Data Imbalance

What is Data Imbalance?

Data imbalance occurs when the classes in a dataset are not equally represented. This is common in classification problems where one class has significantly more samples than another.

Example of Data Imbalance

Consider a **fraud detection** dataset:

- **99% of transactions are NOT fraud** (majority class)
- **1% of transactions are fraud** (minority class)

Since fraudulent transactions are rare, a model trained on this dataset might simply predict **"Not Fraud"** all the time, achieving **99% accuracy** but failing to detect actual fraud cases.

Why is Data Imbalance a Problem?

- **Bias Towards the Majority Class** → The model tends to ignore minority class instances.
- **Misleading Accuracy** → High accuracy might not mean good performance if the minority class is ignored.
- **Poor Generalization** → The model may fail to recognize rare events in real-world scenarios.

Solutions to Handle Data Imbalance

1. **Resampling Techniques**
    - **Oversampling** (e.g., SMOTE) → Duplicate or generate synthetic minority samples.
    - **Undersampling** → Remove some majority samples to balance the dataset.
2. **Use Different Metrics**

- Instead of accuracy, use **Precision, Recall, F1-score, AUC-ROC** to evaluate the model.

3. **Algorithm-Level Solutions**
    - Use models that **handle imbalance better** (e.g., Decision Trees, Random Forest).
    - Modify loss functions (e.g., **Weighted Cross-Entropy Loss** for deep learning models).

SMOTE

What is SMOTE? (Synthetic Minority Over-sampling Technique)

SMOTE (Synthetic Minority Over-sampling Technique) is a method used to handle **imbalanced datasets** by **generating synthetic examples** for the minority class rather than simply duplicating existing ones.

Why Use SMOTE?

In imbalanced datasets, the model tends to favor the majority class. SMOTE helps by:

✅ Creating **synthetic** (not duplicated) minority class samples.

✅ Making the dataset **more balanced**, improving classification performance.

✅ Reducing **overfitting**, which can happen with simple oversampling.

How Does SMOTE Work?

1. **Select a minority class sample** at random.
2. **Find its k-nearest neighbors** (default **k=5**).
3. **Randomly choose one of the neighbors**.
4. **Generate a new synthetic data point** along the line connecting the original sample and the chosen neighbor.
5. Repeat until the dataset is balanced.

SMOTE Example in Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.datasets import make_classification

# Generate an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=2, weights=[0.90, 0.1(

# Check class distribution before SMOTE
print("Before SMOTE:", Counter(y))

# Apply SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check class distribution after SMOTE
print("After SMOTE:", Counter(y_resampled))

# Plot results
plt.figure(figsize=(8, 5))
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled, cmap='coolw
plt.title("Data Distribution After SMOTE")
plt.show()
```

Key Observations

- Before SMOTE, the minority class is **underrepresented**.
- After SMOTE, both classes are **balanced**, improving model performance.
- **SMOTE works best** when combined with **undersampling** of the majority class.

Tomek Links

What are Tomek Links?

Tomek Links are pairs of **nearest neighbor samples** from **different classes** that are very close to each other. They are used in **undersampling techniques** to clean the dataset by removing ambiguity in class separation.

Why Use Tomek Links?

✅ **Helps reduce class overlap** → Removes noisy samples near decision boundaries.

✅ **Improves class separation** → Creates a more distinct margin between classes.

✅ **Works well with SMOTE** → Often used after SMOTE to clean the synthetic data.

How Does It Work?

1. Identify **Tomek Links**:
    - A **majority class** point (A) and a **minority class** point (B) are **nearest neighbors**.
    - No other point is closer to either of them than they are to each other.
2. **Remove the majority class sample** (A) from the Tomek Link pair.
    - This **removes noisy samples** and helps create better class separation.

Example of Tomek Links in Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from imblearn.under_sampling import TomekLinks
from collections import Counter
from sklearn.datasets import make_classification

# Generate an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=2, weights=[0.90, 0.10], random_state=42)

# Check class distribution before Tomek Links
print("Before Tomek Links:", Counter(y))
```

```
# Apply Tomek Links
tomek = TomekLinks()
X_resampled, y_resampled = tomek.fit_resample(X, y)

# Check class distribution after Tomek Links
print("After Tomek Links:", Counter(y_resampled))

# Plot results
plt.figure(figsize=(8, 5))
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled, cmap='co
olwarm', alpha=0.5)
plt.title("Data Distribution After Tomek Links")
plt.show()
```

Key Takeaways

- **Tomek Links help refine class separation** by removing ambiguous samples.
- They are often used **after SMOTE** to remove noisy data.
- **Best for cleaning datasets** where classes are overlapping.

ENN

What is ENN (Edited Nearest Neighbors)?

Edited Nearest Neighbors (ENN) is an **undersampling technique** used in imbalanced datasets to **remove noisy samples** from the majority class, improving class separation.

Why Use ENN?

✅ **Reduces noise** by removing misclassified samples.

✅ **Improves decision boundaries** for classification models.

✅ **Works well with SMOTE** to clean synthetic data after oversampling.

How Does ENN Work?

1. **For each sample in the dataset**, find its **k-nearest neighbors** (default **k = 3**).
2. **Check the sample's class label** against its neighbors:
   - If the sample is misclassified by its neighbors, **remove it**.
3. Repeat for all data points, **removing ambiguous samples**.

Example of ENN in Python

```
import numpy as np
import matplotlib.pyplot as plt
from imblearn.under_sampling import EditedNearestNeighbours
from collections import Counter
from sklearn.datasets import make_classification

# Generate an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=2, weights=[0.90, 0.10], random_state=42)

# Check class distribution before ENN
print("Before ENN:", Counter(y))

# Apply Edited Nearest Neighbors
enn = EditedNearestNeighbours(n_neighbors=3)
X_resampled, y_resampled = enn.fit_resample(X, y)

# Check class distribution after ENN
print("After ENN:", Counter(y_resampled))

# Plot results
plt.figure(figsize=(8, 5))
plt.scatter(X_resampled[:, 0], X_resampled[:, 1], c=y_resampled, cmap='coolwarm', alpha=0.5)
plt.title("Data Distribution After ENN")
plt.show()
```

Key Takeaways

- **ENN removes noisy samples** that don't fit well within their class.
- **It works well after SMOTE** to clean up synthetic data.
- **Reduces misclassified samples** in highly overlapping datasets.
