Tackling Class Imbalance in Machine Learning: A Hands-On Guide with SMOTE

Introduction: Class imbalance is a common challenge in machine learning where the distribution of classes in a dataset is uneven. This can lead to biased models that favor the majority class, impacting the model's ability to generalize well to minority classes. In this blog post, we'll explore the issue of class imbalance and delve into a practical solution using the Synthetic Minority Over-sampling Technique (SMOTE).

Understanding Class Imbalance: Before we delve into the solution, let's understand the implications of class imbalance. In a binary classification problem, if one class significantly outnumbers the other, the model may become biased towards predicting the majority class. This is problematic, especially when the minority class holds crucial information or is of particular interest.

Oversampling for Class Imbalance: Oversampling, specifically SMOTE, involves increasing minority class instances. This balances class distribution, ensuring the model receives adequate minority class examples for accurate predictions.

Overview of Oversampling:

1. Identify Minority Class: Identify the minority class in your dataset.
2. Choose Technique: Explore various oversampling techniques, including SMOTE, ADASYN, and others, selecting one that aligns with your dataset and problem.
3. Apply to Training Data: Generate synthetic samples for the minority class in the training dataset.
4. Train the Model: Use oversampled data to train your machine learning model.

To illustrate the concepts discussed, let's explore a customer churn project, a common problem faced by businesses addressing class imbalance using SMOTE. The dataset contains various features such as gender, seniority, contract type, and others, with the target variable being whether a customer has churned or not.

Steps Involved

1. Data Loading and Exploration
2. Visualization
3. Data Cleaning
4. Training the Dataset
5. Handling Imbalance with SMOTE
6. Building a RandomForest Model
7. Model Evaluation

1. Data Loading and Exploration: The dataset under consideration is loaded from a CSV file named 'customer_churn.csv'. The journey begins with loading the dataset and performing exploratory data analysis (EDA).

1a. Import the required libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
!pip install -U scikit-learn
!pip install -U imbalanced-learn
```

1b. Load the data set

```
df = pd.read_csv('customer_churn.csv')
df.info()
print(df.head())
```

1c. Exploring Class Distribution: The first challenge in any classification problem is understanding the distribution of classes. In the case of customer churn prediction, the 'Churn' column is of particular interest.

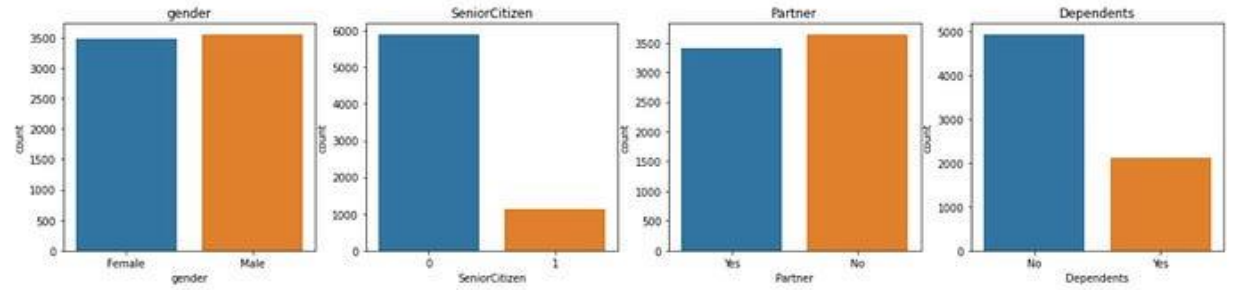
```
df['Churn'].value_counts()
```

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

2. Visualizing: Visualizations such as count plots and box plots provide insights into the distribution of data and potential patterns.

2a. Categorical Features: Understanding the impact of categorical features on churn is essential. We focus on visualizing 'gender', 'SeniorCitizen', 'Partner', and 'Dependents' using count plots.

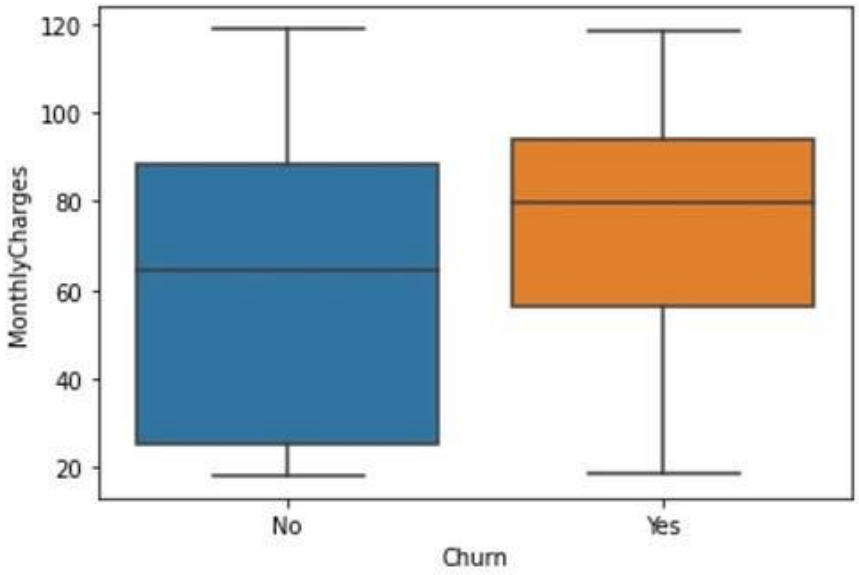
```
cols = ['gender','SeniorCitizen',"Partner","Dependents"]
numerical = cols

plt.figure(figsize=(20,4))

for i, col in enumerate(numerical):
    ax = plt.subplot(1, len(numerical), i+1)
    sns.countplot(x=str(col), data=df)
    ax.set_title(f"{col}")
```



2b. Analyzing Numeric Features: Numeric features like 'MonthlyCharges' and their relationship with churn are explored using a box plot.

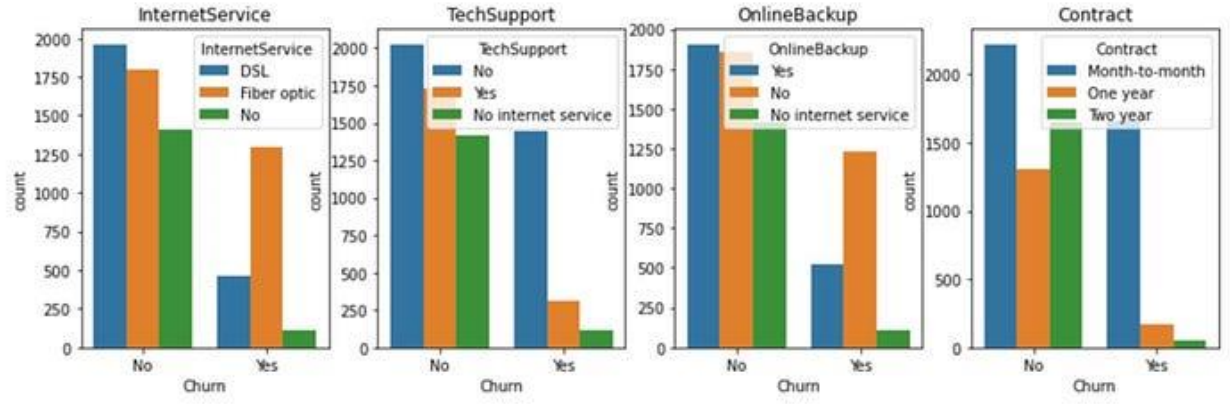
```
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
```

```
<AxesSubplot:xlabel='Churn', ylabel='MonthlyCharges'>
```



2c. Exploring Service-related Features: The impact of services such as 'InternetService', 'TechSupport', 'OnlineBackup', and 'Contract' on churn is visualized using count plots.

```
cols = ['InternetService',"TechSupport","OnlineBackup","Contract"]

plt.figure(figsize=(14,4))

for i, col in enumerate(cols):
    ax = plt.subplot(1, len(cols), i+1)
    sns.countplot(x ="Churn", hue = str(col), data = df)
    ax.set_title(f"{col}")
```



3. Data Cleaning: In the pursuit of building a robust model, it's essential to preprocess the data, ensuring a clean dataset suitable for machine learning algorithms.

3a. Label Encoding: To prepare the data for modeling, 'TotalCharges' is converted to numeric.

3b. Categorical features are label-encoded before using the machine learning model.

3c. Merging Numeric and Categorical Features: The numeric and label-encoded categorical features are merged into a final data frame for modeling.

```
df['TotalCharges'] = df['TotalCharges'].apply(lambda x: pd.to_numeric(x, errors='coerce')).dropna()
```

```
cat_features = df.drop(['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure'],axis=1)
cat_features.head()

from sklearn import preprocessing

le = preprocessing.LabelEncoder()
df_cat = cat_features.apply(le.fit_transform)
df_cat.head()
```

```
num_features = df[['customerID','TotalCharges','MonthlyCharges','SeniorCitizen','tenure']]
finaldf = pd.merge(num_features, df_cat, left_index=True, right_index=True)
finaldf
```

4. Training the Dataset: Before oversampling, we perform a train-test split on the dataset. Oversampling will be done on the training dataset, as the test dataset represents the true population.

```
from sklearn.model_selection import train_test_split

finaldf = finaldf.dropna()
finaldf = finaldf.drop(['customerID'],axis=1)

X = finaldf.drop(['Churn'],axis=1)
y = finaldf['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
y_train.value_counts()
```

5. Handling Imbalance with SMOTE: Recognizing the imbalance in the class distribution, SMOTE is employed to generate synthetic samples for the minority class, leveling the playing field for the machine learning model.

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE(k_neighbors=5)
X_smote, y_smote = oversample.fit_resample(X_train, y_train)
X_train, y_train = X_smote, y_smote
y_train.value_counts()
```

6. Building a RandomForest Model: With the dataset preprocessed and class imbalance mitigated, the next steps involve training a machine learning model (Random Forest Classifier in this case) and evaluating its performance.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=46)
rf.fit(X_train,y_train)
```

7. Model Evaluation: Finally, the trained model is evaluated using accuracy as the metric.

```
from sklearn.metrics import accuracy_score

preds = rf.predict(X_test)
print(accuracy_score(preds,y_test))
```

```
0.7703576044808272
```

Conclusion: This blog post has provided insights into the challenges posed by a class imbalance in machine learning datasets and offered a practical solution using SMOTE. Addressing class imbalance is crucial for creating models that make informed predictions across all classes, ensuring a fair and reliable outcome.