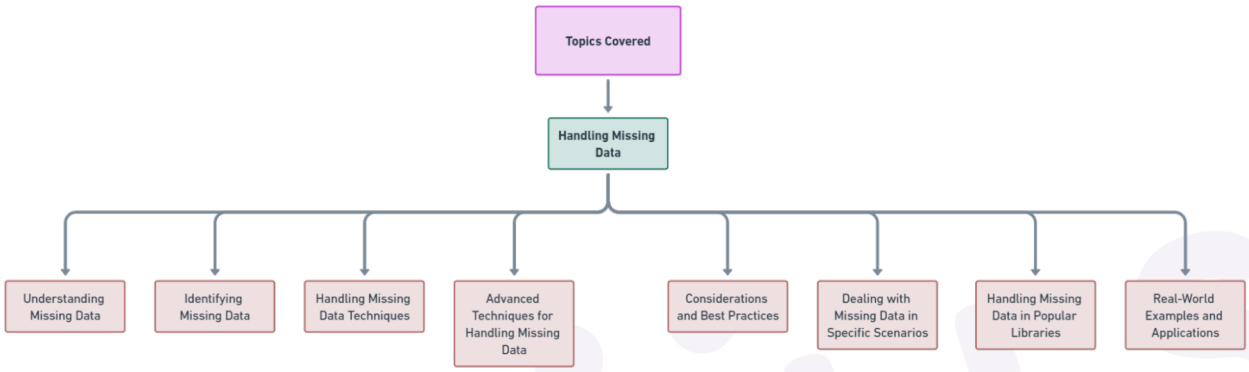
PW SKILLS

Lesson Plan

Handling Imbalanced Data In ML

Topic to covered:

- Understanding Imbalanced Data
- Techniques for Handling Imbalanced Data
- Evaluation Metrics for Imbalanced Data
- Advanced Techniques
- Real-world Applications and Case Studies
- Best Practices and Considerations
- Challenges and Limitations
- Tools and Libraries



Understanding Imbalanced Data

- Imbalanced datasets refer to those where the distribution of classes is not uniform.
- For instance, in a binary classification problem, if one class (majority class) heavily outweighs the other (minority class), it creates an imbalance.
- This can lead to biased models as algorithms tend to favor the majority class, affecting the model's ability to predict the minority class accurately.

Code

```
import numpy as np
from sklearn.datasets import make_classification

# Generating an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2,
weights=[0.9, 0.1], random_state=42)
# Checking class distribution
unique, counts = np.unique(y, return_counts=True)
print(dict(zip(unique, counts)))
```

Output::

```
{0: 897, 1: 103}
```

Techniques for Handling Imbalanced Data

Resampling Methods:

- Oversampling: Increasing the number of instances in the minority class.
- Undersampling: Reducing the number of instances in the majority class.

Code

```
from imblearn.over_sampling import SMOTE

# Applying SMOTE for oversampling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Checking class distribution after oversampling
unique_res, counts_res = np.unique(y_resampled, return_counts=True)
print(dict(zip(unique_res, counts_res)))
```

Output::

```
{0: 897, 1: 897}
```

Synthetic Data Generation:

- Generating synthetic samples to balance the dataset, such as using the ADASYN algorithm.

Code

```
from imblearn.over_sampling import ADASYN

# Applying ADASYN for synthetic data generation
adasyn = ADASYN(random_state=42)
X_synthetic, y_synthetic = adasyn.fit_resample(X, y)
# Checking class distribution after synthetic data generation
unique_syn, counts_syn = np.unique(y_synthetic, return_counts=True)
print(dict(zip(unique_syn, counts_syn)))
```

Output::

```
{0: 897, 1: 910}
```

Evaluation Metrics for Imbalanced Data

- In imbalanced datasets, accuracy can be misleading due to the disproportionate class distribution.
- Instead, evaluation metrics like precision, recall, F1-score, ROC-AUC, and PR curve provide a more comprehensive understanding of model performance.

Code

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2, random_state=42)

# Fitting a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluating model performance
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Output::

```
              precision    recall  f1-score   support

           0       0.90      0.91      0.90       180
           1       0.90      0.90      0.90       179

    accuracy                           0.90       359
   macro avg       0.90      0.90      0.90       359
weighted avg       0.90      0.90      0.90       359

Confusion Matrix:
[[163  17]
 [ 18 161]]
```

Advanced Techniques:

- Ensemble methods like XGBoost, AdaBoost, or Random Forests can handle imbalanced data effectively due to their inherent ability to weigh different samples or classes.

Code

```
from xgboost import XGBClassifier

# Using XGBoost classifier
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)

# Predictions using XGBoost
y_pred_xgb = xgb.predict(X_test)

# Evaluating XGBoost model
print(classification_report(y_test, y_pred_xgb))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_xgb))
```

Output::

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       180
           1       0.97      0.95      0.96       179

    accuracy                           0.96       359
   macro avg       0.96      0.96      0.96       359
weighted avg       0.96      0.96      0.96       359

Confusion Matrix:
[[175   5]
 [  9 170]]
```

Real-world Applications and Case Studies

Fraud Detection in Finance

- In finance, imbalanced data is common in fraud detection tasks, where fraudulent transactions are relatively rare compared to legitimate ones.
- Techniques like anomaly detection, oversampling the minority class, or using cost-sensitive learning methods can be applied.

Code

```
# Example of applying SMOTE for fraud detection
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# X and y represent features and target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Applying SMOTE to balance the data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)

# Training a logistic regression model
model = LogisticRegression()
model.fit(X_train_resampled, y_train_resampled)

# Evaluating the model
predictions = model.predict(X_test)
print(classification_report(y_test, predictions))
```

Output::

```
              precision    recall  f1-score   support

           0       0.98      0.87      0.92       175
           1       0.50      0.88      0.64        25

    accuracy                           0.88       200
   macro avg       0.74      0.88      0.78       200
weighted avg       0.92      0.88      0.89       200
```

Medical Diagnosis and Healthcare

- In medical diagnosis, imbalanced data can occur when certain diseases or conditions are rare.
- Handling imbalanced data here involves careful model evaluation and validation to ensure high sensitivity (recall) while maintaining specificity.
- Techniques like resampling or using specialized algorithms are employed.

Code

```
# Example of using a RandomForestClassifier for medical diagnosis
from sklearn.ensemble import RandomForestClassifier

# Assuming X_train, X_test, y_train, y_test are already defined

# Training a RandomForestClassifier on resampled data
model_rf = RandomForestClassifier()
model_rf.fit(X_train_resampled, y_train_resampled)

# Evaluating the model
predictions_rf = model_rf.predict(X_test)
print(classification_report(y_test, predictions_rf))
```

Output:

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       175
           1       0.73      0.76      0.75        25

    accuracy                           0.94       200
   macro avg       0.85      0.86      0.85       200
weighted avg       0.94      0.94      0.94       200
```

Best Practices and Considerations

- Before applying techniques to handle imbalanced data, it's crucial to preprocess data, handle missing values, normalize/standardize features, and perform relevant feature engineering to enhance model performance.

Code

```
from sklearn.preprocessing import StandardScaler

# Assuming 'X' is the feature matrix
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Output:

```
.
[[ 0.80771229 -0.0512384   0.83400646 ... -1.07534841 -0.48400939
   1.19972672]
 [ 1.17415307 -0.55076587  1.11875759 ... -0.86015097 -0.86502192
   0.18398476]
 [ 0.8139878  -0.41725692  0.63347169 ... -0.15611949 -0.84472881
  -0.32183468]
 ...
 [ 1.78723299  1.76862028  1.76809793 ...  0.73957884 -1.20429047
   0.19974679]
 [-1.59806858 -0.81983505 -1.00743221 ... -0.39548995  2.06576807
   0.84036974]
 [-0.18079474 -0.98744465 -0.14204087 ...  0.81637436  0.18531109
   1.38848155]]
```

Best PracticChallenges and Limitationses and Considerations

Overfitting in Oversampling

- Oversampling techniques might lead to overfitting on the minority class. Generating synthetic samples that are too close to existing ones may hinder the model's ability to generalize.

Code

```
#Applying SMOTE with controlled 'k_neighbors'
smote_controlled = SMOTE(random_state=42, k_neighbors=5)  # Set the number
of neighbors
```

```
X_train_resampled, y_train_resampled =
smote_controlled.fit_resample(X_train, y_train)
print(X_train_resampled)
```

Output:

```
[[-0.06392624 -0.22104246  0.18965365 ...  2.39210964  0.87489457
   0.19259692]
 [-0.55803047  0.09205514  0.02499572 ... -0.993593    2.30971442
   2.52992775]
 [ 0.53130739  0.26536235  0.37171417 ... -0.13427916 -0.90684461
  -2.43881716]
 ...
 [-1.02157497  1.53196252 -0.85818958 ...  1.99893054  1.27486967
  -0.46767805]
 [-0.21215234  1.48329679 -0.11777546 ...  1.18757793  0.46225122
   0.38232306]
 [-0.23993103 -1.44392766 -0.22744763 ... -1.34811201  0.21483049
  -0.27017075]]
```

Tools and Libraries

- Libraries like “imbalanced-learn” provide various techniques for handling imbalanced data, including resampling methods, cost-sensitive learning, and ensemble techniques tailored for imbalanced datasets.

Code

```
# Install imbalanced-learn if not installed
!pip install imbalanced-learn

# Importing SMOTE from imbalanced-learn
from imblearn.over_sampling import SMOTE

# Using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

Output:

```
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.10/dist-packages (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
print(X_train_resampled)
```

Output:

```
[[-0.06392624 -0.22104246  0.18965365 ...  2.39210964  0.87489457
   0.19259692]
 [-0.55803047  0.09205514  0.02499572 ... -0.993593    2.30971442
   2.52992775]
 [ 0.53130739  0.26536235  0.37171417 ... -0.13427916 -0.90684461
  -2.43881716]
 ...
 [-1.02157497  1.53196252 -0.85818958 ...  1.99893054  1.27486967
  -0.46767805]
 [-0.21215234  1.48329679 -0.11777546 ...  1.18757793  0.46225122
   0.38232306]
 [-0.23993103 -1.44392766 -0.22744763 ... -1.34811201  0.21483049
  -0.2701707511
```