## Chapter - VI

# Data Balancing

**Syllabus**: Interpretation of classification dataset, Impact of imbalanced dataset, Techniques to handle imbalanced dataset - under sampling, over-sampling, K-fold Cross-Validation, SMOTE, Balanced Bagging Classifier, Threshold moving.

- ✓ Class balancing in Exploratory Data Analysis (EDA) is a crucial step when dealing with classification problems where the target variable classes are imbalanced—i.e., some classes have many more samples than others.
- ✓ Imbalanced datasets can lead to biased models that favor the majority class and perform poorly on the minority class.

### 1. Understanding Class Imbalance

Class imbalance occurs when **one or more classes in the target variable have significantly fewer instances than others.** It's especially common in domains like:

- **Medical Diagnosis:** Rare diseases vs. healthy patients.
- **Fraud Detection:** Fraudulent transactions are rare.
- **Churn Prediction:** Few customers churn compared to non-churners.

Formally, if a target variable $y$ has $k$ classes, the dataset is imbalanced if:

$$\max(\text{count}(y_i)) \gg \min(\text{count}(y_i))$$

where $y_i$ is the number of instances in class $i$.

**Extreme imbalance** is sometimes called **rare event detection.**

### 2. Causes of Class Imbalance

- **Natural rarity:** Some events are inherently rare (e.g., disease, fraud).
- **Data collection bias:** Sampling methods may favor some classes.
- **Data filtering or preprocessing:** Some classes may be inadvertently removed.

### 3. Problems Caused by Class Imbalance

Class imbalance can significantly **bias model training**:

1. **Bias towards majority class**
   - The model minimizes overall error by predicting the majority class more often.
   - Example: In a 95:5 dataset, a naive model predicting only the majority class achieves **95% accuracy** but **0% recall for minority class**.
2. **Poor generalization on minority class**
   - The model fails to learn patterns for the minority class, leading to poor recall and F1-score.
3. **Misleading evaluation metrics**
   - Accuracy is not reliable. Metrics like **Precision, Recall, F1, ROC-AUC** are better.
4. **Algorithm limitations**
   - Certain algorithms (e.g., decision trees) tend to favor majority class unless corrected.

### 4. Types of Class Imbalance

1. **Binary imbalance**
   - Only two classes, one dominates.
   - Example: Fraud (1%) vs Non-fraud (99%).
2. **Multi-class imbalance**
   - Multiple classes with different distributions.
   - Example: Handwriting recognition where digits 0–9 appear unevenly.
3. **Extreme imbalance**
   - Minority class is very rare (less than 1–2%).
   - Requires specialized techniques like anomaly detection.

## Interpretation of classification datasets

### 1. Understanding the Target Variable

The target variable (class) is the most important aspect in a classification problem.

- **Class distribution:** Check how balanced the classes are.

```python
df['target'].value_counts()
```

  - If classes are imbalanced, consider class balancing techniques.
- **Visualizations:**
  - **Bar plot:** Shows the count of each class.
  - **Pie chart:** Shows proportion of each class.

**Interpretation:**

- A very skewed distribution indicates potential model bias toward the majority class.
- Balanced classes allow standard classifiers to work without adjustment.

## 2. Understanding Features

Each feature should be analyzed to see how it contributes to class separation.

### A. Numerical Features

- **Descriptive statistics:** mean, median, std, min, max.
- **Distribution plots:** histograms, boxplots.
- **Class-wise distribution:** Compare feature distributions across classes.

```python
sns.boxplot(x='target', y='feature1', data=df)
```

- **Interpretation:** Features with distinct distributions for each class are **strong predictors.**

### B. Categorical Features

- **Frequency tables:** Count of categories within each class.

```python
pd.crosstab(df['feature_cat'], df['target'])
```

- **Bar plots per class:** Helps see class separation based on categories.
- **Interpretation:** Categories dominated by one class indicate strong predictive power.

## 3. Feature Relationships

- **Correlation analysis:** Identify multicollinearity.
  - Highly correlated features may not add independent information.
- **Pairplots / Scatterplots:** Useful for seeing **class separation** visually.
- **Interpretation:**
  - Features that cluster differently by class are informative.
  - Overlapping features may require transformation or feature engineering.

---

## 4. Handling Missing Values

- Missing data patterns can affect model performance.
- **Interpretation:**
  - Missing values concentrated in one class may bias the model.
  - Imputation may be required per class or overall.

### 5. Outlier Detection

- Outliers can distort classifiers.
- **Interpretation:**
  - Extreme values in one class may be genuine signals (e.g., fraud detection) or noise.
  - Decide whether to remove, transform, or keep them.

---

### 6. Feature Importance Estimation

- Before modeling, you can get an initial sense of feature importance using:
  - Mutual information
  - Chi-square test (categorical)
  - ANOVA (numerical)
- **Interpretation:** Helps select features likely to separate classes well.

---

### 7. Interaction Between Features

- Check for combinations of features that differentiate classes better than single features.
- Use **cross-tabulations, scatterplots, and heatmaps.**
- **Interpretation:** Interaction effects may suggest need for feature engineering.

## Impact of imbalanced datasets:

### 1. Biased Model Learning

When the dataset is imbalanced:

- The model tends to **predict the majority class more often**, because that minimizes overall error during training.
- Example: In a dataset with 95% negatives and 5% positives, a naive classifier predicting all negatives achieves **95% accuracy**, but **fails to detect any positives**.

**Impact:**

- The model becomes **useless for minority class prediction**.
- Critical in applications like fraud detection, medical diagnosis, and rare event prediction.

## 2. Misleading Performance Metrics

- Standard metrics like **accuracy** are **misleading** in imbalanced datasets.
- Example:
  - Dataset: 1000 samples, 950 negative, 50 positive.
  - Model predicts all negative → Accuracy = 95%.
  - But **Recall for positive class = 0%** → Model fails entirely for minority class.

**Better metrics:**

- Precision, Recall, F1-score
- ROC-AUC, PR-AUC
- Confusion Matrix

## 3. Poor Minority Class Generalization

- Imbalance causes the model to **underfit the minority class**, leading to poor generalization.
- The model may **ignore important patterns** in minority class data.

**Impact:**

- In critical applications, failure to detect minority class can have **serious consequences** (e.g., missing cancer diagnosis, undetected fraud).

## 4. Overfitting Risk with Oversampling

- Techniques like **simple duplication** of minority class can lead to **overfitting**, where the model memorizes minority samples rather than generalizing.
- More advanced techniques like **SMOTE** or **ADASYN** reduce this risk but still require careful validation.

## 5. Challenges in Multi-Class Imbalance

- Some classes may be underrepresented while others dominate.
- Classifiers can **ignore small classes entirely**, focusing on larger classes.
- Multi-class metrics (like macro-averaged F1-score) become important to assess performance fairly.

## 6. Impact on Algorithm Choice

- Some algorithms are more sensitive to class imbalance:

  - **Decision Trees, Random Forests, Logistic Regression**: Can be biased toward majority class.
  - **SVM with class weighting** or **XGBoost with scale_pos_weight**: Better suited for imbalanced data.
- Imbalance may force the use of **resampling or cost-sensitive algorithms**.

## 7. Potential Business/Practical Implications

- **Healthcare:** Missed detection of rare diseases.
- **Finance:** Fraudulent transactions may go undetected.
- **Marketing:** Misclassifying churn customers can lead to loss of revenue.
- **Safety-critical systems:** Failing to detect anomalies could be catastrophic.

# Techniques to handle imbalanced dataset

# Under-sampling:

### 1. Theory of Undersampling

**Undersampling** is a **data-level technique** for handling class imbalance. It works by **reducing the number of instances in the majority class** to match (or approach) the number of instances in the minority class.

**Goal:** Balance the dataset so that the classifier **does not become biased toward the majority class**.

- Especially useful when the dataset is very large and contains **redundant majority samples**.
- Often used with **binary classification**, but can also be adapted for multi-class problems.

### 2. How It Works (Approach / Formula)

1. Let $N_{\text{minority}}$ = number of minority class samples
2. Let $N_{\text{majority}}$ = number of majority class samples

**Random undersampling:** Randomly select $N_{\text{minority}}$ samples from the majority class.

$$X_{\text{balanced}} = X_{\text{minority}} \cup \text{RandomSample}(X_{\text{majority}}, N_{\text{minority}})$$

- This results in a **balanced dataset**:

$$N_{\text{balanced majority}} = N_{\text{minority}}$$

**Other approaches:**

- **Cluster-based undersampling:** Select representative majority samples using clustering.
- **Tomek links / Edited Nearest Neighbor:** Remove majority samples that are close to minority samples (noise cleaning).

## 3. Example in Python

```python
from imblearn.under_sampling import RandomUnderSampler

# Assume X = features, y = target
rus = RandomUnderSampler(random_state=42)
X_res, y_res = rus.fit_resample(X, y)

# Before undersampling
print(y.value_counts())
# e.g., 900 majority, 100 minority

# After undersampling
print(pd.Series(y_res).value_counts())
# e.g., 100 majority, 100 minority
```

## 4. Advantages of Undersampling

- Simple to implement.
- Reduces training time by decreasing dataset size.
- Can help models focus on minority class patterns.

## 5. Disadvantages of Undersampling

- May **lose valuable information** from the majority class.
- Can lead to **underfitting** if too many majority samples are removed.
- Random undersampling can remove important examples unless careful methods (like cluster-based sampling) are used.

# Example:

### Scenario

Suppose we have a dataset for **binary classification:**

| Sample | Feature1 | Feature2 | Class |
|---|---|---|---|
| 1 | 2 | 3 | 0 |
| 2 | 1 | 5 | 0 |
| 3 | 3 | 2 | 0 |
| 4 | 4 | 3 | 0 |
| 5 | 2 | 1 | 1 |
| 6 | 3 | 1 | 1 |

- **Class 0 (majority)** = 4 samples
- **Class 1 (minority)** = 2 samples

Dataset is **imbalanced.**

### Step 1: Random Undersampling

- We randomly select **2 samples** from the majority class (Class 0) to match the minority class size.

Example selection:

| Sample | Feature1 | Feature2 | Class |
|---|---|---|---|
| 1 | 2 | 3 | 0 |
| 3 | 3 | 2 | 0 |
| 5 | 2 | 1 | 1 |
| 6 | 3 | 1 | 1 |

### Step 2: Resulting Balanced Dataset

- **Class 0** = 2 samples
- **Class 1** = 2 samples

Now, the dataset is **balanced**, and a classifier trained on it **won't be biased toward Class 0.**

### Step 3: Python Implementation

```python
from imblearn.under_sampling import RandomUnderSampler
import pandas as pd

# Sample dataset
data = {'Feature1':[2,1,3,4,2,3],
        'Feature2':[3,5,2,3,1,1],
        'Class':[0,0,0,0,1,1]}
df = pd.DataFrame(data)

X = df[['Feature1','Feature2']]
y = df['Class']

rus = RandomUnderSampler(random_state=42)
X_res, y_res = rus.fit_resample(X, y)

balanced_df = pd.DataFrame(X_res, columns=['Feature1','Feature2'])
balanced_df['Class'] = y_res
print(balanced_df)
```

**Output:**

```powershell
   Feature1  Feature2  Class
0         2         3      0
1         3         2      0
2         2         1      1
3         3         1      1
```

- Now we have a **perfectly balanced dataset.**

## Over-Sampling:

Oversampling in EDA (Exploratory Data Analysis) usually refers to techniques used to handle class imbalance before or during the EDA process so that patterns in minority classes become more visible and models built later are not biased.

**Why Oversample During or Before EDA?**

Although oversampling is usually a *pre-modeling step*, during EDA it can help:

**1. Better Visualization**

If minority classes are extremely small, plots like boxplots, histograms, scatterplots, or correlation analysis may show misleading patterns. Oversampling helps reveal structure.

**2. Avoid Biased Summary Statistics**

Mean/variance for minority class can be unstable with small samples.

**3. Understanding Model Behaviour**

EDA on oversampled data can show how features separate classes once imbalance is neutralized.

## 🔧 Common Oversampling Techniques

### 1. Random Oversampling

Randomly duplicates minority class examples.

- Simple & fast
- Risk of overfitting (duplicates)

```python
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
X_res, y_res = ros.fit_resample(X, y)
```

---

### 2. SMOTE (Synthetic Minority Oversampling Technique)

Generates *synthetic* new samples by interpolating between real ones.

- More realistic than duplication
- Works well for numeric features

```python
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_res, y_res = sm.fit_resample(X, y)
```

---

## K-fold Cross-Validation:

# What is K-Fold Cross-Validation?

- ✓ K-Fold Cross-Validation is a model evaluation technique where the dataset is split into **K equal-sized parts (folds)**.
- ✓ The model is trained on **K−1 folds** and tested on the remaining fold. This process is repeated **K times**, each time using a different fold as the test set.
- ✓ Finally, the **average performance** across all K iterations is used as the model's score.

Example:

### Step 1: Splitting into 3 folds

```lua
Fold 1:
X = [[1], [2]]
y = [0, 0]

Fold 2:
X = [[3], [4]]
y = [1, 1]

Fold 3:
X = [[5], [6]]
y = [0, 1]
```

### Dataset

Suppose we have the following dataset (features X and target y):

```lua
X = [[1], [2], [3], [4], [5], [6]]
y = [0, 0, 1, 1, 0, 1]
```

We will use K = 3 folds.

This means each fold has 2 samples.

### Iteration 2

Test on Fold 2

Train on Folds 1 + 3

Train X = [[1], [2], [5], [6]]

Train y = [0, 0, 0, 1]

Test X = [[3], [4]]

Test y = [1, 1]

Suppose the model predicts: [1, 1]

Accuracy = 2/2 = 1.0

---

### Iteration 3

Test on Fold 3

Train on Folds 1 + 2

Train X = [[1], [2], [3], [4]]

Train y = [0, 0, 1, 1]

Test X = [[5], [6]]

Test y = [0, 1]

Suppose predictions = [0, 1]

Accuracy = 2/2 = 1.0

## Step 3: Final Score

Average accuracy across 3 folds:

```
(1.0 + 1.0 + 1.0) / 3 = 1.0
```

## Python Example Using the Same Dataset

```python
from sklearn.model_selection import KFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier
import numpy as np

# Simple dataset
X = [[1], [2], [3], [4], [5], [6]]
y = [0, 0, 1, 1, 0, 1]

model = DecisionTreeClassifier()

# Define 3-Fold CV
kf = KFold(n_splits=3, shuffle=False)

# Perform CV
scores = cross_val_score(model, X, y, cv=kf)

print("Scores for each fold:", scores)
print("Average score:", np.mean(scores))
```

Expected output structure:

```yaml
Scores for each fold: [1. 1. 1.]
Average score: 1.0
```

**Review Questions:**

1. What is class imbalance? Explain problems caused by class imbalance.
2. List the types of class imbalance.
3. Explain interpretation of classification datasets.
4. Describe the impacts of imbalanced datasets.
5. Explain under sampling in detail.
6. Illustrate under – sampling.
7. List advantageous and disadvantageous of under sampling.
8. What is over sampling? Why oversample is done during or before EDA?
9. Describe random oversampling and SMOTE.
10. Explain K-fold cross validation in detail.