
- This subset of features may not be optimal for a different machine learning model. In other words, **wrapper methods are not model agnostic**.


Search for a subset of features
Build a model with selected subset
Evaluate the performance
Repeat

Example: RFE with Logistic Regression

Wrapper Methods

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
rfe = RFE(model, n_features_to_select=5)
rfe.fit(X_train, y_train)
selected_features = X_train.columns[rfe.support_]
print("Selected␣Features:", selected_features)
```

Embedded Methods

- Feature selection occurs during model training.
- Uses model-specific techniques.
- Examples:
    - Lasso (L1) Regression
    - Decision Trees and Random Forests

Comparison of Methods

Method	Pros	Cons
Filter	Fast, scalable	Ignores interactions
Wrapper	Better selection	Expensive
Embedded	Model-optimized	Model-dependent

Understanding Pipelines in Scikit-Learn

- A pipeline is a sequence of data processing steps that are applied in a structured way.
- It helps automate machine learning workflows by chaining together multiple operations.
- Instead of manually applying transformations (e.g., scaling, feature selection, model training), a pipeline does it all in one step.
- Think of a pipeline as an assembly line:
  - Raw data enters the pipeline.
  - The data is preprocessed (e.g., missing values handled, scaling applied).
  - Feature selection picks the most relevant features.
  - A machine learning model is trained on the transformed data.
  - Predictions are generated efficiently.

Feature Selection Pipeline in Scikit-Learn

- Machine learning pipelines in Scikit-Learn allow seamless integration of preprocessing, feature selection, and model training.
- They enhance workflow efficiency, reproducibility, and ensure that transformations are applied consistently during training and inference.
- A typical pipeline consists of:
    - Preprocessing (e.g., standardization, encoding categorical variables)
    - Feature Selection (e.g., Recursive Feature Elimination, SelectKBest)
    - Model Training (e.g., Logistic Regression, Random Forest)
- Pipelines prevent data leakage by ensuring that feature selection and scaling are only learned from the training data.

Example: Feature Selection Pipeline in Scikit-Learn

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression

# Define the pipeline
pipeline = Pipeline([
# Standardize features
('scaler', StandardScaler()),
# Select top 5 features
('feature_selection', SelectKBest(score_func=f_classif, k=5)),
# Train logistic regression model
('classifier', LogisticRegression())
])
# Fit the pipeline on training data
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test) # Make predictions
```

Advantages of Using Pipelines

- **Automation:** Reduces manual steps and minimizes human errors.
- **Prevents Data Leakage:** Ensures feature selection and scaling are applied correctly.
- **Code Reusability:** Simplifies experimentation with different models and preprocessing techniques.
- **Hyperparameter Tuning:** Easily integrates with GridSearchCV and RandomizedSearchCV for optimizing parameters.

Conclusion and Best Practices

- Choose method based on dataset size and complexity.
- Filter methods for quick insights.
- Wrapper methods for optimal selection.
- Embedded methods for model-specific optimization.
- Always validate selected features.

Dimensionality Reduction

Feature Selection: Follow a KISS Approach!

- As a dimensionality reduction technique, feature selection aims to choose a small subset of the relevant features from the original features by removing irrelevant, redundant, or noisy features.
- Feature selection usually can lead to better learning performance, higher learning accuracy, lower computational cost, and better model interpretability.
- The problem is important, because a high number of features in a dataset, comparable to or higher than the number of samples, leads to model overfitting, which in turn leads to poor results on the validation datasets.
- Additionally, constructing models from datasets with many features is more computationally demanding.

Feature Extraction and Feature Selection

- Even though feature extraction and feature selection processes share some overlap, often these terms are erroneously equated.
- **Feature extraction** is the process of using domain knowledge to extract new variables from raw data that make machine learning algorithms work.
- The **feature selection** process is based on selecting the most consistent, relevant, and non-redundant features.
- The feature selection process is based on selecting the most consistent, relevant, and non-redundant feature subset from feature vectors.
- It is not only reduces training time and model complexity, but it eventually helps to **prevent overfitting**.

Principal Component Analysis (PCA)

- PCA is a widely used technique that transforms the original features into a new set of orthogonal components.
- The first few principal components retain the most variance in the data.
- Helps in reducing dimensionality while preserving important patterns.
- PCA works by:
  - Standardizing the dataset.
  - Computing the covariance matrix.
  - Finding eigenvectors and eigenvalues.
  - Selecting the top principal components.
  - Transforming the data into the new feature space.
- PCA is effective for noise reduction and visualization.

Mathematics Behind PCA

- Given a dataset with n features, PCA computes the covariance matrix:

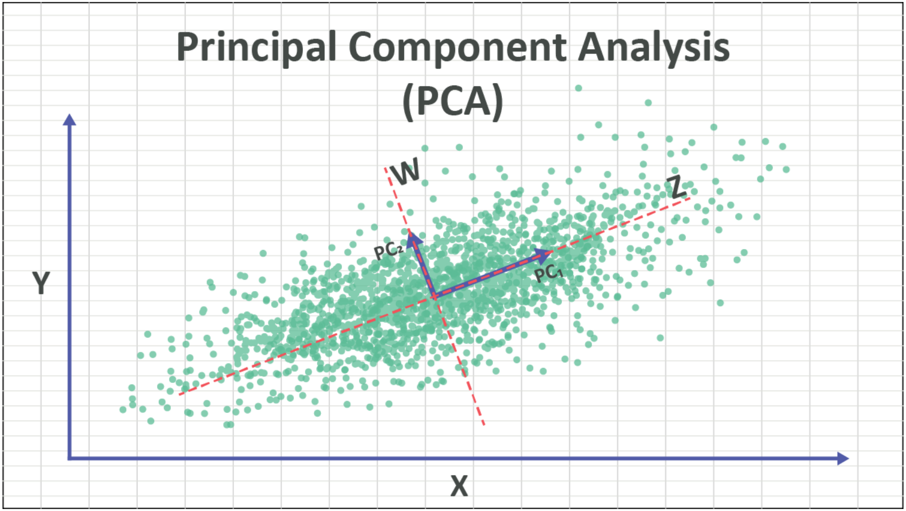
$$C = \frac{1}{n} X^T X \tag{3}$$

- The eigenvectors and eigenvalues of C represent the principal components and their importance.
- The data is projected onto the top k eigenvectors with the highest eigenvalues:

$$Z = XW \tag{4}$$

  where W is the matrix of selected eigenvectors.

Standardization and Gradient Descent



Example: PCA in Scikit-Learn

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

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

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

Advantages and Limitations of PCA

- **Advantages:**
  - Reduces redundancy by removing correlated features.
  - Improves model efficiency by reducing computation time.
  - Helps in visualizing high-dimensional data.
- **Limitations:**
  - Assumes linear relationships in data.
  - Loses interpretability of original features.
  - Sensitive to data scaling; standardization is required.

When to Use PCA?

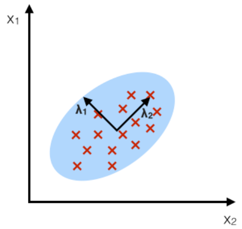
- When dealing with high-dimensional data to reduce computational costs.
- When visualization of data is required in lower dimensions.
- When removing noise and redundancy is essential for better model performance.
- When feature interpretability is less important compared to efficiency.

Linear Discriminant Analysis (LDA)

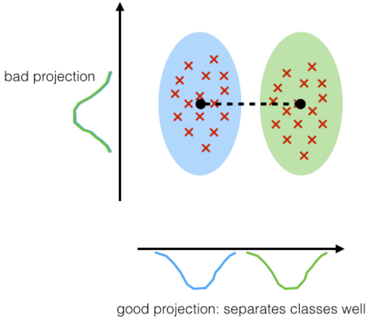
- LDA is a supervised dimensionality reduction technique that maximizes class separability.
- Unlike PCA, which focuses on variance, LDA optimizes for class separation.
- Works well for classification problems by finding a lower-dimensional space that retains discriminative information.
- LDA assumes that different classes generate data based on Gaussian distributions with the same covariance matrix.

LDA Vs PCA

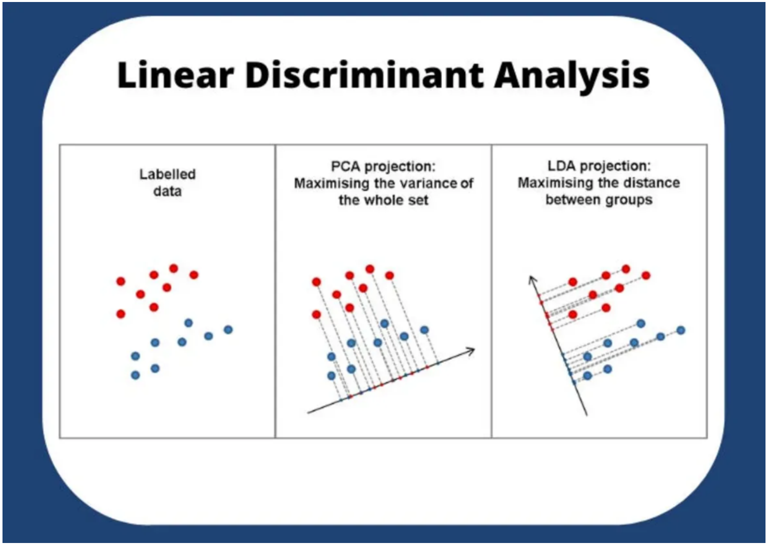
PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



LDA Vs PCA


Linear Discriminant Analysis
Labelled data
PCA projection: Maximising the variance of the whole set
LDA projection: Maximising the distance between groups

Mathematics Behind LDA

- LDA computes two key matrices:
  - Within-class scatter matrix (S_W): Measures variance within each class.
  - Between-class scatter matrix (S_B): Measures variance between different classes.
- LDA solves the generalized eigenvalue problem:

$$S_W^{-1} S_B v = \lambda v \tag{5}$$

where eigenvectors corresponding to the largest eigenvalues define the optimal projection space.

Example: LDA in Scikit-Learn

```
from sklearn.discriminant_analysis
     import LinearDiscriminantAnalysis

from sklearn.preprocessing import StandardScaler

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

# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)
```

Advantages and Limitations of LDA

- **Advantages:**
  - Improves class separability in classification problems.
  - Reduces dimensionality while preserving discriminative information.
  - Computationally efficient compared to other non-linear methods.
- **Limitations:**
  - Assumes normally distributed classes with equal covariance matrices.
  - Less effective when class distributions overlap significantly.
  - Requires labeled data for training.

When to Use LDA?

- When working with classification tasks that require feature reduction.
- When the goal is to maximize class separation in lower-dimensional space.
- When the dataset follows Gaussian distribution assumptions.

Comparison: PCA vs LDA

Method	Objective	Type
PCA	Maximize variance	Unsupervised
LDA	Maximize class separability	Supervised

Conclusion
LDA

- LDA is a powerful tool for supervised dimensionality reduction, particularly in classification tasks.
- It optimizes feature space for class separability, making it useful for many real-world applications.
- Choosing between PCA and LDA depends on the problem: use PCA for general structure preservation and LDA for class separability.