Data Analysis Report: Missing Values and Imbalanced Datasets

This report covers two important topics in data analysis: missing values in datasets and techniques for handling imbalanced datasets. We will explore examples using the Titanic dataset and synthetic datasets.

1. Missing Values in Datasets

Missing values occur in datasets when some information is not stored for a variable. There are three main mechanisms:

1.1 Missing Completely at Random (MCAR)

Missing completely at random (MCAR) is a type of missing data mechanism in which the probability of a value being missing is unrelated to both the observed data and the missing data. In other words, if the data is MCAR, the missing values are randomly distributed throughout the dataset, and there is no systematic reason for why they are missing.For example, in a survey about the prevalence of a certain disease, the missing data might be MCAR if the survey participants with missing values for certain questions were selected randomly and their missing responses are not related to their disease status or any other variables measured in the survey.

1.2 Missing at Random (MAR)

Missing at Random (MAR) is a type of missing data mechanism in which the probability of a value being missing depends only on the observed data, but not on the missing data itself. In other words, if the data is MAR, the missing values are systematically related to the observed data, but not to the missing data. Here are a few examples of missing at random:Income data: Suppose you are collecting income data from a group of people, but some participants choose not to report their income. If the decision to report or not report income is related to the participant's age or gender, but not to their income level, then the data is missing at random.Medical data: Suppose you are collecting medical data on patients, including their blood pressure, but some patients do not report their blood pressure. If the patients who do not report their blood pressure are more likely to be younger or have healthier lifestyles, but the missingness is not related to their actual blood pressure values, then the data is missing at random.

1.3 Missing Not at Random (MNAR)

It is a type of missing data mechanism where the probability of missing values depends on the value of the missing data itself. In other words, if the data is MNAR, the missingness is not random and is dependent on unobserved or unmeasured factors that are associated with the missing values.For example, suppose you are collecting data on the income and job satisfaction of employees in a company. If employees who are less satisfied with their jobs are more likely to refuse to report their income, then the data is not missing at random. In this case, the missingness is dependent on job satisfaction, which is not directly observed or measured.

2. Example with Titanic Dataset

We will use the Titanic dataset to demonstrate how to handle missing values.

Code Snippet: Load Dataset

```
import seaborn as sns
df = sns.load_dataset('titanic')
df.head()
```

This code snippet loads the Titanic dataset using the seaborn library and displays the first few rows.

Code Snippet: Check for Missing Values

```
df.isnull().sum()
```

This code checks for missing values in each column of the dataset.

Code Snippet: Delete Rows with Missing Values

```
df.dropna().shape
```

This code snippet deletes rows with any missing values and shows the new shape of the dataset.

3. Imputation Techniques

We can handle missing values using various imputation techniques:

3.1 Mean Value Imputation

```
df['age_mean'] = df['age'].fillna(df['age'].mean())
```

This code fills missing values in the 'age' column with the mean age.

3.2 Median Value Imputation

```
df['age_median'] = df['age'].fillna(df['age'].median())
```

This code fills missing values in the 'age' column with the median age, which is useful in the presence of outliers.

3.3 Mode Imputation for Categorical Values

```
df['embarked'].unique()

mode_value = df[df['embarked'].notna()]['embarked'].mode()[0]

df['embarked_mode'] = df['embarked'].fillna(mode_value)
```

This code fills missing values in the 'embarked' column with the mode (most frequent value).

4. Handling Imbalanced Datasets

Imbalanced datasets can lead to biased models. We will explore techniques for handling such datasets, specifically focusing on upsampling and downsampling methods.

4.1 Creating an Imbalanced Dataset

We start by creating a synthetic dataset with two classes, where one class is significantly larger than the other.

Code Snippet: Create Imbalanced Dataset

```
import numpy as np
import pandas as pd

# Set the random seed for reproductivity
np.random.seed(123)

# Create a dataframe with two classes
n_samples = 1000
class_0_ratio = 0.9
n_class_0 = int(n_samples * class_0_ratio)
n_class_1 = n_samples - n_class_0

class_0 = pd.DataFrame({
    'feature_1' : np.random.normal(loc=0, scale=1, size=n_class_0),
    'feature_2' : np.random.normal(loc=0, scale=1, size=n_class_0),
    'target' : [0] * n_class_0
})

class_1 = pd.DataFrame({
    'feature_1': np.random.normal(loc=2, scale=1, size=n_class_1),
    'feature_2': np.random.normal(loc=2, scale=1, size=n_class_1),
    'target': [1] * n_class_1
})

df = pd.concat([class_0, class_1]).reset_index(drop=True)
```

In this code, we create a dataset with 1000 samples, where 900 belong to class 0 and 100 belong to class 1. The features are generated using a normal distribution.

4.2 Checking Class Distribution

We can check the distribution of the classes in our dataset.

Code Snippet: Check Class Distribution

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

This code snippet shows the distribution of the target classes in the dataset, confirming the imbalance.

4.3 Upsampling

Upsampling involves increasing the number of instances in the minority class to balance the dataset.

Code Snippet: Upsampling

```
from sklearn.utils import resample

df_minority = df[df['target'] == 1]
df_majority = df[df['target'] == 0]

df_minority_upsampled = resample(df_minority, replace=True,
                                 n_samples=len(df_majority),
                                 random_state=42)

df_upsampled = pd.concat([df_majority, df_minority_upsampled])
df_upsampled['target'].value_counts()
```

In this code, we use the `resample` function from `sklearn.utils` to upsample the minority class. We then concatenate the upsampled minority class with the majority class to create a balanced dataset.

4.4 Downsampling

Downsampling involves reducing the number of instances in the majority class to balance the dataset.

Code Snippet: Downsampling

```
df_majority_downsampled = resample(df_majority, replace=False,
                                   n_samples=len(df_minority),
                                   random_state=42)

df_downsampled = pd.concat([df_minority, df_majority_downsampled])
df_downsampled['target'].value_counts()
```

This code snippet demonstrates how to downsample the majority class. We again use the `resample` function, but this time we set `replace=False` to ensure we are not sampling with replacement.

5. Conclusion

Handling missing values and imbalanced datasets is crucial for effective data analysis and model training. Understanding the mechanisms behind missing data and applying appropriate techniques for imbalanced datasets can significantly improve the quality of the analysis.

Glossary

Missing Values: Data entries that are not recorded or are absent in a dataset.

Imbalanced Dataset: A dataset where the classes are not represented equally, leading to potential bias in model training.

Upsampling: A technique used to increase the number of instances in the minority class to achieve a balanced dataset.

Downsampling: A method that reduces the number of instances in the majority class to balance the dataset.

Synthetic Dataset: A dataset created artificially, often used for testing and validation purposes.

Resampling: The process of randomly selecting samples from a dataset to create a new dataset, which can be done with or without replacement.

Class Distribution: The proportion of different classes within a dataset, which is crucial for understanding the balance of the dataset.

SMOTE (Synthetic Minority Over-sampling Technique)

This report covers the SMOTE technique, which is used in machine learning to address imbalanced datasets. SMOTE generates synthetic instances of the minority class by interpolating between existing instances.

1. Generating a Sample Dataset

We will first create a synthetic dataset using the `make_classification` function from the `sklearn.datasets` module.

Code Snippet: Generate Sample Dataset

```
from sklearn.datasets import make_classification
X, Y = make_classification(n_samples=1000, n_redundant=0, n_features=2,
                           n_clusters_per_class=1, weights=[0.90],
                           random_state=12)
import pandas as pd
df1 = pd.DataFrame(X, columns=['f1', 'f2'])
df2 = pd.DataFrame(Y, columns=['target'])
final_df = pd.concat([df1, df2], axis=1)
final_df.head()
```

In this code, we generate a dataset with 1000 samples, where 90% belong to one class (majority) and 10% to another (minority). The features are stored in a DataFrame, and we can visualize the first few rows using `head()`.

2. Visualizing the Dataset

Next, we visualize the dataset to understand the class distribution.

Code Snippet: Visualize Dataset

```
import matplotlib.pyplot as plt
```

```
plt.scatter(final_df['f1'], final_df['f2'], c=final_df['target'])
plt.title('Original Dataset Visualization')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

The scatter plot above shows the distribution of the two classes in the feature space. The majority class (0) is represented by one color, while the minority class (1) is represented by another.

3. Applying SMOTE

To address the class imbalance, we will apply the SMOTE technique using the `imblearn` library.

Code Snippet: Apply SMOTE

```
!pip install imblearn
from imblearn.over_sampling import SMOTE

# Transform the dataset
oversample = SMOTE()
X, Y = oversample.fit_resample(final_df[['f1', 'f2']], final_df['target'])
X.shape, Y.shape
```

In this code, we first install the `imblearn` library if it is not already installed. Then, we apply SMOTE to the dataset, which generates synthetic samples for the minority class, resulting in a balanced dataset with equal instances of both classes.

4. Visualizing the Oversampled Dataset

After applying SMOTE, we can visualize the new dataset to confirm that the classes are now balanced.

Code Snippet: Visualize Oversampled Dataset

```
df1 = pd.DataFrame(X, columns=['f1', 'f2'])
df2 = pd.DataFrame(Y, columns=['target'])
oversample_df = pd.concat([df1, df2], axis=1)
plt.scatter(oversample_df['f1'], oversample_df['f2'], c=oversample_df['target'])
plt.title('Oversampled Dataset Visualization')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

The scatter plot above illustrates the oversampled dataset, where both classes are now represented equally. This balance is crucial for training machine learning models effectively.

Conclusion

In this report, we explored the SMOTE technique for handling imbalanced datasets. We generated a synthetic dataset, visualized it, applied SMOTE to balance the classes, and visualized the results. This technique is essential for improving the performance of machine learning models on imbalanced data.

SMOTE Glossary

SMOTE: Synthetic Minority Over-sampling Technique, a method to generate synthetic samples for the minority class in imbalanced datasets.

Synthetic Sample: An artificially created instance that is generated based on existing instances in the minority class.

Interpolation: A method used in SMOTE to create new samples by combining features of existing samples.

Minority Class: The class in a dataset that has fewer instances compared to the majority class.

Majority Class: The class in a dataset that has more instances compared to the minority class.