Smt. Indira Gandhi College of Engineering
Computer Engineering Department
Ghansoli – Navi Mumbai
Academic Year 2023-24 (Even Sem)

Student Name: Khyati Garude **Roll No.:** 13 **C lass:** BE **Sem:** VIII

Course Name: Applied Data Science Lab

Course Code: CSL8023

Experiment No. 08

Experiment Title: Implementing Smote Technique To Generate The Synthetic Data

Date of Performance	Date of Submission	Marks (10)					Sign / Remark
		A	B	C	D	E	
		2	3	2	2	1	
14/3/24	21/3/24	2	3	2	2	1	12/4/24
		Total Marks					
		10					

A: Prerequisite Knowledge
B: Implementation
C: Oral
D: Content
E: Punctuality & Discipline

Khyati Garude
BE COMPS
ADS

Date:

DATE	EXPERIMENT-8	SIGN
21/3/24	Implementing SMOTE technique to generate the synthetic data.	21/4/24

AIM: Implementing SMOTE technique to generate the synthetic data.

THEORY:

In many real-world datasets, the classes are not evenly distributed, leading to a class imbalance problem. This can result in biased models that perform poorly on minority class samples.

Synthetic Minority over-Sampling Technique (SMOTE) is a popular technique used to address class imbalance by generating synthetic samples for the minority class. It works by synthesizing new instances of minority class samples, thereby increasing their representation in the dataset.

SMOTE uses linear interpolation to generate synthetic instances. For each feature dimension, the value of the synthetic instance is calculated as a weighted sum of the values

Date:_____

of the original instance and its selected neighbor.

→ Advantages:

→ SMOTE helps to address class imbalance by increasing the representation of minority class samples, leading to better model performance.

→ It mitigates the risk of model bias towards the majority class by ensuring that the classifier is exposed to a more balanced distribution of classes.

→ Limitations:

→ SMOTE may not perform well when the minority class is densely packed or when the nearest neighbors of a minority sample are from the majority classes.

→ It can potentially introduce noise into the dataset if the synthetic samples are generated irresponsibly.

SMOTE can be combined with other techniques, such as under-sampling or cost-sensitive learning, to further enhance its effectiveness in handling class imbalance.

Date:

ALGORITHM:

(i) Load dataset:
Load the dataset containing the target variable ('sleep duration' in this case).

(ii) Encode target variable:
Encode the target variable into binary classes ('Adequate' and 'Inadequate' sleep) based on a threshold value (7 hours in this case) using LabelEncoder.

(iii) Count class distribution before SMOTE:
Count the number of samples in each class ('Adequate' and 'Inadequate' sleep) before applying SMOTE.

(iv) Plot Target variable distribution before SMOTE:
Visualize the distribution of the target variable before applying SMOTE using a Histogram.

(v) Apply SMOTE:
Use SMOTE algorithm to oversample the minority class ('Inadequate' sleep) to balance the class distribution.

(vi) Count class distribution after SMOTE:
Count the number of samples in each class after applying SMOTE.

Date:_____

(vii) Plot target variable distribution after SMOTE:
Visualize the distribution of the target variable after applying SMOTE using a histogram.

(viii) Display results:
Display the counts of each class before and after SMOTE to observe the effect of oversampling on the class distribution.

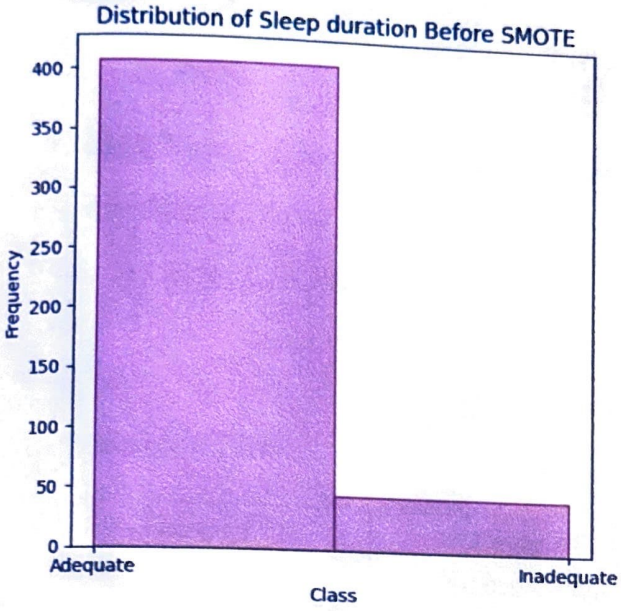
(ix) Plot Comparisons:
Plot the distribution of the target variable before and after SMOTE to visualize the balance achieved by the oversampling technique.

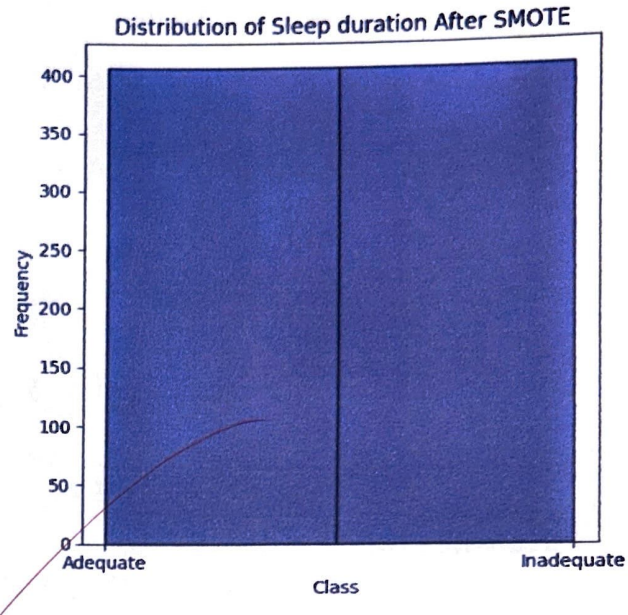
CONCLUSION:
SMOTE is a valuable tool in the machine learning practitioner's toolkit for addressing class imbalance and improving the robustness of models trained on imbalanced datasets. Overall, SMOTE is a powerful technique for handling class imbalance in datasets, but it's essential to understand its underlying principles and limitations when applying it to real-world problems.

1/4/24

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE

# Load the dataset
data = pd.read_csv("/content/Sleep_Efficiency.csv")

# Display the first few rows of the dataset to understand its structure
#print(data.head())

# Assuming 'Sleep duration' is the target variable
y = data['Sleep duration']  # Target variable

# Encoding the target variable
label_encoder = LabelEncoder()
y_binary = y.apply(lambda x: 'Adequate' if x >= 7 else 'Inadequate')
y_encoded = label_encoder.fit_transform(y_binary)

# Count of each class before SMOTE
class_labels = label_encoder.classes_
class_counts_before = pd.Series(y_encoded).value_counts().to_dict()
class_counts_before = {class_labels[i]: class_counts_before[i] for i in range(len(class_labels))}
print("\nCount of each class before SMOTE:")
print("Adequate sleep:", class_counts_before['Adequate'])
print("Inadequate sleep:", class_counts_before['Inadequate'])

# Plot the distribution of the target variable before SMOTE
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.hist(y_encoded, bins=2, color='lightpink', edgecolor='brown')
plt.title('Distribution of Sleep duration Before SMOTE')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks(range(2), label_encoder.classes_)

# Apply SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(data[['Sleep duration']], y_encoded)

# Convert resampled data back to DataFrame
X_resampled = pd.DataFrame(X_resampled, columns=['Sleep duration'])

# Count of each class after SMOTE
class_counts_after = pd.Series(y_resampled).value_counts().to_dict()
class_counts_after = {class_labels[i]: class_counts_after[i] for i in range(len(class_labels))}
print("\nCount of each class after SMOTE:")
print("Adequate sleep:", class_counts_after['Adequate'])
print("Inadequate sleep:", class_counts_after['Inadequate'])

# Plot the distribution of the target variable after SMOTE
plt.subplot(1, 2, 2)
plt.hist(y_resampled, bins=2, color='grey', edgecolor='black')
plt.title('Distribution of Sleep duration After SMOTE')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks(range(2), label_encoder.classes_)

plt.tight_layout()
plt.show()
```

```
Count of each class before SMOTE:
Adequate sleep: 407
Inadequate sleep: 45

Count of each class after SMOTE:
Adequate sleep: 407
Inadequate sleep: 407
```





12/4/24